10

Functions

Built-In and Group Functions

Objectives

After completing this lesson, you will be able to:

- Use built-in functions in MySQL
- Describe and use:
  - String functions
  - Temporal functions
  - Numeric functions
  - Control flow functions
- Use aggregate functions with the `SELECT` statement

10 - 2

Functions in MySQL Expressions

- Functions perform calculations on data.
- A function returns a value that can be used as part of an expression.
- General syntax:

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

  - You must include the parentheses, even if there are no arguments.
  - Separate multiple arguments with commas.
- Examples:
  - `SELECT NOW()`: Returns the current date and time
  - `SELECT VERSION()`: Returns the MySQL Server version currently being used on the host

10 - 3

The general syntax of a function call is the name of the function followed by parentheses. The parentheses contain any arguments that the function requires, separated by commas.

You can put spaces around function arguments but there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function calls and references to tables or columns with the same name as a function.

For more information about functions, see the MySQL Reference Manual at http://dev.mysql.com/doc/refman/5.6/en/functions.html.

Using Functions

- Functions can be used anywhere a value expression is accepted.
- Most functions require arguments.
- Columns (of the appropriate data type) can be used as arguments.
- The output of one function can be used as the input of another function.
- An expression with null values always produces a null output.
  - With rare, documented exceptions
- Mathematical functions return a null value on error
  - For example, division by zero

10 - 4

Types of Functions

- **String:** Operations on character strings
- **Temporal:** Operations on dates and times
- **Numeric:** Mathematical operations
- **Control Flow:** Choose between different values based on the result of an expression
- **Aggregate:** Return a single value based on multiple values in a column

10 - 5

The following slides cover these different types of functions in detail.

String Functions

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

10 - 6

String Functions (Numeric): Examples (CHAR_LENGTH, INSTR, STRCMP)

- 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                  |
+--------------------+--------------------+--------------------+
```

10 - 7

Usage:

- `CHAR_LENGTH(<string>)`
- `INSTR(<string>, <substring>)`: Returns `0` if the substring is not in the specified string
- `STRCMP(<arg1>, <arg2>)`: Returns `0` if arguments are the same, `-1` if `arg1` is smaller than `arg2` (according to the current sort order), and `1` otherwise

String Functions: 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 specified number of characters from the left of a string

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

- Returns the specified number of characters from the right of a string

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

10 - 8

Usage:

- `CONCAT(<arg1> [, <arg2>, …, <argn>])`
- `REVERSE(<string>)`
- `RIGHT(<string>, <length>)`
- `LEFT(<string>, <length>)`

String Functions: 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 specified characters:

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

- Returns string right-padded with the specified characters:

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

10 - 9

Usage:

- `LOWER(<string>)`
- `UPPER(<string>)`
- `LPAD(<string>, <length>, <padding>)`
- `RPAD(<string>, <length>, <padding>)`

String Functions: Examples (TRIM)

- Removes all leading and trailing spaces from the string:

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

- Removes the specified characters from the beginning of the string:

```
mysql> SELECT TRIM(LEADING 'Q' FROM 'QQQMySQLQQQ');
+--------------------------------------+
| TRIM(LEADING 'Q' FROM 'QQQMySQLQQQ') |
+--------------------------------------+
| MySQLQQQ                             |
+--------------------------------------+
```

10 - 10

Usage:

- `TRIM( [[LEADING | TRAILING] <string> FROM ] <fullstring>)`:
  - `TRIM` without the `FROM` clause removes all spaces from the beginning and end of `<fullstring>`.
  - With the `FROM` clause, `TRIM` removes all `<string>` characters from the beginning and/or end (`LEADING` or `TRAILING` or `BOTH`) of `<fullstring>`.

String Functions: Examples (SUBSTRING)

- Returns part of a string
  - From a specified starting position to the end of a string

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

  - A specified number of characters from the starting position:

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

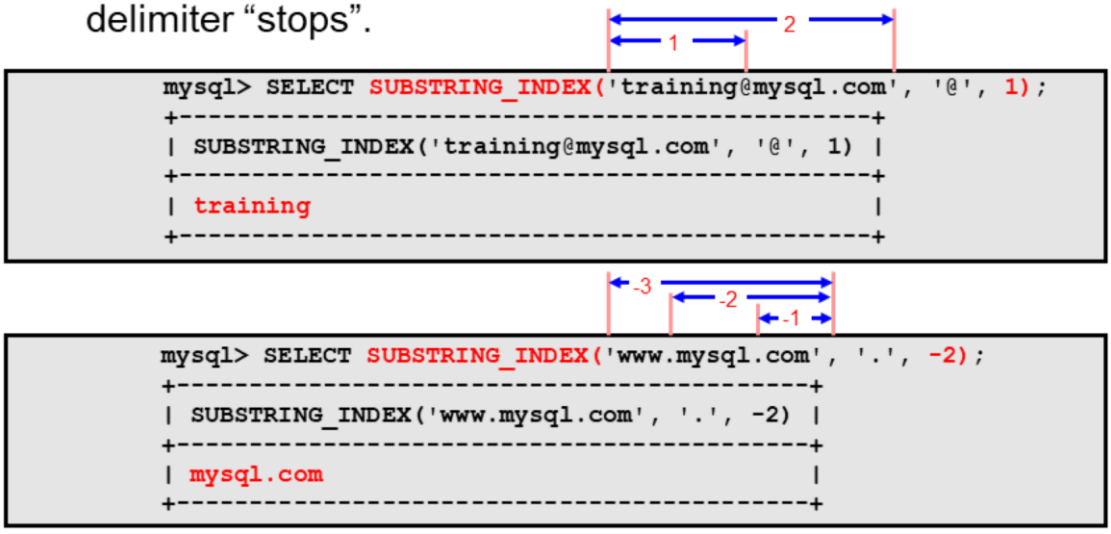
10 - 11

Usage:

