Date And Time Functions

1. Overview

SQLite supports seven scalar date and time functions as follows:

1. **date(***time-value, modifier, modifier, ...***)**
2. **time(***time-value, modifier, modifier, ...***)**
3. **datetime(***time-value, modifier, modifier, ...***)**
4. **julianday(***time-value, modifier, modifier, ...***)**
5. **unixepoch(***time-value, modifier, modifier, ...***)**
6. **strftime(***format, time-value, modifier, modifier, ...***)**
7. **timediff(***time-value, time-value***)**

The first six date and time functions take an optional time-value as an argument, followed by zero or more modifiers. The strftime() function also takes a format string as its first argument. The timediff() function takes exactly two arguments which are both time-values.

SQLite does not have a dedicated date/time datatype. Instead, date and time values can stored as any of the following:

ISO-8601	A text string that is one of the ISO 8601 date/time values shown in items 1 through 10 below. Example: `'2025-05-29 14:16:00'`
Julian day number	The number of days including fractional days since -4713-11-24 12:00:00 Example: `2460825.09444444`
Unix timestamp	The number of seconds including fractional seconds since 1970-01-01 00:00:00 Example: `1748528160`

These three formats are collectively known as time-values. All of the date time functions accept time-values as either ISO-8601 text or as Julian day numbers. They can also be made to accept unix timestamps by adding optional modifiers arguments 'auto' or 'unixepoch'. Since the timediff() function does not accept modifiers, it can only use ISO-8601 and julian day number time-values.

The **date()** function returns the date as text in this format: YYYY-MM-DD.

The **time()** function returns the time as text in formatted as HH:MM:SS or as HH:MM:SS.SSS if the subsec modifier is used.

The **datetime()** function returns the date and time formatted as YYYY-MM-DD HH:MM:SS or as YYYY-MM-DD HH:MM:SS.SSS if the subsec modifier is used.

The **julianday()** function returns the Julian day - the fractional number of days since noon in Greenwich on November 24, 4714 B.C. (Proleptic Gregorian calendar).

The **unixepoch()** function returns a unix timestamp - the number of seconds since 1970-01-01 00:00:00 UTC. The unixepoch() function normally returns an integer number of seconds, but with the optional subsec modifier it will return a floating point number which is the fractional number of seconds.

The **strftime()** function returns the date formatted according to the format string specified as the first argument. The format string supports the most common substitutions found in the strftime() function from the standard C library plus two new substitutions, %f and %J. The following is a complete list of valid strftime() substitutions as of version 3.46.0 (2024-05-23). Earlier versions of SQLite might not support all substitutions. If an undefined or unsupported substitution is seen, the result is NULL.

%d	day of month: 01-31
%e	day of month without leading zero: 1-31
%f	fractional seconds: SS.SSS
%F	ISO 8601 date: YYYY-MM-DD
%G	ISO 8601 year corresponding to %V
%g	2-digit ISO 8601 year corresponding to %V
%H	hour: 00-24
%I	hour for 12-hour clock: 01-12
%j	day of year: 001-366
%J	Julian day number (fractional)
%k	hour without leading zero: 0-24
%l	%I without leading zero: 1-12
%m	month: 01-12
%M	minute: 00-59
%p	"AM" or "PM" depending on the hour
%P	"am" or "pm" depending on the hour
%R	ISO 8601 time: HH:MM
%s	seconds since 1970-01-01
%S	seconds: 00-59
%T	ISO 8601 time: HH:MM:SS
%U	week of year (00-53) - week 01 starts on the first Sunday

%u	day of week 1-7 with Monday==1
%V	ISO 8601 week of year
%w	day of week 0-6 with Sunday==0
%W	week of year (00-53) - week 01 starts on the first Monday
%Y	year: 0000-9999
%%	%

Other date and time functions can be expressed in terms of strftime():

Function	**Equivalent strftime()**
date(...)	strftime('%F', ...)
time(...)	strftime('%T', ...)
datetime(...)	strftime('%F %T', ...)
julianday(...)	CAST(strftime('%J', ...) as REAL)
unixepoch(...)	CAST(strftime('%s', ...) as INT)

The date(), time(), and datetime() functions all return text, and so their strftime() equivalents are exact. However, the julianday() and unixepoch() functions return numeric values. Their strftime() equivalents return a string that is the text representation of the corresponding number.

