12.5 String Functions

Table 12.7 String Operators

Name	Description
ASCII()	Return numeric value of left-most character
BIN()	Return a string containing binary representation of a number
BIT_LENGTH()	Return length of argument in bits
CHAR()	Return the character for each integer passed
CHAR_LENGTH()	Return number of characters in argument
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CONCAT()	Return concatenated string
CONCAT_WS()	Return concatenate with separator
ELT()	Return string at index number
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD()	Return the index (position) of the first argument in the subsequent arguments
FIND_IN_SET()	Return the index position of the first argument within the second argument
FORMAT()	Return a number formatted to specified number of decimal places
FROM_BASE64()	Decode to a base-64 string and return result
HEX()	Return a hexadecimal representation of a decimal or string value
INSERT()	Insert a substring at the specified position up to the specified number of characters
INSTR()	Return the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LOAD_FILE()	Load the named file
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase

Name	Description
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MATCH	Perform full-text search
MID()	Return a substring starting from the specified position
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
ORD()	Return character code for leftmost character of the argument
POSITION()	Synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Pattern matching using regular expressions
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Synonym for REGEXP
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds
SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings
SUBSTR()	Return the substring as specified
SUBSTRING()	Return the substring as specified

Name	Description
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
TO_BASE64()	Return the argument converted to a base-64 string
TRIM()	Remove leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Return a string containing hex representation of a number
UPPER()	Convert to uppercase
WEIGHT_STRING()	Return the weight string for a string

String-valued functions return NULL if the length of the result would be greater than the value of the max_allowed_packet system variable. See Section 5.1.1, “Configuring the Server”.

For functions that operate on string positions, the first position is numbered 1.

For functions that take length arguments, noninteger arguments are rounded to the nearest integer.

- ASCII(str)

  Returns the numeric value of the leftmost character of the string str. Returns 0 if str is the empty string. Returns NULL if str is NULL. ASCII() works for 8-bit characters.

```
mysql> SELECT ASCII('2');
        -> 50
mysql> SELECT ASCII(2);
        -> 50
mysql> SELECT ASCII('dx');
        -> 100
```

  See also the ORD() function.

- BIN(N)

  Returns a string representation of the binary value of N, where N is a longlong (BIGINT) number. This is equivalent to CONV(N,10,2). Returns NULL if N is NULL.

```
mysql> SELECT BIN(12);
        -> '1100'
```

- BIT_LENGTH(str)

  Returns the length of the string str in bits.

```
mysql> SELECT BIT_LENGTH('text');
        -> 32
```

- `CHAR(N,... [USING charset_name])`

  `CHAR()` interprets each argument `N` as an integer and returns a string consisting of the characters given by the code values of those integers. `NULL` values are skipped.

```
mysql> SELECT CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
        -> 'MMM'
```

  `CHAR()` arguments larger than 255 are converted into multiple result bytes. For example, `CHAR(256)` is equivalent to `CHAR(1,0)`, and `CHAR(256*256)` is equivalent to `CHAR(1,0,0)`:

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+----------------+----------------+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+----------------+----------------+
| 0100           | 0100           |
+----------------+----------------+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+------------------+--------------------+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+------------------+--------------------+
| 010000           | 010000             |
+------------------+--------------------+
```

  By default, `CHAR()` returns a binary string. To produce a string in a given character set, use the optional `USING` clause:

```
mysql> SELECT CHARSET(CHAR(X'65')), CHARSET(CHAR(X'65' USING utf8));
+----------------------+----------------------------------+
| CHARSET(CHAR(X'65')) | CHARSET(CHAR(X'65' USING utf8)) |
+----------------------+----------------------------------+
| binary               | utf8                             |
+----------------------+----------------------------------+
```

  If `USING` is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from `CHAR()` becomes `NULL`.

- `CHAR_LENGTH(str)`

  Returns the length of the string `str`, measured in characters. A multibyte character counts as a single character. This means that for a string containing five 2-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

- `CHARACTER_LENGTH(str)`

`CHARACTER_LENGTH()` is a synonym for `CHAR_LENGTH()`.

- `CONCAT(str1,str2,...)`

  Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent nonbinary string form.

  `CONCAT()` returns `NULL` if any argument is `NULL`.

