Week 6: Data Cleaning and Window Functions in SQL

Parch and Posey Database

Data Cleaning

LEFT and RIGHT

1. In the `accounts` table, there is a column holding the website for each company. The last three digits specify what type of web address they are using. Pull these extensions and provide how many of each website type exist in the accounts table.

```
select distinct right(website, 3) as domain, count(*) num_companies
from accounts
group by 1
```

domain	num_companies
com	349
net	1
org	1

2. There is much debate about how much the name (or even the first letter of a company name) matters. Use the accounts table to pull the first letter of each company name to see the distribution of company names that begin with each letter (or number).

```
select upper(left(name, 1)) as letter, count(*)
from accounts
group by 1
order by 1
```

letter	count
3	1
A	37
B	16
C	37
D	17
E	16
F	12
G	14
H	15
I	7
J	7
K	7
L	16
M	22
N	15
O	7
P	27
Q	1
R	8
S	17
T	17
U	13
V	7
W	12
X	2
Y	1

3. What is the number of company names that start with a vowel and consonant letters?

```
with t1 as (
    select left(name, 1) as first_letter,
            CASE WHEN lower(left(name, 1)) in ('a', 'e', 'i', 'o', 'u') then 'vowel' ELSE 'consonant'
            END as type
    from accounts
)
select type, count(*)
from t1
group by type;
```

type	count
vowel	80
consonant	271

POSITION

4. Use the `accounts` table to create first and last name columns that hold the first and last names for the `primary_poc`.

```
select primary_poc,
       left(primary_poc, position(' ' in primary_poc) -1 ) as firstname,
       right(primary_poc, length(primary_poc) - position(' ' in primary_poc) ) as lastname
```

```
from accounts
limit 5
```

primary_poc	firstname	lastname
Tamara Tuma	Tamara	Tuma
Sung Shields	Sung	Shields
Jodee Lupo	Jodee	Lupo
Serafina Banda	Serafina	Banda
Angeles Crusoe	Angeles	Crusoe

CONCAT

5. Each company in the `accounts` table wants to create an email address for each `primary_poc`. The email address should be the first name of the `primary_poc` last name `primary_poc` @ company name .com. (e.g. `tamara.tuma@walmart.com`)

```
with t1 as (
  select primary_poc,
    left(primary_poc, position(' ' in primary_poc) -1 ) as firstname,
    right(primary_poc, length(primary_poc) - position(' ' in primary_poc)) as lastname,
    name
  from accounts
    )
select lower(concat(firstname, '.', lastname, '@', name, 'com'))
from t1
limit 5
```

lower
tamara.tuma@walmartcom
sung.shields@exxon mobilcom
jodee.lupo@applecom
serafina.banda@berkshire hathawaycom
angeles.crusoe@mckessoncom

6. You may have noticed that in the previous solution some of the company names include spaces, which will certainly not work in an email address. See if you can create an email address that will work by removing all of the spaces in the account name, but otherwise your solution should be just as in the previous question.

```
with t1 as (
  select primary_poc,
    left(primary_poc, position(' ' in primary_poc) -1 ) as firstname,
    right(primary_poc, length(primary_poc) - position(' ' in primary_poc)) as lastname,
    replace(name, ' ', '') as company
    from accounts
    )
select lower(concat(firstname, '.', lastname, '@', company, 'com'))
from t1
limit 5
```

lower
tamara.tuma@walmartcom
sung.shields@exxonmobilcom
jodee.lupo@applecom
serafina.banda@berkshirehathawaycom
angeles.crusoe@mckessoncom

7. We would also like to create an initial password, which they will change after their first log in. The password will be a combination of:

