

# Difference between timestamps with/without time zone in PostgreSQL

Asked 13 years, 7 months ago   Modified 4 days ago   Viewed 605k times



366



Are timestamp values stored differently in PostgreSQL when the data type is `WITH TIME ZONE` versus `WITHOUT TIME ZONE` ? Can the differences be illustrated with simple test cases?

postgresql

types

timestamp

timezone

Share

Improve this question

Follow

edited Feb 21, 2020 at 11:03



lospejos

1,986 ● 4 ● 19 ● 38

asked May 3, 2011 at 21:56



Larsenal

51.1k ● 43 ● 154 ● 223

6 [This related answer](#) may be of help. – Erwin Brandstetter Sep 6, 2012 at 2:52

8 Answers

Sorted by:

Highest score (default)





314



The differences are covered at [the PostgreSQL documentation for date/time types](#). Yes, the treatment of `TIME` or `TIMESTAMP` differs between one `WITH TIME ZONE` or `WITHOUT TIME ZONE`. It doesn't affect how the values are stored; it affects how they are interpreted.

The effects of time zones on these data types is [covered specifically](#) in the docs. The difference arises from what the system can reasonably know about the value:

- With a time zone as part of the value, the value can be rendered as a local time in the client.
- Without a time zone as part of the value, the obvious default time zone is UTC, so it is rendered for that time zone.

The behaviour differs depending on at least three factors:

- The timezone setting in the client.
- The data type (i.e. `WITH TIME ZONE` or `WITHOUT TIME ZONE`) of the value.
- Whether the value is specified with a particular time zone.

Here are examples covering the combinations of those factors:

```
foo=> SET TIMEZONE TO 'Japan';  
SET  
foo=> SELECT '2011-01-01 00:00:00'::TIMESTAMP;  
timestamp
```

```

-----
2011-01-01 00:00:00
(1 row)

foo=> SELECT '2011-01-01 00:00:00'::TIMESTAMP WITH TIM
      timestamptz
-----
2011-01-01 00:00:00+09
(1 row)

foo=> SELECT '2011-01-01 00:00:00+03'::TIMESTAMP;
      timestamp
-----
2011-01-01 00:00:00
(1 row)

foo=> SELECT '2011-01-01 00:00:00+03'::TIMESTAMP WITH
      timestamptz
-----
2011-01-01 06:00:00+09
(1 row)

foo=> SET TIMEZONE TO 'Australia/Melbourne';
SET
foo=> SELECT '2011-01-01 00:00:00'::TIMESTAMP;
      timestamp
-----
2011-01-01 00:00:00
(1 row)

foo=> SELECT '2011-01-01 00:00:00'::TIMESTAMP WITH TIM
      timestamptz
-----
2011-01-01 00:00:00+11
(1 row)

foo=> SELECT '2011-01-01 00:00:00+03'::TIMESTAMP;
      timestamp
-----
2011-01-01 00:00:00
(1 row)

foo=> SELECT '2011-01-01 00:00:00+03'::TIMESTAMP WITH
      timestamptz

```

-----  
2011-01-01 08:00:00+11  
(1 row)

Share Improve this answer

edited Jan 4, 2018 at 22:42

Follow


answered May 3, 2011 at 22:04



bignose

32.2k ● 15 ● 79 ● 116

---

178 Correct only if referring to the process of inserting/retrieving values. But readers should understand that both data types, `timestamp with time zone` and `timestamp without time zone`, in Postgres do *not* actually store time zone information. You can confirm this with a glance at the data type doc page: Both types takes up the same number of octets and have the save range of values, thus no room for storing time zone info. The text of the page confirms this. Something of a misnomer: "without tz" means "ignore offset when inserting data" and "with tz" means "use offset to adjust to UTC". – Basil Bourque Feb 15, 2014 at 9:51 

---

91 The data types are a misnomer in a second way: They say "time zone" but actually we are talking about offset from UTC/GMT. A time zone is actually an offset *plus* rules/history about Daylight Saving Time (DST) and other anomalies. – Basil Bourque Feb 15, 2014 at 9:52

---

7 I would rather say an offset is a time zone plus rules for DST. You cannot discover the time zone given an offset, but you can discover the offset given the time zone and DST rules. – igorsantos07 Nov 22, 2015 at 23:30

---

8 Citing the [official doc](#) : *All timezone-aware dates and times are stored internally in UTC. They are converted to local*

*time in the zone specified by the TimeZone configuration parameter before being displayed to the client.*

– [Guillaume Husta](#) Apr 24, 2017 at 11:27

---

- 6 @igorsantos07 A time zone *is* the set of rules/history about DST changes and other changes. Your wording seems superfluous. And your statement that "an offset is a time zone plus rules for DST" is simply wrong: an offset is merely a number of hours, minutes, and seconds – nothing more, nothing less. – [Basil Bourque](#) Jan 2, 2018 at 17:54



I try to explain it more understandably than [the PostgreSQL documentation](#).