```
mysql> SELECT CONCAT('My', 'S', 'QL');
        -> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
        -> NULL
mysql> SELECT CONCAT(14.3);
        -> '14.3'
```

  For quoted strings, concatenation can be performed by placing the strings next to each other:

```
mysql> SELECT 'My' 'S' 'QL';
        -> 'MySQL'
```

- `CONCAT_WS(separator,str1,str2,...)`

  `CONCAT_WS()` stands for Concatenate With Separator and is a special form of `CONCAT()`. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is `NULL`, the result is `NULL`.

```
mysql> SELECT CONCAT_WS(',','First name','Second name','Last
Name');
        -> 'First name,Second name,Last Name'
mysql> SELECT CONCAT_WS(',','First name',NULL,'Last Name');
        -> 'First name,Last Name'
```

  `CONCAT_WS()` does not skip empty strings. However, it does skip any `NULL` values after the separator argument.

- `ELT(N,str1,str2,str3,...)`

  `ELT()` returns the `N`th element of the list of strings: `str1` if `N` = `1`, `str2` if `N` = `2`, and so on. Returns `NULL` if `N` is less than `1` or greater than the number of arguments. `ELT()` is the complement of `FIELD()`.

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
        -> 'ej'
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
        -> 'foo'
```

- `EXPORT_SET(bits,on,off[,separator[,number_of_bits]])`

Returns a string such that for every bit set in the value `bits`, you get an `on` string and for every bit not set in the value, you get an `off` string. Bits in `bits` are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the `separator` string (the default being the comma character `,`). The number of bits examined is given by `number_of_bits`, which has a default of 64 if not specified. `number_of_bits` is silently clipped to 64 if larger than 64. It is treated as an unsigned integer, so a value of −1 is effectively the same as 64.

```
mysql> SELECT EXPORT_SET(5,'Y','N',',',4);
        -> 'Y,N,Y,N'
mysql> SELECT EXPORT_SET(6,'1','0',',',10);
        -> '0,1,1,0,0,0,0,0,0,0'
```

- `FIELD(str,str1,str2,str3,...)`

  Returns the index (position) of `str` in the `str1`, `str2`, `str3`, `...` list. Returns `0` if `str` is not found.

  If all arguments to `FIELD()` are strings, all arguments are compared as strings. If all arguments are numbers, they are compared as numbers. Otherwise, the arguments are compared as double.

  If `str` is `NULL`, the return value is `0` because `NULL` fails equality comparison with any value. `FIELD()` is the complement of `ELT()`.

  ```
  mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
          -> 2
  mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
          -> 0
  ```

- `FIND_IN_SET(str,strlist)`

  Returns a value in the range of 1 to `N` if the string `str` is in the string list `strlist` consisting of `N` substrings. A string list is a string composed of substrings separated by `,` characters. If the first argument is a constant string and the second is a column of type `SET`, the `FIND_IN_SET()` function is optimized to use bit arithmetic. Returns `0` if `str` is not in `strlist` or if `strlist` is the empty string. Returns `NULL` if either argument is `NULL`. This function does not work properly if the first argument contains a comma (`,`) character.

  ```
  mysql> SELECT FIND_IN_SET('b','a,b,c,d');
          -> 2
  ```

- `FORMAT(X,D[,locale])`

  Formats the number `X` to a format like `'#,###,###.##'`, rounded to `D` decimal places, and returns the result as a string. If `D` is `0`, the result has no decimal point or fractional part.

  The optional third parameter enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the `lc_time_names` system variable (see Section 10.7, “MySQL Server Locale Support”). If no locale is specified, the default is `'en_US'`.

```
mysql> SELECT FORMAT(12332.123456, 4);
        -> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
        -> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
        -> '12,332'
mysql> SELECT FORMAT(12332.2,2,'de_DE');
        -> '12.332,20'
```

- `FROM_BASE64(str)`

  Takes a string encoded with the base-64 encoded rules used by `TO_BASE64()` and returns the decoded result as a binary string. The result is `NULL` if the argument is `NULL` or not a valid base-64 string. See the description of `TO_BASE64()` for details about the encoding and decoding rules.

```
mysql> SELECT TO_BASE64('abc'), FROM_BASE64(TO_BASE64('abc'));
        -> 'JWJj', 'abc'
```

- `HEX(str)`, `HEX(N)`

  For a string argument `str`, `HEX()` returns a hexadecimal string representation of `str` where each byte of each character in `str` is converted to two hexadecimal digits. (Multibyte characters therefore become more than two digits.) The inverse of this operation is performed by the `UNHEX()` function.