- the first letter of the `primary_poc`'s first name (lowercase),
- the last letter of their first name (lowercase),
- the first letter of their last name (uppercase),
- the last letter of their last name (uppercase),
- the number of letters in their first name,
- the number of letters in their last name, and
- the name of the company they are working with, no spaces
- the forth and fifth digit of their sales rep id

```
with t1 as(
    select replace(name, ' ', '') as company, primary_poc,
           left(primary_poc, position(' ' in primary_poc) -1) as firstname,
           right(primary_poc, length(primary_poc) - position(' ' in primary_poc)) as lastname,
           sales_rep_id
    from accounts
)
select firstname, lastname, company, sales_rep_id,
       concat(
           lower(left(firstname, 1)),
           lower(right(firstname, 1)),
           upper(left(lastname, 1)),
           upper(right(lastname, 1)),
           lower(company),
           substr(sales_rep_id::text, 4,2) -- substr(text, start_pos, num_of_characters)
           ) as password
from t1
limit 5
```

firstname	lastname	company	sales_rep_id	password
Tamara	Tuma	Walmart	321500	taTAwalmart50
Sung	Shields	ExxonMobil	321510	sgSSexxonmobil51
Jodee	Lupo	Apple	321520	jeLOapple52
Serafina	Banda	BerkshireHathaway	321530	saBAberkshirehathaway53
Angeles	Crusoe	McKesson	321540	asCEmckesson54

Window Functions

PostgreSQL's documentation does an excellent job of introducing the concept of Window Functions:

"A window function performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single

output row — the rows retain their separate identities. Behind the scenes, the window function is able to access more than just the current row of the query result."

8. For the orders table, create a new column which shows the total number of transactions for all accounts.

```
select *,
       sum(total_amt_usd) over() as over_all_total
from orders
limit 10
```

9. Update the previous query to create two new column: (1) over_all_total_by_account_id, and (2) overall_count_by_account_id (without using Group By).

```
select *,
       sum(total_amt_usd) over(partition by account_id) as over_all_total_by_id,
       count(total_amt_usd) over(partition by account_id) as count_by_id,
       sum(total_amt_usd) over() as over_all_total
from orders
limit 10
```

id	account_id	occurred_at	standard_qty	gloss_qty	poster_qty	total	standard_amt_us
14	1001	2016-10-26 20:31:30	97	143	54	294	484.
4318	1001	2016-11-25 23:19:37	485	543	177	1205	2420.
4317	1001	2016-09-26 23:22:47	507	614	226	1347	2529.
4316	1001	2016-08-28 06:50:58	557	572	255	1384	2779.
4315	1001	2016-07-30 03:21:57	457	532	249	1238	2280.
4314	1001	2016-05-31 21:09:48	531	603	209	1343	2649.
4313	1001	2016-05-01 15:40:04	483	570	201	1254	2410.
4312	1001	2016-04-01 11:15:27	497	618	152	1267	2480.
4311	1001	2016-03-02 15:40:29	498	605	204	1307	2485.
4310	1001	2016-02-01 19:07:32	473	595	212	1280	2360.

10. Create a running total of `standard_amt_usd` (in the orders table) over order time.

```
SELECT occurred_at, standard_amt_usd,
       SUM(standard_amt_usd) OVER (ORDER BY occurred_at) AS running_total
FROM orders
limit 10
```

occurred_at	standard_amt_usd	running_total
2013-12-04 04:22:44	0.00	0.00
2013-12-04 04:45:54	2445.10	2445.10
2013-12-04 04:53:25	2634.72	5079.82
2013-12-05 20:29:16	0.00	5079.82
2013-12-05 20:33:56	2455.08	7534.90
2013-12-06 02:13:20	2504.98	10039.88
2013-12-06 12:55:22	264.47	10304.35
2013-12-06 12:57:41	1536.92	11841.27
2013-12-06 13:14:47	374.25	12215.52
2013-12-06 13:17:25	1402.19	13617.71

11. Create a running total of `standard_amt_usd` (in the orders table) over order time for each month.

```
SELECT standard_amt_usd, date_trunc('month', occurred_at),
       SUM(standard_amt_usd) OVER (PARTITION BY date_trunc('month', occurred_at)
                                   ORDER BY occurred_at) AS running_total
```

```
FROM orders
limit 10
```

standard_amt_usd	date_trunc	running_total
0.00	2013-12-01	0.00
2445.10	2013-12-01	2445.10
2634.72	2013-12-01	5079.82
0.00	2013-12-01	5079.82
2455.08	2013-12-01	7534.90
2504.98	2013-12-01	10039.88
264.47	2013-12-01	10304.35
1536.92	2013-12-01	11841.27
374.25	2013-12-01	12215.52
1402.19	2013-12-01	13617.71

