Week 1 Lab: Introduction to SQL

Parch and Posey Database

Contents

Parch & Posey is a fabricated non-real company that sells paper:

- There are **50 sales reps** spread across the United States in 4 regions.
- There are **3 types of paper**. Regular, Poster and Glossy.
- The clients are primarily large Fortune's 100 companies whom are attracted by Google, Facebook and Twitter.

Questions answered using Parch & Posey data are meant to simulate real word problems. Using SQL, we will help Parch & Posey answer tricky questions like: Which of their product lines is worst performing? Which of their market channels they should make a great investment in.

In the Parch & Posey database there are five tables (essentially 5 spreadsheets):

- web_events
- accounts
- orders
- sales_reps
- region

Figure 1 shows the ERD (entity relationship diagram) for Parch and Posey.

Creating the Database

Note: you only need to create the database if you are installing PostgreSQL locally (ie. you are running your own database on your own computer). If you are accessing the remote database (`database.ychennay.com`), these tables are already set up for you. - Open pgAdmin 4 or any SQL editor of your choice. - Create a new database, and call it `parch_and_posey_example`. - Right click on the Parch database, and choose "Query Tool. . . " - Load the file "parch.sql" and run all the commands (We will discuss these commands later during the semester, but for now, just think of this step as a database creation and connection) - Right click on the parch_and_posey_example database, and choose "Query Tool. . . " to create an empty SQL file.

SELECT & FROM

1. Take a quick look at all the 5 tables in the database, and return all the columns and the first five rows of each table.