The main reasons for providing functions other than strftime() are for convenience and for efficiency. The julianday() and unixepoch() functions return real and integer values respectively, and do not incur the format conversion costs or inexactitude resulting from use of the '%J' or '%s' format specifiers with the strftime() function.

The **timediff(A,B)** function returns a string that describes the amount of time that must be added to B in order to reach time A. The format of the timediff() result is designed to be human-readable. The format is:

```
(+|-)YYYY-MM-DD HH:MM:SS.SSS
```

This time difference string is also an allowed modifier for the other date/time functions. The following invariant holds for time-values A and B:

```
datetime(A) = datetime(B, timediff(A,B))
```

The length of months and years vary. February is shorter than March. Leap years are longer than non-leap years. The output from timediff() takes this all into account. The timediff() function is intended to provide a human-friendly description of the time span. If you want to know the number of days or seconds between two dates, A and B, then you can always do one of these:

```
SELECT julianday(B) - julianday(A);
SELECT unixepoch(B) - unixepoch(A);
```

The timediff(A,B) might return the same result even for values A and B that span a different number of days - depending on the starting date. For example, both of the

following two timediff() calls return the same result ("-0000-01-00 00:00:00.000") even though the first timespan is 28 days and the seconds is 31 days:

```
SELECT timediff('2023-02-15','2023-03-15');
SELECT timediff('2023-03-15','2023-04-15');
```

Summary: If you want a human-friendly time span, use timediff(). If you want a precise time difference (in days or seconds) use the difference between two julianday() or unixepoch() calls.

2. Time Values

A time-value can be in any of the following formats shown below. The value is usually a string, though it can be an integer or floating point number in the case of format 12.

1. *YYYY-MM-DD*
2. *YYYY-MM-DD HH:MM*
3. *YYYY-MM-DD HH:MM:SS*
4. *YYYY-MM-DD HH:MM:SS.SSS*
5. *YYYY-MM-DD***T***HH:MM*
6. *YYYY-MM-DD***T***HH:MM:SS*
7. *YYYY-MM-DD***T***HH:MM:SS.SSS*
8. *HH:MM*
9. *HH:MM:SS*
10. *HH:MM:SS.SSS*
11. **now**
12. *DDDDDDDDDD*

In formats 5 through 7, the "T" is a literal character separating the date and the time, as required by ISO-8601. Formats 8 through 10 that specify only a time assume a date of 2000-01-01. Format 11, the string 'now', is converted into the current date and time as obtained from the xCurrentTime method of the sqlite3_vfs object in use. The 'now' argument to date and time functions always returns exactly the same value for multiple invocations within the same sqlite3_step() call. Universal Coordinated Time (UTC) is used. Format 12 is the Julian day number expressed as an integer or floating point value. Format 12 might also be interpreted as a unix timestamp if it is immediately followed either the 'auto' or 'unixepoch' modifier.

Formats 2 through 10 may be optionally followed by a timezone indicator of the form "*[+-]HH:MM*" or just "*Z*". The date and time functions use UTC or "zulu" time internally, and so the "Z" suffix is a no-op. Any non-zero "HH:MM" suffix is subtracted from the indicated date and time in order to compute zulu time. For example, all of the following time-values are equivalent:

```
2013-10-07 08:23:19.120
2013-10-07T08:23:19.120Z
2013-10-07 04:23:19.120-04:00
2456572.84952685
```

In formats 4, 7, and 10, the fractional seconds value SS.SSS can have one or more digits following the decimal point. Exactly three digits are shown in the examples because only

the first three digits are significant to the result, but the input string can have fewer or more than three digits and the date/time functions will still operate correctly. Similarly, format 12 is shown with 10 significant digits, but the date/time functions will really accept as many or as few digits as are necessary to represent the Julian day number.

ISO-8601 supports a wide variety of alternative date/time formats, but SQLite only supports the ones specifically enumerated above.

In all functions other than timediff(), the time-value (and all modifiers) may be omitted, in which case a time value of 'now' is assumed.

