5-SQL Manipulation Functions

Author: Vincent Lau

Learning Objectives

- Understand various types of SQL functions to manipulate data, such as String, Mathematical, Date & Time, Conditional of Null & Case Handling.

Introduction

Aggregation operations in SQL involve performing calculations on groups of rows to produce summary results. These operations are commonly used to obtain insights from data, such as calculating totals, averages, counts, and more.

String Functions

String functions in SQL are used to manipulate text data stored in columns. These functions allow you to perform tasks like extracting substrings, changing cases, concatenating strings, and

more. Here's an introduction to some common string functions in SQL, along with code examples in MySQL for each operation:

1. CONCAT

The `CONCAT` function is used to concatenate two or more strings together.

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

2. SUBSTRING

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

```
SELECT SUBSTRING(product_name, 1, 10) AS short_name
FROM products;
```

3. LENGTH

The `LENGTH` function returns the length of a string.

```
SELECT product_name, LENGTH(product_name) AS name_length
FROM products;
```

4. UPPER and LOWER

The `UPPER` function converts a string to uppercase, while the `LOWER` function converts it to lowercase.

```
SELECT UPPER(product_name) AS uppercase_name, LOWER(product_name) AS
lowercase_name
FROM products;
```

5. TRIM

The `TRIM` function removes leading and trailing spaces from a string.

```
SELECT TRIM(product_name) AS trimmed_name
```

```
2 FROM products;
```

6. REPLACE

The `REPLACE` function replaces occurrences of a substring within a string.

```
1 SELECT REPLACE(product_name, 'Small', 'Large') AS modified_name
2 FROM products;
```

7. LEFT and RIGHT

The `LEFT` function retrieves a specified number of characters from the beginning of a string, while the `RIGHT` function retrieves characters from the end.

```
1 SELECT LEFT(product_name, 5) AS left_part, RIGHT(product_name, 5) AS right_part
2 FROM products;
```

8. CONCAT_WS

The `CONCAT_WS` function concatenates strings with a specified separator.

```
1 SELECT CONCAT_WS('-', first_name, last_name) AS full_name
2 FROM employees;
```

9. CHAR_LENGTH

The `CHAR_LENGTH` function returns the number of characters in a string (not bytes).

Both `CHAR_LENGTH` and `LENGTH` are functions used to determine the length of a string. However, there is a key difference between them that relates to how they handle multi-byte character encodings like UTF-8.

```
1 SELECT char_length('我'), -- 1
2 LENGTH('我'), -- 3
3 char_length('hello'), -- 5
4 LENGTH('hello') -- 5
5 FROM dual;
```

10. INSTR

The `INSTR` function returns the position of a substring within a string.

```
1 SELECT product_name, INSTR(product_name, 'Shirt') AS position
2 FROM products;
```

11. SQL Case Insensitivity

SQL commands are ***case insensitive***, but they are often written in upper case letters for better readability.

12. String Comparison Sensitivity

MySQL - Case-Insensitive

In MySQL, string comparisons are case-insensitive by default for character sets that are not case-sensitive. This behavior is determined by the collation settings of the character columns or string literals used in your SQL queries. Collation defines the rules for comparing and sorting characters in a character set.

If you use a case-insensitive collation, string comparisons will be case-insensitive. For example, if you're using the `utf8_general_ci` (CI stands for case-insensitive) collation for a character column, comparisons involving that column will be case-insensitive.

Here's an example:

If the `username` column is using a case-insensitive collation, it will match rows where the username is 'johndoe', 'JOHNDOE', 'jOhNdOe', and so on.

```
1 SELECT * FROM users WHERE username = 'JohnDoe';
```

However, if you explicitly want a case-sensitive comparison, you can use a case-sensitive collation like `utf8_bin` (BIN stands for binary) for the comparison. For example:

```
1 SELECT * FROM users WHERE username COLLATE utf8mb4_bin = 'JohnDoe';
```

In this query, the `COLLATE utf8_bin` clause makes the comparison case-sensitive, so it will only match rows where the username is exactly 'JohnDoe'.

