



*Small. Fast. Reliable.
Choose any three.*

Home Menu About Documentation Download License Support Purchase Search

SQL As Understood By SQLite

[\[Top\]](#)

Core Functions

The core functions shown below are available by default. [Date & Time functions](#), [aggregate functions](#), [window functions](#), and [JSON functions](#) are documented separately. An application may define additional functions written in C and added to the database engine using the [sqlite3_create_function\(\)](#) API.

- [abs\(X\)](#)
- [changes\(\)](#)
- [char\(X1,X2,...,XN\)](#)
- [coalesce\(X,Y,...\)](#)
- [glob\(X,Y\)](#)
- [hex\(X\)](#)
- [ifnull\(X,Y\)](#)
- [instr\(X,Y\)](#)
- [last_insert_rowid\(\)](#)
- [length\(X\)](#)
- [like\(X,Y\)](#)
- [like\(X,Y,Z\)](#)
- [likelihood\(X,Y\)](#)
- [likely\(X\)](#)
- [load_extension\(X\)](#)
- [load_extension\(X,Y\)](#)
- [lower\(X\)](#)
- [ltrim\(X\)](#)
- [ltrim\(X,Y\)](#)
- [max\(X,Y,...\)](#)
- [min\(X,Y,...\)](#)
- [nullif\(X,Y\)](#)
- [printf\(FORMAT,...\)](#)
- [quote\(X\)](#)
- [random\(\)](#)
- [randomblob\(N\)](#)
- [replace\(X,Y,Z\)](#)
- [round\(X\)](#)
- [round\(X,Y\)](#)
- [rtrim\(X\)](#)
- [rtrim\(X,Y\)](#)
- [soundex\(X\)](#)
- [sqlite_compileoption_get\(N\)](#)
- [sqlite_compileoption_used\(X\)](#)
- [sqlite_offset\(X\)](#)
- [sqlite_source_id\(\)](#)
- [sqlite_version\(\)](#)
- [substr\(X,Y\)](#)
- [substr\(X,Y,Z\)](#)
- [total_changes\(\)](#)
- [trim\(X\)](#)
- [trim\(X,Y\)](#)
- [typeof\(X\)](#)
- [unicode\(X\)](#)
- [unlikely\(X\)](#)
- [upper\(X\)](#)
- [zeroblob\(N\)](#)

abs(X)

The `abs(X)` function returns the absolute value of the numeric argument `X`. `Abs(X)` returns `NULL` if `X` is `NULL`. `Abs(X)` returns `0.0` if `X` is a string or blob that cannot be converted to a numeric value. If `X` is the integer `-9223372036854775808` then `abs(X)` throws an integer overflow error since there is no equivalent positive 64-bit two complement value.

changes()

The `changes()` function returns the number of database rows that were changed or inserted or deleted by the most recently completed `INSERT`, `DELETE`, or `UPDATE` statement, exclusive of statements in lower-level triggers. The `changes()` SQL function is a wrapper around the [sqlite3_changes\(\)](#) C/C++ function and hence follows the same rules for counting changes.

char(X1,X2,...,XN)

The `char(X1,X2,...,XN)` function returns a string composed of characters having the unicode code point values of integers `X1` through `XN`, respectively.

coalesce(X,Y,...)

The `coalesce()` function returns a copy of its first non-`NULL` argument, or `NULL` if all arguments are `NULL`. `Coalesce()` must have at least 2 arguments.

glob(X,Y)

The `glob(X,Y)` function is equivalent to the expression `"Y GLOB X"`. Note that the `X` and `Y` arguments are reversed in the `glob()` function relative to the infix [GLOB](#) operator. `Y` is the string and `X` is the pattern. So, for example, the following expressions are equivalent:

```
name GLOB '*helium*'
glob('*helium*',name)
```

If the [sqlite3_create_function\(\)](#) interface is used to override the glob(X,Y) function with an alternative implementation then the [GLOB](#) operator will invoke the alternative implementation.

hex(X)

The hex() function interprets its argument as a BLOB and returns a string which is the upper-case hexadecimal rendering of the content of that blob.

ifnull(X,Y)