3. Modifiers

For all date/time functions other than timediff(), the time-value argument can be followed by zero or more modifiers that alter date and/or time. Each modifier is a transformation that is applied to the time-value to its left. Modifiers are applied from left to right; order is important. The available modifiers are as follows.

1. NNN days
2. NNN hours
3. NNN minutes
4. NNN seconds
5. NNN months
6. NNN years
7. ±HH:MM
8. ±HH:MM:SS
9. ±HH:MM:SS.SSS
10. ±YYYY-MM-DD
11. ±YYYY-MM-DD HH:MM
12. ±YYYY-MM-DD HH:MM:SS
13. ±YYYY-MM-DD HH:MM:SS.SSS
14. ceiling
15. floor
16. start of month
17. start of year
18. start of day
19. weekday N
20. unixepoch
21. julianday
22. auto
23. localtime
24. utc
25. subsec
26. subsecond

The first thirteen modifiers (1 through 13) add the specified amount of time to the date and time specified by the arguments to its left. The 's' character at the end of the modifier names in 1 through 6 is optional. The NNN value can be any floating point number, with an optional '+' or '-' prefix.

The **time shift modifiers** (7 through 13) move the time-value by the number of years, months, days, hours, minutes, and/or seconds specified. An initial "+" or "-" is required for formats 10 through 13 but is optional for formats 7, 8, and 9. The changes are applies from left to right. First the year is shifted by YYYY, then the month by MM, and then day by DD, and so forth. The timediff(A,B) function returns a time shift in format 13 that shifts the time-value B into A.

Because the length of a month or year changes from one month or year to the next, ambiguities can arise when shifting a date by months and/or years. For example, what is the date one year after 2024-02-29? Is it 2025-02-28 or 2025-03-01? Or what is the date that is two months after 2023-12-31? Is it 2024-02-29 or 2024-03-02? There is no consensus on how to resolve this ambiguity, so the "**ceiling**" and "**floor**" modifiers (14 and 15) are available to let the programmer decide. If the next modifier after a time shift is "ceiling", then any ambiguity in the date is resolved by choosing the later date. The "floor" modifier resolves ambiguities by resolving to the last day of the previous month. The default behavior is "ceiling".

The "**start of**" modifiers (16 through 18) shift the date backwards to the beginning of the subject month, year or day.

The "**weekday**" modifier advances the date forward, if necessary, to the next date where the weekday number is N. Sunday is 0, Monday is 1, and so forth. If the date is already on the desired weekday, the "weekday" modifier leaves the date unchanged.

The "**unixepoch**" modifier (20) only works if it immediately follows a time-value in the DDDDDDDDDD format. This modifier causes the DDDDDDDDDD to be interpreted not as a Julian day number as it normally would be, but as Unix Time - the number of seconds since 1970. If the "unixepoch" modifier does not follow a time-value of the form DDDDDDDDDD which expresses the number of seconds since 1970 or if other modifiers separate the "unixepoch" modifier from prior DDDDDDDDDD then the behavior is undefined.

The "**julianday**" modifier must immediately follow the initial time-value which must be of the form DDDDDDDDD. Any other use of the 'julianday' modifier is an error and causes the function to return NULL. The 'julianday' modifier forces the time-value number to be interpreted as a julian-day number. As this is the default behavior, the 'julianday' modifier is scarcely more than a no-op. The only difference is that adding 'julianday' forces the DDDDDDDDD time-value format, and causes a NULL to be returned if any other time-value format is used.

The "**auto**" modifier must immediately follow the initial time-value. If the time-value is numeric (the DDDDDDDDDD format) then the 'auto' modifier causes the time-value to interpreted as either a julian day number or a unix timestamp, depending on its magnitude. If the value is between 0.0 and 5373484.499999, then it is interpreted as a julian day number (corresponding to dates between -4713-11-24 12:00:00 and 9999-12-31 23:59:59, inclusive). For numeric values outside of the range of valid julian day numbers, but within the range of -210866760000 to 253402300799, the 'auto' modifier causes the value to be interpreted as a unix timestamp. Other numeric values are out of range and cause a NULL return. The 'auto' modifier is a no-op for ISO 8601 text time-values. The "auto" modifier is designed to work with time-values even in cases where it is not known which time-value format is stored in the database file, or in cases where the same column stores time-values in different formats on different rows. The 'auto'

modifier will automatically select the appropriate format. However, there is some ambiguity. Unix timestamps for the first 63 days of 1970 will be interpreted as julian day numbers. The 'auto' modifier is very useful when the dataset is guaranteed to contain no dates within that range, but should be avoided for applications that might make use of dates in the opening months of 1970.