Oracle, PostgreSQL, SQL Server - Case-sensitive

In Oracle Database, PostgreSQL, and SQL Server, the default behavior for string comparisons is case-sensitive. However, you can perform case-insensitive string comparisons by using specific collations or functions. Here's how it works in each of these database systems:

1. Oracle Database:

- By default, Oracle Database uses **case-sensitive comparisons for character columns**.
- To perform case-insensitive comparisons, you can use functions like `UPPER` or `LOWER` to convert both the column value and the search string to the same case before comparing.

Example (Oracle):

```
1 SELECT * FROM employees WHERE UPPER(last_name) = 'SMITH';
```

2. PostgreSQL:

- PostgreSQL also performs **case-sensitive comparisons by default**.
- To perform case-insensitive comparisons, you can use the `ILIKE` operator, which is similar to `LIKE` but case-insensitive.

Example (PostgreSQL):

```
1 SELECT * FROM employees WHERE last_name ILIKE 'Smith';
```

3. SQL Server:

- SQL Server performs **case-sensitive comparisons by default.**
- To perform case-insensitive comparisons, you can use the `COLLATE` clause with a case-insensitive collation, such as `COLLATE Latin1_General_CI_AI`.

Example (SQL Server):

```
1 SELECT * FROM employees WHERE last_name COLLATE Latin1_General_CI_AI = 'Smith';
```

In all three database systems, the methods mentioned above allow you to perform case-insensitive string comparisons when needed. However, it's essential to be aware of the specific syntax and functions required in each system to achieve the desired behavior.

13. Single Quotes

- Strings are enclosed in *single quotes*, and are still case sensitive.
- A string without any content is called an *empty string*, i.e. `''`.

14. Wildcard Characters

- `_` represents any single character.
- `%` represents zero or more characters.
- Patterns are case sensitive. If we want to compare strings ignoring the case, we can either use the `UPPER` or `LOWER` function.

```sql
-- matches any instructor's name that begins with 'MO' (ignoring case)
SELECT * FROM instructor WHERE UPPER(name) LIKE 'MO%';

-- matches any instructor's name that begins with one character followed by 'old' (ignoring case)
SELECT * FROM instructor WHERE LOWER(name) LIKE '_old';
```

These examples cover a range of string functions in MySQL. These functions are helpful for manipulating and extracting information from text-based data stored in your database tables. They can be used in various ways to modify, transform, and analyze string data as needed.

Mathematical Functions

These are common mathematical functions available in most database management systems (DBMS) for manipulating numeric data:

15. ROUND

The `ROUND` function is used to round a numeric value to a specified number of decimal places.

- It has two arguments: the numeric value to be rounded and the number of decimal places to round to.
- If the number of decimal places is positive, it rounds to that number of decimal places. If it's negative, it rounds to the left of the decimal point

Example (MySQL and PostgreSQL):

```sql
SELECT ROUND(3.14159, 2) AS rounded_value; -- Result: 3.14
```

16. CEIL (CEILING)

The `CEIL` or `CEILING` function is used to round a numeric value up to the nearest integer greater than or equal to the original value.

- It returns the smallest integer that is greater than or equal to the input value.

Example (MySQL and PostgreSQL):

```
1 SELECT CEIL(4.3) AS ceiling_value; -- Result: 5
```

17. FLOOR

The `FLOOR` function is used to round a numeric value down to the nearest integer less than or equal to the original value.

- It returns the largest integer that is less than or equal to the input value.

Example (MySQL and PostgreSQL):

```
1 SELECT FLOOR(4.9) AS floor_value; -- Result: 4
```

18. ABS (Absolute Value)

The `ABS` function is used to find the absolute (non-negative) value of a numeric expression.

- It returns the magnitude of a number, ignoring its sign.

Example (MySQL and PostgreSQL):

```
1 SELECT ABS(-5) AS absolute_value; -- Result: 5
```

19. POWER

The `POWER` function is used to raise a number to a specified power.

