Handling timezones in storage?

Asked 16 years, 4 months ago Modified 5 years, 10 months ago

Viewed 6k times

20

Store everything in GMT?

Store everything the way it was entered with an embedded offset?

Do the math everytime you render?

Display relative Times "1 minutes ago"?

localization internationalization timezone globalization

Share

Improve this question

Follow

edited Jun 27, 2009 at 1:36

zombat

94.1k • 25 • 157 • 165

asked Aug 15, 2008 at 4:54

DevelopingChris

40.7k • 30 • 89 • 119

13 Answers

Sorted by: Highest score (default)

You have to store in UTC - if you don't, your historic reporting and behaviour during things like Daylight

39

Savings goes... funny. GMT is a local time, subject to Daylight Savings relative to UTC (which is not).

Presentation to users in different time-zones can be a real bastard if you're storing local time. It's easy to adjust to local if your raw data is in UTC - just add your user's offset and you're done!

Joel talked about this in one of the podcasts (in a round-about way) - he said to store your data in the highest resolution possible (search for 'fidelity'), because you can always munge it when it goes out again. That's why I say store it as UTC, as local time you need to adjust for anyone who's not in that timezone, and that's a lot of hard work. And you need to store whether, for example, daylight savings was in effect when you stored the time. Yuk.

Often in databases in the past I've stored two - UTC for sorting, local time for display. That way neither the user nor the computer get confused.

Now, as to display: Sure, you can do the "3 minutes ago" thing, but only if you store UTC - otherwise, data entered in different timezones is going to do things like display as "-4 hours ago", which will freak people out. If you're going to display an actual time, people love to have it in their local time - and if data's being entered in multiple timezones you can only do that with ease if you're storing UTC.

Follow

edited Nov 6, 2012 at 14:00

answered Aug 15, 2008 at 5:19

Josh

8,016 ● 5 ● 43 ● 63

But, suppose you have a reservation system, and you book a room in a timezone with DST at 3 P.M. When DST happens, the offset between GMT and the meeting room's timezone changes by one hour. Suddenly the meeting is booked at a different time! – Joeri Sebrechts Jul 7, 2009 at 9:49

Never mind, I hadn't figured out that you would convert the time in the DST offset of the moment the meeting occurred, so it would always be in the right offset. – Joeri Sebrechts Jul 7, 2009 at 10:00

No, GMT does not have daylight savings. If we are being pedantic, then neither is UTC a time-zone. GMT is based on mean solar time at Greenwich. UTC is based on Atomic Time (TAI) adjusted by an integer number of seconds (think leap seconds) to remain close to the solar phase of the earth, though they can differ by up to 0.9 seconds. You'd have to keep meeting for quite a few years for the rotation time of the earth to change appreciably, but that's why the complexity exists here. – mc0e Feb 20, 2017 at 11:43

14

The answer, as always, is "depends".

It depends on what you are describing with the time, and how the data was provided to you. The key to deciding how to store time values is deciding if you are losing


information by dropping the timezone, as well as not surprising your users.

There are definite benefits in storing data in a UTC time_t - it is a single int, allowing quick sorting and easy storage.

I see the problem as being broken down into specific areas:

1. Historical Data
2. Future, Short Term Data
3. Future, Long Term Data

With the following subclasses on each:

1. System Provided
2. User Provided

Let's look at them one at a time.

System Provided: I would recommend running systems in UTC, then you avoid the timezone problem and again, no information loss is seen (it's always UTC).

Historical Data: These are things like system log files, process statistics, tracing, comment dates/times, etc. The data isn't going to change, and the timezone descriptor isn't going to change retroactively. For this type of data, there is no information lost by storing the information in UTC regardless of the timezone it was provided in. So, drop the timezone.

Future, Long Term Data: These are events that are either far enough in the future or will keep happening. If they are kept around long enough, the timezone descriptors are guaranteed change. A good example of this type of data is, "The Weekly Management Meeting". This is data that is entered once, and expected to keep working into perpetuity. For these values, it is important to determine if it is system or user provided. For user-provided data, the time should be stored with the creator's timezone, anything else results in information loss. This information loss becomes apparent when the timezone definition changes and the time is displayed to the creator as having an entirely different value!

As Bwooce has indicated, there is some confusion where the creator and viewer are in different timezones. In that case, I would expect the application to indicate which time values have moved due to a timezone shift from their previous locations.