140



Neither `TIMESTAMP` variants store a time zone (or an offset), despite what the names suggest. The difference is in the interpretation of the stored data (and in the intended application), not in the storage format itself:



- `TIMESTAMP WITHOUT TIME ZONE` stores *local* date-time (aka. wall calendar date and wall clock time). Its time zone is unspecified as far as PostgreSQL can tell (though your application may know what it is). Hence, PostgreSQL does no time zone related conversion on input or output. If the value was entered into the database as `'2011-07-01 06:30:30'`, then no matter in what time zone you display it later, it will still say year 2011, month 07, day 01, 06 hours, 30 minutes, and 30 seconds (in some format). Also, any offset or time zone you specify in the input is ignored by PostgreSQL, so

'2011-07-01 06:30:30+00' and '2011-07-01 06:30:30+05' are the same as just '2011-07-01 06:30:30'. For Java developers: it's analogous to `java.time.LocalDateTime`.

- `TIMESTAMP WITH TIME ZONE` stores a point on the UTC time line. How it looks (how many hours, minutes, etc.) depends on your time zone, but it always refers to the same "physical" instant (like the moment of an actual physical event). The input is internally converted to UTC, and that's how it's stored. For that, the offset of the input must be known, so when the input contains no explicit offset or time zone (like '2011-07-01 06:30:30') it's assumed to be in the current time zone of the PostgreSQL session, otherwise the explicitly specified offset or time zone is used (as in '2011-07-01 06:30:30+05'). The output is displayed converted to the current time zone of the PostgreSQL session. For Java developers: It's analogous to `java.time.Instant` (with lower resolution though), but with JDBC and JPA 2.2 you are supposed to map it to `java.time.OffsetDateTime` (or to `java.util.Date` or `java.sql.Timestamp` of course).

Some say that both `TIMESTAMP` variations store UTC date-time. Kind of, but it's confusing to put it that way in my opinion. `TIMESTAMP WITHOUT TIME ZONE` is stored like a `TIMESTAMP WITH TIME ZONE`, which rendered with UTC time zone happens to give the same year, month, day, hours, minutes, seconds, and microseconds as they are in the local date-time. But it's not meant to represent the

point on the time line that the UTC interpretation says, it's just the way the local date-time fields are encoded. (It's some cluster of dots on the time line, as the real time zone is not UTC; we don't know what it is.)

Share Improve this answer

edited Aug 29 at 19:44

Follow



Philippe Cloutier

516 ● 3 ● 14

answered Jan 3, 2018 at 0:36



ddekany

31k ● 4 ● 59 ● 65

---

1 There is nothing wrong with retrieving a `TIMESTAMP WITH TIME ZONE` as a `Instant`. Both represent a point on the timeline in UTC. `Instant` is preferred, in my opinion, over `OffsetDateTime` as it is more self-documenting: A `TIMESTAMP WITH TIME ZONE` is always retrieved from the database as UTC, and an `Instant` is always in UTC so a natural match, while an `OffsetDateTime` can carry other offsets. – [Basil Bourque](#) Jan 3, 2018 at 0:56 ✎

---

@BasilBourque Unfortunately, the current JDBC specification, the JPA 2.2 specification, and also the PostgreSQL JDBC documentation only mentions `OffsetDateTime` as the mapped Java type. I'm not sure if `Instant` is still unofficially supported somewhere. – [ddekany](#) Jan 3, 2018 at 9:12

---

question, you say any offset i specify in the input such as `'2011-07-01 06:30:30+00'` and `'2011-07-01 06:30:30+05'` is ignored but i'm able to do `insert into test_table (date) values ('2018-03-24T00:00:00-05:00'::timestampz);` and it will convert it to utc correctly. where date is timestamp without timezone. I'm trying to understand what the main value of timestamp with

timezone is and having trouble. – [pk1m](#) Mar 25, 2018 at 4:05



@pk1m You complicate matters with the `::timestampz`. With that you convert the string to `TIMESTAMP WITH TIME ZONE`, and when that will be further converted to `WITHOUT TIME ZONE`, that will store the "wall calendar" day and wall clock time of that instant as seen from your session time zone (which is maybe UTC). It still only will be a local timestamp with unspecified offset (no zone). – [ddekany](#) Mar 26, 2018 at 8:11

