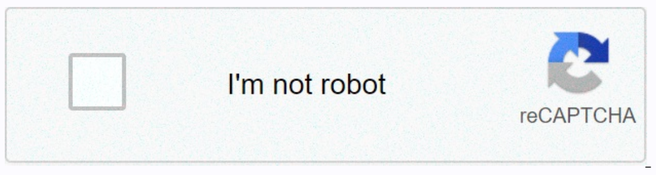

Continue

Postgresql timestamp format select

PostgreSQL comes with a bunch of built-in date and time related data types. Why should you use them over strings or integers? What should you watch out for when using them? Read on to learn more about how to effectively work with these types of data in Postgres. A bunch of formulas The SQL standard, ISO 8601, the embedded postgreSQL directory, and backward compatibility together define a plethora of overlapping, customizable data types related to date/time and confusing conventions at best. This confusion usually diffuses into database driver code, application code, SQL routines and leads to subtle errors that are difficult to debug. On the other hand, using native built-in formulas simplifies SQL statements and makes them much easier to read and write, and therefore less prone to errors. Using, say integers (number of seconds from the time) to represent time, it results in cumbersome SQL expressions and more application code. The benefits of native formulas make it useful to define a set of not-so-painful rules and enforce them across the application code and ops basis. Here is such a set, which should provide reasonable presets and a resonable starting point for further customization if required. Formulas Use only the following 3 formulas (although many are available): date - a specific date, no time stamp - a specific date and time with a microsecond resolution interval - a time interval with microsecond resolution These three formulas together should support most instances of application usage. If you don't have specific needs (such as maintaining storage), it's highly recommended that we stick to only these types. The date represents a date without time and is quite useful in practice (see examples below). The timestamp type is the variation that includes time zone information – without time zone information there are simply too many variables that can affect the interpretation and export of the value. Finally, space represents intervals from as low as a microsecond to millions of years. Literal strings Use only the following literal representations and use the cast operator to reduce detail without sacrificing readability: '2012-12-25'::d ate - ISO 8601 '2012-12-2 25 13:04 :05.123-08:00'::time stamp - ISO 8601 '1 month 3 days'::space - Postgres traditional format for entering space Omitting the time zone leaves you at the mercy of the server time zone setting the TimeZone configuration that can be set at the database level, session level, role level, or connection string, client time zone setting, and more such factors. When querying from application code, convert the space types to an appropriate unit (such as days or seconds) by using the export function and read the value as an integer or actual value. Configure and other settings Do not change the default settings for the GUC DateStyle, TimeZone, and lc_time. Do not set or use the PGDATESTYLE and PGTZ environment variables. Do not use set sets Zone.... If you can, set the system time zone to UTC on the computer running the Postgres server, as well as all machines running application code that are connected to it. Verify that the database driver (such as a JDBC connection or a Go/SQL database driver) behaves sensibly while the client is running in one time zone and on the server in another. Make sure it works correctly when a valid timezone parameter that is not UTC is included in the connection string. Finally, note that these are all just guidelines and can be tweaked to suit your needs – but make sure you explore the consequences of doing so first. Native formulas and operators So how exactly does using native formulas help simplify SQL code? Here are some examples. Date type values can be subtracted to give space between them. You can also add an integer number of days to a particle date or add a space to a date to give a timestamp: -- 10 days from now (exits 2020-07-26) SELECT now()::d ate + 10; -- 10 days from now (exits 2020-07-26 04:44:30.568847+00) SELECT now() + '10 days'::space; -- days until Christmas (exits 161 days 14:06:26.759466) SELECT '2020-12-25'::d ate - now() -- the 10 largest select name courses, end_date - start_date duration AS from the courses ORDER with end_date - start_date DESC LIMIT 10; The prices of these types are comparable, so you can order the last question end_date - start_date, which has a space type. Here's another example: -- certificates that expire within the next 7 days SELECT THE NAME FROM CERTIFICATES WHERE expiry_date BETWEEN NOW() AND NOW() + '7 Days'::space? Type timestamp values can also be subtracted (to give a space), added (in a space to give another time stamp), and compared. -- the difference in time markings gives a time interval SELECT password_last_modified - created_at as password_age by users; -- can also use the age() function SELECT age(password_last_modified, created_at) AS password_age from users; While in the topic, note that there are 3 different built-in functions that return different current timestamp values. In fact different things return: -- transaction_timestamp() returns the time stamps of the start of the current