The ifnull() function returns a copy of its first non-NULL argument, or NULL if both arguments are NULL. Ifnull() must have exactly 2 arguments. The ifnull() function is equivalent to [coalesce\(\)](#) with two arguments.

instr(X,Y)

The instr(X,Y) function finds the first occurrence of string Y within string X and returns the number of prior characters plus 1, or 0 if Y is nowhere found within X. Or, if X and Y are both BLOBs, then instr(X,Y) returns one more than the number bytes prior to the first occurrence of Y, or 0 if Y does not occur anywhere within X. If both arguments X and Y to instr(X,Y) are non-NULL and are not BLOBs then both are interpreted as strings. If either X or Y are NULL in instr(X,Y) then the result is NULL.

last_insert_rowid()

The last_insert_rowid() function returns the [ROWID](#) of the last row insert from the database connection which invoked the function. The last_insert_rowid() SQL function is a wrapper around the [sqlite3_last_insert_rowid\(\)](#) C/C++ interface function.

length(X)

For a string value X, the length(X) function returns the number of characters (not bytes) in X prior to the first NUL character. Since SQLite strings do not normally contain NUL characters, the length(X) function will usually return the total number of characters in the string X. For a blob value X, length(X) returns the number of bytes in the blob. If X is NULL then length(X) is NULL. If X is numeric then length(X) returns the length of a string representation of X.

like(X,Y) like(X,Y,Z)

The like() function is used to implement the "**Y LIKE X [ESCAPE Z]**" expression. If the optional ESCAPE clause is present, then the like() function is invoked with three arguments. Otherwise, it is invoked with two arguments only. Note that the X and Y parameters are reversed in the like() function relative to the infix [LIKE](#) operator. X is the pattern and Y is the string to match against that pattern. Hence, the following expressions are equivalent:

```
name LIKE '%neon%'
like('%neon%',name)
```

The [sqlite3_create_function\(\)](#) interface can be used to override the like() function and thereby change the operation of the [LIKE](#) operator. When overriding the like() function, it may be important to override both the two and three argument versions of the like() function. Otherwise, different code may be called to implement the [LIKE](#) operator depending on whether or not an ESCAPE clause was specified.

likelihood(X,Y)

The likelihood(X,Y) function returns argument X unchanged. The value Y in likelihood(X,Y) must be a floating point constant between 0.0 and 1.0, inclusive. The likelihood(X) function is a no-op that the code generator optimizes away so that it consumes no CPU cycles during run-time (that is, during calls to [sqlite3_step\(\)](#)). The purpose of the likelihood(X,Y) function is to provide a hint to the query planner that the argument X is a boolean that is true with a probability of approximately Y. The [unlikely\(X\)](#) function is short-hand for likelihood(X,0.0625). The [likely\(X\)](#) function is short-hand for likelihood(X,0.9375).

likely(X)

The likely(X) function returns the argument X unchanged. The likely(X) function is a no-op that the code generator optimizes away so that it consumes no CPU cycles at run-time (that is, during calls to [sqlite3_step\(\)](#)). The purpose of the likely(X) function is to provide a hint to the query planner that the argument X is a boolean value that is usually true. The likely(X) function is equivalent to [likelihood\(X,0.9375\)](#). See also: [unlikely\(X\)](#).

load_extension(X)
load_extension(X,Y)

The load_extension(X,Y) function loads [SQLite extensions](#) out of the shared library file named X using the entry point Y. The result of load_extension() is always a NULL. If Y is omitted then the default entry point name is used. The load_extension() function raises an exception if the extension fails to load or initialize correctly.

The load_extension() function will fail if the extension attempts to modify or delete an SQL function or collating sequence. The extension can add new functions or collating sequences, but cannot modify or delete existing functions or collating sequences because those functions and/or collating sequences might be used elsewhere in the currently running SQL statement. To load an extension that changes or deletes functions or collating sequences, use the [sqlite3_load_extension\(\)](#) C-language API.