```
SUBSTRING(<string>, <startpos> [, <chars>])
```

String Functions: Examples (SUBSTRING_INDEX)

- Returns part or all of a string based on a delimiter
- You provide the delimiter character(s) and a count of delimiter “stops”.



10 - 12

Usage:

`SUBSTRING_INDEX(<string>, <delimiter>, <count>)`: The search is from left to right if `<count>` is positive and right to left if `<count>` is negative.

Temporal Functions

- Perform operations such as:
  - Extracting parts of dates and times
  - Reformatting values
  - Converting values to seconds or days
- You can generate temporal data in many ways:
  - Copy existing data.
  - Use built-in functions.
  - Build a string representation for the server to evaluate.

10 - 13

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

10 - 14

The slide lists the date and time formats you can use with temporal functions.

See the lesson about data types for specific information about the values and ranges for these types.

Functions that expect date values usually accept `DATETIME` values and ignore the time part. Functions that expect time values usually accept `DATETIME` values and ignore the date part.

Temporal Functions: Function Types

Function Syntax	Return value
`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, from the date you supply as an argument.
`MONTH(<date_expression>)`	Month of the year in integer format, from the date you supply as an argument
`DAYOFMONTH(<date_expression>)` or `DAY(<date_expression>)`	Day of the month in integer format, from the date you supply as an argument
`DAYNAME(<date_expression>)`	Day of the week in natural language format, from the date you supply as an argument
`HOUR(<date_expression>)`	Hour of the day in integer format (in 0–23 range), from the date you supply as an argument
`MINUTE(<date_expression>)`	Minute of the day in integer format, from the date you supply as an argument
`SECOND(<date_expression>)`	Second of the minute in integer format, from the date you supply as an argument

10 - 15

Temporal Functions: Examples

- Return current date, time, and day of week:

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

- Add specified number of days to current date:

```
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: Examples

Change the output date format:

```
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            |
+---------------------+---------------------------------------+
```

10 - 17

Syntax and additional information for the temporal functions:

- `DATE_FORMAT (<date>, <format>)`
- Prefix the date/time formats with `%`.
  - Others include `%H` (hour), `%p` (AM or PM), `%T` (24 hour clock).
  - For other formats, see the MySQL Reference Manual at http://dev.mysql.com/doc/refman/5.6/en/date-and-time-functions.html#function_date-format.

Numeric Functions

Perform mathematical operations such as:

- Rounding
- Truncation
- Trigonometric calculations
- Generating random numbers

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

10 - 18

`ROUND(<number>)`:

- Works with `DECIMAL` data type
- Defaults to zero, if not specified
- For exact-value numbers, a value with a fractional part less than .5 is rounded down to the next integer if positive, or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that a value with any fractional part is rounded to the nearest even 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 sign (–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        |
+-----------+----------+---------+---------+----------+
```

10 - 19

The absolute value of a number is its distance from zero, without considering which direction from zero the number lies. An absolute number is always zero or positive.

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                     |
+-------------------------+
```

FALSE

TRUE

Control Flow Functions: CASE Functions

- The **CASE** function executes a series of conditional tests.
- Each test returns a Boolean result.
- The result determines the flow of control.
- It can be used in two different ways:
  - To compare expression values
  - To compare expression conditions

10 - 22

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
```

10 - 23

The first example in the slide returns the result where `value = compare_value`. The second slide example returns the result for the first condition it tests that is true. If no conditions are true, it returns the result after `ELSE`, or `NULL` if there is no `ELSE`.

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;
```

10 - 24

The default return type of a `CASE` expression is the compatible aggregated type of all return values, but the return type also depends on the context in which you use `CASE`. If you use it in a string context, the result is returned as a string. If used in a numeric context, the result is returned as a decimal, real, or integer value.

