SQL As Understood By SQLite

[Top]

Date And Time Functions

SQLite supports five date and time functions as follows:

1. **date(***timestring, modifier, modifier, ...***)**
2. **time(***timestring, modifier, modifier, ...***)**
3. **datetime(***timestring, modifier, modifier, ...***)**
4. **julianday(***timestring, modifier, modifier, ...***)**
5. **strftime(***format, timestring, modifier, modifier, ...***)**

All five date and time functions take a time string as an argument. The time string is followed by zero or more modifiers. The strftime() function also takes a format string as its first argument.

The date and time functions use a subset of IS0-8601 date and time formats. The date() function returns the date in this format: YYYY-MM-DD. The time() function returns the time as HH:MM:SS. The datetime() function returns "YYYY-MM-DD HH:MM:SS". The julianday() function returns the Julian day - the number of days since noon in Greenwich on November 24, 4714 B.C. (Proleptic Gregorian calendar). The strftime() routine 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:

%d	day of month: 00
%f	fractional seconds: SS.SSS
%H	hour: 00-24
%j	day of year: 001-366
%J	Julian day number
%m	month: 01-12
%M	minute: 00-59
%s	seconds since 1970-01-01
%S	seconds: 00-59
%w	day of week 0-6 with Sunday==0
%W	week of year: 00-53
%Y	year: 0000-9999

%% %

Notice that all other date and time functions can be expressed in terms of strftime():

Function	Equivalent strftime()
date(...)	strftime('%Y-%m-%d', ...)
time(...)	strftime('%H:%M:%S', ...)
datetime(...)	strftime('%Y-%m-%d %H:%M:%S', ...)
julianday(...)	strftime('%J', ...)

The only reasons for providing functions other than strftime() is for convenience and for efficiency.

Time Strings

A time string can be in any of the following formats:

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 a floating point value.

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 strings 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.

Modifiers

The time string 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.NNNN seconds
5. NNN months
6. NNN years
7. start of month
8. start of year
9. start of day
10. weekday N
11. unixepoch
12. localtime
13. utc

The first six modifiers (1 through 6) simply add the specified amount of time to the date and time specified by the preceding timestring and modifiers. The 's' character at the end of the modifier names is optional. Note that "±NNN months" works by rendering the original date into the YYYY-MM-DD format, adding the ±NNN to the MM month value, then normalizing the result. Thus, for example, the data 2001-03-31 modified by '+1 month' initially yields 2001-04-31, but April only has 30 days so the date is normalized to 2001-05-01. A similar effect occurs when the original date is February 29 of a leapyear and the modifier is ±N years where N is not a multiple of four.

The "start of" modifiers (7 through 9) shift the date backwards to the beginning of the current month, year or day.

The "weekday" modifier advances the date forward to the next date where the weekday number is N. Sunday is 0, Monday is 1, and so forth.

The "unixepoch" modifier (11) only works if it immediately follows a timestring 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 timestring 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. Due to precision limitations imposed by the implementations use of 64-bit integers, the "unixepoch" modifier only works for dates between 0000-01-01 00:00:00 and 5352-11-01 10:52:47 (unix times of -62167219200 through 106751991167).

The "localtime" modifier (12) assumes the time string to its left is in Universal Coordinated Time (UTC) and adjusts the time string so that it displays 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 string to its left is in the local timezone and adjusts that string to be in UTC. If the prior string is not in localtime, then the result of "utc" is undefined.

Examples

Compute the current date.

```
SELECT date('now');
```

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');
```

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 strftime('%s','now');
```

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 strftime('%s','now') - strftime('%s','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 (like strftime('%s','now') except includes fractional part):

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

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 (julidan 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. It is also assumed that every day is exactly 86400 seconds in duration.