SQL FOR EXPLORATORY DATA ANALYSIS

Date/time types and formats

Christina Maimone
Data Scientist

Main types

`date`

- YYYY-MM-DD
- example: 2018-12-30

`timestamp`

- YYYY-MM-DD HH:MM:SS
- example: 2018-12-30 13:10:04.3

Intervals

`interval` examples:

```
6 days 01:48:08

00:51:03

1 day 21:57:47

07:48:46

406 days 00:31:56
```

Date/time format examples

1pm on January 10, 2018

01/10/18 1:00

10/01/18 01:00:00

01/10/2018 1pm

January 10th, 2018 1pm

10 Jan 2018 1:00

01/10/18 01:00:00

01/10/18 13:00:00

ISO 8601

ISO = International Organization for Standards

YYYY-MM-DD HH:MM:SS

Example: 2018-01-05 09:35:15

UTC and timezones

UTC = Coordinated Universal Time

Timestamp with timezone:

YYYY-MM-DD HH:MM:SS+HH

Example: 2004-10-19 10:23:54+02

Date and time comparisons

Compare with >, <, =

```sql
SELECT '2018-01-01' > '2017-12-31';
```

`now()`: current timestamp

```sql
SELECT now() > '2017-12-31';
```

Date subtraction

```
SELECT now() - '2018-01-01';
```

```
 343 days 21:26:32.710898
```

```
SELECT now() - '2015-01-01';
```

```
 1439 days 21:32:22.616076
```

Date addition

```
SELECT '2010-01-01'::date + 1;
```

```
 2010-01-02
```

```
SELECT '2018-12-10'::date + '1 year'::interval;
```

```
 2019-12-10 00:00:00
```

```
SELECT '2018-12-10'::date + '1 year 2 days 3 minutes'::interval ;
```

```
 2019-12-12 00:03:00
```

SQL FOR EXPLORATORY DATA ANALYSIS

Let's practice!

SQL FOR EXPLORATORY DATA ANALYSIS

Date/time components and aggregation

Christina Maimone
Data Scientist

Common date/time fields

Date/Time Functions and Operators Documentation

Fields

- century: 2019-01-01 = century 21
- decade: 2019-01-01 = decade 201
- year, month, day
- hour, minute, second
- week
- dow: day of week

Extracting fields

```
-- functions to extract datetime fields

date_part('field', timestamp)

EXTRACT(FIELD FROM timestamp)
```

```
-- now is 2019-01-08 22:15:10.647281-06

SELECT date_part('month', now()),
       EXTRACT(MONTH FROM now());
```

```
 date_part | date_part
-----------+-----------
         1 |         1
```

Extract to summarize by field

Individual sales

```
SELECT *
  FROM sales
 WHERE date >= '2010-01-01'
   AND date < '2017-01-01';
```

```
    date     | amt
-------------+------
 2010-01-05 |  32
 2010-03-04 |  10
 2010-07-29 |   7
 2016-12-20 |  46
 2015-11-01 |  30
 2014-07-10 |  35
 2013-09-09 |  15
 2011-04-22 |   9
...
```

By month

```
SELECT date_part('month', date)
         AS month,
       sum(amt)
  FROM sales
 GROUP BY month
 ORDER BY month;
```

```
 month     sum
-------+--------
     1 |   432      <- Jan
     2 |   847
     3 |  1987      <- Mar
     4 |  3899
     5 |   974
     6 |   397
     7 |   974
     8 |   198
...
```

Truncating dates

```
date_trunc('field', timestamp)
```

```
-- now() is  2018-12-17 21:45:15.6829-06

SELECT date_trunc('month', now());
```

```
       date_trunc
------------------------
 2018-12-01 00:00:00-06
```

Truncate to keep larger units

Individual sales

```
SELECT *
  FROM sales
 WHERE date >= '2017-06-01'
   AND date < '2019-02-01';
```

```
        date         | amt
---------------------+-----
 2019-01-25 06:58:32 |  10
 2018-03-14 18:07:03 |  11
 2018-04-15 19:03:33 |   5
 2018-08-10 14:30:49 |  28
 2018-02-02 09:52:28 |  17
 2017-08-08 08:40:35 |  20
 2018-07-05 02:05:52 |   7
 2018-07-28 17:49:16 |  16
 2018-08-01 20:57:40 |   8
 2018-03-04 08:56:04 |  32
 ...
```

By month with year

```
SELECT date_trunc('month', date)
         AS month
       sum(amt)
  FROM sales
 GROUP BY month
 ORDER BY month;
```

```
        month        |  sum
---------------------+------
 2017-06-01 00:00:00 |  594
 2017-07-01 00:00:00 | 3824
 2017-08-01 00:00:00 |  482
 2017-09-01 00:00:00 | 1384
 2017-10-01 00:00:00 | 3058
 2017-11-01 00:00:00 |  259
 2017-12-01 00:00:00 |  874
 2018-01-01 00:00:00 | 1225
...
```

