SQL string functions are used primarily for string manipulation. The following table details the important string functions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>Returns numeric value of left-most character</td>
</tr>
<tr>
<td>BIN</td>
<td>Returns a string representation of the argument</td>
</tr>
<tr>
<td>BIT_LENGTH</td>
<td>Returns length of argument in bits</td>
</tr>
<tr>
<td>CHAR_LENGTH</td>
<td>Returns number of characters in argument</td>
</tr>
<tr>
<td>CHAR</td>
<td>Returns the character for each integer passed</td>
</tr>
<tr>
<td>CHARACTER_LENGTH</td>
<td>A synonym for CHAR_LENGTH</td>
</tr>
<tr>
<td>CONCAT_WS</td>
<td>Returns concatenate with separator</td>
</tr>
<tr>
<td>CONCAT</td>
<td>Returns concatenated string</td>
</tr>
<tr>
<td>CONV</td>
<td>Converts numbers between different number bases</td>
</tr>
<tr>
<td>ELT</td>
<td>Returns string at index number</td>
</tr>
<tr>
<td>EXPORT_SET</td>
<td>Returns 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</td>
</tr>
<tr>
<td>FIELD</td>
<td>Returns the index position of the first argument in the subsequent arguments</td>
</tr>
<tr>
<td>FIND_IN_SET</td>
<td>Returns the index position of the first argument within the second argument</td>
</tr>
<tr>
<td>FORMAT</td>
<td>Returns a number formatted to specified number of decimal places</td>
</tr>
<tr>
<td>HEX</td>
<td>Returns a string representation of a hex value</td>
</tr>
<tr>
<td>INSERT</td>
<td>Inserts a substring at the specified position up to the specified number of characters</td>
</tr>
<tr>
<td>INSTR</td>
<td>Returns the index of the first occurrence of substring</td>
</tr>
<tr>
<td>LCASE</td>
<td>Synonym for LOWER</td>
</tr>
<tr>
<td>LEFT</td>
<td>Returns the leftmost number of characters as specified</td>
</tr>
<tr>
<td>LENGTH</td>
<td>Returns the length of a string in bytes</td>
</tr>
<tr>
<td>LOAD_FILE</td>
<td>Loads the named file</td>
</tr>
<tr>
<td>LOCATE</td>
<td>Returns the position of the first occurrence of substring</td>
</tr>
<tr>
<td>LOWER</td>
<td>Returns the argument in lowercase</td>
</tr>
<tr>
<td>LPAD</td>
<td>Returns the string argument, left-padded with the specified string</td>
</tr>
<tr>
<td>LTRIM</td>
<td>Removes leading spaces</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MAKE_SET</td>
<td>Returns a set of comma-separated strings that have the corresponding bit in bits set</td>
</tr>
<tr>
<td>MID</td>
<td>Returns a substring starting from the specified position</td>
</tr>
<tr>
<td>OCT</td>
<td>Returns a string representation of the octal argument</td>
</tr>
<tr>
<td>OCTET_LENGTH</td>
<td>A synonym for LENGTH</td>
</tr>
<tr>
<td>ORD</td>
<td>If the leftmost character of the argument is a multi-byte character, returns the code for that character</td>
</tr>
<tr>
<td>POSITION</td>
<td>A synonym for LOCATE</td>
</tr>
<tr>
<td>QUOTE</td>
<td>Escapes the argument for use in an SQL statement</td>
</tr>
<tr>
<td>REGEXP</td>
<td>Pattern matching using regular expressions</td>
</tr>
<tr>
<td>REPEAT</td>
<td>Repeats a string the specified number of times</td>
</tr>
<tr>
<td>REPLACE</td>
<td>Replaces occurrences of a specified string</td>
</tr>
<tr>
<td>REVERSE</td>
<td>Reverses the characters in a string</td>
</tr>
<tr>
<td>RIGHT</td>
<td>Returns the specified rightmost number of characters</td>
</tr>
<tr>
<td>RPAD</td>
<td>Appends string the specified number of times</td>
</tr>
<tr>
<td>RTRIM</td>
<td>Removes trailing spaces</td>
</tr>
<tr>
<td>SOUNDEX</td>
<td>Returns a soundex string</td>
</tr>
<tr>
<td>SOUNDS LIKE</td>
<td>Compares sounds</td>
</tr>
<tr>
<td>SPACE</td>
<td>Returns a string of the specified number of spaces</td>
</tr>
<tr>
<td>STRCMP</td>
<td>Compares two strings</td>
</tr>
<tr>
<td>SUBSTRING_INDEX</td>
<td>Returns a substring from a string before the specified number of occurrences of the delimiter</td>
</tr>
<tr>
<td>SUBSTRING, SUBSTR</td>
<td>Returns the substring as specified</td>
</tr>
<tr>
<td>TRIM</td>
<td>Removes leading and trailing spaces</td>
</tr>
<tr>
<td>UCASE</td>
<td>Synonym for UPPER</td>
</tr>
<tr>
<td>UNHEX</td>
<td>Converts each pair of hexadecimal digits to a character</td>
</tr>
<tr>
<td>UPPER</td>
<td>Converts to uppercase</td>
</tr>
</tbody>
</table>