  For a numeric argument `N`, `HEX()` returns a hexadecimal string representation of the value of `N` treated as a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,16)`. The inverse of this operation is performed by `CONV(HEX(N),16,10)`.

```
mysql> SELECT X'616263', HEX('abc'),
UNHEX(HEX('abc'));
        -> 'abc', 616263, 'abc'
mysql> SELECT HEX(255), CONV(HEX(255),16,10);
        -> 'FF', 255
```

- `INSERT(str,pos,len,newstr)`

  Returns the string `str`, with the substring beginning at position `pos` and `len` characters long replaced by the string `newstr`. Returns the original string if `pos` is not within the length of the string. Replaces the rest of the string from position `pos` if `len` is not within the length of the rest of the string. Returns `NULL` if any argument is `NULL`.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
        -> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
        -> 'QuWhat'
```

  This function is multibyte safe.

- `INSTR(str,substr)`

  Returns the position of the first occurrence of substring `substr` in string `str`. This is the same as the two-argument form of `LOCATE()`, except that the order of the arguments is reversed.

  ```
  mysql> SELECT INSTR('foobarbar', 'bar');
          -> 4
  mysql> SELECT INSTR('xbar', 'foobar');
          -> 0
  ```

  This function is multibyte safe, and is case sensitive only if at least one argument is a binary string.

- `LCASE(str)`

  `LCASE()` is a synonym for `LOWER()`.

  In MySQL 5.7, `LCASE()` used in a view is rewritten as `LOWER()` when storing the view's definition. (Bug #12844279)

- `LEFT(str,len)`

  Returns the leftmost `len` characters from the string `str`, or `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT LEFT('foobarbar', 5);
          -> 'fooba'
  ```

  This function is multibyte safe.

- `LENGTH(str)`

  Returns the length of the string `str`, measured in bytes. A multibyte character counts as multiple bytes. This means that for a string containing five 2-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

  ```
  mysql> SELECT LENGTH('text');
          -> 4
  ```

  Note

  The `Length()` OpenGIS spatial function is named `ST_Length()` in MySQL.

- `LOAD_FILE(file_name)`

  Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full path name to the file, and you must have the `FILE` privilege. The file must be readable by all and its size less than `max_allowed_packet` bytes. If the `secure_file_priv` system variable is set to a nonempty directory name, the file to be loaded must be located in that directory.

  If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

The character_set_filesystem system variable controls interpretation of file names that are given as literal strings.

```
mysql> UPDATE t
            SET
blob_col=LOAD_FILE('/tmp/picture')
            WHERE id=1;
```

- LOCATE(substr,str), LOCATE(substr,str,pos)

  The first syntax returns the position of the first occurrence of substring substr in string str. The second syntax returns the position of the first occurrence of substring substr in string str, starting at position pos. Returns 0 if substr is not in str. Returns NULL if substr or str is NULL.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
        -> 7
```

  This function is multibyte safe, and is case-sensitive only if at least one argument is a binary string.

- LOWER(str)

  Returns the string str with all characters changed to lowercase according to the current character set mapping. The default is latin1 (cp1252 West European).

```
mysql> SELECT LOWER('QUADRATICALLY');
        -> 'quadratically'
```

  LOWER() (and UPPER()) are ineffective when applied to binary strings (BINARY, VARBINARY, BLOB). To perform lettercase conversion, convert the string to a nonbinary string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING
latin1));
+-------------+-----------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
+-------------+-----------------------------------+
| New York    | new york                          |
+-------------+-----------------------------------+
```

  For collations of Unicode character sets, LOWER() and UPPER() work according to the Unicode Collation Algorithm (UCA) version in the collation name, if there is one, and UCA 4.0.0 if no version is specified. For example, utf8_unicode_520_ci works according to UCA 5.2.0, whereas utf8_unicode_ci works according to UCA 4.0.0. See Section 10.1.10.1, “Unicode Character Sets”.

  This function is multibyte safe.

