07 MySQL Scalar Functions

Introduction to MySQL Scalar Functions

Scalar functions in MySQL are built-in functions that operate on a single value and return a single value. Unlike aggregate functions (which work on multiple rows), scalar functions process data on a row-by-row basis. These functions are essential tools for data manipulation, transformation, and formatting in SQL queries.

1. String Functions

String functions manipulate text data, allowing you to transform, extract, and format string values.

CONCAT() - Combining Strings

The `CONCAT()` function joins two or more strings together.

```sql
-- Combine programmer name with their primary programming language
SELECT CONCAT(Programmer_Name, ' programs in ', Primary_Language) AS
Programmer_Info
FROM programmers;

-- Example output:
-- "Tony Stark programs in Python"
-- "Peter Parker programs in JavaScript"
-- etc.
```

CONCAT_WS() - Combining with Separator

The `CONCAT_WS()` function (Concatenate With Separator) joins strings with a specified separator.

```sql
-- Create a list of languages a programmer knows
SELECT Programmer_Name,
       CONCAT_WS(', ', Primary_Language, Secondary_Language) AS Known_Languages
FROM programmers;

-- Example output:
-- "Tony Stark | Python, JavaScript"
-- "Diana Prince | Ruby, Python"
```

UPPER() and LOWER() - Changing Case

These functions convert strings to uppercase or lowercase.

```sql
-- Convert programmer names to uppercase
SELECT UPPER(Programmer_Name) AS Programmer_Upper, Primary_Language
FROM programmers;

-- Convert languages to lowercase
SELECT Programmer_Name, LOWER(Primary_Language) AS language_lower
FROM programmers;
```

SUBSTRING() - Extracting Parts of Strings

The `SUBSTRING()` function extracts a portion of a string.

```sql
-- Extract the first 4 characters of programmer names
SELECT Programmer_Name,
       SUBSTRING(Programmer_Name, 1, 4) AS Short_Name
FROM programmers;

-- Example output:
-- "Tony Stark | Tony"
-- "Peter Parker | Pete"
```

LENGTH() - String Length

The `LENGTH()` function returns the length of a string in bytes.

```sql
-- Find the length of programmer names
SELECT Programmer_Name,
       LENGTH(Programmer_Name) AS Name_Length
FROM programmers
ORDER BY Name_Length DESC;

-- Find programmers with short names (less than 10 characters)
SELECT Programmer_Name
FROM programmers
WHERE LENGTH(Programmer_Name) < 10;
```

TRIM(), LTRIM(), RTRIM() - Removing Whitespace

These functions remove whitespace from the beginning and/or end of a string.

```sql
-- In a real scenario, you might need to clean up imported data
SELECT TRIM('  Python  ') AS Trimmed_String;  -- Returns "Python"

-- Clean up programming languages if they have extra spaces
SELECT Programmer_Name,
       TRIM(Primary_Language) AS Clean_Language
FROM programmers;
```

REPLACE() - Substituting Text

The `REPLACE()` function substitutes one string with another.

```sql
-- Change all occurrences of "Python" to "Python 3"
SELECT Programmer_Name,
       REPLACE(Primary_Language, 'Python', 'Python 3') AS Updated_Language
FROM programmers
WHERE Primary_Language = 'Python';
```

2. Numeric Functions

Numeric functions perform mathematical operations and transformations on numeric data.

ROUND() - Rounding Numbers

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

```sql
-- Round salaries to the nearest thousand
SELECT Programmer_Name,
       Salary,
       ROUND(Salary, -3) AS Rounded_Salary
FROM programmers;



select round(10200,-3);

-- Calculate and round average salary
SELECT ROUND(AVG(Salary), 2) AS Average_Salary
FROM programmers;
```

FLOOR() and CEILING() - Rounding Down and Up

`FLOOR()` rounds down to the nearest integer, while `CEILING()` rounds up.

```sql
-- Get the floor and ceiling of salaries divided by 1000
SELECT Programmer_Name,
       Salary,
       FLOOR(Salary/1000) AS Salary_Floor_K,
       CEILING(Salary/1000) AS Salary_Ceiling_K
FROM programmers;

-- Example output:
-- "Tony Stark | 15000 | 15 | 15"
-- "Peter Parker | 12000 | 12 | 12"
```

