10

Functions

Built-In and Group Functions

Objectives

After completing this lesson, you should be able to:

- Use built-in functions in MySQL
- Describe and use string functions
- Describe and use temporal functions
- Describe and use numeric functions
- Describe and use control flow functions
- Use aggregate functions with the **`SELECT`** statement

Functions in MySQL Expressions

- Functions can be invoked within expressions.
- An expression returns a value that is used in place of the function call when evaluated.
- General syntax:

```
function_name([<arg1> [, <arg2>, …, <argn>]])
```

  - The parentheses following the function name are required.
- Examples:
  - **SELECT NOW()**: The **NOW** function returns the current date and time.
  - **SELECT VERSION()**: The **VERSION** function returns the MySQL server version currently being used on the host.

Function Tips

- Functions can be used anywhere a value expression is accepted.
  - Most function calls also take arguments.
- Columns can be used as arguments with the correct data type.
- The output of a function can be used as the input of another function.
- An expression with **NULL** always produces a **NULL** output.
  - Unless otherwise indicated in the documentation for a particular function or operator
- Mathematical functions return **NULL** on error.
  - For instance, a division by zero

Function Categories

- **String:** Perform operations relating to strings.
- **Temporal:** Perform operations on dates and times.
- **Numeric:** Perform several types of mathematical operations.
- **Control Flow:** Choose between different values based on the result of an expression.
- **Aggregate:** This is based on aggregates or groups of values.

String Functions

- Perform operations on strings, such as:
  - Calculating string lengths
  - Extracting pieces of strings
  - Searching for substrings or replacing them
  - Performing case conversion
- String functions are divided into two categories:
  - **Numeric:** Return numbers
  - **String:** Return strings

String Function: Numeric Category Examples

- Returns the number of characters in the string:

```
mysql> SELECT CHAR_LENGTH('MySQL');
+----------------------+
| CHAR_LENGTH('MySQL') |
+----------------------+
|                    5 |
+----------------------+
```

- Returns the position in the string where substring occurs:

```
mysql> SELECT INSTR('MySQL', 'SQL');
+-----------------------+
| INSTR('MySQL', 'SQL') |
+-----------------------+
| 3                     |
+-----------------------+
```

- Returns results of string sort comparison (0=same, –1=smaller, 1=other):

```
mysql> SELECT STRCMP('abc','def'),
    -> STRCMP('def','def'),
    -> STRCMP('def','abc');
+---------------------+---------------------+---------------------+
| STRCMP('abc','def') | STRCMP('def','def') | STRCMP('def','abc') |
+---------------------+---------------------+---------------------+
| -1                  | 0                   | 1                   |
+---------------------+---------------------+---------------------+
```

String Function: String Category Examples (CONCAT, REVERSE, LEFT, RIGHT)

- Concatenates the given arguments into one string:

```
mysql> SELECT CONCAT('A', '-', 'Z');
+------------------------+
| CONCAT('A', '-', 'Z') |
+------------------------+
| A-Z                    |
+------------------------+
```

- Returns string with the characters in reverse order:

```
mysql> SELECT REVERSE('MySQL');
+-------------------+
| REVERSE('MySQL') |
+-------------------+
| LQSyM             |
+-------------------+
```

- Returns the left-most length characters of string:

```
mysql> SELECT LEFT('MySQL', 3);
+--------------------+
| LEFT ('MySQL', 3) |
+--------------------+
| MyS                |
+--------------------+
```

- Returns the right-most length characters of string:

```
mysql> SELECT RIGHT('MySQL', 3);
+--------------------+
| RIGHT('MySQL', 3) |
+--------------------+
| SQL                |
+--------------------+
```

String Function: String Category Examples (LOWER, UPPER, LPAD, RPAD)

- Returns the string with all characters in lowercase:

```
mysql> SELECT LOWER('MySQL');
+----------------+
| LOWER('MySQL') |
+----------------+
| mysql          |
+----------------+
```

- Returns the string with all characters in uppercase:

```
mysql> SELECT UPPER('MySQL');
+----------------+
| UPPER('MySQL') |
+----------------+
| MYSQL          |
+----------------+
```

- Returns string left-padded with the indicated characters:

```
mysql> SELECT LPAD('MySQL', 8, '.');
+------------------------+
| LPAD('MySQL', 8, '.')  |
+------------------------+
| ...MySQL               |
+------------------------+
```

- Returns string right-padded with the indicated characters:

```
mysql> SELECT RPAD('MySQL', 8, '.');
+------------------------+
| RPAD('MySQL', 8, '.')  |
+------------------------+
| MySQL...               |
+------------------------+
```

String Function: String Category Examples (TRIM)

