Survey of useful functions

ANALYZING BUSINESS DATA IN SQL

Michel Semaan
Data Scientist

datacamp

Dealing with dates

- `DATE_TRUNC('quarter', '2018-08-13')` ? `'2018-07-01 00:00:00+00:00'`
- `'2018-07-01 00:00:00+00:00' :: DATE` ? `'2018-07-01'`

Dates in reports

- Human-readable dates are important in reporting
- The default date format, `'2018-08-13'` , isn't very readable
- How do you get from `'2018-08-13'` to `'Friday 13, August 2018'` ?

Solution

- `TO_CHAR('2018-08-13', 'FMDay DD, FMMonth YYYY')` ? `'Friday 13, August 2018'`

TO_CHAR()

- `TO_CHAR(DATE, TEXT) ? TEXT` (the formatted date string)
- **Example:** `Dy` ? Abbreviated day name (`Mon`, `Tues`, *etc.*)
  - `TO_CHAR('2018-06-01', 'Dy') ? 'Fri'`
  - `TO_CHAR('2018-06-02', 'Dy') ? 'Sat'`
- Patterns in the format string will be replaced by what they represent in the date; other characters remain as-is
- **Example:** `DD` ? Day number (`01` - `31`)
  - `TO_CHAR('2018-06-01', 'Dy - DD') ? 'Fri - 01'`
  - `TO_CHAR('2018-06-02', 'Dy - DD') ? 'Sat - 02'`

Pattern	Description
`FMDay`	Full day name (`Monday`, `Tuesday`, *etc.*)
`MM`	Month of year (`01` - `12`)
`Mon`	Abbreviated month name (`Jan`, `Feb`, *etc.*)
`FMMonth`	Full month name (`January`, `February`, *etc.*)
`YY`	Last 2 digits of year (`18`, `19`, *etc.*)
`YYYY`	Full 4-digit year (`2018`, `2019`, *etc.*)

Documentation: https://www.postgresql.org/docs/9.6/functions-formatting.html

Query

```
SELECT DISTINCT
  order_date,
  TO_CHAR(order_date,
          'FMDay DD, FMMonth YYYY') AS format_1,
  TO_CHAR(order_date,
          'Dy DD Mon/YYYY') AS format_2
FROM orders
ORDER BY order_date ASC
LIMIT 3;
```

Result

```
order_date  format_1                format_2
----------  ----------------------  --------------
2018-06-01  Friday 01, June 2018    Fri 01/Jun 2018
2018-06-02  Saturday 02, June 2018  Sat 02/Jun 2018
2018-06-02  Sunday 03, June 2018    Sun 03/Jun 2018
```

Window functions revisited

- `SUM(...) OVER (...)` : Calculates a column's running total
  - **Example:** `SUM(registrations) OVER (ORDER BY registration_month)` calculates the registrations running total
- `LAG(...) OVER (...)` : Fetches a preceding row's value
  - **Example:** `LAG(mau) OVER (ORDER BY active_month)` returns the previous month's active users (MAU)
- `RANK() OVER (...)` : Assigns a rank to each row based on that row's position in a sorted order
  - **Example:** `RANK() OVER (ORDER BY revenue DESC)` ranks users, eateries, or months by the revenue they've generated

Query

```
SELECT
  user_id,
  SUM(meal_price * order_quantity) AS revenue
FROM meals
JOIN orders ON meals.meal_id = orders.meal_id
GROUP BY user_id
ORDER BY revenue DESC
LIMIT 3;
```

Result

```
user_id  revenue
-------  -------
18       626
76       553.25
73       537
```

Query

```
WITH user_revenues AS (
  SELECT
    user_id,
    SUM(meal_price * order_quantity) AS revenue
  FROM meals
  JOIN orders ON meals.meal_id = orders.meal_id
  GROUP BY user_id)

SELECT
  user_id,
  RANK() OVER (ORDER BY revenue DESC)
    AS revenue_rank
FROM user_revenues
ORDER BY revenue_rank DESC
LIMIT 3;
```

Result

```
user_id  revenue_rank
-------  ------------
18       1
76       2
73       3
```

Survey of useful functions

ANALYZING BUSINESS DATA IN SQL

Pivoting

ANALYZING BUSINESS DATA IN SQL

Michel Semaan
Data Scientist

datacamp

What is pivoting?