The "**localtime**" modifier assumes the time-value to its left is in Universal Coordinated Time (UTC) and adjusts that time value so that it is in localtime. If "localtime" follows a time that is not UTC, then the behavior is undefined. The "**utc**" modifier is the opposite of "localtime". "utc" assumes that the time-value to its left is in the local timezone and adjusts that time-value to be in UTC. If the time to the left is not in localtime, then the result of "utc" is undefined.

The "**subsecond**" modifier (which may be abbreviated as just "**subsec**") increases the resolution of the output for datetime(), time(), and unixepoch(), and for the "%s" format string in strftime(). The "subsecond" modifier has no effect on other date/time functions. The current implemention increases the resolution from seconds to milliseconds, but this might increase to a higher resolution in future releases of SQLite. When "subsec" is used with datetime() or time(), the seconds field at the end is followed by a decimal point and one or more digits to show fractional seconds. When "subsec" is used with unixepoch(), the result is a floating point value which is the number of seconds and fractional seconds since 1970-01-01. The "subsecond" and "subsec" modifiers have the special property that they can occur as the first argument to date/time functions (or as the first argument after the format string for strftime()). When this happens, the time-value that is normally in the first argument is understood to be "now". For example, a short cut to get the current time in seconds since 1970 with millisecond precision is to say:

```
SELECT unixepoch('subsec');
```

4. Examples

Compute the current date.

```
SELECT date();
```

Compute the last day of the current month.

```
SELECT date('now','start of month','+1 month','-1 day');
```

Compute the date and time given a unix timestamp 1092941466.

```
SELECT datetime(1092941466, 'unixepoch');
SELECT datetime(1092941466, 'auto'); -- Does not work for early 1970!
```

Compute the date and time given a unix timestamp 1092941466, and compensate for your local timezone.

```
SELECT datetime(1092941466, 'unixepoch', 'localtime');
```

Compute the current unix timestamp.

```
SELECT unixepoch();
SELECT strftime('%s');
```

Compute the number of days since the signing of the US Declaration of Independence.

```
SELECT julianday('now') - julianday('1776-07-04');
```

Compute the number of seconds since a particular moment in 2004:

```
SELECT unixepoch() - unixepoch('2004-01-01 02:34:56');
```

Compute the date of the first Tuesday in October for the current year.

```
SELECT date('now','start of year','+9 months','weekday 2');
```

Compute the time since the unix epoch in seconds with millisecond precision:

```
SELECT (julianday('now') - 2440587.5)*86400.0;
SELECT unixepoch('now','subsec');
```

Compute how old Abraham Lincoln would be if he were still alive today:

```
SELECT timediff('now','1809-02-12');
```

5. Caveats And Bugs

The computation of local time depends heavily on the whim of politicians and is thus difficult to get correct for all locales. In this implementation, the standard C library function localtime_r() is used to assist in the calculation of local time. The localtime_r() C function normally only works for years between 1970 and 2037. For dates outside this range, SQLite attempts to map the year into an equivalent year within this range, do the calculation, then map the year back.

These functions only work for dates between 0000-01-01 00:00:00 and 9999-12-31 23:59:59 (julian day numbers 1721059.5 through 5373484.5). For dates outside that range, the results of these functions are undefined.

Non-Vista Windows platforms only support one set of DST rules. Vista only supports two. Therefore, on these platforms, historical DST calculations will be incorrect. For example, in the US, in 2007 the DST rules changed. Non-Vista Windows platforms apply the new 2007 DST rules to all previous years as well. Vista does somewhat better getting results correct back to 1986, when the rules were also changed.

All internal computations assume the Gregorian calendar system. They also assume that every day is exactly 86400 seconds in duration; no leap seconds are incorporated.

This page last modified on 2024-10-24 10:55:33 UTC