Finding Matched and Unmatched Rows with FULL OUTER JOIN

You’re not likely to use FULL JOIN (which can also be written as FULL OUTER JOIN) too often, but the syntax is worth practicing anyway. LEFT JOIN and RIGHT JOIN each return unmatched rows from one of the tables—FULL JOIN returns unmatched rows from both tables. FULL JOIN is commonly used in conjunction with aggregations to understand the amount of overlap between two tables.

Say you're an analyst at Parch & Posey and you want to see:

Each account who has a sales rep and each sales rep that has an account (all of the columns in these returned rows will be full)
but also each account that does not have a sales rep and each sales rep that does not have an account (some of the columns in these returned rows will be empty)

This type of question is rare, but FULL OUTER JOIN is perfect for it. In the following SQL Explorer, write a query with FULL OUTER JOIN to fit the above described Parch & Posey scenario (selecting all of the columns in both of the relevant tables, accounts and sales_reps).

1.

```
SELECT a.name account_name,
       s.name rep
FROM accounts a
FULL OUTER JOIN sales_reps s ON s.id = a.sales_rep_id
WHERE s.name IS NULL OR a.name IS NULL
```

-- INEQUALITY JOINS

Example:

A. Create a table that returns all web events that occurred before an order. Where the subquery is the very first order by month.

```
SELECT w.*,
```

```
       o.id order_id,
       o.occurred_at order_date
FROM orders o
LEFT JOIN web_events w ON w.account_id = o.account_id
AND w.occurred_at < o.occurred_at
WHERE DATE_TRUNC('month', o.occurred_at) = (
                                            SELECT
DATE_TRUNC('month', MIN(o.occurred_at))
                                            FROM orders o
                                           )
ORDER BY o.account_id, o.occurred_at
```

2. Write a query that left joins the accounts table and the sales_reps tables on each sale rep's ID number and joins it using the < comparison operator on accounts.primary_poc and sales_reps.name, like so: accounts.primary_poc < sales_reps.name

```
SELECT accounts.name as account_name,
       accounts.primary_poc as poc_name,
       sales_reps.name as sales_rep_name
FROM accounts
LEFT JOIN sales_reps ON accounts.sales_rep_id =
sales_reps.id
AND accounts.primary_poc < sales_reps.name

-- Observation: resulting query only only includes
sales_rep name IF primary_poc name comes before the
sales_reps name.

-- SELF JOINS

Example:

SELECT o1.id AS o1_id,
       o1.account_id AS o1_account_id,
       o1.occurred_at AS o1_occurred_at,
       o2.id AS o2_id,
       o2.account_id AS o2_account_id,
       o2.occurred_at AS o2_occurred_at
  FROM orders o1
```

```
 LEFT JOIN orders o2 ON o1.account_id = o2.account_id
-- just the same table
  AND o2.occurred_at > o1.occurred_at
-- find orders that happen after the original order was
placed
  AND o2.occurred_at <= o1.occurred_at + INTERVAL '28 days'
-- find orders where o2 is less than or equal to o1+28 days
ORDER BY o1.account_id, o1.occurred_at
```

3.

```
SELECT w1.id AS w1_id,
       w1.account_id AS w1_account_id,
       w1.occurred_at AS w1_occurred_at,
       w1.channel AS w1_channel,
       w2.id AS w2_id,
       w2.account_id AS w2_account_id,
       w2.occurred_at AS w2_occurred_at,
       w2.channel AS w2_channel
FROM web_events w1
LEFT JOIN web_events w2 ON w1.account_id = w2.account_id
AND w2.occurred_at > w1.occurred_at
AND w2.occurred_at <= w1.occurred_at + INTERVAL '1 day'
order by 1,3
```

```
-- UNION
```

4. Write a query that uses UNION ALL on two instances (and selecting all columns) of the accounts table. Then inspect the results and answer the subsequent quiz.

```
SELECT *
FROM accounts

UNION ALL

SELECT *
FROM accounts
```

5. Add a WHERE clause to each of the tables that you unioned in the query above, filtering the first table where name equals Walmart and filtering the second table where name equals Disney. Inspect the results then answer the subsequent quiz.

```
FROM accounts
WHERE name = 'Walmart'

UNION ALL

SELECT *
FROM accounts
WHERE name = 'Disney'
```

6. How else could the above query results be generated?

```
SELECT *
FROM accounts
WHERE name = 'Walmart' OR name = 'Disney'
```

7. Perform the union in your first query (under the Appending Data via UNION header) in a common table expression and name it double_accounts. Then do a COUNT the number of times a name appears in the double_accounts table. If you do this correctly, your query results should have a count of 2 for each name.

```
WITH double_accounts AS (
                          SELECT *
                           FROM accounts

                          UNION ALL

                          SELECT *
                           FROM accounts
                        )
SELECT name,
       COUNT(*)
FROM double_accounts
GROUP BY 1
```