# 9.8. Data Type Formatting Functions

The PostgreSQL formatting functions provide a powerful set of tools for converting various data types (date/time, integer, floating point, numeric) to formatted strings and for converting from formatted strings to specific data types. Table 9.26 lists them. These functions all follow a common calling convention: the first argument is the value to be formatted and the second argument is a template that defines the output or input format.

Table 9.26. Formatting Functions

| Function<br>Description<br>Example(s) |
|---|
| `to_char ( timestamp, text ) → text`<br>`to_char ( timestamp with time zone, text ) → text`<br>Converts time stamp to string according to the given format.<br>`to_char(timestamp '2002-04-20 17:31:12.66', 'HH12:MI:SS') → 05:31:12` |
| `to_char ( interval, text ) → text`<br>Converts interval to string according to the given format.<br>`to_char(interval '15h 2m 12s', 'HH24:MI:SS') → 15:02:12` |
| `to_char ( numeric_type, text ) → text`<br>Converts number to string according to the given format; available for `integer`, `bigint`, `numeric`, `real`, `double precision`.<br>`to_char(125, '999') → 125`<br>`to_char(125.8::real, '999D9') → 125.8`<br>`to_char(-125.8, '999D99S') → 125.80-` |
| `to_date ( text, text ) → date`<br>Converts string to date according to the given format.<br>`to_date('05 Dec 2000', 'DD Mon YYYY') → 2000-12-05` |
| `to_number ( text, text ) → numeric`<br>Converts string to numeric according to the given format.<br>`to_number('12,454.8-', '99G999D9S') → -12454.8` |
| `to_timestamp ( text, text ) → timestamp with time zone`<br>Converts string to time stamp according to the given format. (See also `to_timestamp(double precision)` in Table 9.33.)<br>`to_timestamp('05 Dec 2000', 'DD Mon YYYY') → 2000-12-05 00:00:00-05` |

> **Tip**
>
> `to_timestamp` and `to_date` exist to handle input formats that cannot be converted by simple casting. For most standard date/time formats, simply casting the source string to the required data type works, and is much easier. Similarly, `to_number` is unnecessary for standard numeric representations.

In a `to_char` output template string, there are certain patterns that are recognized and replaced with appropriately-formatted data based on the given value. Any text that is not a template pattern is simply copied verbatim. Similarly, in an input template string (for the other functions), template patterns identify the values to be supplied by the input data string. If there are characters in the template string that are not template patterns, the corresponding characters in the input data string are simply skipped over (whether or not they are equal to the template string characters).

Table 9.27 shows the template patterns available for formatting date and time values.

Table 9.27. Template Patterns for Date/Time Formatting

| Pattern | Description |
|---|---|
| `HH` | hour of day (01–12) |
| `HH12` | hour of day (01–12) |
| `HH24` | hour of day (00–23) |
| `MI` | minute (00–59) |
| `SS` | second (00–59) |
| `MS` | millisecond (000–999) |
| `US` | microsecond (000000–999999) |
| `FF1` | tenth of second (0–9) |
| `FF2` | hundredth of second (00–99) |
| `FF3` | millisecond (000–999) |
| `FF4` | tenth of a millisecond (0000–9999) |
| `FF5` | hundredth of a millisecond (00000–99999) |
| `FF6` | microsecond (000000–999999) |
| `SSSS, SSSSS` | seconds past midnight (0–86399) |
| `AM, am, PM or pm` | meridiem indicator (without periods) |
| `A.M., a.m., P.M. or p.m.` | meridiem indicator (with periods) |
| `Y,YYY` | year (4 or more digits) with comma |