Solution #Q1

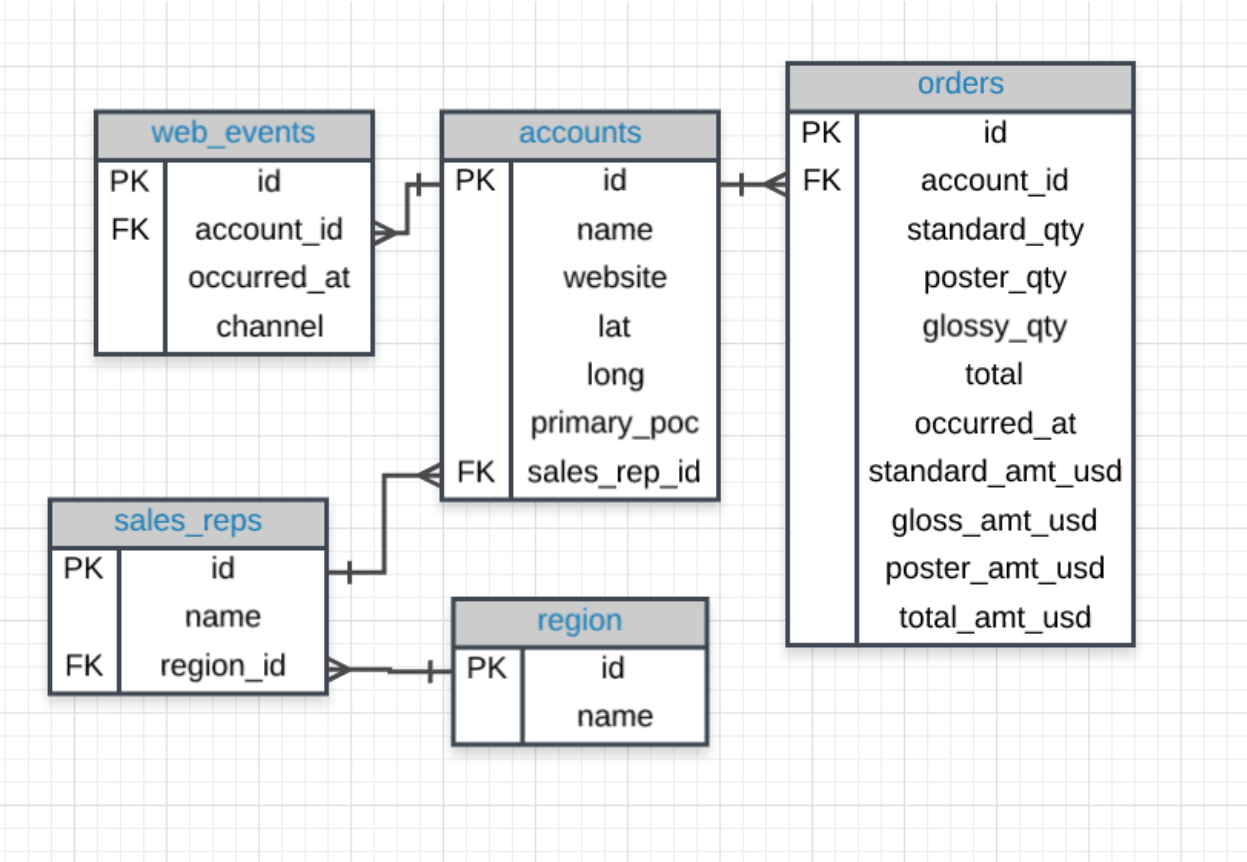

Figure 1: Parch and Posey ERD

```sql
SELECT *
FROM sales_reps
LIMIT 5; -- return only the first 5 rows in this table
```

id	name	region_id
321500	Samuel Racine	1
321510	Eugena Esser	1
321520	Michel Averette	1
321530	Renetta Carew	1
321540	Cara Clarke	1

```sql
SELECT *
FROM orders
LIMIT 5;
```

id	account_id	occurred_at	standard_qty	gloss_qty	poster_qty	total	standard_amt_usd	gloss_amt_usd	poster_amt_usd	total_amt_usd
1	1001	2015-10-06 17:31:14	123	22	24	169	613.77	164.78	194.88	973.43
2	1001	2015-11-05 03:34:33	190	41	57	288	948.10	307.09	462.84	1718.03
3	1001	2015-12-04 04:21:55	85	47	0	132	424.15	352.03	0.00	776.18
4	1001	2016-01-02 01:18:24	144	32	0	176	718.56	239.68	0.00	958.24
5	1001	2016-02-01 19:27:27	108	29	28	165	538.92	217.21	227.36	983.49

```sql
SELECT *
FROM accounts
LIMIT 5;
```

id	name	website	lat	long	primary_poc	sales_rep_id
1001	Walmart	www.walmart.com	40.23850	-75.10330	Tamara Tuma	321500
1011	Exxon Mobil	www.exxonmobil.com	41.16916	-73.84937	Sung Shields	321510
1021	Apple	www.apple.com	42.29049	-76.08401	Jodee Lupo	321520
1031	Berkshire Hathaway	www.berkshirehathaway.com	40.94902	-75.76390	Serafina Banda	321530
1041	McKesson	www.mckesson.com	42.21709	-75.28500	Angeles Crusoe	321540

```sql
SELECT *
FROM region
LIMIT 5;
```

id	name
1	Northeast
2	Midwest
3	Southeast
4	West

```sql
SELECT *
FROM web_events
LIMIT 5;
```

id	account_id	occurred_at	channel
1	1001	2015-10-06 17:13:58	direct
2	1001	2015-11-05 03:08:26	direct
3	1001	2015-12-04 03:57:24	direct
4	1001	2016-01-02 00:55:03	direct
5	1001	2016-02-01 19:02:33	direct

2. Generate a list of all the names of the sales representatives and their IDs that Parch and Posey has in their database.

To answer this question, we will use a SELECT statement. The SELECT statement is used to query the database and retrieve selected data that match the criteria that you specify. In this case, we are looking to retrieve only the sales reps `name` and `id`, so we just choose these two columns in the SELECT statement.

The FROM statement is where you tell the query what table you are querying from. In this case, we are looking to get the information from the `sales_reps` table in the database.

```
SELECT id, name
FROM sales_reps
LIMIT 5;
```

The following output only shows the first few records on the output.

id	name
321500	Samuel Racine
321510	Eugena Esser
321520	Michel Averette
321530	Renetta Carew
321540	Cara Clarke

Sometimes, we would like to retrieve all the columns that we have in a specific table. Instead of listing all the columns in the SELECT statement, we can use the asterisk symbol (*).

3. Generate a list of all the order IDs that Parch and Posey has in their database. Provide only the order ID and when they occurred.

```
SELECT id, occurred_at
FROM orders
LIMIT 5;
```

4. **YOUR TURN** Write your own query to select only the `id`, `account_id`, and `occurred_at` columns for all orders in the `orders` table.