In previous versions of MySQL, `LOWER()` used within a view was rewritten as `LCASE()` when storing the view's definition. In MySQL 5.7, `LOWER()` is never rewritten in such cases, but `LCASE()` used within views is instead rewritten as `LOWER()`. (Bug #12844279)

- `LPAD(str,len,padstr)`

  Returns the string `str`, left-padded with the string `padstr` to a length of `len` characters. If `str` is longer than `len`, the return value is shortened to `len` characters.

```
mysql> SELECT LPAD('hi',4,'??');
        -> '??hi'
mysql> SELECT LPAD('hi',1,'??');
        -> 'h'
```

- `LTRIM(str)`

  Returns the string `str` with leading space characters removed.

```
mysql> SELECT LTRIM('
barbar');
        -> 'barbar'
```

  This function is multibyte safe.

- `MAKE_SET(bits,str1,str2,...)`

  Returns a set value (a string containing substrings separated by `,` characters) consisting of the strings that have the corresponding bit in `bits` set. `str1` corresponds to bit 0, `str2` to bit 1, and so on. `NULL` values in `str1`, `str2`, `...` are not appended to the result.

```
mysql> SELECT MAKE_SET(1,'a','b','c');
        -> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
        -> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
        -> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
        -> ''
```

- `MID(str,pos,len)`

  `MID(str,pos,len)` is a synonym for `SUBSTRING(str,pos,len)`.

- `OCT(N)`

  Returns a string representation of the octal value of `N`, where `N` is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,8)`. Returns `NULL` if `N` is `NULL`.

```
mysql> SELECT OCT(12);
        -> '14'
```

- `OCTET_LENGTH(str)`

  `OCTET_LENGTH()` is a synonym for `LENGTH()`.

- `ORD(str)`

  If the leftmost character of the string `str` is a multibyte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

  ```
    (1st byte code)
  + (2nd byte code * 256)
  + (3rd byte code * 2562)
  ...
  ```

  If the leftmost character is not a multibyte character, `ORD()` returns the same value as the `ASCII()` function.

  ```
  mysql> SELECT ORD('2');
          -> 50
  ```

- `POSITION(substr IN str)`

  `POSITION(substr IN str)` is a synonym for `LOCATE(substr,str)`.

- `QUOTE(str)`

  Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotation marks and with each instance of backslash (\), single quote ('), ASCII `NUL`, and Control+Z preceded by a backslash. If the argument is `NULL`, the return value is the word "NULL" without enclosing single quotation marks.

  ```
  mysql> SELECT QUOTE('Don\'t!');
          -> 'Don\'t!'
  mysql> SELECT QUOTE(NULL);
          -> NULL
  ```

  For comparison, see the quoting rules for literal strings and within the C API in Section 9.1.1, "String Literals", and Section 27.8.7.56, "mysql_real_escape_string_quote()".

- `REPEAT(str,count)`

  Returns a string consisting of the string `str` repeated `count` times. If `count` is less than 1, returns an empty string. Returns `NULL` if `str` or `count` are `NULL`.

  ```
  mysql> SELECT REPEAT('MySQL', 3);
          -> 'MySQLMySQLMySQL'
  ```

- `REPLACE(str,from_str,to_str)`

Returns the string `str` with all occurrences of the string `from_str` replaced by the string `to_str`. `REPLACE()` performs a case-sensitive match when searching for `from_str`.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

This function is multibyte safe.

- `REVERSE(str)`

  Returns the string `str` with the order of the characters reversed.

  ```
  mysql> SELECT REVERSE('abc');
          -> 'cba'
  ```

  This function is multibyte safe.

- `RIGHT(str,len)`

  Returns the rightmost `len` characters from the string `str`, or `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT RIGHT('foobarbar', 4);
          -> 'rbar'
  ```

  This function is multibyte safe.

- `RPAD(str,len,padstr)`

  Returns the string `str`, right-padded with the string `padstr` to a length of `len` characters. If `str` is longer than `len`, the return value is shortened to `len` characters.

  ```
  mysql> SELECT RPAD('hi',5,'?');
          -> 'hi???'
  mysql> SELECT RPAD('hi',1,'?');
          -> 'h'
  ```

  This function is multibyte safe.

- `RTRIM(str)`

  Returns the string `str` with trailing space characters removed.

