Time with Time Zone

FB4 introduces the TIME WITH TIME ZONE data type.

The TIME data type is a longstanding Firebird data type that records a time in the range 0:00..23:59 and to a precision of 10^{-4} seconds. The TIME WITH TIME ZONE data type extends this to include a time zone identifier as part of the value saved in the database.

A TIME WITH TIME ZONE data type is always saved in the database as a GMT timestamp plus a time zone id, regardless of whether it was originally input as GMT or as a local time in the context of some time zone.

A TIME data type may be used for many purpose:

- It can be an elapsed time (< 24 hours) as recorded on a stop watch, or a time code on a video stream.
- It can be a time of day with reference to a date stored elsewhere. For example, a master record may include a date and a detailed record may hold a time on that day on which some event occurs alongside information about the event itself.
- It can be a scheduled time on some day in the future. This can be any day or a day in a week. For example, shop opening and closing hours may be recorded as a pair of times on a specified day of the week.
- It could even be a *sidereal time*. This is defined by wikipedia as a "time scale that is based on Earth's rate of rotation measured relative to the fixed stars". Here a TIME data type could record the sidereal time at which an observation is to be made. Given that a sidereal day is about four minutes shorter than a solar day a separate computation system is needed to convert a scheduled sidereal time to a calendar time.

A TIME (without time zone) may be used for any of the above. In the first and last cases, the concept of local time does not apply and hence a TIME WITH TIME ZONE is not of interest. In the second and third cases, prior to FB4, the database designer will have had to have specified whether the time is implicitly recorded in UTC or some local time zone. FB4 allows the time zone to be explicitly added to the value by using a TIME WITH TIME ZONE data type.

Inputting a TIME WITH TIME ZONE Data Type

The time zone can be given each time a time is entered or set as a session default. Either way, the following applies.

Let's say that the time is to be set as 7 am Eastern (New York time). This could be expressed in several ways, each resulting in a different database entry when translated to GMT + time zone id. This is illustrated in the following table.

Original Value	Time (GMT)	Time Zone ID
7:00 EST	12:00	65136 (EST)
7:00 EST5EDT	11:00 or 12:00	65135 (EST5EDT)

7:00 –05:00	12:00	1139 (-05:00)
7:00 America/New York	11:00 or 12:00	America/New_York (65361)

The second and fourth cases are somewhat problematic. The time is translated to either 11:00 GMT or 12:00 GMT depending on whether or not daylight savings time applies. For this you either need to assume that the daylight savings time is ignored or apply an assumed date for the translation.

Firebird uses the current date when determining whether or not to apply daylight savings time. Hence, 7:00 America/New York will translate to 11:00 GMT during the summer months and to 12:00 GMT during the winter months.

The translation to GMT may be an issue that has to be investigated further when looking at how a TIME WITH TIME ZONE data type is used. The date on which a translation to and from GMT takes place is an issue that the database designer is going to have to think about carefully.

Reading a TIME WITH TIME ZONE Data Type from a Database

FB4 provides two TIME WITH TIME ZONE formats for the over the wire protocol:

- In the standard format (which was the only format available in FB4 beta 1), an encoded GMT time and time zone id is provided.
- In the extended format (available only in the FB4 development version so far), an encoded GMT time, offset from GMT and time zone id is provided.

The rationale for the extended format was to allow for situations where the time zone database was not available to the client and hence all time zone database computations have to be performed by the server.

As with the TIMESTAMP WITH TIME ZONE data type, when rendering a time as a string, IBX provides the option of either always rendering the time zone as an offset or rendering it using the time zone name or offset as input by the user. It thus needs to obtain on a data read, the time (local time or GMT), the offset to GMT and the time zone name.

The low level Firebird API provides two API calls that can be used to decode the data received from the server.