SQL FOR EXPLORATORY DATA ANALYSIS

Time to practice extracting and aggregating dates

SQL FOR EXPLORATORY DATA ANALYSIS

Aggregating with date/time series

Christina Maimone
Data Scientist

Generate series

```
SELECT generate_series(from, to, interval);
```

```
SELECT generate_series('2018-01-01',
                       '2018-01-15',
                       '2 days'::interval);
```

```
   generate_series
---------------------
 2018-01-01 00:00:00
 2018-01-03 00:00:00
 2018-01-05 00:00:00
 2018-01-07 00:00:00
 2018-01-09 00:00:00
 2018-01-11 00:00:00
 2018-01-13 00:00:00
 2018-01-15 00:00:00
(8 rows)
```

Generate series

```
SELECT generate_series('2018-01-01',
                       '2018-01-02',
                       '5 hours'::interval);
```

```
    generate_series
----------------------
 2018-01-01 00:00:00
 2018-01-01 05:00:00
 2018-01-01 10:00:00
 2018-01-01 15:00:00
 2018-01-01 20:00:00
(5 rows)
```

Generate series from the beginning

```
SELECT generate_series('2018-01-31',
                       '2018-12-31',
                       '1 month'::interval);
```

```
   generate_series
---------------------
 2018-01-31 00:00:00
 2018-02-28 00:00:00
 2018-03-28 00:00:00
 2018-04-28 00:00:00
 2018-05-28 00:00:00
 2018-06-28 00:00:00
 2018-07-28 00:00:00
 2018-08-28 00:00:00
 2018-09-28 00:00:00
 2018-10-28 00:00:00
 2018-11-28 00:00:00
 2018-12-28 00:00:00
(12 rows)
```

Generate series from the beginning

```
-- Subtract 1 day to get end of month
SELECT generate_series('2018-02-01',  -- start 1 month late
                       '2019-01-01',
                       '1 month'::interval) - '1 day'::interval;
```

```
   generate_series
---------------------
 2018-01-31 00:00:00
 2018-02-28 00:00:00
 2018-03-31 00:00:00
 2018-04-30 00:00:00
 2018-05-31 00:00:00
 2018-06-30 00:00:00
 2018-07-31 00:00:00
 2018-08-31 00:00:00
 2018-09-30 00:00:00
 2018-10-31 00:00:00
 2018-11-30 00:00:00
 2018-12-31 00:00:00
(12 rows)
```

Normal aggregation

```
SELECT * FROM sales;
```

```
        date         | amount
---------------------+--------
 2018-04-23 09:13:14 |     12
 2018-04-23 13:57:53 |     41
 2018-04-23 12:05:44 |     23
 2018-04-23 09:07:33 |     31
 2018-04-23 10:31:40 |      5
 2018-04-23 09:35:16 |     18
 2018-04-23 12:17:43 |     19
 2018-04-23 12:57:49 |     32
 2018-04-23 10:12:35 |     13
 2018-04-23 13:21:30 |      6
(10 rows)
```

```
SELECT date_trunc('hour', date)
          AS hour,
       count(*)
  FROM sales
 GROUP BY hour
 ORDER BY hour;
```

```
        hour         | count
---------------------+-------
 2018-04-23 09:00:00 |     3
 2018-04-23 10:00:00 |     2
 2018-04-23 12:00:00 |     3
 2018-04-23 13:00:00 |     2
(4 rows)
```

Aggregation with series

```
-- Create the series as a table called hour_series
WITH hour_series AS (
     SELECT generate_series('2018-04-23 09:00:00',          -- 9am
                            '2018-04-23 14:00:00',          -- 2pm
                            '1 hour'::interval) AS hours)

-- Hours from series, count date (NOT *) to count non-NULL
SELECT hours, count(date)

  -- Join series to sales data
  FROM hour_series
       LEFT JOIN sales
              ON hours=date_trunc('hour', date)

 GROUP BY hours
 ORDER BY hours;
```

Aggregation with series: result

```
         hours          | count
------------------------+-------
 2018-04-23 09:00:00-05 |     3
 2018-04-23 10:00:00-05 |     2
 2018-04-23 11:00:00-05 |     0
 2018-04-23 12:00:00-05 |     3
 2018-04-23 13:00:00-05 |     2
 2018-04-23 14:00:00-05 |     0
(6 rows)
```

Aggregation with bins

```
-- Create bins
WITH bins AS (
      SELECT generate_series('2018-04-23 09:00:00',
                             '2018-04-23 15:00:00',
                             '3 hours'::interval) AS lower,
             generate_series('2018-04-23 12:00:00',
                             '2018-04-23 18:00:00',
                             '3 hours'::interval) AS upper)

-- Count values in each bin
SELECT lower, upper, count(date)

  -- left join keeps all bins
  FROM bins
       LEFT JOIN sales
              ON date >= lower
             AND date < upper

 -- Group by bin bounds to create the groups
 GROUP BY lower, upper
 ORDER BY lower;
```

Bin result

```
        lower        |        upper        | count
---------------------+---------------------+-------
 2018-04-23 09:00:00 | 2018-04-23 12:00:00 |     5
 2018-04-23 12:00:00 | 2018-04-23 15:00:00 |     5
 2018-04-23 15:00:00 | 2018-04-23 18:00:00 |     0
(3 rows)
```

SQL FOR EXPLORATORY DATA ANALYSIS

Practice generating series!