- **Pivoting**: Rotating a table around a pivot column; transposing a column's values into columns
  - Converts a "long" table into a "wide" one

Benefits

- Control a table's shape while preserving its data
- Unstacked data viewed horizontally is often easier to read than stacked data viewed vertically

Before

```
meal_id  delivr_month  count_orders
-------  ------------  ------------
0        2018-06-01    39
0        2018-07-01    47
1        2018-06-01    25
1        2018-07-01    55
```

After

```
meal_id  2018-06-01  2018-07-01
-------  ----------  ----------
0        39          47
1        25          55
```

- Pivoted by `delivr_month`

Before table

```
SELECT
  meal_id,
  DATE_TRUNC('month', order_date) :: DATE AS delivr_month,
  COUNT(DISTINCT orders) :: INT AS revenue
FROM meals
JOIN orders ON meals.meal_id = orders.meal_id
WHERE meals.meal_id IN (0, 1)
  AND order_date < '2018-08-01'
GROUP BY meal_id, delivr_month
ORDER BY meal_id, delivr_month;
```

CROSSTAB()

```
CREATE EXTENSION IF NOT EXISTS tablefunc;
```

- `CREATE EXTENSION` is like `import` in Python

```
SELECT * FROM CROSSTAB($$
  TEXT source_sql
$$)

AS ct (column_1 DATA_TYPE_1,
       column_2 DATA_TYPE_2,
       ...,
       column_n DATA_TYPE_N)
;
```

Using CROSSTAB()

```
CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT * FROM CROSSTAB($$
  SELECT
    meal_id,
    DATE_TRUNC('month', order_date) :: DATE AS delivr_month,
    COUNT(DISTINCT order_id) :: INT AS orders
  FROM orders
  WHERE meal_id IN (0, 1)
    AND order_date < '2018-08-01'
  GROUP BY meal_id, delivr_month
  ORDER BY meal_id, delivr_month $$)
  AS ct (meal_id INT,
         "2018-06-01" INT,
         "2018-07-01" INT)
ORDER BY meal_id ASC;
```

Before table

```
meal_id  delivr_month  count_orders
-------  ------------  ------------
0        2018-06-01    39
0        2018-07-01    47
1        2018-06-01    25
1        2018-07-01    55
```

After table

```
meal_id  2018-06-01  2018-07-01
-------  ----------  ----------
0        39          47
1        25          55
```

- Pivoted by `delivr_month`

Pivoting

ANALYZING BUSINESS DATA IN SQL

Producing executive reports

ANALYZING BUSINESS DATA IN SQL

Michel Semaan
Data Scientist

datacamp

Readability

- **Dates**: Use readable date formats (`August 2018` , not `2018-08-01`)
- **Numbers**: Round numbers to the second decimal at most (`98.76` , not `98.761234`)
- **Table shape**: Reshape long tables into wide ones, pivoting by date when possible
- **Order**: Don't forget to sort!

Executive report - query

Query

```
SELECT
  eatery,
  TO_CHAR(order_date, 'MM-Mon YYYY') AS delivr_month,
  COUNT(DISTINCT order_id) AS count_orders
FROM meals
JOIN orders ON meals.meal_id = orders.meal_id
WHERE order_date >= '2018-10-01'
GROUP BY eatery, delivr_month
ORDER BY eatery, delivr_month;
```

Result

```
eatery              delivr_month  count_orders
-----------------   ------------  ------------
Bean Me Up Scotty   10-Oct 2018   709
Bean Me Up Scotty   11-Nov 2018   1143
Bean Me Up Scotty   12-Dec 2018   2168
Burgatorio          10-Oct 2018   679
...                 ...           ...
```

Executive report (II)

```
WITH eatery_orders AS (
  SELECT
    eatery,
    TO_CHAR(order_date, 'MM-Mon YYYY') AS delivr_month,
    COUNT(DISTINCT order_id) AS count_orders
  FROM meals
  WHERE order_date >= '2018-10-01'
  JOIN orders ON meals.meal_id = orders.meal_id
  GROUP BY eatery, delivr_month)

SELECT
  eatery,
  delivr_month,
  RANK() OVER
    (PARTITION BY delivr_month
     ORDER BY count_orders DESC) :: INT AS orders_rank
FROM eatery_orders
ORDER BY eatery, delivr_month;
```

```
eatery               delivr_month  orders_rank
-----------------  ------------  -----------
Bean Me Up Scotty  10-Oct 2018     2
Bean Me Up Scotty  11-Nov 2018     4
Bean Me Up Scotty  12-Dec 2018   2
Burgatorio         10-Oct 2018     4
...                ...           ...
```

```
CREATE EXTENSION IF NOT EXISTS tablefunc;

SELECT * FROM CROSSTAB($$
  ...
$$) AS ct (eatery TEXT,
           "10-Oct 2018" INT,
           "11-Nov 2018" INT,
           "12-Dec 2018" INT)
ORDER BY eatery ASC;
```

Executive report - result

eatery	Q2 2018	Q3 2018	Q4 2018
The Moon Wok	1	1	1
Bean Me Up Scotty	3	2	2
Leaning Tower of Pizza	4	4	3
Burgatorio	2	3	4
Life of Pie	5	5	5

Producing executive reports

ANALYZING BUSINESS DATA IN SQL

Course recap

ANALYZING BUSINESS DATA IN SQL

Michel Semaan
Data Scientist

datacamp

Course recap

- **Chapter 1**: Revenue, cost, and profit
- **Chapter 2**: User-centric metrics
- **Chapter 3**: Unit economics and distributions
- **Chapter 4**: Generating an executive report

To infinity, and beyond!

ANALYZING BUSINESS DATA IN SQL