- Removes all the leading and trailing spaces around the string:

```
mysql> SELECT TRIM('   MySQL    ') AS str;
+-------+
| str   |
+-------+
| MySQL |
+-------+
```

String Function: String Category Examples (SUBSTRING)

- Returns the part of the string starting at the specified position through the end of string:

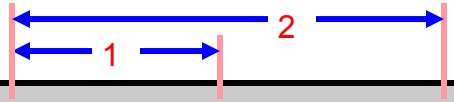
```
mysql> SELECT SUBSTRING('MySQL', 3);
+-----------------------+
| SUBSTRING('MySQL', 3) |
+-----------------------+
| SQL                   |
+-----------------------+
```

- Returns the part of the string starting at the specified position, and the number of characters indicated:

```
mysql> SELECT SUBSTRING('MySQL', 2, 2);
+--------------------------+
| SUBSTRING('MySQL', 2, 2) |
+--------------------------+
| yS                       |
+--------------------------+
```

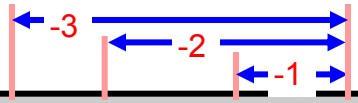
String Function: String Category Examples (SUBSTRING_INDEX)

- Returns the part of the string starting from the left to the specified delimiter count:




```
mysql> SELECT SUBSTRING_INDEX('training@mysql.com', '@', 1);
+-----------------------------------------------+
| SUBSTRING_INDEX('training@mysql.com', '@', 1) |
+-----------------------------------------------+
| training                                      |
+-----------------------------------------------+
```

- Returns the part of the string starting from the right to the specified delimiter count:




```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
+-------------------------------------------+
| SUBSTRING_INDEX('www.mysql.com', '.', -2) |
+-------------------------------------------+
| mysql.com                                 |
+-------------------------------------------+
```

Temporal Functions

- Perform operations such as:
  - Extracting parts of dates and times
  - Reformatting values
  - Converting values to seconds or days
- Use several formats for the same information, such as:
  - 2012-02-10 22:50:15
  - Friday, February 10, 2012
- Generate temporal data in many ways:
  - Copy existing data.
  - Execute the built-in function.
  - Build a string representation to be evaluated by the server.

Temporal Functions: Date/Time Formats

Type	Default Format
DATE	YYYY-MM-DD
TIME	HH:MM:SS
DATETIME	YYYY-MM-DD hh:mm:ss
TIMESTAMP	YYYY-MM-DD hh:mm:ss
YEAR	YYYY

Temporal Functions: Function Types

Function Syntax	Definition
NOW()	Current date and time as set on the server host (DATETIME format)
CURDATE()	Current date as set on the server host (DATE format)
CURTIME()	Current time as set on the server host (TIME format)
YEAR(<date_expression>)	Year in four-digit YEAR format
MONTH(<date_expression>)	Month of the year in integer format, per expression
DAYOFMONTH(<date_expression>) or DAY(<date_expression>)	Day of the month in integer format, per expression
DAYNAME(<date_expression>)	Day of the week in string format, per expression (English)
HOUR(<date_expression>)	Hour of the day in integer format (in 0–23 range), per expression
MINUTE(<date_expression>)	Minute of the day in integer format, per expression
SECOND(<date_expression>)	Second of the minute in integer format, per expression

Temporal Functions: Extracting Temporal Data Examples

- Extracting current temporal data (date, time, and day of week):

```
mysql> SELECT CURDATE(), CURTIME(), DAYNAME(NOW());
+------------+-----------+----------------+
| CURDATE()  | CURTIME() | DAYNAME(NOW()) |
+------------+-----------+----------------+
| 2012-02-06 | 17:55:40  | Friday         |
+------------+-----------+----------------+
```

- Addition of a specified interval of days from the current date and time:

```
mysql> SELECT NOW(), NOW() + INTERVAL 5 DAY;
+---------------------+------------------------+
| NOW()               | NOW() + INTERVAL 5 DAY |
+---------------------+------------------------+
| 2012-02-06 18:02:35 | 2006-02-11 18:02:35    |
+---------------------+------------------------+
```

Temporal Functions: Extracting Temporal Data Examples

- Customizing the output format of temporal data:

```
mysql> SELECT NOW(), DATE_FORMAT(NOW(), '%W the %D of %M');
+---------------------+----------------------------------------+
| NOW()               | DATE_FORMAT(NOW(), '%W the %D of %M')  |
+---------------------+----------------------------------------+
| 2012-02-06 18:02:35 | Friday the 6th of February             |
+---------------------+----------------------------------------+
```

Numeric Functions

Perform mathematical operations such as:

- Rounding
- Truncation
- Trigonometric calculations
- Generating random numbers

Function Syntax	Definition
`ABS(<number>)`	Returns the absolute value of number
`SIGN(<number>)`	Returns –1, 0, or 1 depending on whether the number is negative, zero or positive
`TRUNCATE(<number>, <decimals>)`	Returns number truncated to decimals
`FLOOR(<number>)`	Rounds number down to the closest lower integer
`CEILING(<number>)`	Rounds number up to the closest higher integer
`ROUND(<number>)`	Rounds number to the closest integer

Numeric Functions: Examples

- Returns the absolute value of the negative and positive values:

```
mysql> SELECT ABS(-42), ABS(42);
+----------+---------+
| ABS(-42) | ABS(42) |
+----------+---------+
|       42 |      42 |
+----------+---------+
```

- Returns results of sign determination (–1=negative, 0=zero, 1=positive):

```
mysql> SELECT SIGN(-42), SIGN(-1), SIGN(0), SIGN(1), SIGN(42);
+-----------+----------+---------+---------+----------+
| SIGN(-42) | SIGN(-1) | SIGN(0) | SIGN(1) | SIGN(42) |
+-----------+----------+---------+---------+----------+
| -1        | -1       | 0       | 1       | 1        |
+-----------+----------+---------+---------+----------+
```

Numeric Functions: Additional Functions

- Geometric functions:
  - `DEGREES()`, `PI()`, `RADIANS()`
- Trigonometric functions:
  - `COS()`, `SIN()`, `COT()`
  - `ACOS()`, `ASIN()`, `ATAN()`, `ATAN2()`
- Other functions:
  - `EXP()`, `LN()`, `LOG()`, `LOG2()`, `LOG10()`
  - `POWER()`, `SQRT()`
  - `MOD()`

Control Flow Functions

- Choose between different values based on the result of an expression.
- **IF()** function example:

```
mysql> SELECT IF(1 > 0, 'YES', 'NO');
+-------------------------+
| IF(1 > 0, 'YES', 'NO') |
+-------------------------+
| YES                     |
+-------------------------+
```

TRUE

FALSE

Control Flow Functions: `CASE` Functions

The `CASE` function provides branching flow of control in two different ways:

- Expression value comparison
- Expression condition evaluation

Control Flow Functions: CASE Function Syntax

- Expression value comparison example:

```
CASE value
   WHEN <compare_value> THEN <result>
   [WHEN <compare_value> THEN <result> ...]
   [ELSE <result>]
END
```

- Expression condition evaluation example :

```
CASE
   WHEN <condition> THEN <result>
   [WHEN <condition> THEN <result> ...]
   [ELSE <result>]
END
```

Control Flow Functions: CASE Function Examples

```
mysql> SELECT CASE 3 WHEN 1 THEN 'one'
    -> WHEN 2 THEN 'two' ELSE 'more' END;
'more'
```

```
mysql> SELECT CASE WHEN 1>0 THEN 'true`
    -> ELSE 'false' END;
'true'
```

```
mysql> SELECT CASE BINARY 'B`
    -> WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
NULL
```

```
SELECT pName, oID, pGender,
  CASE
     WHEN pGender = 'm' THEN 'Boy'
     WHEN pGender = 'f' THEN 'Girl'
     ELSE 'Unknown'
  END
FROM pet_info
ORDER BY pGender;
```

Quiz

The `CHAR_LENGTH()` function is part of the _____ function category.

a. String
b. Temporal
c. Numeric
d. Control Flow
e. Aggregate

Aggregate Functions

- Perform summary operations on a set of values, such as:
  - Counting
  - Averaging
  - Finding minimum or maximum values
- Calculate a single value based on a group of values from different rows.
- Result value is based only on non-`NULL` values from the selected rows.

Aggregate Function Types

Some of the aggregate function types:

Function Syntax	Definition
`MIN(<column_name>)`	Find the smallest value.
`MAX(<column_name>)`	Find the largest value.
`SUM(<column_name>)`	Summarize numeric value totals.
`AVG(<column_name>)`	Summarize numeric value averages.
`COUNT(<column_name>)`	Count rows and non-null values.
`GROUP_CONCAT(<column_name>)`	Concatenate a set of strings to produce a single string.