12. YOUR TURN - Create a running total of `standard_qty` (in the orders table) over order time for each year.

```
select standard_qty, date_trunc('year', occurred_at) as month,
        sum(standard_qty) over(partition by date_trunc('year', occurred_at) order by occurred_at) as ru
from orders
limit 10
```

standard_qty	month	running_total
0	2013-01-01	0
490	2013-01-01	490
528	2013-01-01	1018
0	2013-01-01	1018
492	2013-01-01	1510
502	2013-01-01	2012
53	2013-01-01	2065
308	2013-01-01	2373
75	2013-01-01	2448
281	2013-01-01	2729

Ranking data: RAW_NUMBER() and RANK(), DENSE_RANK()

`ROW_NUMBER()` does just what it sounds like— displays the number of a given row. It starts at 1 and numbers the rows according to the ORDER BY part of the window statement. ROW_NUMBER() does not require you to specify a variable within the parentheses. Using the PARTITION BY clause will allow you to begin counting 1 again in each partition.

13. For account with id 1001, use the `row_number()`, `rank()` and `dense_rank()` to rank the transactions by the number of standard paper purchased.

```
select standard_qty,
       row_number() over(order by standard_qty),
       rank() over(order by standard_qty),
       dense_rank()  over(order by standard_qty)
from orders
where account_id = 1001
limit 10
```

standard_qty	row_number	rank	dense_rank
85	1	1	1
91	2	2	2
94	3	3	3
95	4	4	4
97	5	5	5
101	6	6	6
101	7	6	6
103	8	8	7
104	9	9	8
108	10	10	9

14. For each account, use the `row_number()`, `rank()` and `dense_rank()` to rank the transactions by the number of standard paper purchased.

```
select account_id, standard_qty,
       row_number() over(partition by account_id order by standard_qty),
       rank() over(partition by account_id order by standard_qty),
       dense_rank()  over(partition by account_id order by standard_qty)
from orders
limit 10
```

account_id	standard_qty	row_number	rank	dense_rank
1001	85	1	1	1
1001	91	2	2	2
1001	94	3	3	3
1001	95	4	4	4
1001	97	5	5	5
1001	101	6	6	6
1001	101	7	6	6
1001	103	8	8	7
1001	104	9	9	8
1001	108	10	10	9

15. **Your Turn** Select the id, account_id, and standard_qty variable from the orders table, then create a column called dense_rank that ranks this standard_qty amount of paper for each account. In addition, create a sum_std_qty which gives you the running total for account. Repeat the last task to get the avg, min, and max.

```
select id, account_id, standard_qty,
       dense_rank() over(partition by account_id order by standard_qty) as myrank,
       sum(standard_qty) over(partition by account_id order by standard_qty) as total_standard_qty,
       avg(standard_qty) over(partition by account_id order by standard_qty) as total_standard_qty,
       min(standard_qty) over(partition by account_id order by standard_qty) as total_standard_qty,
       max(standard_qty) over(partition by account_id order by standard_qty) as total_standard_qty
from orders
limit 10
```

id	account_id	standard_qty	myrank	total_standard_qty	total_standard_qty	total_standard_qty	total_standard_qty
3	1001	85	1	85	85.00000	85	85
9	1001	91	2	176	88.00000	85	91
10	1001	94	3	270	90.00000	85	94
8	1001	95	4	365	91.25000	85	95
14	1001	97	5	462	92.40000	85	97
11	1001	101	6	664	94.85714	85	101
7	1001	101	6	664	94.85714	85	101
6	1001	103	7	767	95.87500	85	103
13	1001	104	8	871	96.77778	85	104
5	1001	108	9	979	97.90000	85	108