ABS() - Absolute Value

The `ABS()` function returns the absolute (positive) value of a number.

```sql
-- Calculate the absolute difference between a programmer's salary and the average salary
SELECT Programmer_Name,
       Salary,
       (SELECT AVG(Salary) FROM programmers) AS Avg_Salary,
       ABS(Salary - (SELECT AVG(Salary) FROM programmers)) AS Absolute_Difference
FROM programmers
ORDER BY Absolute_Difference;
```

MOD() - Modulus

The `MOD()` function returns the remainder of one number divided by another.

```sql
-- Find programmers with odd or even employee IDs (assuming ID is the position in the result set)
SELECT Programmer_Name,
       @rownum:=@rownum+1 AS ID,
       CASE
           WHEN MOD(@rownum, 2) = 0 THEN 'Even'
           ELSE 'Odd'
       END AS ID_Type
FROM programmers, (SELECT @rownum:=0) r;
```

POWER() and SQRT() - Exponents and Square Roots

`POWER()` raises a number to a specified power, while `SQRT()` calculates the square root.

```sql
-- Calculate the square of the salary (divided by 1000 for readability)
SELECT Programmer_Name,
       Salary,
       POWER(Salary/1000, 2) AS Salary_Squared
FROM programmers;

-- Calculate the square root of the software cost
SELECT Software_Name,
       Software_Cost,
       ROUND(SQRT(Software_Cost), 2) AS Sqrt_Cost
FROM software;
```

3. Date and Time Functions

Date and time functions manipulate and extract information from date and time values.

NOW(), CURDATE(), CURTIME() - Current Date and Time

These functions return the current date and time.

```sql
-- Get the current date and time
SELECT NOW() AS Current_DateTime,
       CURDATE() AS Current_Date_,
       CURTIME() AS Current_Time_;
```

YEAR(), MONTH(), DAY() - Extracting Components

These functions extract specific parts of a date.

```sql
-- Extract year, month, and day from programmer dates of birth
SELECT Programmer_Name,
       DOB,
       YEAR(DOB) AS Birth_Year,
       MONTH(DOB) AS Birth_Month,
       DAY(DOB) AS Birth_Day
FROM programmers;

-- Find programmers born in a specific month
SELECT Programmer_Name
FROM programmers
WHERE MONTH(DOB) = 7;  -- July birthdays
```

DATE_FORMAT() - Formatting Dates

The `DATE_FORMAT()` function formats a date according to a specified format string.

```sql
-- Format the date of joining in a readable format
SELECT Programmer_Name,
       DOJ,
       DATE_FORMAT(DOJ, '%M %d, %Y') AS Formatted_DOJ
FROM programmers;

-- Example output:
-- "Tony Stark | 1990-05-11 | May 11, 1990"
```

DATEDIFF() - Difference Between Dates

The `DATEDIFF()` function returns the number of days between two dates.

```sql
-- Calculate years of experience for each programmer
SELECT Programmer_Name,
       DOJ,
       CURDATE() AS Today,
       ROUND(DATEDIFF(CURDATE(), DOJ)/365, 1) AS Years_Experience
FROM programmers
ORDER BY Years_Experience DESC;
```

DATE_ADD() and DATE_SUB() - Adding/Subtracting from Dates

These functions add or subtract a specified time interval from a date.

```sql
-- Calculate when each programmer will reach 10 years of experience
SELECT Programmer_Name,
       DOJ,
       DATE_ADD(DOJ, INTERVAL 10 YEAR) AS Ten_Year_Anniversary
FROM programmers;

-- Find programmers who joined in the last 30 years
SELECT Programmer_Name
FROM programmers
WHERE DOJ > DATE_SUB(CURDATE(), INTERVAL 30 YEAR);
```

4. Conditional Functions

Conditional functions implement logic that evaluates conditions and returns different values based on those conditions.