**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 characters with numeric values from 0 to 255.

```sql
SQL> SELECT ASCII('2');
+-------------------+
| ASCII('2')        |
| 50                |
+-------------------+ 1 row in set (0.00 sec)

SQL> SELECT ASCII('dx');
+-------------------+
| ASCII('dx')       |
+-------------------+
```
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.

SQL> SELECT BIN(12);
+ BIN(12) +
| 1100 +
1 row in set (0.00 sec)

BIT_LENGTH(str)
Returns the length of the string str in bits.

SQL> SELECT BIT_LENGTH('text');
+ BIT_LENGTH('text') +
| 32 +
1 row in set (0.00 sec)

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.

SQL> SELECT CHAR(77, 121, 83, 81, '76');
+ CHAR(77, 121, 83, 81, '76') +
| SQL +
1 row in set (0.00 sec)

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

SQL> SELECT CHAR_LENGTH("text");
+ CHAR_LENGTH("text") +
| 4 +
1 row in set (0.00 sec)

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 non-binary strings, the result is a non-binary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form; if you want to avoid that, you can use an explicit type cast, as in this example:

```
SQL> SELECT CONCAT('My', 'S', 'QL');
+---------------------------------------------+
| CONCAT('My', 'S', 'QL')                     |
| SQL                                         |
+---------------------------------------------+
1 row in set (0.00 sec)
```

**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.

```
SQL> SELECT CONCAT_WS(' ', 'First name', 'Last Name');
+---------------------------------------------+
| CONCAT_WS(' ', 'First name', 'Last Name')  |
+---------------------------------------------+
| First name, Last Name                      |
+---------------------------------------------+
1 row in set (0.00 sec)
```

**CONV(N, from_base, to_base)**

Converts numbers between different number bases. Returns a string representation of the number N, converted from base from_base to to_base. Returns NULL if any argument is NULL. The argument N is interpreted as an integer, but may be specified as an integer or a string. The minimum base is 2 and the maximum base is 36. If to_base is a negative number, N is regarded as a signed number. Otherwise, N is treated as unsigned. CONV works with 64-bit precision.

```
SQL> SELECT CONV('a', 16, 2);
+---------------------------------------------+
| CONV('a', 16, 2)                            |
| 1010                                         |
+---------------------------------------------+
1 row in set (0.00 sec)
```

**ELT(N, str1, str2, str3,...)**

Returns 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.

```
SQL> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
+---------------------------------------------+
| ELT(1, 'ej', 'Heja', 'hej', 'foo')         |
| ej                                          |
+---------------------------------------------+
1 row in set (0.00 sec)
```

**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 fromlow – orderto high – orderbits. 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 default is 64.
### SQL Examples

**SELECT EXPORT_SET**

```sql
SQL> SELECT EXPORT_SET(5, 'Y', 'N', ',', 4);
+---------------------------------------------------------+
| EXPORT_SET(5, 'Y', 'N', ',', 4)                          |
| Y, N, Y, N                                              |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

**FIELD str, str1, str2, str3,…**

Returns the index *position starting with* 1 of str in the str1, str2, str3, ... list. Returns 0 if str is not found.