16. Give an allias for the window function in the previous question, and call it `account_window`.

```
select id, account_id, standard_qty,
     dense_rank() over(partition by account_id order by standard_qty) as myrank,
       sum(standard_qty) over account_window as total_standard_qty,
       avg(standard_qty) over account_window as total_standard_qty,
       min(standard_qty) over account_window as total_standard_qty,
       max(standard_qty) over account_window as total_standard_qty
from orders
window account_window as (partition by account_id order by standard_qty)
limit 10
```

id	account_id	standard_qty	myrank	total_standard_qty	total_standard_qty	total_standard_qty	total_standard_qty
3	1001	85	1	85	85.00000	85	85
9	1001	91	2	176	88.00000	85	91
10	1001	94	3	270	90.00000	85	94
8	1001	95	4	365	91.25000	85	95
14	1001	97	5	462	92.40000	85	97
11	1001	101	6	664	94.85714	85	101
7	1001	101	6	664	94.85714	85	101
6	1001	103	7	767	95.87500	85	103
13	1001	104	8	871	96.77778	85	104
5	1001	108	9	979	97.90000	85	108

Percentiles

You can use window functions to identify what percentile (or quartile, or any other subdivision) a given row falls into. The syntax is NTILE(*# of buckets*). In this case, ORDER BY determines which column to use to determine the quartiles (or whatever number of tiles you specify).

In cases with relatively few rows in a window, the NTILE function doesn't calculate exactly as you might expect. For example, If you only had two records and you were measuring percentiles, you'd expect one record to define the 1st percentile, and the other record to define the 100th percentile. Using the NTILE function, what you'd actually see is one record in the 1st percentile, and one in the 2nd percentile.

In other words, when you use a NTILE function but the number of rows in the partition is less than the NTILE(number of groups), then NTILE will divide the rows into as many groups as there are members (rows) in the set but then stop short of the requested number of groups. If you're working with very small windows, keep this in mind and consider using quartiles or similarly small bands.

17. Use the NTILE functionality to divide the accounts into 4 levels in terms of the amount of standard_qty for their orders. Your resulting table should have the account_id, the occurred_at time for each order, the total amount of standard_qty paper purchased, and one of four levels in a standard_quartile column.

```
SELECT id,
       account_id,
       occurred_at,
       standard_qty,
```

```
       NTILE(4) OVER (PARTITION BY account_id ORDER BY standard_qty) AS standard_quartile
  FROM orders
 ORDER BY account_id DESC
limit 10
```

id	account_id	occurred_at	standard_qty	standard_quartile
6912	4501	2016-12-21 13:30:42	61	2
4301	4501	2016-07-29 20:06:39	111	3
6908	4501	2016-06-29 04:03:39	11	1
6910	4501	2016-08-27 00:58:11	16	2
6911	4501	2016-11-22 06:52:22	63	2
4300	4501	2016-06-29 03:57:11	104	3
6909	4501	2016-07-29 19:58:32	5	1
4305	4501	2016-11-22 06:57:04	6	1
4299	4501	2016-05-30 04:18:34	15	1
4306	4501	2016-12-21 13:43:26	126	3

18. **YOUR TURN** Use the NTILE functionality to divide the accounts into two levels in terms of the amount of gloss_qty for their orders. Your resulting table should have the account_id, the occurred_at time for each order, the total amount of gloss_qty paper purchased, and one of two levels in a gloss_half column.

```
SELECT id,
       account_id,
       occurred_at,
       gloss_qty,
       NTILE(2) OVER (PARTITION BY account_id ORDER BY gloss_qty) AS gloss_half
  FROM orders
 ORDER BY account_id DESC
limit 10
```

id	account_id	occurred_at	gloss_qty	gloss_half
4299	4501	2016-05-30 04:18:34	11	1
6911	4501	2016-11-22 06:52:22	67	2
4303	4501	2016-09-25 01:44:03	0	1
4302	4501	2016-08-27 00:48:17	11	1
4300	4501	2016-06-29 03:57:11	14	1
4301	4501	2016-07-29 20:06:39	16	2
4305	4501	2016-11-22 06:57:04	0	1
4306	4501	2016-12-21 13:43:26	0	1
4304	4501	2016-10-24 08:50:37	6	1
6909	4501	2016-07-29 19:58:32	91	2