Sometimes, and for the purpose of viewing the results, we might decide to limit our output to a certain number of rows/tuple/observations. For this purpose, we can use the LIMIT command. Please note that the LIMIT command is always the very last part of a query.

LIMIT

5. Show just the first 10 observations of the `sales_reps` table with all of the columns.

```
SELECT *
FROM sales_reps
LIMIT 10;
```

id	name	region_id
321500	Samuel Racine	1
321510	Eugena Esser	1
321520	Michel Averette	1
321530	Renetta Carew	1
321540	Cara Clarke	1
321550	Lavera Oles	1
321560	Elba Felder	1
321570	Shawanda Selke	1
321580	Sibyl Lauria	1
321590	Necole Victory	1

ORDER BY

The ORDER BY statement allows us to order our table by any row. If you are familiar with Excel, this is similar to the sorting you can do with filters.

The ORDER BY statement is always after the SELECT and FROM statements, but it is before the LIMIT statement. As you learn additional commands, the order of these statements will matter more. If we are using the LIMIT statement, it will always appear last.

6. Write a SQL query to look up the most 10 recent orders.

The `ORDER BY` clause will help you accomplish this by allowing you to sort the orders by date. The orders table is sorted by Account ID by default. Let's add an `ORDER BY` clause to reorder the results based on the date the order was placed, which you can see in the `occurred_at` column. Notice that the order by clause goes between the From and Limit clauses. You have to write the clauses in this order, or the query will not run. By default, order by goes from a to z, lowest to highest, or earliest to latest, if working with dates. If you want to order the other way, you can add DESC, short for descending, to the end of the Order By clause.

```
SELECT *
FROM orders
ORDER BY occurred_at DESC
LIMIT 10;
```

id	account_id	occurred_at	standard_qty	gloss_qty	poster_qty	total	standard_amt_usd	gloss_amt_usd	poster_amt_usd	total_amt_usd
6451	3841	2017-01-02 00:02:40	42	506	302	850	209.58	3789.94	2452.24	6451.76
3546	3841	2017-01-01 23:50:16	291	36	26	353	1452.09	269.64	211.12	1932.85
6454	3861	2017-01-01 22:29:50	38	167	51	256	189.62	1250.83	414.12	1854.57
3554	3861	2017-01-01 22:17:26	497	0	23	520	2480.03	0.00	186.76	2666.79
6556	4051	2017-01-01 21:04:25	0	65	50	115	0.00	486.85	406.00	892.85
3745	4051	2017-01-01 20:52:23	495	15	0	510	2470.05	112.35	0.00	2582.40
1092	1761	2017-01-01 17:34:10	62	28	124	214	309.38	209.72	1006.88	1525.98
1364	1961	2017-01-01 16:40:57	102	39	29	170	508.98	292.11	235.48	1036.57
3159	3431	2017-01-01 14:05:39	302	29	18	349	1506.98	217.21	146.16	1870.35
6223	3431	2017-01-01 13:57:21	51	444	135	630	254.49	3325.56	1096.20	4676.25

7. **YOUR TURN** Write a query to return the 10 earliest orders in the orders table. Include the `id`, `occurred_at`, and `total_amt_usd`.
8. **YOUR TURN** Write a query to return the top 5 orders in terms of largest `total_amt_usd`. Include the `id`, `account_id`, and `total_amt_usd`.
9. **YOUR TURN** Write a query to return the bottom 5 orders in terms of least total amount USD Include the id, account_id, and total amount in US dollars.
10. Write a query that that returns all accounts ordered by account id. Within each account, we would like to see the orders sorted by total amount used in descending order.

You can also order by multiple columns. This is particularly useful if your data falls into categories and you'd like to organize rows by date, for example, but keep all of the results within a given category together. The

statement sorts according to columns listed from left first and those listed on the right after that. We still have the ability to flip the way we order using DESC.

```
SELECT account_id, total_amt_usd
FROM orders
ORDER BY account_id, total_amt_usd DESC
LIMIT 5;
```

account_id	total_amt_usd
1001	9426.71
1001	9230.67
1001	9134.31
1001	8963.91
1001	8863.24

11. **YOUR TURN** Write a query that returns the top 5 rows from orders ordered according newest to oldest, but with the largest total_amt_usd for each date listed first for each date. (Return only `id`, `occured_at`, `total_amt_used`)

WHERE

Imagine yourself as an account manager at Parch and Posey. You're about to head out to visit one of your most important customers and you want to show up prepared, which means making sure that you are up to speed on all of their recent purchases.

