**MySQL UCASE and LCASE Function with Examples**

**MySQL UCASE() Function**

The MySQL UCASE function converts string data values to uppercase. The UCASE function only works on string columns. Following is the syntax to use the UCASE function in MySQL.

**SELECT UCASE(columnname) FROM tablename;**  
**SELECT UCASE(‘String Value’);**

We need to pass the string column name to the UCASE function whose values we want in the Upper case. The changes will only affect the result set, it will not affect the original data present in the database table.

**Example: Converting Employee Name and City to Upper Case**

Let us see an example that will convert the employee’s name and city to upper case. Following is the SQL query which will convert the Name and City of Employees to upper case using the UCASE function.

**SELECT Id, UCASE(Name) as Name, Department, Salary, UCASE(City) AS CITY FROM Employee;**

Once you execute the above query, you will get the following output. Notice that the Name and City column data are in upper case.

**MySQL MID Function with Examples**

**Parameter**

1. **string**: Required. The string to extract from
2. **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
3. **length**: Optional. The number of characters to extract. If not specified, then it will extract all the characters from the start position.

**Example:** Extract a substring from a string starting from position 7 and extract 3 characters  
**SELECT MID(“MySQL Tutorials”, 7, 3) AS ExtractString;**  
**Output: Tut**

**MySQL LENGTH and CHAR\_LENGTH Function with Examples**

Mathematical Functions

1. CREATE DATABASE mathDB;

CREATE TABLE product\_information (

product\_id int,

product\_name varchar(30),

product\_type varchar(30),

total\_inventory int(200),

product\_cost decimal(3, 2),

product\_retail decimal(3, 2),

store\_units int(100),

online\_units int(100),

PRIMARY KEY (product\_id)

);

INSERT INTO product\_information

(product\_id, product\_name, product\_type, total\_inventory, product\_cost, product\_retail, store\_units, online\_units)

VALUES

(1, 'chamomile', 'tea', 200, 5.12, 7.50, 38, 52),

(2, 'chai', 'tea', 100, 7.40, 9.00, 17, 27),

(3, 'lavender', 'tea', 200, 5.12, 7.50, 50, 112),

(4, 'english\_breakfast', 'tea', 150, 5.12, 7.50, 22, 74),

(5, 'jasmine', 'tea', 150, 6.17, 7.50, 33, 92),

(6, 'matcha', 'tea', 100, 6.17, 7.50, 12, 41),

(7, 'oolong', 'tea', 75, 7.40, 9.00, 10, 29),

(8, 'tea sampler', 'tea', 50, 6.00, 8.50, 18, 25),

(9, 'ceramic teapot', 'tea item', 30, 7.00, 9.75, 8, 15),

(10, 'golden teaspoon', 'tea item', 100, 2.00, 5.00, 18, 67);

**Calculating with Mathematical Expressions**

In SQL, you typically use SELECT to query your database and retrieve the desired result set. However, you can also use the SELECT keyword to perform a variety of mathematical operations.

Keep in mind that in a real-life scenario, SQL is primarily used to query and make calculations from values in your actual database. But for this section, you’ll use SELECT solely for numerical values to get familiar with the syntax of mathematical expressions and operators.

Before you begin, here’s an overview of the operators you can use to perform six arithmetic operations in SQL. Please note this list is not comprehensive and that many RDBMSs have a unique set of mathematical operators:

Addition uses the + symbol

Subtraction uses the - symbol

Multiplication uses the \* symbol

Division uses the / symbol

Modulo operations use the % symbol

Exponentiation uses POW(x,y)

You can practice running different types of calculations with your own value combinations. We’ll demonstrate using the following examples, starting with an addition equation:

SELECT 893 + 579;

SELECT 437.82 - 66.34;

SELECT 60 \* 1234 \* 2 \* 117;

SELECT 2604.56 / 41;

SELECT 38 % 5;

SELECT POW(99,9);

**Understanding Order of Operations in SQL**

the term PEMDAS, which stands for parentheses, exponents, multiplication, division, addition, and subtraction. This term serves as a guideline for the order of operations necessary to solve more complex equations. PEMDAS is the term used in the U.S., while other countries may use different acronyms to represent their order of operations rule.

When it comes to combining different mathematical operations nested within parentheses, SQL reads them left to right, and then values beginning from the inside to the outside. For this reason, ensure your values within parentheses accurately capture the problem you’re trying to solve.