transaction -- exits 2020-07-16 05:09:32.677409+00 SELECT transaction_timestamp() -- statement_timestamp() returns the time stamp of the start of the current SELECT statement_timestamp(). -- clock_timestamp() returns the timestamp of the SELECT system clock clock_timestamp(). There are aliases for these functions: -- now() actually returns the start of the current transaction, which means that -- does it not change during the SELECT transaction now(), transaction_timestamp()? -- the transaction time stamp is also returned by these keyword structures CURRENT_DATE, CURRENT_TIMESTAMP, transaction_timestamp(). Space Types Values that are typed intermittently can be used as column data types, can be compared to each other and added to (and removed removed time stamps and dates. Here are some examples: -- values with space typing can be saved and SELECT values compared from WHERE valid_for passports > '10 years'::ORDER BY valid_for DESC; -- you can multiply them by numbers (outputs 4 years) SELECT 4 * '1 year'::space? -- you can divide them by numbers (outputs 3 mons) SELECT '1 year'::space / 4; -- you can add and remove them (exits 1 year 1 mon 6 days) SELECT '1 year'::space + '1.2 months'::space; Other PostgreSQL functions and constructions also comes with some useful functions and constructions that can be used to manipulate the values of these types. The export function can be used to retrieve a specified portion from the given value, such as the month from a date. The full list of sections that can be exported is documented here. Here are some useful and non-obvious examples: -- years from a space (exits 2) SELECT extract (YEARS FROM 1.5 years 6 months::space); -- day of the week (0=Sun .. 6=Saturday) by timestamp (exits 4) SELECT quote (DOW FROM NOW()); -- day of the week (1=Mon. 7=Sun) by timestamp (exits 4) SELECT quote (ISODOW FROM NOW()); -- space conversion in seconds (outputs 86400) SELECT extract (EPOCH FROM '1 day'::space) The latter example is particularly useful in queries run by applications, as it may be easier for applications to handle a space as a floating-point value of the number of seconds/minutes/days/etc. Time zone conversion There is also an easy-to-use function to express a timestamp in another time zone. Usually this will be done in the application code – it is easier to control in this way and reduces the dependency on the time zone database to which the Postgres server will refer. Nevertheless, it can be useful from time to time: -- convert time markings to a different SELECT time zone ('Europe/Helsinki', now()); -- as before, but this is an SQL SELECT standard now() in the Europe/Helsinki time zone; Convert to and from text The to_char function (documents) can convert dates, time stamps, and spaces to text based on a format string - the Postgres equivalent of classic C-mode strftime. -- exits Thu, 16 July SELECT to_char(now(), 'Dy, DDth Month') -- exits 01 06 00 12 00 00 SELECT to_char('1.5 years':space, 'YY MM DD HH MI SS'); To convert from text to dates, use to_date and to convert text to time stamps, use to_timestamp. Note that if you use the forms listed at the beginning of this publication, you can only use the cast operators instead. -- exits 2000-12-25 SELECT to_timestamp('2000.12.25.15.42.50', 'YYYY. MM.DD.HH24.MI.SS»)· -- exits 2000-12-25 SELECT to_date('2000.12.25.15.42.50', 'YYYY. MM.DD»)· Refer to the documents for the full list of format string patterns. It is best to use these functions for simple cases. For more complex analysis or formatting, it's best to rely on the application code, which can (undoubtedly) be best tested on a drive. Interface with application code Sometimes not convenient convenient pass the date/time stamp/space values to and from the application code, especially when reserved parameters are used. For example, it is usually more convenient to spend a space as an integer number of days (or hours or minutes) than in string format. It is also easier to read in a space as an integer/number of floating-point days (or hours, or minutes, etc.). This function make_interval be used to create a space value from an integer number of item values (see documents here). The to_timestamp operation we saw earlier has another format that can generate a timestamp value from the Unix era. -- pass the interval as a number of days from the application code SELECT name FROM courses WHERE the duration <= make_interval(days => $1). -- pass timestamptz as unix epoch (number of seconds from 1-Jan-1970) SELECT id FROM events WHERE logged_at >= to_timestamp($1). -- return interval as number of days (fractional part) SELECT EXTRACT (EPOCH FROM duration) / 60 / 60 / 24; About pgDash pgDash is a modern, in-depth monitoring solution designed specifically for postgreSQL deployments. pgDash shows you information and metrics about every aspect of your PostgreSQL database server, collected using the open source pgmetrics tool. pgDash provides basic reporting and visualization functions, including collecting and displaying PostgreSQL information, and providing time series charts, detailed reports, diagnostics, notifications, groups, and more. Complete purchase of features here or register today for a free trial. Test.