For security reasons, extension loaded is turned off by default and must be enabled by a prior call to [sqlite3_enable_load_extension\(\)](#).

lower(X)

The lower(X) function returns a copy of string X with all ASCII characters converted to lower case. The default built-in lower() function works for ASCII characters only. To do case conversions on non-ASCII characters, load the ICU extension.

ltrim(X)
ltrim(X,Y)

The ltrim(X,Y) function returns a string formed by removing any and all characters that appear in Y from the left side of X. If the Y argument is omitted, ltrim(X) removes spaces from the left side of X.

max(X,Y,...)

The multi-argument max() function returns the argument with the maximum value, or return NULL if any argument is NULL. The multi-argument max() function searches its arguments from left to right for an argument that defines a collating function and uses that collating function for all string comparisons. If none of the arguments to max() define a collating function, then the BINARY collating function is used. Note that **max()** is a simple function when it has 2 or more arguments but operates as an [aggregate function](#) if given only a single argument.

min(X,Y,...)

The multi-argument min() function returns the argument with the minimum value. The multi-argument min() function searches its arguments from left to right for an argument that defines a collating function and uses that collating function for all string comparisons. If none of the arguments to min() define a collating function, then the BINARY collating function is used. Note that **min()** is a simple function when it has 2 or more arguments but operates as an [aggregate function](#) if given only a single argument.

nullif(X,Y)

The nullif(X,Y) function returns its first argument if the arguments are different and NULL if the arguments are the same. The nullif(X,Y) function searches its arguments from left to right for an argument that defines a collating function and uses that collating function for all string comparisons. If neither argument to nullif() defines a collating function then the BINARY is used.

printf(FORMAT,...)

The printf(FORMAT,...) SQL function works like the [sqlite3_mprintf\(\)](#) C-language function and the printf() function from the standard C library. The first argument is a format string that specifies how to construct

the output string using values taken from subsequent arguments. If the `FORMAT` argument is missing or `NULL` then the result is `NULL`. The `%n` format is silently ignored and does not consume an argument. The `%p` format is an alias for `%X`. The `%z` format is interchangeable with `%s`. If there are too few arguments in the argument list, missing arguments are assumed to have a `NULL` value, which is translated into 0 or 0.0 for numeric formats or an empty string for `%s`. See the [built-in printf\(\)](#) documentation for additional information.

quote(X)

The `quote(X)` function returns the text of an SQL literal which is the value of its argument suitable for inclusion into an SQL statement. Strings are surrounded by single-quotes with escapes on interior quotes as needed. BLOBs are encoded as hexadecimal literals. Strings with embedded NUL characters cannot be represented as string literals in SQL and hence the returned string literal is truncated prior to the first NUL.

random()

The `random()` function returns a pseudo-random integer between -9223372036854775808 and +9223372036854775807.

randomblob(N)

The `randomblob(N)` function return an N-byte blob containing pseudo-random bytes. If N is less than 1 then a 1-byte random blob is returned.

Hint: applications can generate globally unique identifiers using this function together with [hex\(\)](#) and/or [lower\(\)](#) like this:

```
hex(randomblob(16))  
  
lower(hex(randomblob(16)))
```

replace(X,Y,Z)

The `replace(X,Y,Z)` function returns a string formed by substituting string Z for every occurrence of string Y in string X. The [BINARY](#) collating sequence is used for comparisons. If Y is an empty string then return X unchanged. If Z is not initially a string, it is cast to a UTF-8 string prior to processing.

round(X) **round(X,Y)**

The `round(X,Y)` function returns a floating-point value X rounded to Y digits to the right of the decimal point. If the Y argument is omitted, it is assumed to be 0.

rtrim(X) **rtrim(X,Y)**

The `rtrim(X,Y)` function returns a string formed by removing any and all characters that appear in Y from the right side of X. If the Y argument is omitted, `rtrim(X)` removes spaces from the right side of X.

soundex(X)

The `soundex(X)` function returns a string that is the soundex encoding of the string X. The string "?000" is returned if the argument is `NULL` or contains no ASCII alphabetic characters. This function is omitted from SQLite by default. It is only available if the [SQLITE_SOUNDEX](#) compile-time option is used when SQLite is built.

sqlite_compileoption_get(N)