The last example in the slide assumes that you want to display a pet's gender as Boy or Girl instead of M or F. You can use the `CASE` function to return "Boy" when `pGender = 'M'` and "Girl" when `pGender = 'F'`:

```
+--------+-----+---------+----------+
| pName  | oID | pGender | CASE     |
+--------+-----+---------+----------+
| Claws  |   2 | M       | Boy      |
| Fluffy |   1 | F       | Girl     |
| Buffy  |   1 | F       | Girl     |
+--------+-----+---------+----------+
```

Aggregate Functions

- Perform summary operations on a set of values, such as:
  - Counting
  - Averaging
  - Finding minimum or maximum values
- Return a single value based on multiple values from different rows
- Include only non-null values in results

10 - 25

Aggregate functions are also known as group functions. Use them when you want a summary of the data in all rows, rather than the individual row values. These functions consider only values that are not null, except `COUNT(*)`, which includes all rows.

Aggregate Function Types

Some of the aggregate function types:

Function Syntax	Definition
MIN(<column_name>)	Finds the smallest value
MAX(<column_name>)	Finds the largest value
SUM(<column_name>)	Calculates the sum of values in <column_name>
AVG(<column_name>)	Calculates the average value of <column_name>
COUNT(<column_name>)	Counts non-null values in <column_name>. COUNT(*) includes records with null values.
GROUP_CONCAT(<column_name>)	Concatenates a set of strings to produce a single string

10 - 26

All the functions can use the `DISTINCT` keyword, although it is not useful for the `MAX()` and `MIN()` functions.

`DISTINCT` examples:

```
SUM(DISTINCT <column_name>)
AVG(DISTINCT <column_name>)
COUNT(DISTINCT <column_name>)
GROUP_CONCAT(DISTINCT <column_name>)
```

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 non-null values in the `Capital` column:

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

10 - 27

The second example in the slide returns a different result, because not every country has a capital city associated with it. The count does not include null values in the `Capital` column.

Aggregate Functions: GROUP BY Clause

- The **GROUP BY** clause places rows into groups.
  - Each group consists of rows that have the same value in one or more columns.
  - It calculates an aggregate 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   |
+---------------+-----------------+
```

10 - 28

If a `WHERE` clause selects 20 rows and `GROUP BY` arranges them into four groups of five rows each, aggregate functions produce a value for each of the four groups.

Without a `GROUP BY` clause, an aggregate function calculates the aggregate value based on all selected rows. That is, MySQL treats all rows as a single group:

```
mysql> SELECT AVG(Population) FROM COUNTRY;
+-----------------+
| AVG(Population) |
+-----------------+
|   25434098.1172 |
+-----------------+
1 row in set (0.00 sec)
```

The example in the slide groups the `Country` table rows by continent, and calculates the average population of countries in each continent.

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
```

10 - 29

The example in the slide creates a list of the countries for each form of government on the South American continent.

Aggregate Functions: GROUP BY and HAVING Clauses

- Use the **HAVING** clause to filter 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       |
+---------------+-----------------+
```

10 - 30

Use the `HAVING` modifier to require that the groups a `GROUP BY` clause produces satisfy particular criteria.

It is similar to the `WHERE` clause. The difference is that MySQL evaluates the `HAVING` clause after the grouping implied by the `GROUP BY` clause. This means that the `HAVING` condition can refer to aggregate functions. Do not put any part of a condition in the `HAVING` clause that could also appear in the `WHERE` clause. A good `HAVING` clause is always based on aggregate functions (because these are not allowed in the `WHERE` clause).

The slide example results in a list of continents whose sum (aggregate value) of country populations is greater than 100,000,000.

Aggregate Functions: GROUP BY Clause and WITH ROLLUP Modifier

- Use the **WITH ROLLUP** modifier to produce multiple levels of aggregate 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 |
+---------------+-----------------+
```

10 - 31

`WITH ROLLUP` adds extra rows to the aggregate output. These extra rows display the results of higher-level (or super-aggregate) aggregate operations.

The example in the slide shows the population of each continent and the total population of all continents.

Another way to do this is to run one query to get the per-continent totals and another to get the total for all continents. Using `WITH ROLLUP` lets you use a single query to get both.

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   |
+---------------+-----------------+
```

10 - 32

Adding a `WITH ROLLUP` modifier to the `GROUP BY` clause performs a super-aggregate operation. It forces the query to produce another row that shows the overall average, rather than just the sum of all the averages.

Note: `ORDER BY` does not work with `WITH ROLLUP`.

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 by 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 |
+----------+
```

10 - 33