- The IUtil.decodeTimeTzEx API call is used to decode the extended format and returns the local time plus the time zone name as input (if a time zone offset was input then the offset is returned as a text string). IBX uses this to get the local time and time zone name. It also extracts the offset and time zone id from the received data.
- IUtil.decodeTimeTz is used to decode the standard format. This also returns the local time plus the time zone name as input. In order to determine the offset, IBX uses the (TIME WITHOUT TIME ZONE) API call IUtil.decodeTime to get the timestamp in GMT. The offset can then be derived. This exploits the fact that the time zone is an extension to the original timestamp over the wire encoding and that the time part returned “over the wire” is in GMT. The time zone id may also be extracted from the received data.

However, the date used to translation the GMT time in the database to local time is the current date; the original input date has been lost. If, for example, the time was entered as 07:00 in the America/New_York time zone during the summer months, it will be recorded in the database as 11:00 GMT. If it is read back during the winter months then the value returned with be

06:00 America/New_York

This may or may not be appropriate depending on the intended use.

TIME WITH TIME ZONE use cases

Case #1: Local Time in a known Date Context

This use case is built around a pair of tables that are in a master/detail relationship. The master table contains the date on which some observation is made and the detail table records record the event itself and the time of the event.

For example, a master table entry may identify a daily log and records the date of the log plus other summary information. A detail table entry references the master table entry for the day the log entry was made, records the observation itself and the time of the observation. If the time is recorded in a TIME WITH TIME ZONE data type then the time can be recorded in local time plus the time zone. The observation may be a weather observation, such a temperature and pressure.

In this use case example, the tables are in a database of worldwide observations and here it is important to be able to relate observations made in different time zones.

It may be observed that a serious problem would result if the time zone name format (e.g. America/New_York) was used for data entry, or its alias EST5EDT. This is because the time will be translated to GMT using the date on which the date was entered into the database and this is not necessarily the same date as the date in the master record. Depending on the dates when daylight savings time applies, this could result in an incorrect translation to GMT.

A workaround could be to prohibit the use of time zone names in data entry and hence to force the use of the current offset. This would ensure a correct translation to GMT, albeit in a way that requires the person doing the data entry to correctly identify the offset and which removes the time zone identity from the log entry.

Alternatively, the time zone could be entered as a character string in the master record (or more efficiently as a reference to the RDB$TIME_ZONE_ID in the RDB$TIME_ZONES virtual table), and the time recorded in GMT using a TIME (without time zone) data type. As all the log entries that reference the same master record are in the same time zone this would be an appropriate and efficient way to record the time zone information.

In order to efficient compare times entered in log files from different time zones, it is still desirable to translate to or save the time in GMT.

The time would thus either have to be explicitly entered in GMT or a trigger defined to translate from local time to GMT using the master record time zone and date. FB4 provides built-in store procedure “RDB$TIME_ZONE_UTIL.TRANSITIONS” that would support such a trigger.

The same built-in stored procedure could also be used to support a function that converts a time in GMT to a local time on a given date and for a specific time zone. This could be used to support (e.g.) a VIEW that returned log file entries in local time. For example:

CREATE FUNCTION GMT_TO_LOCALTIME(gmt_time TIME, timezone_name CHAR(63), atDate DATE)
RETURNS TIME;

Case #2: Scheduled Local Times

In this use case, a database table is used to record an event time schedule e.g. by day of the week, or just *any day*. A specific example could be a table of shop opening and closing times with each record providing the opening and closing times (in local time) on a specific day of the week and for a given shop.

Here, the time is a “Wall clock time”. That is the opening/closing time is the time on a local “wall clock” irrespective of whether daylight savings time applies on any given day. The whole point of daylight savings time is to advance the time shown on a wall clock so that scheduled events are shifted to an earlier solar time without having to change the local time of each such event.

The value in using a TIME WITH TIME ZONE data type here should be that the table may be a list of many shops located in different time zones in the same or different countries. By including the time zone it should be possible to determine whether a given shop is open “now” for a user in a different time zone, or to perform a query returning all shops in the database that are open at a given date and time, regardless of which time zone they are in.

Here, again, the data entry problem rears its ugly head. Translating the opening and closing time to GMT on the date that the database entry was made is problematic.