Aggregate Functions: COUNT Function Examples

- Retrieves a count of all rows in the **Country** table:

```
mysql> SELECT COUNT(*) FROM Country;
+----------+
| COUNT(*) |
+----------+
|      239 |
+----------+
```

- Retrieves a count of all the **Country** table rows that have non-NULL values in the **Capital** column:

```
mysql> SELECT COUNT(Capital) FROM Country;
+----------------+
| COUNT(Capital) |
+----------------+
|            232 |
+----------------+
```

Aggregate Functions: GROUP BY Clause

- The **GROUP BY** clause places rows into groups.
  - Each group consists of rows having the same value in one or more columns.
  - Calculates a summary value for each group
- Example:

```
mysql> SELECT Continent, AVG(Population)
    -> FROM Country
    -> GROUP BY Continent;
+---------------+-----------------+
| Continent     | AVG(Population) |
+---------------+-----------------+
| Asia          | 72647562.7451   |
| Europe        | 15871186.9565   |
| North America | 13053864.8649   |
| Africa        | 13525431.0345   |
| Oceania       | 1085755.3571    |
| Antarctica    | 0.0000          |
| South America | 24698571.4286   |
+---------------+-----------------+
```

Aggregate Functions: GROUP BY Clause and GROUP_CONCAT Function

- The **GROUP_CONCAT()** function concatenates results.
- Example:

```
mysql> SELECT GovernmentForm, GROUP_CONCAT(Name)
    -> AS Countries
    -> FROM Country WHERE Continent = 'South America'
    -> GROUP BY GovernmentForm\G
************************* 1. row *************************
GovernmentForm: Dependent Territory of the UK
      Countries: Falkland Islands
************************* 2. row *************************
GovernmentForm: Federal Republic
      Countries: Argentina,Venezuela,Brazil
************************* 3. row *************************
GovernmentForm: Overseas Department of France
      Countries: French Guiana
************************* 4. row *************************
GovernmentForm: Republic
Countries: Chile,Uruguay,Suriname,Peru,Paraguay,Bolivia,
   Guyana,Ecuador,Colombia
```

Aggregate Functions: GROUP BY and HAVING Clauses

- Use the **HAVING** clause to eliminate rows based on aggregate values.
  - Evaluated after the grouping implied by **GROUP BY**
- Example:

```
mysql> SELECT Continent, SUM(Population)
-> FROM Country
-> GROUP BY Continent
-> HAVING SUM(Population) > 100000000;
+---------------+-----------------+
| Continent     | SUM(Population) |
+---------------+-----------------+
| Asia          | 3705025700      |
| Europe        | 730074600       |
| North America | 482993000       |
| Africa        | 784475000       |
| South America | 345780000       |
+---------------+-----------------+
```

Aggregate Functions: GROUP BY Clause and WITH ROLLUP Modifier

- Use the **WITH ROLLUP** modifier to produce multiple levels of summary values.
- Example:

```
mysql> SELECT Continent, SUM(Population)
    ->  FROM Country
    ->  GROUP BY Continent
    ->  WITH ROLLUP;
+---------------+-----------------+
| Continent     | SUM(Population) |
+---------------+-----------------+
| Asia          |      3705025700 |
| Europe        |       730074600 |
| North America |       482993000 |
| Africa        |       784475000 |
| Oceania       |        30401150 |
| Antarctica    |               0 |
| South America |       345780000 |
| NULL          |      6078749450 |
+---------------+-----------------+
```

Aggregate Functions: Super Aggregate Operation

- Use the **WITH ROLLUP** and the **AVG()** function
  - Produce a final line that comprises the application of the given aggregate function.
- Example:

```
mysql> SELECT Continent, AVG(Population)
    -> FROM Country
    -> GROUP BY Continent WITH ROLLUP;
+---------------+-----------------+
| Continent     | AVG(Population) |
+---------------+-----------------+
| Asia          | 72647562.7451   |
| Europe        | 15871186.9565   |
| North America | 13053864.8649   |
| Africa        | 13525431.0345   |
| Oceania       | 1085755.3571    |
| Antarctica    | 0.0000          |
| South America | 24698571.4286   |
| NULL          | 25434098.1172   |
+---------------+-----------------+
```

Spaces in Function Names

- By default, there must be no space between a function name and the parenthesis:

```
mysql> SELECT PI ();
ERROR 1305 (42000): FUNCTION world.PI does not exist
```

- You can change this using the **IGNORE SPACE** SQL mode:

```
mysql> SET sql_mode = 'IGNORE_SPACE';
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT PI ();
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
```

Summary

In this lesson, you should have learned how to:

- Use built-in functions in MySQL
- Describe and use string functions
- Describe and use temporal functions
- Describe and use numeric functions
- Describe and use control flow functions
- Use aggregate functions with the **`SELECT`** statement

Practice 10-1 Overview: Quiz

This practice covers answering questions about MySQL functions.

Practice 10-2 Overview: Use Built-In, String, and Temporal Functions

This practice covers using built-in , string, and temporal function statements.

Practice 10-3 Overview: Use Numeric and Control Flow Functions

This practice covers using numeric and control flow statements.

Practice 10-4 Overview: Use Aggregate Functions

This practice covers the following topics:

- Using aggregate functions
- Using the `GROUP BY` clause, with options