SQL FOR EXPLORATORY DATA ANALYSIS

Time between events

Christina Maimone
Data Scientist

The problem

```
SELECT *
  FROM sales
 ORDER BY date;
```

```
        date         | amount
---------------------+--------
 2018-04-23 09:07:33 |     31
 2018-04-23 09:13:14 |     12
 2018-04-23 09:35:16 |     18
 2018-04-23 10:12:35 |     13
 2018-04-23 10:31:40 |      5
 2018-04-23 12:05:44 |     23
 2018-04-23 12:17:43 |     19
 2018-04-23 12:57:49 |     32
 2018-04-23 13:21:30 |      6
 2018-04-23 13:57:53 |     41
(10 rows)
```

Lead and lag

```
SELECT date,
       lag(date) OVER (ORDER BY date),
       lead(date) OVER (ORDER BY date)
  FROM sales;
```

```
        date         |         lag         |        lead
---------------------+---------------------+---------------------
 2018-04-23 09:07:33 |                     | 2018-04-23 09:13:14
 2018-04-23 09:13:14 | 2018-04-23 09:07:33 | 2018-04-23 09:35:16
 2018-04-23 09:35:16 | 2018-04-23 09:13:14 | 2018-04-23 10:12:35
 2018-04-23 10:12:35 | 2018-04-23 09:35:16 | 2018-04-23 10:31:40
 2018-04-23 10:31:40 | 2018-04-23 10:12:35 | 2018-04-23 12:05:44
 2018-04-23 12:05:44 | 2018-04-23 10:31:40 | 2018-04-23 12:17:43
 2018-04-23 12:17:43 | 2018-04-23 12:05:44 | 2018-04-23 12:57:49
 2018-04-23 12:57:49 | 2018-04-23 12:17:43 | 2018-04-23 13:21:30
 2018-04-23 13:21:30 | 2018-04-23 12:57:49 | 2018-04-23 13:57:53
 2018-04-23 13:57:53 | 2018-04-23 13:21:30 |
(10 rows)
```

Lead and lag

```
SELECT date,
       lag(date) OVER (ORDER BY date),
       lead(date) OVER (ORDER BY date)
  FROM sales;
```

Time between events

```
SELECT date,
       date - lag(date) OVER (ORDER BY date) AS gap
  FROM sales;
```

```
        date         |    gap
---------------------+----------
 2018-04-23 09:07:33 |
 2018-04-23 09:13:14 | 00:05:41
 2018-04-23 09:35:16 | 00:22:02
 2018-04-23 10:12:35 | 00:37:19
 2018-04-23 10:31:40 | 00:19:05
 2018-04-23 12:05:44 | 01:34:04
 2018-04-23 12:17:43 | 00:11:59
 2018-04-23 12:57:49 | 00:40:06
 2018-04-23 13:21:30 | 00:23:41
 2018-04-23 13:57:53 | 00:36:23
(10 rows)
```

Average time between events

```
SELECT avg(gap)
   FROM (SELECT date - lag(date) OVER (ORDER BY date) AS gap
           FROM sales) AS gaps;
```

```
        avg
-----------------
 00:32:15.555556
(1 row)
```

Change in a time series

```
SELECT date,
       amount,
       lag(amount) OVER (ORDER BY date),
       amount - lag(amount) OVER (ORDER BY date) AS change
  FROM sales;
```

```
        date         | amount | lag | change
---------------------+--------+-----+--------
 2018-04-23 09:07:33 |     31 |     |
 2018-04-23 09:13:14 |     12 |  31 |    -19
 2018-04-23 09:35:16 |     18 |  12 |      6
 2018-04-23 10:12:35 |     13 |  18 |     -5
 2018-04-23 10:31:40 |      5 |  13 |     -8
 2018-04-23 12:05:44 |     23 |   5 |     18
 2018-04-23 12:17:43 |     19 |  23 |     -4
 2018-04-23 12:57:49 |     32 |  19 |     13
 2018-04-23 13:21:30 |      6 |  32 |    -26
 2018-04-23 13:57:53 |     41 |   6 |     35
(10 rows)
```

SQL FOR EXPLORATORY DATA ANALYSIS

On to the exercises!

SQL FOR EXPLORATORY DATA ANALYSIS

Wrap-up

Christina Maimone
Data Scientist

Download the data

Links on the course landing page!

Parting tips

- Spend time exploring your data
- Use the PostgreSQL documentation
- Be curious
- Check data distributions first

SQL FOR EXPLORATORY DATA ANALYSIS

Start exploring!