IF() - Simple Conditional Logic

The `IF()` function returns one value if a condition is true and another value if it's false.

```sql
-- Categorize programmers by salary
SELECT Programmer_Name,
       Salary,
       IF(Salary > 15000, 'High', 'Standard') AS Salary_Category
FROM programmers;
```

`IFNULL()` - Handling NULL Values

The `IFNULL()` function returns a specified value if the expression is NULL.

```sql
-- Display "Not specified" if Secondary_Language is NULL
SELECT Programmer_Name,
       Primary_Language,
       IFNULL(Secondary_Language, 'Not specified') AS Secondary_Language
FROM programmers;
```

`NULLIF()` - Returning NULL for Matches

The `NULLIF()` function returns NULL if two expressions are equal, otherwise it returns the first expression.

```sql
-- Return NULL if a programmer's primary and secondary languages are the same
SELECT Programmer_Name,
       Primary_Language,
       Secondary_Language,
       NULLIF(Primary_Language, Secondary_Language) AS Different_Primary
FROM programmers;
```

CASE - Complex Conditional Logic

The `CASE` expression allows for more complex conditional logic with multiple conditions.

SQL

```sql
-- Categorize programmers by their primary language type
SELECT Programmer_Name,
       Primary_Language,
       CASE
           WHEN Primary_Language IN ('Python', 'R', 'Julia') THEN 'Data Science'
           WHEN Primary_Language IN ('JavaScript', 'Ruby', 'PHP') THEN 'Web
Development'
           WHEN Primary_Language IN ('Java', 'C#', 'C++', 'C') THEN 'Systems
Programming'
           ELSE 'Other'
       END AS Language_Category
FROM programmers;

-- Categorize programmers by experience level
SELECT Programmer_Name,
       DOJ,
       CASE
           WHEN YEAR(DOJ) <= 1980 THEN 'Veteran'
           WHEN YEAR(DOJ) <= 2000 THEN 'Senior'
           WHEN YEAR(DOJ) <= 2010 THEN 'Mid-level'
           ELSE 'Junior'
       END AS Experience_Level
FROM programmers
ORDER BY DOJ;
```

5. Type Conversion Functions

Type conversion functions convert values from one data type to another.

CAST() - Converting Between Data Types

The `CAST()` function explicitly converts a value from one data type to another.

SQL

```sql
-- Convert numeric salary to string and concatenate with currency symbol
SELECT Programmer_Name,
       CONCAT('$', CAST(Salary AS CHAR)) AS Salary_Display
FROM programmers;

-- Convert string to date
SELECT CAST('2023-01-15' AS DATE) AS Converted_Date;
```

CONVERT() - Alternative Conversion Function

The `CONVERT()` function is similar to `CAST()` but with a slightly different syntax.

```sql
-- Format salary as currency using CONVERT and FORMAT
SELECT Programmer_Name,
       CONCAT('$', FORMAT(CONVERT(Salary, CHAR), 2)) AS Formatted_Salary
FROM programmers;
```

6. Information Functions

Information functions provide metadata about the database or its objects.

DATABASE() - Current Database

The `DATABASE()` function returns the name of the current database.

```sql
-- Show the current database name
SELECT DATABASE() AS `Current_DB`;
```

USER() - Current User

The `USER()` function returns the current MySQL user.

```sql
-- Show the current user
SELECT USER() AS `Current_User`;
```

VERSION() - MySQL Version

The `VERSION()` function returns the MySQL server version.

```sql
-- Show the MySQL version
SELECT VERSION() AS MySQL_Version;
```

7. Combining Multiple Functions (Real-World Scenarios)

In real-world applications, you'll often combine multiple functions to solve complex problems.

Scenario 1: Creating User-Friendly Reports

```sql
-- Create a detailed programmer report
SELECT
    Programmer_Name,
    CONCAT(Primary_Language, '/', Secondary_Language) AS Skills,
    CONCAT('$', FORMAT(Salary, 2)) AS Formatted_Salary,
    CONCAT(
        FLOOR(DATEDIFF(CURDATE(), DOJ) / 365), ' years, ',
        FLOOR((DATEDIFF(CURDATE(), DOJ) % 365) / 30), ' months'
    ) AS Experience,
    DATE_FORMAT(DOB, '%M %d, %Y') AS Birth_Date
FROM programmers;
```