- 4 I think this is explained much better and accurate. I find the top accepted answer confusing and misleading. Thank you.  
– [Srki Rakic](#) Jun 5, 2020 at 2:44



25



Here is an example that should help. If you have a timestamp with a timezone, you can convert that timestamp into any other timezone. If you haven't got a base timezone it won't be converted correctly.

```
SELECT now(),
       now()::timestamp,
       now() AT TIME ZONE 'CST',
       now()::timestamp AT TIME ZONE 'CST'
```

Output:

```
-[ RECORD 1 ]-----
now          | 2018-09-15 17:01:36.399357+03
now          | 2018-09-15 17:01:36.399357
timezone     | 2018-09-15 08:01:36.399357
timezone     | 2018-09-16 02:01:36.399357+03
```



Share Improve this answer

Follow

edited Sep 15, 2018 at 14:04



message

4,603 ● 3 ● 30 ● 40

answered May 3, 2011 at 22:06



serby

4,306 ● 2 ● 26 ● 25

---

13 The statement *"won't be converted correctly"* is simply not true. You have to understand what `timestamp` and `timestampz` mean. `timestampz` means an absolute point in time (UTC) whereas `timestamp` denotes what the clock showed in a certain time zone. Thus, when converting `timestampz` to a time zone you are asking *what did the clock show in New York at this absolute point in time?* whereas when "converting" a `timestamp`, you're asking *what was the absolute point in time when the clock in New York showed x?* – [fphilipe](#) Mar 27, 2016 at 16:03

---

3 The `AT TIME ZONE` construct is a brain teaser its own, even if you already understand the `WITH` vs. `WITHOUT TIME ZONE` types. So it's a curious choice for explaining them. (: (`AT TIME ZONE` converts a `WITH TIME ZONE` timestamp to a `WITHOUT TIME ZONE` timestamp, and vice versa... not exactly obvious.) – [ddekany](#) Jan 3, 2018 at 15:40

---

1 `now()::timestamp AT TIME ZONE 'CST'` does not make sense, unless you what at what instant a clock for zone 'CST' would show the time that your local clock is currently showing – [Jasen](#) Apr 10, 2019 at 3:41

---



Timestamptz vs Timestamp

22



The `timestamptz` field in Postgres is basically just the `timestamp` field where Postgres actually just stores the “normalised” UTC time, even if the timestamp given in the input string has a timezone.

If your input string is: `2018-08-28T12:30:00+05:30` , when this timestamp is stored in the database, it will be stored as `2018-08-28T07:00:00`.

The advantage of this over the simple `timestamp` field is that your input to the database will be timezone independent, and will not be inaccurate when apps from different timezones insert timestamps, or when you move your database server location to a different timezone.

To quote from the docs:

For timestamp with time zone, the internally stored value is always in UTC (Universal Coordinated Time, traditionally known as Greenwich Mean Time, GMT). An input value that has an explicit time zone specified is converted to UTC using the appropriate offset for that time zone. If no time zone is stated in the input string, then it is assumed to be in the time zone indicated by the system's `TimeZone` parameter, and is converted to UTC using the offset for the timezone zone. To give a simple analogy, a `timestamptz` value represents an instant in time, the same instant for anyone viewing it. But a `timestamp` value just represents

a particular orientation of a clock, which will represent different instances of time based on your timezone.

For pretty much any use case, `timestampz` is almost always a better choice. This choice is made easier with the fact that both `timestampz` and `timestamp` take up the same 8 bytes of data.

source: <https://hasura.io/blog/postgres-date-time-data-types-on-graphql-fd926e86ee87/>

Share Improve this answer

answered Nov 7, 2020 at 6:21

Follow



ChatGPT

5,546 ● 13 ● 52 ● 72



The differences are shown in [PostgreSQL official docs](#). Please refer the docs for deep digging.

14



In a nutshell `TIMESTAMP WITHOUT TIME ZONE` doesn't save any timezone related informations if you give date time with timezone info, it takes date & time only and ignores timezone



For example

***When I save this `12:13, 11 June 2021 IST` to PostgreSQL `TIMESTAMP WITHOUT TIME ZONE` will reject the timezone information and saves the date time `12:13, 11 June 2021`***

But the the case of `TIMESTAMP WITH TIME ZONE` it saves the timezone info in `UTC` format.