- It has two arguments: the base number and the exponent.

Example (MySQL and PostgreSQL):

```
1 SELECT POWER(2, 3) AS result; -- Result: 8
```

These mathematical functions are valuable for performing calculations and transformations on numeric data within your SQL queries. They allow you to manipulate and format numerical values to suit your specific needs in various applications.

Date and Time Functions

Date and time functions in database management systems (DBMS) allow you to work with date and time values, perform calculations, format output, and extract specific components of date and time data. Here are some commonly used date and time functions:

1. DATEADD

The `DATEADD` function is used to add or subtract a specified interval (such as days, months, or years) to a date or timestamp.

- It takes three arguments: the part of the date to add (e.g., 'day', 'month', 'year'), the number of intervals, and the date or timestamp.

Example (SQL Server):

```
SELECT date_add('2023-07-15', interval 3 month) AS new_date; -- Result: '2023-10-15'
```

2. DATEDIFF

The `DATEDIFF` function calculates the difference between two dates or timestamps in terms of a specified interval.

- It takes three arguments: the interval (e.g., 'day', 'month', 'year'), the start date or timestamp, and the end date or timestamp.

Example (MySQL and PostgreSQL):

```
SELECT DATEDIFF('2023-12-31', '2023-01-01') AS days_difference; -- Result: 364
SELECT DATEDIFF('2024-01-01','2023-12-31') AS days_difference; -- Result: 1
```

3. NOW (CURRENT_TIMESTAMP)

The `NOW` or `CURRENT_TIMESTAMP` function returns the current date and time as a timestamp.

Example (MySQL and PostgreSQL):

```
SELECT NOW() AS current_timestamp; -- Result: '2023-08-15 12:34:56'
```

4. DATE_FORMAT (TO_CHAR in PostgreSQL)

The `DATE_FORMAT` function (or `TO_CHAR` in PostgreSQL) is used to format date and time values into strings based on a specified format pattern.

- It has two arguments: the date or timestamp value and the format pattern.

Example (MySQL and PostgreSQL):

```
SELECT DATE_FORMAT('2023-08-15', '%Y-%m-%d') AS formatted_date; -- Result: '2023-08-15'
```

5. INTERVAL (MySQL & PostgreSQL)

In MySQL, you can use the `INTERVAL` keyword to specify a time interval, including intervals of days. To add or subtract a specific number of days from a date or timestamp, you can use the `DATE_ADD` and `DATE_SUB` functions along with the `INTERVAL` keyword.

```
-- MySQL
SELECT '2020-06-30' + INTERVAL 1 DAY AS tomorrow;
SELECT STR_TO_DATE('2020-06-30', '%Y-%m-%d') + INTERVAL 1 DAY AS tomorrow;
SELECT DATE_ADD('2023-08-15', INTERVAL 1 DAY) AS new_date; -- Result: '2023-08-16'
SELECT DATE_SUB('2023-08-15', INTERVAL 1 DAY) AS new_date; -- Result: '2023-08-14'

-- PostgreSQL
SELECT to_date('2020-06-30','yyyy-mm-dd') + INTERVAL '1 day' AS tomorrow;
SELECT to_date('2020-06-30','yyyy-mm-dd') + 1 AS tomorrow;

-- Oracle
SELECT to_date('2020-06-30','yyyy-mm-dd') + 1 AS tomorrow;

-- SQL SERVER
SELECT DATEADD(DAY, 1, CAST('2020-06-30' AS DATE)) AS tomorrow;
```

6. EXTRACT (DATE_PART in PostgreSQL)

The `EXTRACT` function (or `DATE_PART` in PostgreSQL) is used to extract a specific component (e.g., year, month, day) from a date or timestamp.

- It has two arguments: the component to extract and the date or timestamp value.

Example (MySQL and PostgreSQL):

```sql
SELECT EXTRACT(YEAR FROM '2023-08-15') AS extracted_year; -- Result: 2023
```

These date and time functions are essential for working with temporal data in databases. They help you perform calculations, manipulate dates and times, and format output according to your specific requirements, making them valuable tools for various applications. Keep in mind that the availability and syntax of these functions may vary depending on the DBMS you are using.