```sql
SQL> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
+---------------------------------------------------------+
| FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo')          |
| 2                                                       |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

**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.

```sql
SQL> SELECT FIND_IN_SET('b', 'a,b,c,d');
+---------------------------------------------------------+
| SELECT FIND_IN_SET('b', 'a,b,c,d')                       |
| 2                                                       |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

**FORMAT X, D**

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.

```sql
SQL> SELECT FORMAT(12332.123456, 4);
+---------------------------------------------------------+
| FORMAT(12332.123456, 4)                                 |
| 12,332.1235                                             |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

**HEX N_or_S**

If N_or_S is a number, returns a string representation of the hexadecimal value of N, where N is a longlong *BIGINT* number. This is equivalent to CONV N, 10, 16.

If N_or_S is a string, returns a hexadecimal string representation of N_or_S where each character in N_or_S is converted to two hexadecimal digits.

```sql
SQL> SELECT HEX(255);
+---------------------------------------------------------+
| HEX(255)                                                |
| FF                                                      |
+---------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT 0x616263;
+---------------------------------------------------------+
|                                                           |
+---------------------------------------------------------+
```
**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.

```sql
SQL> SELECT INSERT('Quadratic', 3, 4, 'What');
+--------------------------+
| INSERT('Quadratic', 3, 4, 'What') |
| QuWhattic                |
+--------------------------+
1 row in set (0.00 sec)
```

**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.

```sql
SQL> SELECT INSTR('foobarbar', 'bar');
+-------------------------+
| INSTR('foobarbar', 'bar') |
| 4                       |
+-------------------------+
1 row in set (0.00 sec)
```

**LCASE**\(str\)

LCASE is a synonym for LOWER.

**LEFT**\(str, len\)

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

```sql
SQL> SELECT LEFT('foobarbar', 5);
+----------------+
| LEFT('foobarbar', 5) |
| fooba           |
+----------------+
1 row in set (0.00 sec)
```

**LENGTH**\(str\)

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

```sql
SQL> SELECT LENGTH('text');
+----------------+
| LENGTH('text') |
| 4              |
+----------------+
1 row in set (0.00 sec)
```
**LOAD_FILE**

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

As of SQL 5.0.19, the character_set_filesystem system variable controls interpretation of filenames that are given as literal strings.

```sql
SQL> UPDATE table_test
   -> SET blob_col=LOAD_FILE('/tmp/picture')
   -> WHERE id=1;
```

**LOCATE**

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.

```sql
SQL> SELECT LOCATE('bar', 'foobarbar');
```

**LOWER**

Returns the string str with all characters changed to lowercase according to the current character set mapping.

```sql
SQL> SELECT LOWER('QUADRATICALLY');
```

**LPAD**

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.

```sql
SQL> SELECT LPAD('hi', 4, '??');
```

**LTRIM**

Returns the string str with leading space characters removed.

```sql
SQL> SELECT LTRIM('   barbar');
```
MAKE_SET

Returns a set value as strings 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.

SQL> SELECT MAKE_SET(1, 'a', 'b', 'c');

| MAKE_SET(1, 'a', 'b', 'c') |
| a |

1 row in set (0.00 sec)

MID

MIDstr, pos, len is a synonym for SUBSTRING str, pos, len.

OCT

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.

SQL> SELECT OCT(12);

| OCT(12) |
| 14 |

1 row in set (0.00 sec)

OCTET_LENGTH

OCTET_LENGTH is a synonym for LENGTH.

ORD

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

\[
\text{code} = (1\text{st byte code}) + (2\text{nd byte code} \times 256) + (3\text{rd byte code} \times 2562) \ldots
\]

If the leftmost character is not a multi-byte character, ORD returns the same value as the ASCII function.