Try a calculation using parentheses and a couple of different operators:

SELECT (2 + 4 ) \* 8;

Remember, parentheses placement matters and if you’re not careful, the entire result can change. For example, the following uses the same three values and operators, but with a different parentheses placement, this produces a different result:

SELECT 2 + (4 \* 8);

If you prefer to perform calculations without parentheses, you can do that as well. Remember that there is still the order of operations rule; therefore, similar to the parentheses placement, verify that this is the equation you want based on the operation order it will be evaluated upon. In the following example, you’ll notice that the division operation takes precedence over the subtraction operator and results in a negative value:

SELECT 100 / 5 - 300;

Applying Mathematical Expressions in a Business Scenario

This section provides a few example scenarios that involve performing data analysis to help the tea shop owners with decision-making as it relates to their business.

As a first scenario, calculate the total units currently available in inventory to understand how many products are remaining for in-store and online sales. This query will also include the DESC statement to organize the data from the largest to smallest amount. Typically RDBMSs will default to ascending order, but this example includes the DESC option which allows you to view the data in descending order:

SELECT product\_name,

total\_inventory - (store\_units + online\_units)

AS remaining\_inventory

FROM product\_information

ORDER BY(remaining\_inventory) DESC;

This query is useful because it calculates the remaining inventory, which can help the tea shop owners make plans to purchase more orders if they’re running low on a product.

For the next scenario, you’ll analyze and compare the amount of revenue from in-store and online sales:

SELECT product\_name,

(online\_units \* product\_retail) AS o,

(store\_units \* product\_retail) AS s

FROM product\_information;

This query is useful because it calculates the remaining inventory, which can help the tea shop owners make plans to purchase more orders if they’re running low on a product.

For the next scenario, you’ll analyze and compare the amount of revenue from in-store and online sales:

SELECT product\_name,

(online\_units \* product\_retail) AS o,

(store\_units \* product\_retail) AS s

FROM product\_information; SELECT product\_name,

(online\_units \* product\_retail) AS o,

(store\_units \* product\_retail) AS s

FROM product\_information;

Next, calculate the total revenue from in-store and online sales using the SUM function and several mathematical operators:

SELECT SUM(online\_units \* product\_retail) +

SUM(store\_units \* product\_retail)

AS total\_sales

FROM product\_information;

Performing these queries is important for two reasons. The first reason is so that the tea shop owners can evaluate which items are best-sellers and prioritize those products when purchasing more in the future. Second, they can analyze how well the tea shop performed overall with products sales in-store and online.

Next, you’ll find the profit margin for each product. The *profit margin* for a given product is the amount of revenue that the business gains for each unit of that product that it sells. To understand how much revenue you earned, you can multiply the sales by the profit margin.

To calculate the profit margin for your individual products, subtract product\_cost from product\_retail for each row. Then divide this value by the product retail to calculate the profit margin percentage:

SELECT product\_name,

(product\_retail - product\_cost) / product\_retail

AS profit\_margin

FROM product\_information;

Based on this output, you’ll learn that the product with the highest profit margin is the golden teaspoon at 60%, and the lowest is for the Chai, Jasmine, Matcha, and Oolong teas at 18%. For the golden teaspoon, this means at a retail value of $5.00 with a profit margin of 60%, you create $3.00 in revenue.

You could also use the aggregate function AVG to calculate the average profit margin for all of the tea shop’s products. This average serves as a benchmark for the tea shop owners to then identify what products fall below that number and strategize how to improve:

SELECT AVG((product\_retail - product\_cost) / product\_retail)

AS avg\_profit\_margin

FROM product\_information;

From this calculation, you can conclude that the mean profit margin for products at this tea shop is 28%.

With this new information, imagine that the tea shop owners want to increase the profit margin to 31% in the next quarter for any products that currently have a profit margin less than 27%. In order to do this, you’ll subtract your target profit margin from 1 (1 - 0.31) and then divide each of the returned products’ costs by this value. The result will be the new price that the product must sell for at retail to achieve a 31% profit margin:

SELECT product\_name, product\_cost / (1 - 0.31)

AS new\_retail

FROM product\_information

WHERE (product\_retail - product\_cost) / product\_retail < 0.27;

These results display the new retail prices necessary for under-performing products to achieve a 31% profit margin. Data analysis such as this equips the tea shop owners with the ability to make decisive business decisions about how to improve their revenue for the next quarter and understand what to aim for.