- If the opening/closing time is entered with a fixed offset to GMT or using a time zone name (e.g. EST) that does not include any reference to daylight savings time, then the time will be converted to GMT using the explicit or implicit fixed offset. E.g. an opening time of 7:00 -0500 becomes 12:00 GMT in the database itself. In New York, in the winter months, this may well be the wall clock time that the shop opens. However, in the summer months, presenting the opening time as 07:00 EST (or -05:00) is wrong as this would be converted to a wall clock time of 8 am by anyone who reads it properly.

- If the opening/closing time is entered with a time zone name (e.g. America/New_York) in the winter months, then it will be translated to 12:00 GMT in the database. If it is then read back in the summer months, 12:00 GMT becomes 08:00 America/New_York when applying daylight savings time. Again, the wrong answer as the wall clock time should be 07:00.

Again, the problem is due to translating to GMT. In use case #1 this was because the date used for the translation is not necessarily the same as the log file date. In this use case, any translation to GMT should only be done when the time is read from the database or compared with other database entries. This is what “wall clock time” implies.

The workaround to avoid this problem is again to separate out the time and time zone into separate database columns. The time zone is a property of the shop itself and so only needs to be recorded once per shop. The opening/closing times may be recorded using the TIME (without time zone) data type.

On data entry, there is no requirement for any translation to GMT and the time is entered into the database as a wall clock time. There is also no need to translate the opening/closing time on read back. The correct answer is always returned.

The only need to translate the opening/closing time to GMT occurs when comparing shop opening times. For example, to answer a question such as which shops are open on a given date and time. The translation to GMT is performed at the given date and time and hence the desired result returned.

Here, a function such as

CREATE FUNCTION LOCALTIME_TO_GMT(localtime TIME, timezone_name CHAR(63), atDate DATE)
RETURNS TIME;

is needed to allow (e.g.) shop opening/closing times to be compared.

Discussion

Neither of the above use cases appears to be a case where a TIME WITH TIME ZONE can be usefully used, even though it was expected that they would be. In each case, the problem is due to the translation to GMT.

- In use case #1, the problem is that the data entry date is used as the baseline for the translation rather than the log file date.
- In use case #2, any translation to GMT is inappropriate until there is a need to compare times.

Use case #2 agues for not translating a TIME WITH TIME ZONE to GMT when it is entered into the database. However, that would not be right for use case #1. Under use case #1, you do want to translate to GMT because the same date will always be used for the translation and doing it at data input is more efficient than having to perform the translation dynamically every time log file entries are compared.

Also, storing the time as a local time could create some interesting issues for indexes and table joins, and is probably best avoided.

On the other hand, use case #1 would benefit from being able to specify the date used for translation to GMT in context. For example, by defining an SQL statement such as:

SET TIME ZONE DATE TO <date>;

By default the date used to translate a TIME WITH TIME ZONE to GMT uses the CURRENT_DATE. If the above statement is executed then this changes the date used to that given on the statement for the current session and until the session ends or another SET TIME ZONE DATE statement is executed.

This would solve the problem for use case #1 and allow it to use the TIME WITH TIME ZONE data type.

For use case #2, the proposed workaround is probably the correct solution as long as the function LOCALTIME_TO_GMT or similar is available to support comparing (e.g.) shop opening/closing times.

There is no value in translating a wall clock time to GMT until it needs to be compared with other wall clock times, and this translation will always be dynamic and needs to take into account the date on which the times are compared.

Recommendation

The

SET TIME ZONE DATE TO <date>;

SQL statement is added to Firebird as describe above and in order to set the date used to translate TIME WITH TIME ZONE to and from GMT times saved in the database. The functions

CREATE FUNCTION GMT_TO_LOCALTIME(gmt_time TIME, timezone_name CHAR(63), atDate DATE)
RETURNS TIME;

and its analogue

CREATE FUNCTION LOCALTIME_TO GMT(localtime TIME, timezone_name CHAR(63), atDate DATE)
RETURNS TIME;

and also recommended as standard functions.