SQL> SELECT ORD('2');

| ORD('2') |
| 50 |

1 row in set (0.00 sec)

POSITION

POSITION is a synonym for LOCATE substr, str.
QUOTE\texttt{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 quotes and with each instance of single quote ‘, backslash \, ASCII NUL, and Control-Z preceded by a backslash. If the argument is NULL, the return value is the word ‘NULL’ without enclosing single quotes.

\begin{verbatim}
SQL> SELECT QUOTE('Don\'t!');
+----------------------------------------+
| QUOTE('Don\'t!')                     |
| 'Don\'t!'                              |
+----------------------------------------+
1 row in set (0.00 sec)
\end{verbatim}

\textbf{NOTE:} Please check if your installation has any bug with this function then don't use this function.

\textbf{expr REGEXP pattern}

This function performs a pattern match of expr against pattern. Returns 1 if expr matches pat; otherwise it returns 0. If either expr or pat is NULL, the result is NULL. REGEXP is not case sensitive, except when used with binary strings.

\begin{verbatim}
SQL> SELECT 'ABCDEF' REGEXP 'A%C%';
+----------------------------------------+
| 'ABCDEF' REGEXP 'A%C%'               |
| 0                                     |
+----------------------------------------+
1 row in set (0.00 sec)
\end{verbatim}

Another example is:

\begin{verbatim}
SQL> SELECT 'ABCDE' REGEXP '.*';
+----------------------------------------+
| 'ABCDE' REGEXP '.*'                  |
| 1                                     |
+----------------------------------------+
1 row in set (0.00 sec)
\end{verbatim}

Let's see one more example:

\begin{verbatim}
SQL> SELECT 'new\n*line' REGEXP 'new\n*\n*line';
+----------------------------------------+
| 'new\n*line' REGEXP 'new\n*\n*line'       |
| 1                                     |
+----------------------------------------+
1 row in set (0.00 sec)
\end{verbatim}

\textbf{REPEAT} \texttt{str, count}

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

\begin{verbatim}
SQL> SELECT REPEAT('SQL', 3);
+----------------------------------------+
| REPEAT('SQL', 3)                      |
| SQLSQLSQL                               |
+----------------------------------------+
1 row in set (0.00 sec)
\end{verbatim}
### `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`.

**Example**

```sql
SQL> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
+---------------------------------------------------------+
| REPLACE('www.mysql.com', 'w', 'Ww')                     |
| wwwWwWw.mysql.com                                      |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

### `REVERSE(str)`

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

**Example**

```sql
SQL> SELECT REVERSE('abcd');
+---------------------------------------------------------+
| REVERSE('abcd')                                         |
| dcba                                                    |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

### `RIGHT(str, len)`

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

**Example**

```sql
SQL> SELECT RIGHT('foobarbar', 4);
+---------------------------------------------------------+
| RIGHT('foobarbar', 4)                                    |
| rbar                                                    |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

### `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.

**Example**

```sql
SQL> SELECT RPAD('hi', 5, '?');
+---------------------------------------------------------+
| RPAD('hi', 5, '?')                                       |
| hi??                                                    |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

### `RTRIM(str)`

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

**Example**

```sql
SQL> SELECT RTRIM('barbar   ');
+---------------------------------------------------------+
| RTRIM('barbar   ')                                       |
| barbar                                                  |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

### `SOUNDEX(str)`

**Example**

```sql
SQL> SELECT SOUNDEX('barbar');
+---------------------------------------------------------+
| SOUNDEX('barbar')                                        |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```
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 non-alphabetic characters in str are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

```
SQL> SELECT SOUNDEX('Hello');
+----------------------------------+
| SOUNDEX('Hello')                |
| H400                            |
+----------------------------------+
1 row in set (0.00 sec)
```

 expr1 SOUNDS LIKE expr2
This is the same as SOUNDEXexpr1 = SOUNDEXexpr2.