Scenario 2: Software ROI Analysis with Formatted Output

```sql
-- Calculate and format ROI for each software project
SELECT
    Software_Name,
    CONCAT('$', FORMAT(Software_Cost, 2)) AS Unit_Price,
    CONCAT('$', FORMAT(Development_Cost, 2)) AS Dev_Cost,
    CONCAT(Sold, ' units') AS Sales,
    CONCAT('$', FORMAT(Software_Cost * Sold, 2)) AS Revenue,
    CONCAT('$', FORMAT((Software_Cost * Sold) - Development_Cost, 2)) AS Profit,
    CASE
        WHEN ((Software_Cost * Sold) - Development_Cost) > 0 THEN
            CONCAT('+', ROUND((((Software_Cost * Sold) - Development_Cost) /
Development_Cost) * 100, 1), '%')
        ELSE
            CONCAT(ROUND((((Software_Cost * Sold) - Development_Cost) /
Development_Cost) * 100, 1), '%')
    END AS ROI
FROM software
ORDER BY ((Software_Cost * Sold) - Development_Cost) / Development_Cost DESC;
```

Scenario 3: Calculating Age and Experience Metrics

```sql
-- Calculate programmer ages and experience metrics
SELECT
    Programmer_Name,
    YEAR(CURDATE()) - YEAR(DOB) -
        IF(DATE_FORMAT(CURDATE(), '%m%d') < DATE_FORMAT(DOB, '%m%d'), 1, 0) AS Age,
    TIMESTAMPDIFF(YEAR, DOJ, CURDATE()) AS Years_Experience,
    ROUND(Salary / (TIMESTAMPDIFF(YEAR, DOJ, CURDATE())), 2) AS
Salary_Per_Year_Experience,
    CASE
        WHEN (YEAR(CURDATE()) - YEAR(DOB) -
            IF(DATE_FORMAT(CURDATE(), '%m%d') < DATE_FORMAT(DOB, '%m%d'), 1, 0)) <
30 THEN 'Young'
        WHEN (YEAR(CURDATE()) - YEAR(DOB) -
            IF(DATE_FORMAT(CURDATE(), '%m%d') < DATE_FORMAT(DOB, '%m%d'), 1, 0)) <
50 THEN 'Mid-age'
        ELSE 'Senior'
    END AS Age_Group
FROM programmers;
```

Scenario 4: Educational Investment Analysis

```sql
-- Analyze educational investment vs. salary
SELECT
    p.Programmer_Name,
    s.Course,
    s.Institute,
    CONCAT('$', FORMAT(s.Course_Fee, 2)) AS Education_Cost,
    CONCAT('$', FORMAT(p.Salary, 2)) AS Annual_Salary,
    CONCAT(ROUND((p.Salary / s.Course_Fee), 1), 'x') AS Salary_To_Education_Ratio,
    CASE
        WHEN (p.Salary / s.Course_Fee) > 4 THEN 'Excellent ROI'
        WHEN (p.Salary / s.Course_Fee) > 2 THEN 'Good ROI'
        ELSE 'Moderate ROI'
    END AS Education_ROI
FROM programmers p
JOIN studies s ON p.Programmer_Name = s.Programmer_Name
ORDER BY (p.Salary / s.Course_Fee) DESC;
```

8. Creating User-Defined Functions (UDFs)

Sometimes, the built-in functions aren't enough. MySQL allows you to create your own scalar functions.

> NOTE: This does not work in workbench, we will use command line client for this.

```sql
-- Create a function to calculate programmer's experience in years
DELIMITER //
CREATE FUNCTION CalculateExperience(join_date DATE)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    RETURN ROUND(DATEDIFF(CURDATE(), join_date) / 365.25, 2);
END //
DELIMITER ;

-- Use the custom function
SELECT Programmer_Name, DOJ, CalculateExperience(DOJ) AS Years_Experience
FROM programmers
ORDER BY Years_Experience DESC;
```

Conclusion

MySQL scalar functions are powerful tools for transforming, manipulating, and formatting data within your queries. By combining these functions, you can create sophisticated queries that solve complex business problems without requiring additional processing in your application code.

Key takeaways:

1. String functions help manipulate and format text data
2. Numeric functions perform mathematical operations and transformations
3. Date and time functions extract and manipulate temporal data
4. Conditional functions implement decision logic in your queries
5. Type conversion functions change data from one type to another
6. Combining multiple functions allows for complex data transformations
7. User-defined functions extend MySQL's capabilities for custom needs

Understanding and utilizing these scalar functions will significantly enhance your SQL query capabilities and allow you to solve a wide range of data manipulation challenges directly in the database layer.