The `sqlite_compileoption_get()` SQL function is a wrapper around the [sqlite3_compileoption_get\(\)](#) C/C++ function. This routine returns the N-th compile-time option used to build SQLite or `NULL` if N is out of range. See also the [compile_options pragma](#).

sqlite_compileoption_used(X)

The `sqlite_compileoption_used()` SQL function is a wrapper around the [sqlite3_compileoption_used\(\)](#) C/C++ function. When the argument `X` to `sqlite_compileoption_used(X)` is a string which is the name of a compile-time option, this routine returns true (1) or false (0) depending on whether or not that option was used during the build.

sqlite_offset(X)

The `sqlite_offset(X)` function returns the byte offset in the database file for the beginning of the record from which value would be read. If `X` is not a column in an ordinary table, then `sqlite_offset(X)` returns NULL. The value returned by `sqlite_offset(X)` might reference either the original table or an index, depending on the query. If the value `X` would normally be extracted from an index, the `sqlite_offset(X)` returns the offset to the corresponding index record. If the value `X` would be extracted from the original table, then `sqlite_offset(X)` returns the offset to the table record.

The `sqlite_offset(X)` SQL function is only available if SQLite is built using the `-DSQLITE_ENABLE_OFFSET_SQL_FUNC` compile-time option.

sqlite_source_id()

The `sqlite_source_id()` function returns a string that identifies the specific version of the source code that was used to build the SQLite library. The string returned by `sqlite_source_id()` is the date and time that the source code was checked in followed by the SHA1 hash for that check-in. This function is an SQL wrapper around the [sqlite3_sourceid\(\)](#) C interface.

sqlite_version()

The `sqlite_version()` function returns the version string for the SQLite library that is running. This function is an SQL wrapper around the [sqlite3_libversion\(\)](#) C-interface.

substr(X,Y,Z) **substr(X,Y)**

The `substr(X,Y,Z)` function returns a substring of input string `X` that begins with the `Y`-th character and which is `Z` characters long. If `Z` is omitted then `substr(X,Y)` returns all characters through the end of the string `X` beginning with the `Y`-th. The left-most character of `X` is number 1. If `Y` is negative then the first character of the substring is found by counting from the right rather than the left. If `Z` is negative then the `abs(Z)` characters preceding the `Y`-th character are returned. If `X` is a string then characters indices refer to actual UTF-8 characters. If `X` is a BLOB then the indices refer to bytes.

total_changes()

The `total_changes()` function returns the number of row changes caused by INSERT, UPDATE or DELETE statements since the current database connection was opened. This function is a wrapper around the [sqlite3_total_changes\(\)](#) C/C++ interface.

trim(X) **trim(X,Y)**

The `trim(X,Y)` function returns a string formed by removing any and all characters that appear in `Y` from both ends of `X`. If the `Y` argument is omitted, `trim(X)` removes spaces from both ends of `X`.

typeof(X)

The `typeof(X)` function returns a string that indicates the [datatype](#) of the expression `X`: "null", "integer", "real", "text", or "blob".

unicode(X)

The `unicode(X)` function returns the numeric unicode code point corresponding to the first character of the string `X`. If the argument to `unicode(X)` is not a string then the result is undefined.

unlikely(X)

The `unlikely(X)` function returns the argument `X` unchanged. The `unlikely(X)` function is a no-op that the code generator optimizes away so that it consumes no CPU cycles at run-time (that is, during calls to [sqlite3_step\(\)](#)). The purpose of the `unlikely(X)` function is to provide a hint to the query planner that the

argument *X* is a boolean value that is usually not true. The `unlikely(X)` function is equivalent to [likelihood\(*X*, 0.0625\)](#).

upper(*X*)

The `upper(X)` function returns a copy of input string *X* in which all lower-case ASCII characters are converted to their upper-case equivalent.

zeroblob(*N*)

The `zeroblob(N)` function returns a BLOB consisting of *N* bytes of 0x00. SQLite manages these zeroblobs very efficiently. Zeroblobs can be used to reserve space for a BLOB that is later written using [incremental BLOB I/O](#). This SQL function is implemented using the [sqlite3_result_zeroblob\(\)](#) routine from the C/C++ interface.