  ```
  mysql> SELECT RTRIM('barbar
  ');
          -> 'barbar'
  ```

  This function is multibyte safe.

- `SOUNDEX(str)`

Returns a soundex string from `str`. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the `SOUNDEX()` function returns an arbitrarily long string. You can use `SUBSTRING()` on the result to get a standard soundex string. All nonalphabetic characters in `str` are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

Important

When using `SOUNDEX()`, you should be aware of the following limitations:

- This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reliable results.
- This function is not guaranteed to provide consistent results with strings that use multibyte character sets, including `utf-8`. See Bug #22638 for more information.

```
mysql> SELECT SOUNDEX('Hello');
        -> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
        -> 'Q36324'
```

Note

This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

- `expr1 SOUNDS LIKE expr2`

  This is the same as `SOUNDEX(expr1) = SOUNDEX(expr2)`.

- `SPACE(N)`

  Returns a string consisting of `N` space characters.

```
mysql> SELECT SPACE(6);
        -> '      '
```

- `SUBSTR(str,pos)`, `SUBSTR(str FROM pos)`, `SUBSTR(str,pos,len)`, `SUBSTR(str FROM pos FOR len)`

  `SUBSTR()` is a synonym for `SUBSTRING()`.

- `SUBSTRING(str,pos)`, `SUBSTRING(str FROM pos)`, `SUBSTRING(str,pos,len)`, `SUBSTRING(str FROM pos FOR len)`

  The forms without a `len` argument return a substring from string `str` starting at position `pos`. The forms with

a `len` argument return a substring `len` characters long from string `str`, starting at position `pos`. The forms that use `FROM` are standard SQL syntax. It is also possible to use a negative value for `pos`. In this case, the beginning of the substring is `pos` characters from the end of the string, rather than the beginning. A negative value may be used for `pos` in any of the forms of this function.

For all forms of `SUBSTRING()`, the position of the first character in the string from which the substring is to be extracted is reckoned as `1`.

```
mysql> SELECT SUBSTRING('Quadratically',5);
        -> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
        -> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
        -> 'ratica'
mysql> SELECT SUBSTRING('Sakila', -3);
        -> 'ila'
mysql> SELECT SUBSTRING('Sakila', -5, 3);
        -> 'aki'
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
        -> 'ki'
```

This function is multibyte safe.

If `len` is less than 1, the result is the empty string.

- `SUBSTRING_INDEX(str,delim,count)`

  Returns the substring from string `str` before `count` occurrences of the delimiter `delim`. If `count` is positive, everything to the left of the final delimiter (counting from the left) is returned. If `count` is negative, everything to the right of the final delimiter (counting from the right) is returned. `SUBSTRING_INDEX()` performs a case-sensitive match when searching for `delim`.

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

  This function is multibyte safe.

- `TO_BASE64(str)`

  Converts the string argument to base-64 encoded form and returns the result as a character string with the connection character set and collation. If the argument is not a string, it is converted to a string before conversion takes place. The result is `NULL` if the argument is `NULL`. Base-64 encoded strings can be decoded using the `FROM_BASE64()` function.

  ```
  mysql> SELECT TO_BASE64('abc'), FROM_BASE64(TO_BASE64('abc'));
          -> 'JWJj', 'abc'
  ```

  Different base-64 encoding schemes exist. These are the encoding and decoding rules used by `TO_BASE64()` and `FROM_BASE64()`:

- The encoding for alphabet value 62 is `'+'`.
- The encoding for alphabet value 63 is `'/'`.
- Encoded output consists of groups of 4 printable characters. Each 3 bytes of the input data are encoded using 4 characters. If the last group is incomplete, it is padded with `'='` characters to a length of 4.
- A newline is added after each 76 characters of encoded output to divide long output into multiple lines.
- Decoding recognizes and ignores newline, carriage return, tab, and space.

- `TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)` `TRIM([remstr FROM] str)`

  Returns the string `str` with all `remstr` prefixes or suffixes removed. If none of the specifiers `BOTH`, `LEADING`, or `TRAILING` is given, `BOTH` is assumed. `remstr` is optional and, if not specified, spaces are removed.