Future, Short Term Data: This is data that is quickly going to become historical, or is only valid for a short period of time. Examples could be interval timers, rating transitions, etc. For this data, since the likelihood is low that the definition will change between the creation of the value and the time it becomes historical, it might be possible to get away with dropping the timezone. However, I have found that these values have a bad habit of becoming "Future, Long Term Data".

Once you have decided to store the timezone, care must be taken with how it is stored.

- Don't store the timezone as an offset, or the full descriptor.

If you store a timezone as an offset, what do you do if the timezone changes? Do you go through the system and do a blanket change on the existing data? If you do, you've now made any historical values incorrect. Good examples of this fault are Oracle and iCal. Oracle stores timezone information as an offset from UTC, and iCal includes the full descriptor for the creation timezone.

- Do store it as a name.

This allows the definition of the timezone to change without having to modify the existing values you have. It does make sorting more difficult, since any index that is generated may be invalid if the timezone data changes.

If developers continue to store everything in UTC, irrespective of timezone, we will continue to see the problems that we've seen with the last batch of timezone changes.

At one organisation, the secretaries had to print out the calendars for their teams before the daylight savings date, and then print them out again after the change. Finally, they compared the two and re-created all of the appointments that had moved. Of course, they missed several, and there were several weeks of pain until the

old daylight savings date was reached and the times became correct again.

Share Improve this answer Follow

answered Sep 25, 2008 at 2:44

jpollock
141 • 3

5

Josh is completely correct above, but I have one subtle caveat to explain. This is a case with no correct answer regarding future events and timezones.

Consider the case of a repeating appointment. It occurs at GMT 0000 (for simplicity), which is 1200 NZST (New Zealand Standard Time) and 1000 AEST in Sydney Australia.

When Daylight Savings comes into effect in one zone, what should occur to the appointment? Should it:

1a. If the TZ change is in the zone of the appointment's "owner" (who booked it) then attempt to remain at the same desk clock time (eg 10:00am)?
1b. If the TZ change is in one of the other meeting attendee's zones then no change

Consequences: It moves for everyone else, unexpectedly, due to the owners TZ change, but it stays "the 10am meeting" as far as the owner is concerned.

'2. As above, but reversed.

Consequences: It moves for the meeting owner (the 10am meeting becomes the 9am meeting, or v/v), which may be expected but inconvenient. It stays at the same desk clock time for the other attendees until they go through their own TZ transition.

Neither is perfect. Consider the case of two appointments, one booked by Person A that occurs at 10am local time, the other booked by Person B with Person A as an attendee that occurs at 9am. If Person A and Person B are in different TZ's then a DST change could easily cause them to become double-booked.

If your mind is a bit bent at this point then I quite understand.

The point behind this example is that to do either of these behaviors properly you need to know not just the UTC version of the local time, but the TZ (and not the offset) that the owner was in when they booked it. Otherwise you have no choice but to take option 2, silently, without even informing anyone that things have changed since GMT times don't change and only the presentation changes...right? (no, this is the trap, presentation matters when your 10am meeting moves by itself)

I have to credit my colleague and friend Jason Pollock for this insight. Read his view here, and the follow-up discussing iCal and VTIMEZONE here.

edited Sep 24, 2012 at 7:45

Share Improve this answer

Follow

Prof. Falken

24.8k ● 20 ● 102 ● 179

answered Sep 25, 2008 at 1:30

Bwooce

2,143 ● 19 ● 28

Outlook Mobile gets confused by exactly this behaviour bug. Phone appointments shift when I change the active zone.
– devstuff Dec 23, 2008 at 5:13

@devstuff - Appointments shifting when you select a different timezone is not what is being described. What is being described is that the timezone itself has changed (due to legislated changes in daylight savings) and therefore the timezone in which the data was originally described no longer exists. i.e. the times have shifted in the same timezone.
– Mark Oct 4, 2013 at 0:32

2

Storing everything in GMT/UTC seems most logical to me. You can then show the date and time in every timezone you want.

A few ceveats:

1. If a time is only specified as a wall clock time and that is the leading representation, then it is not an absolutely specified time. You should (and cannot) convert it in any GMT representation. E.G. 9:00 AM every morning. In other words: this is no (date)time.
2. If you save a date and time of a future appointment, you should use the offset to GMT specified by the input timezone and the *the moment in time itself*. So

if it is an appointment in summer made in winter in e.g. western europe, it is +2:00, allthough the normal (winter time) offset is +1:00. This will solve the calender problem that Bwooce mentioned.

