Different TIMESTAMP types

Overview

The following overview depicts the desired timestamp semantics in comparison to the SQL standard and selected database vendors:

blocked URL

TIMESTAMP and TIMESTAMP WITHOUT TIME ZONE

The TIMESTAMP and TIMESTAMP WITHOUT TIME ZONE types shall behave like the LocalDateTime class of Java, i.e., each value is a recording of what can be seen on a calendar and a clock hanging on the wall, for example "1969-07-20 16:17:39". It can be decomposed into year, month, day, hour, minute and seconds fields, but with no time zone information available, it does not correspond to any specific point in time.

This behaviour is consistent with the SQL standard (revisions 2003 and higher).

TIMESTAMP WITH LOCAL TIME ZONE

The TIMESTAMP WITH LOCAL TIME ZONE type shall behave like the Instant class of Java, i.e., each value identifies a single time instant, but does not contain any explicit timezone information. To achieve this semantics, the processing of timestamp literals involves an implicit normalization from the session-local time zone to a predefined one (typically but not necessarily UTC), while displaying the stored timestamps involves an implicit conversion from the predefined time zone to the session-local one. Although a predefined time zone (typically UTC) was explicitly mentioned above, it is not a part of the individual values but of the definition of the whole type instead. As such, every value that gets stored has to be normalized to this predefined time zone and therefore its original time zone information gets lost.

For example, if the calendar and clock hanging on the wall shows 1969-07-20 16:17:39 according to Eastern Daylight Time, that must be stored as "1969-07-20 20:17:39", because that UTC time corresponds to the same instant. When reading that value back, we no longer know that it originated from an EDT time and we can only display it in some fixed time zone (either local or UTC or specified by the user).

This behaviour is consistent with some major DB engines, which is the best we can do as no type is defined by the SQL standard that would have this behaviour.

TIMESTAMP WITH TIME ZONE

The TIMESTAMP WITH TIME ZONE type shall behave like the OffsetDateTime class of Java, i.e., each individual value includes a time zone offset as well. This definition leads to timestamps that not only identify specific time instants unambiguously, but also allow retrieval of the originating time zone offset.

If we stored the same timestamp as above using this semantics, then the original timestamp literal could be reconstructed including some time zone information, for example "1969-07-20 16:17:39 (UTC -04:00)". (The primary fields are typically still stored normalized to UTC to make comparison of timestamps more efficient, but this is an implementation detail and does not affect the behaviour of the type.)

This behaviour is consistent with the SQL standard (revisions 2003 and higher).

Comparison

Let's summarize the example of how the different semantics described above apply to a value inserted into a SQL table.

If the timestamp literal '1969-07-20 16:17:39' is inserted in Washington D.C. and then queried from Paris, it might be shown in the following ways based on timestamp semantics:

SQL type	Semantics	Result	Explanation
TIMESTAMP [WITHOUT TIME ZONE]	LocalDateTime	1969-07-20 16:17:39	Displayed like the original timestamp literal.
TIMESTAMP WITH LOCAL TIME ZONE	Instant	1969-07-20 21:17:39	Differs from the original timestamp literal, but refers to the same time instant.
TIMESTAMP WITH TIME ZONE	OffsetDateTime	1969-07-20 16:17:39 (UTC -04:00)	Displayed like the original literal but showing the time zone offset as well.

Of course, the different semantics do not only affect the textual representations but perhaps more importantly SQL function behavior as well. These allow users to take advantage of timestamps in different ways or to explicitly create different textual representations instead of the implicit ones shown above.

In fact, even the implicit textual representations could be different than shown above, for example Instant could be displayed normalized to UTC or OffsetDateTime could be adjusted to the local time zone by default. The examples shown above are just common ways of creating an implicit textual representation for the different semantics, but the truly important difference lies in what details can and what details can not be reconstructed from the different semantics:

		Reconstructible details		
SQL type	Semantics	Local clock reading	Time instant	Time zone offset
TIMESTAMP [WITHOUT TIME ZONE]	LocalDateTime			

TIMESTAMP WITH LOCAL TIME ZONE	Instant			
TIMESTAMP WITH TIME ZONE	OffsetDateTime			