For example

***When I save this `12:13, 11 June 2021 IST` to PostgreSQL `TIMESTAMP WITH TIME ZONE` type variable it will interpret this time to `UTC` value and stored as shown in below `6:43, 11 June 2021 UTC`***

*NB : UTC + 5.30 is IST*

During the time conversion time returned by `TIMESTAMP WITH TIME ZONE` will be stored in UTC format and we can convert it to the required timezone like IST or PST etc.

So the recommended timestamp type in PostgreSQL is `TIMESTAMP WITH TIME ZONE` OR `TIMESTAMPZ`

Share Improve this answer

answered Jun 11, 2021 at 6:58

Follow



[ajesh](#)

406 ● 6 ● 8

---

4 So what is new in your answer that is not covered by other/accepted answers – [Vega](#) Jun 11, 2021 at 13:43


---

1 The two types are of the same length, no extra "timezone information" is stored. The difference is only in how the value is treated (store local time vs store UTC) – [maowtm](#) Oct 1, 2023 at 14:09

---

how will you do this When I save this 12:13, 11 June 2021 IST i.e how you will tell in query that provided input is IST ? I understood the answer clearly , only this part is left  
– Yusuf Dec 30, 2023 at 12:33

---

I have found this answer and other such answers contradictory to what I noticed. On my development machine (having Ubuntu) the timezone set is IST (UTC +0530) and while I installed PostgreSQL also I don't remember if I explicitly chosen a timezone or not. As per [stackoverflow.com/a/28218103/936494](https://stackoverflow.com/a/28218103/936494) the default behavior is to use Operating System's timezone which should be IST and indeed that is the case when I run `show timezone;` on `psql`. Contd in [stackoverflow.com/questions/5876218/...](https://stackoverflow.com/questions/5876218/...)  
– Jignesh Gohel May 18 at 12:20 

---

So if I do not change this default timezone setting for PostgreSQL, then when I insert a row in a table having a column with type `timestamp with time zone` for e.g `INSERT INTO my_table_name (name, created_at) values ('test', '2024-05-18 17:18:18.6514407+0530')` the value stored in `created_at` column is found to be `2024-05-18 17:18:18.651441+05:30` instead of its corresponding UTC time `2024-05-18 11:48:18.651441 UTC` as is claimed in this and other such answers.  
– Jignesh Gohel May 18 at 12:21

---



Run the following to see diff in pgAdmin:

2



```
create table public.testts (tz timestamp with time zone  
time zone);  
insert into public.testts values(now(), now());  
select * from public.testts;
```





If you have similar issues I had of *timestamp precision in Angular / Typescript / Node API / PostgreSQL* environment, hope my [complete answer and solution](#) will help you out.

Share Improve this answer

edited Jun 18, 2021 at 23:52

Follow

answered Apr 18, 2021 at 19:49



Jeb50

7,007 ● 10 ● 65 ● 110



0

I like to think about it as the timestamp without time zone is like storing a particular clock hand setup and copy-pasting them on other clocks upon retrieval.



timestamp WITHOUT time zone assumes, you take a snapshot of some clock and store it. This setup has absolutely no idea, what time zone it resides, it's just a value with no timezone context. HOWEVER if you will get it's epoch value, it will be interpreted as this exact value **IN YOUR LOCAL TIMEZONE**, what's really confusing. For example, if 3 separate DB clients will request timestamp 01-01-2025 00:00 from **three distinct** time zone locations: UTC-8, UTC and UTC+8, they will all get same date for their local time:

- 00:00 -8
- 00:00 +0
- 00:00 +8

But it represents three different timestamps in UTC:

- 08:00 UTC
- 00:00 UTC
- 16:00 UTC

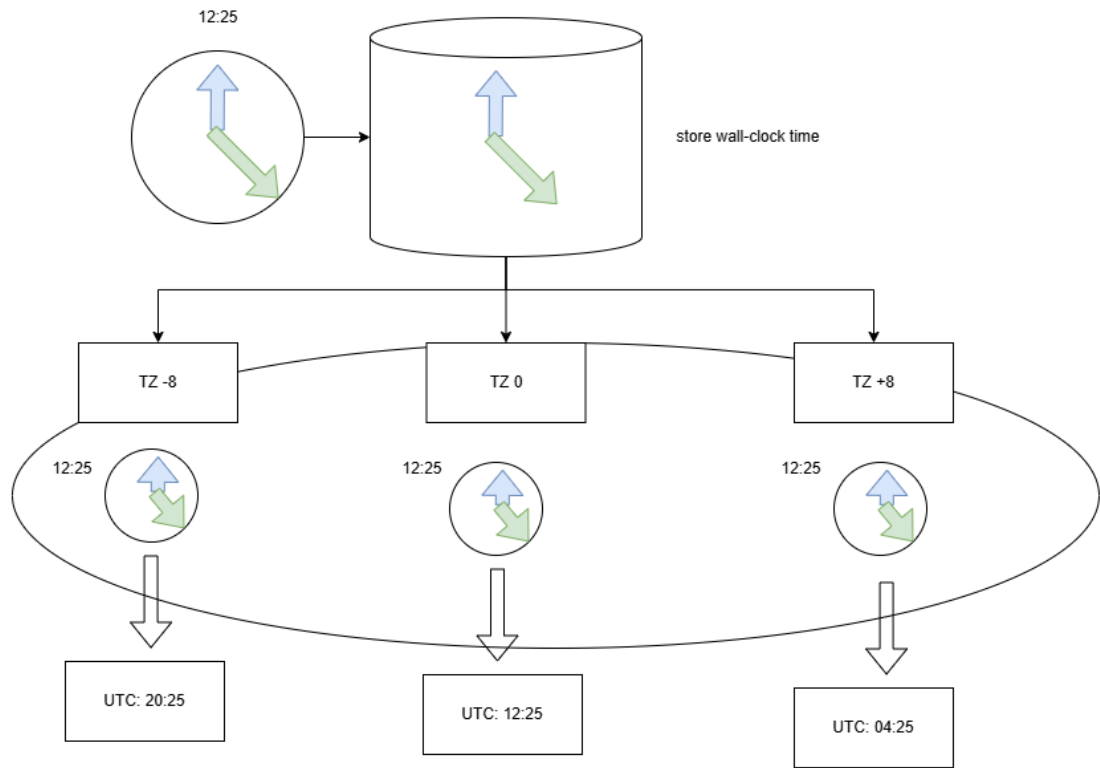
timestamp WITH time zone works entirely different: It will at store time align its offset to match UTC. Afterward, clients obtaining this timestamp will align date according to their time zones. When storing 01-01-2025 00:00 +3, it will be stored as 31-12-2024 21:00 UTC and now, same clients asking for this timestamp are able to retrieve this timestamp in UTC mode, all three then will get same, normalized date:

- 31-12-2024 21:00 UTC

Or, when respecting client session time zone:

- 31-12-2024 13:00 -8
- 31-12-2024 21:00 UTC
- 01-01-2025 05:00 UTC

Here is the graphical interpretation of timestamp without time zone



Share Improve this answer

answered Nov 6 at 14:08

Follow



**Tomas**

3,436 ● 3 ● 34 ● 52



0



The answer of [Tomas](#) is for me a important explanation. I had problems with the difference between time and timezone especially with time zone vs. without time zone. Therefore I created a mini table to understand better my problem and how PostgreSQL operates:



```
DROP TABLE IF EXISTS public."test_A_tim_timestamp_wtz";
CREATE TABLE IF NOT EXISTS public."test_A_tim_timestamp"
(
    index bigint NOT NULL,
    "theTimeTZ" time,
    "theTime" timetz,
    "theTimestamp" timestamp,
    "theTimestampTZ" timestamptz
);
INSERT INTO public."test_A_tim_timestamp"
```



```
VALUES (1, '05:35:55+07:30', '05:35:55+07:30', '1985-17T05:35:55+07:30', '1987-08-17T05:35:55+07:30'),
        (2, '04:10:52-06:30', '04:10:52-06:30', '1999-06:30', '1979-10-21T04:10:52-06:30'),
        (3, '07:57:15.123456+02:30', '07:57:15.12345629T07:57:15.123456+02:30', '2017-12-29T07:57:15.123456+
        (4, '04:43:23.987654-03:30', '04:43:23.98765416T04:43:23.987654-03:30', '2025-04-16T04:43:23.987654-
SELECT * FROM public."test_A_tim_timestamp_wtz";
```

With the explanation of Tomas I understand that a timestamp with time zone is something different than a date and time with time zone.

[Share](#) [Improve this answer](#)

answered Dec 16 at 11:28

[Follow](#)



[modern\\_stonetime\\_programmer](#)

3 ● 2

---