# MYSQL Built-In Functions

MySQL offers a rich set of built-in functions that you can leverage to manipulate data, perform calculations, and extract insights within your queries. These functions can be broadly categorized into different groups based on their functionality:

## 1. Conversion Function in MySQL :

There are two main conversion functions in MySQL:

### A. CONVERT() :

- The CONVERT() function in MySQL is primarily used for character set conversions.
- It can also be used to convert data types, but its primary purpose is to convert strings from one character set to another.

Syntax : CONVERT(expression, data_type)

Example 01 : Write an SQL query to fetch the payment ID, customer name, and the payment date in a date format from the payments table.

```
SELECT payment_id, customer_name, CONVERT(payment_date,
DATE) AS payment_date_date
FROM payments;
```

Example 02 : Write an SQL query to retrieve the payment ID, customer name, and the amount of each payment converted to a string representation from the payments table.

```
SELECT payment_id, customer_name, CONVERT(amount, CHAR)
AS amount_string
FROM payments;
```

B. CAST():

- The CAST() function is used to explicitly convert a value from one data type to another in MySQL.
- It is more versatile than CONVERT() as it can handle a wider range of data type conversions.

Syntax: CAST(expr AS type)

Example 01 : Write an SQL query to fetch the payment ID, customer name, and the amount of each payment converted to a floating-point number from the payments table.

```
SELECT payment_id, customer_name, CAST(amount AS FLOAT)
AS amount_float
FROM payments;
```

## 2. Math Function in MySQL :

SQL Math functions are used primarily for numeric manipulation and/or mathematical calculations. The following table details the numeric functions -

1. ABS(expression):

This function returns the absolute value (non-negative version) of the provided expression.

Example:

```
SELECT customer_name, ABS(amount) AS absolute_amount
FROM payments;
```

2. ROUND(expression, decimal_places):

This function rounds a number to a specified number of decimal places.

Example:

```
SELECT customer_name, ROUND(amount, 2) AS rounded_amt
FROM payments;
```

3. CEILING(expression):

This function returns the smallest integer value that is greater than or equal to the provided expression.

Example:

```
SELECT customer_name, CEILING(amount) AS ceiling_amount
FROM payments;
```

4. FLOOR(expression):

This function returns the largest integer value that is less than or equal to the provided expression.

Example:

```
SELECT customer_name, FLOOR(amount) AS floor_amount
FROM payments;
```

5. SQRT(expression):

This function returns the square root of a non-negative number.

Example: (Assuming no negative amounts):

```
SELECT customer_name, SQRT(amount) AS square_root
FROM payments
WHERE amount >= 0; -- Filter for non-negative amounts
only (square root is undefined for negatives)
```

## 3. String Function in MySQL :

String functions in MySQL are used to manipulate string values stored in columns of a table. These functions allow you to perform tasks such as concatenating strings, extracting substrings, changing the case of strings, searching for specific patterns, and more.

### I. CONCAT():

Concatenates multiple strings (str1, str2, ...) into a single string.

Example:

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM suppliers;
```

### II. UCASE(str) / UPPER(str):

Converts all characters in str to uppercase.

Examples:

```
SELECT UPPER(first_name) AS first_name_upper,
UPPER(last_name) AS last_name_upper FROM suppliers;

SELECT UCASE(first_name) AS first_name_upper,
UCASE(last_name) AS last_name_upper FROM suppliers;
```

### III. LCASE(str) / LOWER(str):

Converts all characters in str to lowercase.

Example:

```
SELECT LOWER(first_name) AS first_name_lower,
LOWER(last_name) AS last_name_lower
FROM suppliers;

SELECT LCASE(first_name) AS first_name_lower,
LCASE(last_name) AS last_name_lower
FROM suppliers;
```

IV. TRIM(str):

Removes leading and trailing whitespace characters from str.

Examples:

```
SELECT TRIM(last_name) AS trimmed_last_name
FROM suppliers;

SELECT TRIM(first_name) AS trimmed_first_name
FROM suppliers;
```

V. SUBSTRING(str, start, length):

Extracts a substring from str starting at position start (1-based) with a length of length characters.

Example:

```
SELECT phone, SUBSTRING(phone, 5, 3) AS middle_number
FROM suppliers;

SELECT SUBSTRING(supplier_code, 3, 5) AS middle_code
FROM suppliers;
```

VI. LEFT(str, length):

Returns the leftmost length characters from str.

Example:

```
SELECT LEFT(supplier_code, 4) AS short_supplier_code
FROM suppliers;
```

VII. Right(str, length):

Returns the rightmost length characters from str.

Example:

```
SELECT RIGHT(supplier_code, 4) AS short_supplier_code
FROM suppliers;
```

### VIII. REPLACE(str, old_substr, new_substr):

Replaces all occurrences of old_substr in str with new_substr.

Example:

```
SELECT customer_name, REPLACE(customer_name, 'John',
'Jane') AS replaced_name
FROM payments;

SELECT REPLACE(email, 'example.com', 'mycompany.com')
AS modified_email FROM suppliers;
```

## 4. Date Function in MySQL :

MySQL provides functions to work with dates and extract meaningful information from them. Here are some commonly used functions:

### A. CURDATE() / CURRENT_DATE:

Returns the current date.

Example:

```
SELECT * FROM payments
WHERE payment_date = CURDATE();

SELECT * FROM payments
WHERE payment_date = CURRENT_DATE();
```

### B. DATE(expression):

Converts a string or datetime expression to a DATE value.

Example:

```
SELECT customer_name, DATE(payment_date) AS std_date
FROM payments;
```

C. YEAR(date):

Extracts the year from a date value.

Example:

```
SELECT * FROM payments
WHERE YEAR(payment_date) = 2023;
```

D. MONTH(date):

Extracts the month (1-12) from a date value.

Example:

```
SELECT customer_name, MONTH(payment_date) AS
payment_month, YEAR(payment_date) AS payment_year
FROM payments;
```

E. DAY(date):

Extracts the day of the month (1-31) from a date value.

Example:

```
SELECT customer_name, DAY(payment_date) AS payment_day
FROM payments;
```

F. DATE_ADD(date, interval):

Adds a specified time interval to a date.

Example:

```
SELECT customer_name, payment_date,
DATE_ADD(payment_date, INTERVAL 7 DAY) AS due_date
FROM payments;
```

G. DATE_SUB(date, interval):

Subtracts a specified time interval from a date.

Example:

```
SELECT * FROM payments
WHERE payment_date >= DATE_SUB(CURDATE(), INTERVAL 30
DAY);
```

H. DATEDIFF(date1, date2):

Calculates the difference between two dates in days

Example:

```
SELECT customer_name, DATEDIFF(CURDATE(), created_at)
AS payment_age_days
FROM payments
```