| Pattern | Description |
|---|---|
| YYYY | year (4 or more digits) |
| YYY | last 3 digits of year |
| YY | last 2 digits of year |
| Y | last digit of year |
| IYYY | ISO 8601 week-numbering year (4 or more digits) |
| IYY | last 3 digits of ISO 8601 week-numbering year |
| IY | last 2 digits of ISO 8601 week-numbering year |
| I | last digit of ISO 8601 week-numbering year |
| BC, bc, AD or ad | era indicator (without periods) |
| B.C., b.c., A.D. or a.d. | era indicator (with periods) |
| MONTH | full upper case month name (blank-padded to 9 chars) |
| Month | full capitalized month name (blank-padded to 9 chars) |
| month | full lower case month name (blank-padded to 9 chars) |
| MON | abbreviated upper case month name (3 chars in English, localized lengths vary) |
| Mon | abbreviated capitalized month name (3 chars in English, localized lengths vary) |
| mon | abbreviated lower case month name (3 chars in English, localized lengths vary) |
| MM | month number (01–12) |
| DAY | full upper case day name (blank-padded to 9 chars) |
| Day | full capitalized day name (blank-padded to 9 chars) |
| day | full lower case day name (blank-padded to 9 chars) |
| DY | abbreviated upper case day name (3 chars in English, localized lengths vary) |
| Dy | abbreviated capitalized day name (3 chars in English, localized lengths vary) |
| dy | abbreviated lower case day name (3 chars in English, localized lengths vary) |
| DDD | day of year (001–366) |
| IDDD | day of ISO 8601 week-numbering year (001–371; day 1 of the year is Monday of the first ISO week) |
| DD | day of month (01–31) |
| D | day of the week, Sunday (1) to Saturday (7) |
| ID | ISO 8601 day of the week, Monday (1) to Sunday (7) |
| W | week of month (1–5) (the first week starts on the first day of the month) |
| WW | week number of year (1–53) (the first week starts on the first day of the year) |
| IW | week number of ISO 8601 week-numbering year (01–53; the first Thursday of the year is in week 1) |
| CC | century (2 digits) (the twenty-first century starts on 2001-01-01) |
| J | Julian Date (integer days since November 24, 4714 BC at local midnight; see Section B.7) |
| Q | quarter |
| RM | month in upper case Roman numerals (I–XII; I=January) |
| rm | month in lower case Roman numerals (i–xii; i=January) |
| TZ | upper case time-zone abbreviation (only supported in to_char) |
| tz | lower case time-zone abbreviation (only supported in to_char) |
| TZH | time-zone hours |
| TZM | time-zone minutes |
| OF | time-zone offset from UTC (only supported in to_char) |

Modifiers can be applied to any template pattern to alter its behavior. For example, `FMMonth` is the `Month` pattern with the `FM` modifier. Table 9.28 shows the modifier patterns for date/time formatting.

Table 9.28. Template Pattern Modifiers for Date/Time Formatting

| Modifier | Description | Example |
|---|---|---|
| FM prefix | fill mode (suppress leading zeroes and padding blanks) | FMMonth |
| TH suffix | upper case ordinal number suffix | DDTH, e.g., 12TH |
| th suffix | lower case ordinal number suffix | DDth, e.g., 12th |
| FX prefix | fixed format global option (see usage notes) | FX Month DD Day |
| TM prefix | translation mode (use localized day and month names based on lc_time) | TMMonth |
| SP suffix | spell mode (not implemented) | DDSP |

Usage notes for date/time formatting:

- `FM` suppresses leading zeroes and trailing blanks that would otherwise be added to make the output of a pattern be fixed-width. In PostgreSQL, `FM` modifies only the next specification, while in Oracle `FM` affects all subsequent specifications, and repeated `FM` modifiers toggle fill mode on and off.
- `TM` suppresses trailing blanks whether or not `FM` is specified.
- `to_timestamp` and `to_date` ignore letter case in the input; so for example `MON`, `Mon`, and `mon` all accept the same strings. When using the `TM` modifier, case-folding is done according to the rules of the function's input collation (see Section 24.2).
- `to_timestamp` and `to_date` skip multiple blank spaces at the beginning of the input string and around date and time values unless the `FX` option is used. For example, `to_timestamp(' 2000 JUN', 'YYYY MON')` and `to_timestamp('2000 - JUN', 'YYYY-MON')` work, but `to_timestamp('2000 JUN', 'FXYYYY MON')` returns an error because `to_timestamp` expects only a single space. `FX` must be specified as the first item in the template.
- A separator (a space or non-letter/non-digit character) in the template string of `to_timestamp` and `to_date` matches any single separator in the input string or is skipped, unless the `FX` option is used. For example, `to_timestamp('2000JUN', 'YYYY///MON')` and `to_timestamp('2000/JUN', 'YYYY MON')` work, but `to_timestamp('2000//JUN', 'YYYY/MON')` returns an error because the number of separators in the input string exceeds the number of separators in the template.