  ```
  mysql> SELECT TRIM('  bar   ');
          -> 'bar'
  mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
          -> 'barxxx'
  mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
          -> 'bar'
  mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
          -> 'barx'
  ```

  This function is multibyte safe.

- `UCASE(str)`

  `UCASE()` is a synonym for `UPPER()`.

  In MySQL 5.7, `UCASE()` used in a view is rewritten as `UPPER()` when storing the view's definition. (Bug #12844279)

- `UNHEX(str)`

  For a string argument `str`, `UNHEX(str)` interprets each pair of characters in the argument as a hexadecimal number and converts it to the byte represented by the number. The return value is a binary string.

  ```
  mysql> SELECT UNHEX('4D7953514C');
          -> 'MySQL'
  mysql> SELECT X'4D7953514C';
          -> 'MySQL'
  mysql> SELECT UNHEX(HEX('string'));
          -> 'string'
  mysql> SELECT HEX(UNHEX('1267'));
          -> '1267'
  ```

  The characters in the argument string must be legal hexadecimal digits: `'0'` .. `'9'`, `'A'` .. `'F'`, `'a'` .. `'f'`. If the argument contains any nonhexadecimal digits, the result is `NULL`:

```
mysql> SELECT UNHEX('GG');
+-------------+
| UNHEX('GG') |
+-------------+
| NULL        |
+-------------+
```

A `NULL` result can occur if the argument to `UNHEX()` is a `BINARY` column, because values are padded with 0x00 bytes when stored but those bytes are not stripped on retrieval. For example, `'41'` is stored into a `CHAR(3)` column as `'41 '` and retrieved as `'41'` (with the trailing pad space stripped), so `UNHEX()` for the column value returns `'A'`. By contrast `'41'` is stored into a `BINARY(3)` column as `'41\0'` and retrieved as `'41\0'` (with the trailing pad `0x00` byte not stripped). `'\0'` is not a legal hexadecimal digit, so `UNHEX()` for the column value returns `NULL`.

For a numeric argument `N`, the inverse of `HEX(N)` is not performed by `UNHEX()`. Use `CONV(HEX(N),16,10)` instead. See the description of `HEX()`.

- `UPPER(str)`

  Returns the string `str` with all characters changed to uppercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

  ```
  mysql> SELECT UPPER('Hej');
          -> 'HEJ'
  ```

  See the description of `LOWER()` for information that also applies to `UPPER()`. This included information about how to perform lettercase conversion of binary strings (`BINARY`, `VARBINARY`, `BLOB`) for which these functions are ineffective, and information about case folding for Unicode character sets.

  This function is multibyte safe.

  In previous versions of MySQL, `UPPER()` used within a view was rewritten as `UCASE()` when storing the view's definition. In MySQL 5.7, `UPPER()` is never rewritten in such cases, but `UCASE()` used within views is instead rewritten as `UPPER()`. (Bug #12844279)

- `WEIGHT_STRING(str [AS {CHAR|BINARY}(N)] [LEVEL levels] [flags])`

  ```
  levels: N [ASC|DESC|REVERSE] [, N [ASC|DESC|REVERSE]]
  ...
  ```

  This function returns the weight string for the input string. The return value is a binary string that represents the comparison and sorting value of the string. It has these properties:

  `WEIGHT_STRING()` is a debugging function intended for internal use. Its behavior can change without notice between MySQL versions. It can be used for testing and debugging of collations, especially if you are adding a new collation. See Section 10.4, “Adding a Collation to a Character Set”.

  This list briefly summarizes the arguments. More details are given in the discussion following the list.

  - `str`: The input string expression.
  - `AS` clause: Optional; cast the input string to a given type and length.

- `LEVEL` clause: Optional; specify weight levels for the return value.
- `flags`: Optional; unused.

The input string, `str`, is a string expression. If the input is a nonbinary (character) string such as a `CHAR`, `VARCHAR`, or `TEXT` value, the return value contains the collation weights for the string. If the input is a binary (byte) string such as a `BINARY`, `VARBINARY`, or `BLOB` value, the return value is the same as the input (the weight for each byte in a binary string is the byte value). If the input is `NULL`, `WEIGHT_STRING()` returns `NULL`.

Examples:

```
mysql> SET @s = _latin1 'AB' COLLATE
latin1_swedish_ci;
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+-----------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+-----------------------+
| AB   | 4142    | 4142                  |
+------+---------+-----------------------+


mysql> SET @s = _latin1 'ab' COLLATE
latin1_swedish_ci;
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+-----------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+-----------------------+
| ab   | 6162    | 4142                  |
+------+---------+-----------------------+


mysql> SET @s = CAST('AB' AS BINARY);
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+-----------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+-----------------------+
| AB   | 4142    | 4142                  |
+------+---------+-----------------------+


mysql> SET @s = CAST('ab' AS BINARY);
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+-----------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+-----------------------+
| ab   | 6162    | 6162                  |
+------+---------+-----------------------+
```

The preceding examples use `HEX()` to display the `WEIGHT_STRING()` result. Because the result is a binary value, `HEX()` can be especially useful when the result contains nonprinting values, to display it in printable form:

```
mysql> SET @s = CONVERT(X'C39F' USING utf8) COLLATE utf8_czech_ci;
mysql> SELECT HEX(WEIGHT_STRING(@s));
+-----------------------+
| HEX(WEIGHT_STRING(@s)) |
+-----------------------+
| 0FEA0FEA              |
+-----------------------+
```

For non-`NULL` return values, the data type of the value is `VARBINARY` if its length is within the maximum length for `VARBINARY`, otherwise the data type is `BLOB`.

The `AS` clause may be given to cast the input string to a nonbinary or binary string and to force it to a given length:

- `AS CHAR(N)` casts the string to a nonbinary string and pads it on the right with spaces to a length of `N` characters. `N` must be at least 1. If `N` is less than the length of the input string, the string is truncated to `N` characters. No warning occurs for truncation.
- `AS BINARY(N)` is similar but casts the string to a binary string, `N` is measured in bytes (not characters), and padding uses `0x00` bytes (not spaces).

```
mysql> SET NAMES 'latin1';
mysql> SELECT HEX(WEIGHT_STRING('ab' AS CHAR(4)));
+-------------------------------------+
| HEX(WEIGHT_STRING('ab' AS CHAR(4))) |
+-------------------------------------+
| 41422020                            |
+-------------------------------------+
mysql> SET NAMES 'utf8';
mysql> SELECT HEX(WEIGHT_STRING('ab' AS CHAR(4)));
+-------------------------------------+
| HEX(WEIGHT_STRING('ab' AS CHAR(4))) |
+-------------------------------------+
| 0041004200200020                    |
+-------------------------------------+


mysql> SELECT HEX(WEIGHT_STRING('ab' AS BINARY(4)));
+---------------------------------------+
| HEX(WEIGHT_STRING('ab' AS BINARY(4))) |
+---------------------------------------+
| 61620000                              |
+---------------------------------------+
```

The `LEVEL` clause may be given to specify that the return value should contain weights for specific collation levels.

The `levels` specifier following the `LEVEL` keyword may be given either as a list of one or more integers separated by commas, or as a range of two integers separated by a dash. Whitespace around the punctuation characters does not matter.

Examples:

```
LEVEL 1
LEVEL 2, 3, 5
LEVEL 1-3
```

Any level less than 1 is treated as 1. Any level greater than the maximum for the input string collation is treated as maximum for the collation. The maximum varies per collation, but is never greater than 6.

In a list of levels, levels must be given in increasing order. In a range of levels, if the second number is less than the first, it is treated as the first number (for example, 4-2 is the same as 4-4).

If the `LEVEL` clause is omitted, MySQL assumes `LEVEL 1 - max`, where `max` is the maximum level for the collation.

If `LEVEL` is specified using list syntax (not range syntax), any level number can be followed by these modifiers:

- `ASC`: Return the weights without modification. This is the default.
- `DESC`: Return bitwise-inverted weights (for example, `0x78f0 DESC` = `0x870f`).
- `REVERSE`: Return the weights in reverse order (that is,the weights for the reversed string, with the first character last and the last first).

Examples:

```
mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1));
+-------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1)) |
+-------------------------------------+
| 007FFF                              |
+-------------------------------------+


mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC));
+----------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC)) |
+----------------------------------------+
| FF8000                                 |
+----------------------------------------+


mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1 REVERSE));
+-------------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1 REVERSE)) |
+-------------------------------------------+
| FF7F00                                    |
+-------------------------------------------+
```

```
mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC REVERSE));
+-------------------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC REVERSE)) |
+-------------------------------------------------+
| 0080FF                                          |
+-------------------------------------------------+
```

The `flags` clause currently is unused.