3. Of course, the same that applies to using the right offset while converting to GMT applies when converting back to a date and time in any particular timezone.

Luckily, when used correctly, the (.NET) DateTime type takes care of all the gory details of keeping calendars etc. for you and all of this should be very easy when you know how it works.

Share Improve this answer Follow

edited Aug 2, 2009 at 9:58

answered Jul 9, 2009 at 14:46

Rick

73 ● 1 ● 5

1

Personally, I can't see any reason not to store everything in GMT and then use the users local timezone to display the time as it relates to them.

If you want to display relative time, you obviously still need the time and do a translation, but if you do want to do the translation I think GMT is still your best option.

Share Improve this answer

answered Aug 15, 2008 at 5:05

Follow

lomaxx
116k • 58 • 147 • 180

the issue is, what time do automated processes use when communicating, say, deadlines to people? do you store the users timezone as they decided it and use it in all communications? – DevelopingChris Jul 9, 2009 at 14:54

1

So I ran a little experiment with MSSQL server.

I created a table and added a row with the current localized timezone (Australia). Then I changed my datetime to be GMT and added another row.

Even tho those rows were added around 10 seconds apart, they appear in SQL server as tho they're 10 hours apart.

If nothing else, it at least tells me that I should be storing dates in a conisitent manner, which for me, adds weight to the argument for storing them as GMT.

Share Improve this answer

Follow

answered Aug 15, 2008 at 5:23

lomaxx
116k • 58 • 147 • 180

storing them with an embedded offset, could also mean, 2 columns, gmt time, and persons chosen offset, not the automated one form sqlserver or the client machine.
– DevelopingChris Jul 9, 2009 at 14:57

MS Dynamics stores GMT and then at a user level knows your times zone relative to GMT. Then it displays items to you in your time zone.

Just thought I'd throw that out there as that's a pretty big group at MS and this is how they decided to handle it.

Share Improve this answer

Follow

answered Aug 15, 2008 at 5:27

brendan

29.9k ● 20 ● 69 ● 109

---

i prefer to store everything with the timezone. the client can decide, which way it should be presented later. my favorite library for converting is the PostgreSQL-Database.

Share Improve this answer

Follow

answered Oct 16, 2008 at 7:13

maletin

585 ● 4 ● 15

Have a look here, the w3c have done an excellent job answering the question.

Look at the use cases.

http://www.w3.org/TR/timezone/

Note that they recommend storing datetimes as UTC not GMT, GMT is subject to daylight savings time.

Share Improve this answer

Follow

answered Sep 19, 2012 at 11:18

HKalsi

333 ● 3 ● 10

0

I like storing in GMT and showing only relative ("about 10 seconds ago", "5 months ago"). Users don't need to see actual timestamps for most use cases.

There are certainly exceptions, and an individual application might have many of them, so it can't be a 'one-true-way' answer. Things that need strong audit-ability (e.g. voting), and systems where time is part of the domain of discourse (astronomy, scientific research) might demand true timestamps to be shown to the user.

Most apps, though, are easier to understand with a simple relative time.

Share Improve this answer

Follow

answered Aug 15, 2008 at 5:15

James A. Rosen

65.2k ● 62 ● 184 ● 263

I usually just use Unix time. not necessarily future safe, but it works pretty well.

0

Share Improve this answer

Follow

answered Aug 15, 2008 at 6:20

icco

3,084 ● 4 ● 35 ● 49

Always store in GMT (or UTC). From there it is easy to convert to any local time zone value.

0

Share Improve this answer

Follow

answered Sep 16, 2008 at 20:26

Craig Fisher

1,792 ● 2 ● 19 ● 27

Dates should be stored as UTC UNLESS it is user provided data and you CANNOT know what timezone the user intended that data to be in. Sometimes (very very rarely) you need to just store the hour, minute, second, day, month and year components without any timezone so you can spit it out back to the user. Now for new developers or if you're unsure, store UTC and you will be 99% correct.

0

But don't be fooled by believing this works 100% of the time for all cases all the time. It does not.

Share Improve this answer

Follow

answered Feb 22, 2019 at 20:24

Jacob

1,062 ● 8 ● 10