  If `FX` is specified, a separator in the template string matches exactly one character in the input string. But note that the input string character is not required to be the same as the separator from the template string. For example, `to_timestamp('2000/JUN', 'FXYYYY MON')` works, but `to_timestamp('2000/JUN', 'FXYYYY MON')` returns an error because the second space in the template string consumes the letter `J` from the input string.

- A `TZH` template pattern can match a signed number. Without the `FX` option, minus signs may be ambiguous, and could be interpreted as a separator. This ambiguity is resolved as follows: If the number of separators before `TZH` in the template string is less than the number of separators before the minus sign in the input string, the minus sign is interpreted as part of `TZH`. Otherwise, the minus sign is considered to be a separator between values. For example, `to_timestamp('2000 -10', 'YYYY TZH')` matches `-10` to `TZH`, but `to_timestamp('2000 -10', 'YYYY  TZH')` matches `10` to `TZH`.

- Ordinary text is allowed in `to_char` templates and will be output literally. You can put a substring in double quotes to force it to be interpreted as literal text even if it contains template patterns. For example, in `'"Hello Year "YYYY'`, the `YYYY` will be replaced by the year data, but the single `Y` in `Year` will not be. In `to_date`, `to_number`, and `to_timestamp`, literal text and double-quoted strings result in skipping the number of characters contained in the string; for example `"XX"` skips two input characters (whether or not they are `XX`).

> **Tip**
>
> Prior to PostgreSQL 12, it was possible to skip arbitrary text in the input string using non-letter or non-digit characters. For example, `to_timestamp('2000y6m1d', 'yyyy-MM-DD')` used to work. Now you can only use letter characters for this purpose. For example, `to_timestamp('2000y6m1d', 'yyyytMMtDDt')` and `to_timestamp('2000y6m1d', 'yyyy"y"MM"m"DD"d"')` skip `y`, `m`, and `d`.

- If you want to have a double quote in the output you must precede it with a backslash, for example `'\"YYYY Month\"'`. Backslashes are not otherwise special outside of double-quoted strings. Within a double-quoted string, a backslash causes the next character to be taken literally, whatever it is (but this has no special effect unless the next character is a double quote or another backslash).

- In `to_timestamp` and `to_date`, if the year format specification is less than four digits, e.g., `YYY`, and the supplied year is less than four digits, the year will be adjusted to be nearest to the year 2020, e.g., `95` becomes 1995.

- In `to_timestamp` and `to_date`, negative years are treated as signifying BC. If you write both a negative year and an explicit `BC` field, you get AD again. An input of year zero is treated as 1 BC.

- In `to_timestamp` and `to_date`, the `YYYY` conversion has a restriction when processing years with more than 4 digits. You must use some non-digit character or template after `YYYY`, otherwise the year is always interpreted as 4 digits. For example (with the year 20000): `to_date('200001130', 'YYYYMMDD')` will be interpreted as a 4-digit year; instead use a non-digit separator after the year, like `to_date('20000-1130', 'YYYY-MMDD')` or `to_date('20000Nov30', 'YYYYMonDD')`.

- In `to_timestamp` and `to_date`, the `CC` (century) field is accepted but ignored if there is a `YYY`, `YYYY` or `Y,YYY` field. If `CC` is used with `YY` or `Y` then the result is computed as that year in the specified century. If the century is specified but the year is not, the first year of the century is assumed.

- In `to_timestamp` and `to_date`, weekday names or numbers (`DAY`, `D`, and related field types) are accepted but are ignored for purposes of computing the result. The same is true for quarter (`Q`) fields.

- In `to_timestamp` and `to_date`, an ISO 8601 week-numbering date (as distinct from a Gregorian date) can be specified in one of two ways:

    - Year, week number, and weekday: for example `to_date('2006-42-4', 'IYYY-IW-ID')` returns the date `2006-10-19`. If you omit the weekday it is assumed to be 1 (Monday).

    - Year and day of year: for example `to_date('2006-291', 'IYYY-IDDD')` also returns `2006-10-19`.