SPACE N
Returns a string consisting of N space characters.

```
SQL> SELECT SPACE(6);
+----------------------------------+
| SELECT SPACE(6)                 |
|                                 |
|                                 |
+----------------------------------+
1 row in set (0.00 sec)
```

STRCMP str1, str2
Compares two strings and returns 0 if both strings are equal, it returns -1 if the first argument is smaller than the second according to the current sort order otherwise it returns 1.

```
SQL> SELECT STRCMP('MOHD', 'MOHD');
+----------------------------------+
| STRCMP('MOHD', 'MOHD')           |
| 0                                |
+----------------------------------+
1 row in set (0.00 sec)
```

Another example is:

```
SQL> SELECT STRCMP('AMOHD', 'MOHD');
+----------------------------------+
| STRCMP('AMOHD', 'MOHD')          |
| -1                               |
+----------------------------------+
1 row in set (0.00 sec)
```

Let's see one more example:

```
SQL> SELECT STRCMP('MOHD', 'AMOHD');
+----------------------------------+
| STRCMP('MOHD', 'AMOHD')          |
| 1                                |
+----------------------------------+
1 row in set (0.00 sec)
```

SUBSTRING str, pos
**SUBSTRING**\texttt{strFROMpos}

**SUBSTRING**\texttt{str, pos, len}

**SUBSTRING**\texttt{strFROMposFORlen}

The forms without a \texttt{len} argument return a substring from \texttt{string str} starting at position \texttt{pos}. The forms with a \texttt{len} argument return a substring \texttt{len} characters long from \texttt{string str}, starting at position \texttt{pos}. The forms that use \texttt{FROM} are standard SQL syntax. It is also possible to use a negative value for \texttt{pos}. In this case, the beginning of the substring is \texttt{pos} characters from the end of the string, rather than the beginning. A negative value may be used for \texttt{pos} in any of the forms of this function.

```sql
SQL> SELECT SUBSTRING('Quadratically', 5);
   +-------------------------+
   | SUBSTRING('Quadratically', 5) |
   | ratically                |
   +-------------------------+ 1 row in set (0.00 sec)

SQL> SELECT SUBSTRING('foobarbar' FROM 4);
   +-------------------------+
   | SUBSTRING('foobarbar' FROM 4) |
   | barbar                   |
   +-------------------------+ 1 row in set (0.00 sec)

SQL> SELECT SUBSTRING('Quadratically', 5, 6);
   +-------------------------+
   | SUBSTRING('Quadratically', 5, 6) |
   | ratica                   |
   +-------------------------+ 1 row in set (0.00 sec)
```

**SUBSTRING\_INDEX**\texttt{str, delim, count}

Returns the substring from \texttt{string str} before count occurrences of the delimiter \texttt{delim}. If count is positive, everything to the left of the final delimiter \texttt{countfromthel left} is returned. If count is negative, everything to the right of the final delimiter \texttt{countfromtheright} is returned. \texttt{SUBSTRING\_INDEX} performs a case-sensitive match when searching for \texttt{delim}.

```sql
SQL> SELECT SUBSTRING\_INDEX('www.mysql.com', '.', 2);
   +-------------------------+
   | SUBSTRING\_INDEX('www.mysql.com', '.', 2) |
   | www.mysql                |
   +-------------------------+ 1 row in set (0.00 sec)
```

**TRIM**\texttt{[BOTH|LEADING|TRAILING][remstrFROMstr]}\texttt{str}

**TRIM**\texttt{[remstrFROMstr]}\texttt{str}

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

```sql
SQL> SELECT TRIM(' bar ');
   +-------------------------+
   | TRIM(' bar ')           |
   | bar                     |
   +-------------------------+
```
UCASE*str*

UCASE is a synonym for UPPER.

UNHEX*str*

Performs the inverse operation of HEX*str*. That is, it interprets each pair of hexadecimal digits in the argument as a number and converts it to the character represented by the number. The resulting characters are returned as a binary string.

```
SQL> SELECT UNHEX('4D7953514C');
+---------------------------------------------------------+
| UNHEX('4D7953514C')                                   |
| SQL                                                     |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

The characters in the argument string must be legal hexadecimal digits: ‘0’ .. ‘9’, ‘A’ .. ‘F’, ‘a’ .. ‘f’. If UNHEX encounters any non-hexadecimal digits in the argument, it returns NULL.

UPPER*str*

Returns the string str with all characters changed to uppercase according to the current character set mapping.

```
SQL> SELECT UPPER('Allah-hus-samad');
+---------------------------------------------------------+
| UPPER('Allah-hus-samad')                                 |
| ALLAH-HUS-SAMAD                                         |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```