Chapter 6

Dates and Types

Contents

1 Date Types

- In this section we'll be consider how Dates are dealt with in SQL. As a reminder, Figure 6.1 has the different types that most SQL standards subscribe too.

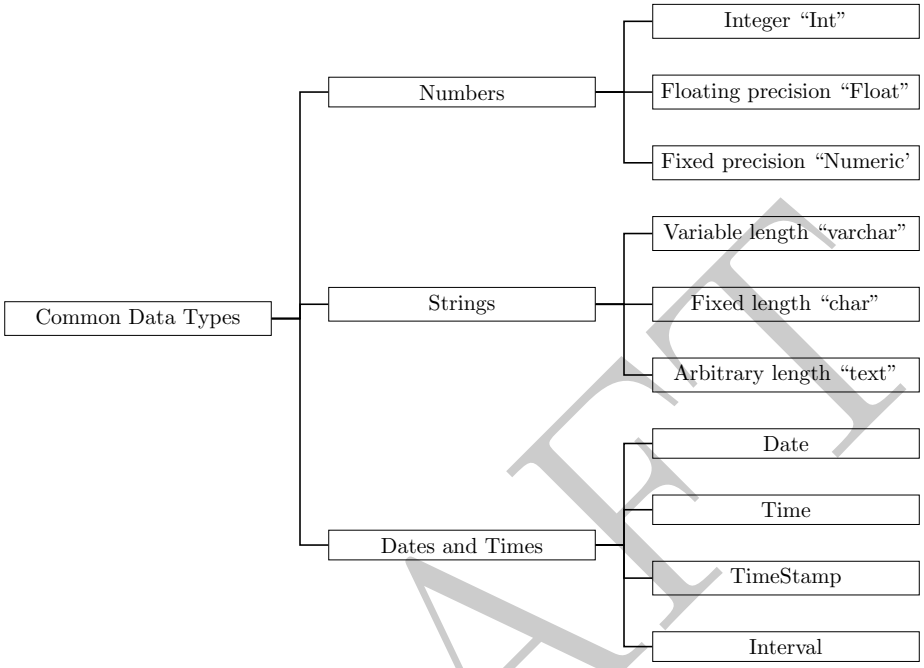

Figure 6.1: Common relational database data types

Dates

- Date and time functions are the *least* standardized portion of the SQL language. Different variants use different functions and conventions when in this area.
- Dates and times are *complicated*. Timezones, server and client location and configuration all yield small changes in how the database operates and what is returned in different operations.
- Date and times should not be considered "standard" between SQL variants. Different servers use different functions, types and standards. Whenever working with a variant of SQL you are unfamiliar with make sure to verify that it is doing what you expect.
- There are four standard data types in Postgres:
  1. **Dates**: Stores a date, ranging from 4713 BC to 5874897 AD.
  2. **Time**: Stores only a time with a resolution of 1 microsecond.
  3. **Timestamp**: Sometimes referred to as a "datetime." Contains both a date and a time and ranges from 4713 BC to 294276 AD.
  4. **Interval**: Time interval (such as "1 Year" or "2 hours"). These do not have a start or end and only represent a length of time.
- Timezones are problematic: Dates do not contain a timezone, but time and timestamp *may* have them. Depending on how the server and client are set-up, timezones may also prove otherwise problematic. Often it feels like timezones are applied in a haphazard way, so be careful!

- Daylight savings times, for example, starts and ends on different dates in Europe then in the United States.
- Because of all these issues, many database administrators (including myself) recommend storing times using **Unix** or **epoch** time, which is the number of seconds since 1/1/1970 in UTC. Since this is a specific point in time, there is no timezone confusion.

2 Date Functions

- There are four classes of operations we want to do with date objects:
  1. **Convert a string to date:** We can do this a few different ways, but these are types of cast operators. We won't get into this too much. In simple cases we can do the "obvious" and it works.

```
select '2012/03/12'::date as dt


dt
----------
2012-03-12
```

```
select '2012/03/01 11:35:00'::timestamp as ts


ts
-------------------
2012-03-01 11:35:00
```

```
select '1 month'::interval, '2 hours'::interval as dt


interval          dt
----------------  -------
30 days, 0:00:00  2:00:00
```

     A reminder that the double colon notation is specific to Postgres. There are alternative functions, such as `to_timestamp` and `to_date` which exist in other variants which do similar things.
  2. **Convert a date to a string:** This isn't something that we do that much in SQL, but if required we use the `to_char` function.
  3. **Extract part of a date:** There are two functions which do this, `date_part` or `extract`. These functions extract a specific value from a date or timestamp and return it as a different type (frequently an integer). A few examples below:

```
select date_part( 'month', '2012/03/12'::date) as mnth;


  mnth
------
     3
```

```
select date_part( 'hour', '2012/03/01 11:35:00'::timestamp ) as hr;


  hr
----
  11
```

```
select extract( 'dow' from '2012/03/01 11:35:00'::timestamp) as day_of_week;


  day_of_week
-------------
           4
```