You can use a WHERE clause to generate a list of all purchases made by that specific customer. The WHERE clause allows you to filter a set of results based on specific criteria as you would with Excel's filter capability.

The WHERE clause goes after FROM but before ORDER BY or LIMIT. In addition, the following are the comparison operators with their SQL syntax:

Operator	SQL Syntax
Equal	=
Greater Than	>
Less Than	<
Great Than or Equal	>=
Less Than or Equal	<=
Not Equal	<> or !=
String Comparison Test	LIKE

12. Write a query to show only orders from our top customer, which is represented by account ID 4251.

```
SELECT *
FROM orders
WHERE account_id = 4251;
```

id	account_id	occurred_at	standard_qty	gloss_qty	poster_qty	total	standard_amt_usd	gloss_amt_usd	poster_amt_usd	total_amt_usd
4009	4251	2016-06-05 01:36:42	626	15	0	641	3123.74	112.35	0.00	3236.09
4010	4251	2016-07-04 12:34:49	498	6	2	506	2485.02	44.94	16.24	2546.20
4011	4251	2016-08-02 00:53:28	679	36	5	720	3388.21	269.64	40.60	3698.45
4012	4251	2016-09-01 02:32:51	503	13	32	548	2509.97	97.37	259.84	2867.18
4013	4251	2016-09-30 13:31:53	503	39	0	542	2509.97	292.11	0.00	2802.08
4014	4251	2016-10-29 12:04:06	483	12	0	495	2410.17	89.88	0.00	2500.05
4015	4251	2016-11-27 15:17:06	520	21	7	548	2594.80	157.29	56.84	2808.93
4016	4251	2016-12-26 08:53:24	521	16	28262	28799	2599.79	119.84	229487.44	232207.07
6719	4251	2016-06-05 01:16:37	0	78	0	78	0.00	584.22	0.00	584.22
6720	4251	2016-08-02 01:13:08	9	0	19	28	44.91	0.00	154.28	199.19
6721	4251	2016-09-01 02:39:55	3	71	50	124	14.97	531.79	406.00	952.76
6722	4251	2016-11-27 15:16:07	19	56	0	75	94.81	419.44	0.00	514.25
6723	4251	2016-12-26 08:39:56	0	31	21	52	0.00	232.19	170.52	402.71

13. **YOUR TURN** Write a query to pull the first 5 rows and all columns from the orders table that have a `total_amt_usd` less than or equal to 500.

14. Filter the accounts table to include the company `name`, `website`, and the primary point of contact (`primary_poc`) for `Exxon Mobil` in the accounts table.

```
SELECT name, website, primary_poc
FROM accounts
WHERE name = 'Exxon Mobil';
```

name	website	primary_poc
Exxon Mobil	www.exxonmobil.com	Sung Shields

Derived Columns

Creating a new column that is a combination of existing columns is known as a derived column. Derived columns can include simple arithmetic or any number of advanced calculations.

15. Calculate how many non-standard papers were sold (poster and gloss).

```
SELECT account_id, occurred_at, gloss_qty, poster_qty, gloss_qty + poster_qty
FROM orders
LIMIT 5;
```

account_id	occurred_at	gloss_qty	poster_qty	?column?
1001	2015-10-06 17:31:14	22	24	46
1001	2015-11-05 03:34:33	41	57	98
1001	2015-12-04 04:21:55	47	0	47
1001	2016-01-02 01:18:24	32	0	32
1001	2016-02-01 19:27:27	29	28	57

We can give the new derived column an alias and we can do this by adding AS to the end of the line that produces the derived column and then giving it a name.

```
SELECT account_id, occurred_at, gloss_qty, poster_qty, gloss_qty + poster_qty as non_standard_qty
FROM orders
LIMIT 5;
```

account_id	occurred_at	gloss_qty	poster_qty	non_standard_qty
1001	2015-10-06 17:31:14	22	24	46
1001	2015-11-05 03:34:33	41	57	98
1001	2015-12-04 04:21:55	47	0	47
1001	2016-01-02 01:18:24	32	0	32
1001	2016-02-01 19:27:27	29	28	57

16. **YOUR TURN** Create a column that divides the standard_amt_usd by the standard_qty to find the unit price for standard paper for each order. Limit the results to the first 5 orders, and include the id and account_id fields.