Conditionals

Null Functions

To handle NULL values by substituting them with a specified default value, there are different function handling among various DBMS, such as MySQL, SQL Server, and PostgreSQL. Here are their differences:

1. Oracle (NVL):

The `NVL()` function in Oracle allows you to replace NULL values with a specified default value.

```sql
SELECT NVL(column_name, default_value) FROM table_name;
```

2. MySQL (IFNULL or COALESCE):

In MySQL, you can use either the `IFNULL()` or `COALESCE()` function to achieve the same result. These functions allow you to specify a default value when a column value is NULL.

```sql
SELECT IFNULL(column_name, default_value) FROM table_name;
-- OR
SELECT COALESCE(column_name, default_value) FROM table_name;
```

3. SQL Server (ISNULL):

In SQL Server, you can use the `ISNULL()` function to replace NULL values with a specified default value.

```sql
SELECT ISNULL(column_name, default_value) FROM table_name;
```

4. PostgreSQL (COALESCE):

In PostgreSQL, you can use the `COALESCE()` function, just like in MySQL, to handle NULL values by substituting them with a specified default value.

```sql
SELECT COALESCE(column_name, default_value) FROM table_name;
```

All of these functions perform a similar task of handling NULL values by providing a default value if the column value is NULL. The specific function you use depends on the database system you are working with. It's important to be aware of the function's syntax and the available functions in your database system for handling NULLs.

Case Syntax (if-else)

The `CASE` expression in SQL is a powerful conditional construct that allows you to perform conditional logic in your queries. It's supported in various database management systems (DBMS), including MySQL, Oracle, SQL Server, and PostgreSQL. Here's an introduction to the `CASE` expression with examples for each DBMS:

Basic Syntax:

```sql
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE default_result
END
```

- `CASE` starts the expression.
- `WHEN` clauses specify conditions to evaluate.
- `THEN` specifies the result when a condition is true.
- `ELSE` provides a default result if no condition is true.
- `END` ends the `CASE` expression.

Now, let's look at examples for each DBMS:

1. MySQL:

In this MySQL example, the `CASE` expression categorizes customers based on their total purchase amount.

```sql
SELECT
    customer_name,
    CASE
        WHEN total_purchase >= 1000 THEN 'Gold'
        WHEN total_purchase >= 500 THEN 'Silver'
        ELSE 'Bronze'
    END AS customer_category
FROM customers;
```

2. Oracle:

In this Oracle example, the `CASE` expression categorizes employees based on their commission percentages.

```sql
SELECT
    employee_id,
    first_name,
    last_name,
    CASE
        WHEN commission_pct IS NULL THEN 'No Commission'
        WHEN commission_pct < 0.1 THEN 'Low Commission'
        ELSE 'High Commission'
    END AS commission_category
FROM employees;
```

3. SQL Server:

In this SQL Server example, the `CASE` expression categorizes orders based on their age.

```sql
SELECT
    order_id,
    order_date,
    CASE
        WHEN DATEDIFF(DAY, order_date, GETDATE()) <= 7 THEN 'New'
        WHEN DATEDIFF(DAY, order_date, GETDATE()) <= 30 THEN 'Recent'
        ELSE 'Old'
    END AS order_age
```

```
FROM orders;
```

4. PostgreSQL:

In this PostgreSQL example, the `CASE` expression categorizes products based on their unit prices.

```
SELECT
    product_name,
    unit_price,
    CASE
        WHEN unit_price < 10 THEN 'Low Price'
        WHEN unit_price <= 50 THEN 'Moderate Price'
        ELSE 'High Price'
    END AS price_category
FROM products;
```

The `CASE` expression is a versatile tool for creating conditional logic within SQL queries, making it easier to manipulate and present data based on specific conditions or rules. It's widely supported across various DBMS and can be adapted to suit a wide range of use cases.