Note that `dow` starts with Sunday (at zero) and goes to Saturday (6)

4. **Basic date math and comparisons:** To do basic date math, such as adding a 3 days or subtracting an hour, we use the addition and subtraction operators with *intervals*:

```
select '2001-01-01 01:00:00'::timestamp + '21 hours'::interval as TS;


ts
-------------------
2001-01-01 22:00:00
```

Comparison operators ($<, >, =$) behave as expected. One caveat is that when you compare a timestamp with a date, the date is converted to a timestamp for midnight of that day:

```
select now() < current_date as c1, now() > current_date as c2;


c1     c2
-----  ----
False  True
```

- While the above are the four most common operations you want to do, there are some additional functions which are nice to know about:
  - `current_date` and `current_time` / `now()` return the full timestamp, the current date and the current time.

```
select now(), current_date, current_time


now                               current_date    current_time
--------------------------------  --------------  --------------------
2023-08-14 20:50:04.440825+00:00  2023-08-14      20:50:04.440825+00:00
```

- `date_trunc`: This function truncates a date or timestamp down to a certain precision. Note that this will return a timestamp with the values beyond the specified precision set to their lowest possible value. For example:

```
select date_trunc( 'month', '2012/03/12'::date) as mnth;


mnth
-------------------------
2012-03-01 00:00:00+00:00
```

```
select date_trunc( 'hour', '2012/03/01 11:35:00'::timestamp ) as hr;


hr
-------------------
2012-03-01 11:00:00
```

  In both of these examples the value being acted on loses its precision. The object returned is a timestamp with all precision below a certain threshold set to the smallest possible value.

- `date()`: Returns the date of a given timestamp in date format. This does a type conversion, which is more than simply truncating the timestamp.

```
select date(  '2012/03/01 11:35:00'::timestamp  ) as dt;


dt
----------
2012-03-01
```

Unfortunately, dates can be difficult to work with, as the following examples demonstrate:

- Seemingly arbitrary math. You can add integers to *dates*, but not to *timestamps*

```
select now() + 1 as dt;
ERROR:  operator does not exist: timestamp with time zone + integer
```

```
select date( now() ) + 1 as dt;


dt
----------
2023-08-15
```

- BETWEEN may not work as expected. For example:

```
select now() between date(now()) -1 and now() as TF;


tf
----
True
```

Now is between yesterday and now

```
select now() between date(now()) -1 and date(now()) as TF;


tf
-----
False
```

but now is not between yesterday and today.

```
select date(now()) between now() and date( now()) + 1 as TF;


tf
-----
False
```

but today is between now and tomorrow.

- Most annoying of all? Every variant of SQL is *slightly* different.
- Epoch time is pretty great at solving some of these problems, but at the cost of interpretation:
  - Math works as expected.
  - BETWEEN works as expected.
  - No time zone ambiguity.
- For the rest of this section we'll do a few time related problems using the NYC MTA data.
- Let's return the average cash volume of cars by day of the week, inbound traffic only:

```
select
    date_part('dow', mtadt ) as dow
    , avg( tvol) as avgvol
from
(select
     sum( vehiclescash ) as tvol
     , mtadt
from
    cls.mta
where
    direction = 'I'
group by 2) as innerQ
group by 1
order by 2 desc;


  dow    avgvol
-----  --------
    6   87392.9
    0   82829.7
    5   78129.6
    4   69274.6
    1   67410.7
[...]
```

- By year, what percentage of cars which pass through a toll plaza use an EZ-pass?

```
select
    date_part('year', mtadt) as yr
    , sum(vehiclesez)::float / (sum(vehiclesez) + sum( vehiclescash)) as pct_EZ
from
    cls.mta
group by 1
order by 1;


  yr    pct_ez
----  --------
2010  0.75715
2011  0.792285
2012  0.808791
2013  0.828819
2014  0.836587
[...]
```

- We can also create a time series of the number of inbound cars, by year and month, for Plaza #1 and #2:

```
select
    date_trunc('month', mtadt)
    , sum( case when plaza = 1
        then vehiclescash else 0 end) as Plaza1Cars
    , sum( case when plaza = 2
        then vehiclescash else 0 end) as Plaza2Cars
from
    cls.mta
where
    direction = 'I'
group by 1;


date_trunc                  plaza1cars    plaza2cars
--------------------------  ------------  ------------
2010-01-01 00:00:00+00:00         427660         313278
2010-02-01 00:00:00+00:00         375918         274724
2010-03-01 00:00:00+00:00         462078         354619
2010-04-01 00:00:00+00:00         455395         353378
2010-05-01 00:00:00+00:00         487051         378600
[...]
```

- When moving in and out of date formats, you may have to rely on using special functions. Part of the reason for this is because date and times require the user to specify the format. Consider the following query:

```
select
    mtadt
    , hr
    , to_timestamp( mtadt::varchar || ' ' ||  hr, 'YYYY-MM-DD HH24') as mta_ts
from cls.mta;


mtadt         hr  mta_ts
----------  ----  -------------------------
2013-10-14    16  2013-10-14 16:00:00+00:00
2013-10-14    16  2013-10-14 16:00:00+00:00
2013-10-14    17  2013-10-14 17:00:00+00:00
2013-10-14    17  2013-10-14 17:00:00+00:00
2013-10-14    18  2013-10-14 18:00:00+00:00
[...]
```

This query returns three columns: the date, hour and then it creates a timestamp object using the command `to_timestamp`. This command takes in two strings. The first is a value to be converted and the second is the format of that conversion. In this example we create a synthetic string made up of the values of mtadt concatenated with a space and then the hour. This is passed to the command and is then converted to a timestamp.