Logical Operators

In this section, you will be learning about Logical Operators. Logical Operators include:

LIKE This allows you to perform operations similar to using WHERE and =, but for cases when you might not know exactly what you are looking for.

IN This allows you to perform operations similar to using WHERE and =, but for more than one condition.

NOT This is used with IN and LIKE to select all of the rows NOT LIKE or NOT IN a certain condition.

AND & BETWEEN These allow you to combine operations where all combined conditions must be true.

OR This allow you to combine operations where at least one of the combined conditions must be true.

17. Suppose that you are trying to find the website for Whole Foods, and you don't remember their full URL. In this case, we can identify all accounts that has the word `food` in their url to find the exact URL for Whole Foods.

```
select name, website
from accounts
where website LIKE '%food%';
```

name	website
Tyson Foods	www.tysonfoods.com
US Foods Holding	www.usfoods.com
ConAgra Foods	www.conagrafoods.com
Whole Foods Market	www.wholefoodsmarket.com
Hormel Foods	www.hormelfoods.com
Dean Foods	www.deanfoods.com

The LIKE operator is extremely useful for working with text. You will use LIKE within a WHERE clause. The LIKE operator is frequently used with %. The % tells us that we might want any number of characters leading up to a particular set of characters or following a certain set of characters, as we saw with the `food` syntax above.

18. **YOUR TURN** Use the accounts table to find all companies whose names contain the string 'one' somewhere in the name.

19. **YOUR TURN** Use the accounts table to find all the companies whose names start with 'C'.

20. Suppose that you are interested in accounts for only Apple and Walmart. Get the account ids for both.

```
select name, id
from accounts
where name in ('Apple', 'Walmart');
```

name	id
Walmart	1001
Apple	1021

The `IN` operator is useful for working with both numeric and text columns. This operator allows you to use an `=`, but for more than one item of that particular column. We can check one, two or many column values for which we want to pull data, but all within the same query.

21. **YOUR TURN** Use the `web_events` table to find all information regarding individuals who were contacted via the `channel` of `organic` or `adwords`.

22. Use the accounts table to find the account name, primary poc, and sales rep id for all stores except Walmart, Target, and Nordstrom.

```
select name, primary_poc, sales_rep_id
from accounts
```

```
where name not in ('Walmart', 'Target', 'Nordstrom')
limit 5;
```

name	primary_poc	sales_rep_id
Exxon Mobil	Sung Shields	321510
Apple	Jodee Lupo	321520
Berkshire Hathaway	Serafina Banda	321530
McKesson	Angeles Crusoe	321540
UnitedHealth Group	Savanna Gayman	321550

The NOT operator is an extremely useful operator for working with the previous two operators we introduced: IN and LIKE. By specifying NOT LIKE or NOT IN, we can grab all of the rows that do not meet a particular criteria.

23. **YOUR TURN** Use the `web_events` table to find all information regarding individuals who were contacted via any method except using organic or adwords methods.

24. **YOUR TURN** Use the accounts table to find all the companies whose names do not start with 'C'.

25. Pull all transactions that occurred between April 1, 2016 and September 1, 2016.

```
select *
from orders
where occurred_at >= '04-01-2016' AND occurred_at <= '09-01-2016'
limit 5;
```

id	account_id	occurred_at	standard_qty	gloss_qty	poster_qty	total	standard_amt_usd	gloss_amt_usd	poster_amt_usd	total_amt_usd
7	1001	2016-04-01 11:20:18	101	33	92	226	503.99	247.17	747.04	1498.20
8	1001	2016-05-01 15:55:51	95	47	151	293	474.05	352.03	1226.12	2052.20
9	1001	2016-05-31 21:22:48	91	16	22	129	454.09	119.84	178.64	752.57
10	1001	2016-06-30 12:32:05	94	46	8	148	469.06	344.54	64.96	878.56
11	1001	2016-07-30 03:26:30	101	36	0	137	503.99	269.64	0.00	773.63

The AND operator is used within a WHERE statement to consider more than one logical clause at a time. Each time you link a new statement with an AND, you will need to specify the column you are interested in looking at. You may link as many statements as you would like to consider at the same time. This operator works with all of the operations we have seen so far including arithmetic operators (+, *, -, /). LIKE, IN, and NOT logic can also be linked together using the AND operator.