  Attempting to enter a date using a mixture of ISO 8601 week-numbering fields and Gregorian date fields is nonsensical, and will cause an error. In the context of an ISO 8601 week-numbering year, the concept of a “month” or “day of month” has no meaning. In the context of a Gregorian year, the ISO week has no meaning.

> **Caution**
>
> While `to_date` will reject a mixture of Gregorian and ISO week-numbering date fields, `to_char` will not, since output format specifications like `YYYY-MM-DD (IYYY-IDDD)` can be useful. But avoid writing something like `IYYY-MM-DD`; that would yield surprising results near the start of the year. (See Section 9.9.1 for more information.)

- In `to_timestamp`, millisecond (`MS`) or microsecond (`US`) fields are used as the seconds digits after the decimal point. For example `to_timestamp('12.3', 'SS.MS')` is not 3 milliseconds, but 300, because the conversion treats it as 12 + 0.3 seconds. So, for the format `SS.MS`, the input values `12.3`, `12.30`, and `12.300` specify the same number of milliseconds. To get three milliseconds, one must write `12.003`, which the conversion treats as 12 + 0.003 = 12.003 seconds.

  Here is a more complex example: `to_timestamp('15:12:02.020.001230', 'HH24:MI:SS.MS.US')` is 15 hours, 12 minutes, and 2 seconds + 20 milliseconds + 1230 microseconds = 2.021230 seconds.

- `to_char(..., 'ID')`'s day of the week numbering matches the `extract(isodow from ...)` function, but `to_char(..., 'D')`'s does not match `extract(dow from ...)`'s day numbering.

- `to_char(interval)` formats `HH` and `HH12` as shown on a 12-hour clock, for example zero hours and 36 hours both output as `12`, while `HH24` outputs the full hour value, which can exceed 23 in an `interval` value.

Table 9.29 shows the template patterns available for formatting numeric values.

Table 9.29. Template Patterns for Numeric Formatting

| Pattern | Description |
|---|---|
| `9` | digit position (can be dropped if insignificant) |
| `0` | digit position (will not be dropped, even if insignificant) |
| `.` (period) | decimal point |
| `,` (comma) | group (thousands) separator |
| `PR` | negative value in angle brackets |
| `S` | sign anchored to number (uses locale) |
| `L` | currency symbol (uses locale) |
| `D` | decimal point (uses locale) |
| `G` | group separator (uses locale) |
| `MI` | minus sign in specified position (if number < 0) |
| `PL` | plus sign in specified position (if number > 0) |
| `SG` | plus/minus sign in specified position |
| `RN` | Roman numeral (input between 1 and 3999) |
| `TH` or `th` | ordinal number suffix |
| `V` | shift specified number of digits (see notes) |
| `EEEE` | exponent for scientific notation |

Usage notes for numeric formatting:

- `0` specifies a digit position that will always be printed, even if it contains a leading/trailing zero. `9` also specifies a digit position, but if it is a leading zero then it will be replaced by a space, while if it is a trailing zero and fill mode is specified then it will be deleted. (For `to_number()`, these two pattern characters are equivalent.)

- If the format provides fewer fractional digits than the number being formatted, `to_char()` will round the number to the specified number of fractional digits.

- The pattern characters `S`, `L`, `D`, and `G` represent the sign, currency symbol, decimal point, and thousands separator characters defined by the current locale (see lc_monetary and lc_numeric). The pattern characters period and comma represent those exact characters, with the meanings of decimal point and thousands separator, regardless of locale.

- If no explicit provision is made for a sign in `to_char()`'s pattern, one column will be reserved for the sign, and it will be anchored to (appear just left of) the number. If `S` appears just left of some `9`'s, it will likewise be anchored to the number.

- A sign formatted using `SG`, `PL`, or `MI` is not anchored to the number; for example, `to_char(-12, 'MI9999')` produces `'-  12'` but `to_char(-12, 'S9999')` produces `'  -12'`. (The Oracle implementation does not allow the use of `MI` before `9`, but rather requires that `9` precede `MI`.)

- `TH` does not convert values less than zero and does not convert fractional numbers.

- `PL`, `SG`, and `TH` are PostgreSQL extensions.