3 Hard GROUP BY problems

In this section we will look at some difficult GROUP BY problems using the MTA data as well as the stocks data sets.

1. How many stocks (symbols) have 19 or more trading days for every month in 2010?

```
select
    count(1) as ct
from
    (select
        symb
        , count(1) as ct2
    from
        (select
            symb
            , date_part( 'month', retdate) as mn
            , count(1) as ct
        from
            stocks.s2010
        group by
            1,2) as innerQ
    where
        ct >= 19
    group by 1 ) as outerQ
where ct2 = 12;


  ct
----
3106
```

2. Write a query which returns 12 rows and two columns. The first column should be month as an integer and the second should be the number of trading days in that month. Do this for 2010 and remember that dates only appear in the stocks table if they are trading days.

```
select
    date_part('month', retdate) as mn
    , count( distinct retdate) as trading_days
from
    stocks.s2010
group by 1;


  mn    trading_days
----  --------------
   1              19
   2              19
   3              23
   4              21
   5              20
[...]
```

3. Create a table with the information above, this time in a wide format: one column per month with a single row.

```
select
        count( distinct case when date_part( 'month', retdate) = 1
                then retdate else null end) as Jan
        , count( distinct case when date_part( 'month', retdate) = 2
                then retdate else null end) as Feb
...[OTHER MONTHS OMITTED]
        , count( distinct case when date_part( 'month', retdate) = 12
                then retdate else null end) as Dec
from
        stocks.s2010;
```

4. Write a query which returns 12 rows and 3 columns from the 2010 data. The first column should be month as an integer, the second should be the number of unique stocks which had an open over $100 that month and the third should be the number of unique stocks with an open less than $50 that month.

```
select
    date_part('month', retdate) as mn
    , count( distinct case when opn > 100
        then symb else null end ) as over100
    , count( distinct case when opn < 50
        then symb else null end ) as less50
from
    stocks.s2010
group by 1;


  mn    over100    less50
----  ---------  --------
   1         68      2929
   2         58      2947
   3         65      2924
   4         71      2925
   5         68      2989
[...]
```

5. Repeat the above, but this time only include those stocks which are *also* in 2011.

```
select
    date_part('month', retdate) as mn
    , count( distinct case when opn > 100
        then symb else null end ) as over100
    , count( distinct case when opn < 50
        then symb else null end ) as less50
from
    stocks.s2010
where
    symb in (select distinct symb from stocks.s2011)
group by 1;


  mn    over100    less50
----  ---------  --------
   1         68      2904
   2         58      2924
   3         65      2901
   4         71      2903
   5         68      2969
[...]
```

6. We define the *yearly spread* as the difference between the maximum closing price for a stock and the minimum closing price for a stock over the year. Write a query which returns all stocks whose yearly spread in 2010 is less than $\frac{1}{2}$ the largest yearly spread (from all stocks) in 2011.

```
select
    symb
from
    (select max(cls) - min(cls) as ys2010
        , symb from stocks.s2010 group by 2) as IQ
where
    ys2010 < .5 * (select max(ys2011) as max_ys2011 from
        (select max(cls) - min(cls) as ys2011
            , symb from stocks.s2011 group by 2) as IQ2);


symb
------
A
AA
AAME
AAN
AAON
[...]
```

7. For stocks in 2010 return the following: (1) symbol, (2) month and (3) the difference between the maximum closing price and minimum closing price for each month. Only include those stocks which were traded more than 10 days that month.

```
select
    symb, mn, diff
from
    (select
        symb
        , date_part('month', retdate) as mn
        , max(cls) - min(cls) as diff
        , count(1) as ct
    from
        stocks.s2010
    group by 1,2 ) as innerQ
where
     ct > 10;


symb      mn    diff
------  ----  ------
A          1  2.339
A          2  1.7096
A          3  1.8097
A          4  2.382
A          5  4.0988
[...]
```

8. Return the data in the previous problem in a wide format – one column per month and one row per symbol.

```
select
        symb
        , sum( case when mn = 1 then diff else null end ) as Jan
        , sum( case when mn = 2 then diff else null end ) as Feb
[OTHER MONTHS OMITTED]
        , sum( case when mn = 12 then diff else null end ) as Dec
from
        (select
                symb
                , date_part('month', retdate) as mn
                , max(cls) - min(cls) as diff
                , count(1) as ct
        from
                stocks.2010
        group by 1,2) as innerQ
WHERE ct > 10
GROUP BY 1;
```