The above query could be written using the BETWEEN operator as follows:

```
select *
from orders
where occurred_at BETWEEN '04-01-2016' AND '09-01-2016'
limit 5;
```

id	account_id	occurred_at	standard_qty	gloss_qty	poster_qty	total	standard_amt_usd	gloss_amt_usd	poster_amt_usd	total_amt_usd
7	1001	2016-04-01 11:20:18	101	33	92	226	503.99	247.17	747.04	1498.20
8	1001	2016-05-01 15:55:51	95	47	151	293	474.05	352.03	1226.12	2052.20
9	1001	2016-05-31 21:22:48	91	16	22	129	454.09	119.84	178.64	752.57
10	1001	2016-06-30 12:32:05	94	46	8	148	469.06	344.54	64.96	878.56
11	1001	2016-07-30 03:26:30	101	36	0	137	503.99	269.64	0.00	773.63

26. **YOUR TURN** Write a query that returns all the orders where the standard_qty is over 1000, the poster_qty is 0, and the gloss_qty is 0.

27. **YOUR TURN** Use the `web_events` table to find all information regarding individuals who were contacted via organic or adwords and started their account at any point in 2016 sorted from newest to oldest.

Using BETWEEN is tricky for dates! While BETWEEN is generally inclusive of endpoints, it assumes the time is at 00:00:00 (i.e. midnight) for dates. This is the reason why we set the right-side endpoint of the period at '01-01-2017'.

28. Find all existing customers whose orders omitted some type of paper.

```
select *
from orders
where standard_qty = 0 OR gloss_qty = 0 OR poster_qty = 0
limit 5;
```

id	account_id	occurred_at	standard_qty	gloss_qty	poster_qty	total	standard_amt_usd	gloss_amt_usd	poster_amt_usd	total_amt_usd
3	1001	2015-12-04 04:21:55	85	47	0	132	424.15	352.03	0	776.18
4	1001	2016-01-02 01:18:24	144	32	0	176	718.56	239.68	0	958.24
11	1001	2016-07-30 03:26:30	101	36	0	137	503.99	269.64	0	773.63
17	1011	2016-12-21 10:59:34	527	14	0	541	2629.73	104.86	0	2734.59
18	1021	2015-10-12 02:21:56	516	23	0	539	2574.84	172.27	0	2747.11

When combining multiple of these operations (AND, OR, BETWEEN, NOT), we frequently might need to use parentheses to assure that logic we want to perform is being executed correctly.

29. Find all existing customers whose orders omitted some type of paper and the order occurred after September 1 2016. Order the result from older transactions to the newest.

```
select *
from orders
where (standard_qty = 0 OR gloss_qty = 0 OR poster_qty = 0) AND
      occurred_at >= '2016-09-01'
order by occurred_at
limit 5;
```

id	account_id	occurred_at	standard_qty	gloss_qty	poster_qty	total	standard_amt_usd	gloss_amt_usd	poster_amt_usd	total_amt_usd
6367	3641	2016-09-01 07:48:59	0	134	49	183	0.00	1003.66	397.88	1401.54
6533	4011	2016-09-01 14:28:48	0	0	13	13	0.00	0.00	105.56	105.56
3708	4011	2016-09-01 14:46:17	459	0	0	459	2290.41	0.00	0.00	2290.41
1079	1741	2016-09-02 06:41:11	503	0	34	537	2509.97	0.00	276.08	2786.05
4971	1741	2016-09-02 06:48:59	0	37	0	37	0.00	277.13	0.00	277.13

30. **YOUR TURN** Find list of orders ids where either gloss_qty or poster_qty is greater than 4000. Only include the id field in the resulting table.

31. **YOUR TURN** Write a query that returns a list of orders where the standard_qty is zero and either the gloss_qty or poster_qty is over 1000.

32. **YOUR TURN** Find all the company names that start with a 'C' or 'W', and the primary contact contains 'ana' or 'Ana', but it doesn't contain 'eana'.

33. **YOUR TURN** Return all the information for web events that occurred during June and July of 2016.