- In `to_number`, if non-data template patterns such as `L` or `TH` are used, the corresponding number of input characters are skipped, whether or not they match the template pattern, unless they are data characters (that is, digits, sign, decimal point, or comma). For example, `TH` would skip two non-data characters.

- `V` with `to_char` multiplies the input values by `10^n`, where ***n*** is the number of digits following `V`. `V` with `to_number` divides in a similar manner. `to_char` and `to_number` do not support the use of `V` combined with a decimal point (e.g., `99.9V99` is not allowed).

- `EEEE` (scientific notation) cannot be used in combination with any of the other formatting patterns or modifiers other than digit and decimal point patterns, and must be at the end of the format string (e.g., `9.99EEEE` is a valid pattern).

Certain modifiers can be applied to any template pattern to alter its behavior. For example, `FM99.99` is the `99.99` pattern with the `FM` modifier. Table 9.30 shows the modifier patterns for numeric formatting.

Table 9.30. Template Pattern Modifiers for Numeric Formatting

| Modifier | Description | Example |
|---|---|---|
| `FM` prefix | fill mode (suppress trailing zeroes and padding blanks) | `FM99.99` |
| `TH` suffix | upper case ordinal number suffix | `999TH` |
| `th` suffix | lower case ordinal number suffix | `999th` |

Table 9.31 shows some examples of the use of the `to_char` function.

Table 9.31. `to_char` Examples

| Expression | Result |
|---|---|
| `to_char(current_timestamp, 'Day, DD  HH12:MI:SS')` | `'Tuesday  , 06  05:39:18'` |
| `to_char(current_timestamp, 'FMDay, FMDD  HH12:MI:SS')` | `'Tuesday, 6  05:39:18'` |
| `to_char(-0.1, '99.99')` | `'  -.10'` |

| Expression | Result |
|---|---|
| to_char(-0.1, 'FM9.99') | '-.1' |
| to_char(-0.1, 'FM90.99') | '-0.1' |
| to_char(0.1, '0.9') | ' 0.1' |
| to_char(12, '9990999.9') | '  0012.0' |
| to_char(12, 'FM9990999.9') | '0012.' |
| to_char(485, '999') | ' 485' |
| to_char(-485, '999') | '-485' |
| to_char(485, '9 9 9') | ' 4 8 5' |
| to_char(1485, '9,999') | ' 1,485' |
| to_char(1485, '9G999') | ' 1 485' |
| to_char(148.5, '999.999') | ' 148.500' |
| to_char(148.5, 'FM999.999') | '148.5' |
| to_char(148.5, 'FM999.990') | '148.500' |
| to_char(148.5, '999D999') | ' 148,500' |
| to_char(3148.5, '9G999D999') | ' 3 148,500' |
| to_char(-485, '999S') | '485-' |
| to_char(-485, '999MI') | '485-' |
| to_char(485, '999MI') | '485 ' |
| to_char(485, 'FM999MI') | '485' |
| to_char(485, 'PL999') | '+485' |
| to_char(485, 'SG999') | '+485' |
| to_char(-485, 'SG999') | '-485' |
| to_char(-485, '9SG99') | '4-85' |
| to_char(-485, '999PR') | '<485>' |
| to_char(485, 'L999') | 'DM 485' |
| to_char(485, 'RN') | '       CDLXXXV' |
| to_char(485, 'FMRN') | 'CDLXXXV' |
| to_char(5.2, 'FMRN') | 'V' |
| to_char(482, '999th') | ' 482nd' |
| to_char(485, '"Good number:"999') | 'Good number: 485' |
| to_char(485.8, '"Pre:"999" Post:" .999') | 'Pre: 485 Post: .800' |
| to_char(12, '99V999') | ' 12000' |
| to_char(12.4, '99V999') | ' 12400' |
| to_char(12.45, '99V9') | ' 125' |
| to_char(0.0004859, '9.99EEEE') | ' 4.86e-04' |


Prev
9.7. Pattern Matching

Up
Home

Next
9.9. Date/Time Functions and Operators


## Submit correction

If you see anything in the documentation that is not correct, does not match your experience with the particular feature or requires further clarification, please use this form to report a documentation issue.