Udacity SQL course codes→

```
SELECT a.primary_poc, w.occurred_at, w.channel, a.name
FROM web_events w
JOIN accounts a
ON w.account_id = a.id
WHERE a.name = 'Walmart';
```

The abbreviation of ‘a’ and ‘w’ occurs after stating the table in lines 2 and 3

Joins

Country

countryid	countryName
1	India
2	Nepal
3	United States
4	Canada
5	Sri Lanka
6	Brazil

State

stateid	countryid	stateName
1	1	Maharashtra
2	1	Punjab
3	2	Kathmandu
4	3	California
5	3	Texas
6	4	Alberta

```
SELECT c.countryid, c.countryName, s.stateName
FROM Country c
JOIN State s
ON c.countryid = s.countryid;
```

It’ll produce→

The resulting table will look like:

countryid	countryName	stateName
1	India	Maharashtra
1	India	Punjab
2	Nepal	Kathmandu
3	United States	California
3	United States	Texas
4	Canada	Alberta

Whereas a LEFT JOIN will look like the following

```
SELECT c.countryid, c.countryName, s.stateName
FROM Country c
LEFT JOIN State s
ON c.countryid = s.countryid;
```

The resulting table will look like:

countryid	countryName	stateName
1	India	Maharashtra
1	India	Punjab
2	Nepal	Kathmandu
3	United States	California
3	United States	Texas
4	Canada	Alberta
5	Sri Lanka	NULL
6	Brazil	NULL

Left Join gives us results when we want to see blank results.

Provide a table that provides the **region** for each **sales_rep** along with their associated **accounts**. This time only for the `Midwest` region. Your final table should include three columns: the region **name**, the sales rep **name**, and the account **name**. Sort the accounts alphabetically (A-Z) according to account name.

```
SELECT r.name region, s.name rep, a.name account
FROM sales_reps s
JOIN region r
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
WHERE r.name = 'Midwest'
ORDER BY a.name;
```

Provide a table that provides the **region** for each **sales_rep** along with their associated **accounts**. This time only for accounts where the sales rep has a first name starting with `S` and in the `Midwest` region. Your final table should include three columns: the region **name**, the sales rep **name**, and the account **name**. Sort the accounts alphabetically (A-Z) according to account name.

```
SELECT r.name region, s.name rep, a.name account
FROM sales_reps s
JOIN region r
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
WHERE r.name = 'Midwest' AND s.name LIKE 'S%'
ORDER BY a.name;
```

Provide a table that provides the **region** for each **sales_rep** along with their associated **accounts**. This time only for accounts where the sales rep has a **last** name starting with `K` and in the `Midwest` region. Your final table should include three columns: the region **name**, the sales rep **name**, and the account **name**. Sort the accounts alphabetically (A-Z) according to account name.

```
SELECT r.name region, s.name rep, a.name account
FROM sales_reps s
JOIN region r
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
WHERE r.name = 'Midwest' AND s.name LIKE '% K%'
ORDER BY a.name;
```

Provide the **name** for each region for every **order**, as well as the account **name** and the **unit price** they paid (total_amt_usd/total) for the order. However, you should only provide the results if the **standard order quantity** exceeds `100` and the **poster order quantity** exceeds `50`. Your final table should have 3 columns: **region name**, **account name**, and **unit price**. Sort for the smallest **unit price** first.

```
SELECT r.name region, a.name account, o.total_amt_usd/(o.total + 0.01) unit_price
FROM region r
JOIN sales_reps s
ON s.region_id = r.id
JOIN accounts a
ON a.sales_rep_id = s.id
JOIN orders o
ON o.account_id = a.id
WHERE o.standard_qty > 100 AND o.poster_qty > 50
ORDER BY unit_price;
```

JOINs

In this lesson, you learned how to combine data from multiple tables using **JOIN**s. The three **JOIN** statements you are most likely to use are:

1. **JOIN** - an **INNER JOIN** that only pulls data that exists in both tables.
2. **LEFT JOIN** - pulls all the data that exists in both tables, as well as all of the rows from the table in the **FROM** even if they do not exist in the **JOIN** statement.
3. **RIGHT JOIN** - pulls all the data that exists in both tables, as well as all of the rows from the table in the **JOIN** even if they do not exist in the **FROM** statement.

There are a few more advanced **JOIN**s that we did not cover here, and they are used in very specific use cases. UNION and UNION ALL, CROSS JOIN, and the tricky SELF JOIN. These are more advanced than this course will cover, but it is useful to be aware that they exist, as they are useful in special cases.

Find the total sales in **usd** for each account. You should include two columns - the total sales for each company's orders in **usd** and the company **name**.

```
SELECT a.name, SUM(total_amt_usd) total_sales
FROM orders o
JOIN accounts a
ON a.id = o.account_id
GROUP BY a.name;
```

Find the number of **sales reps** in each region. Your final table should have two columns - the **region** and the number of **sales_reps**. Order from fewest reps to most reps.

```
SELECT r.name, COUNT(*) num_reps
```

```
2. FROM region r
3. JOIN sales_reps s
4. ON r.id = s.region_id
5. GROUP BY r.name
   ORDER BY num_reps;
```

HAVING - Expert Tip

HAVING is the "clean" way to filter a query that has been aggregated, but this is also commonly done using a subquery. Essentially, any time you want to perform a **WHERE** on an element of your query that was created by an aggregate, you need to use **HAVING** instead.

How many of the **sales reps** have more than 5 accounts that they manage?

```
SELECT s.id, s.name, COUNT(*) num_accounts
FROM accounts a
JOIN sales_reps s
ON s.id = a.sales_rep_id
GROUP BY s.id, s.name
HAVING COUNT(*) > 5
ORDER BY num_accounts;
```

It's important to have the COUNT(*) 'count all' function at the top to allow the columns to be grouped.

Which accounts used `facebook` as a **channel** to contact customers more than 6 times?

```
1. SELECT a.id, a.name, w.channel, COUNT(*) use_of_channel
2. FROM accounts a
3. JOIN web_events w
4. ON a.id = w.account_id
5. GROUP BY a.id, a.name, w.channel
6. HAVING COUNT(*) > 6 AND w.channel = 'facebook'
   ORDER BY use_of_channel;
```

Which account used `facebook` most as a **channel**?

```
SELECT a.id, a.name, w.channel, COUNT(*) use_of_channel
FROM accounts a
JOIN web_events w
ON a.id = w.account_id
WHERE w.channel = 'facebook'
GROUP BY a.id, a.name, w.channel
ORDER BY use_of_channel DESC
LIMIT 1;
```

TIMEFUNCTIONS

Which **year** did Parch & Posey have the greatest sales in terms of total number of orders? Are all years evenly represented by the dataset?

```
SELECT DATE_PART('year', occurred_at) ord_year,  COUNT(*) total_sales
FROM orders
GROUP BY 1
ORDER BY 2 DESC;
```

In which **month** of which **year** did `Walmart` spend the most on gloss paper in terms of dollars?

```
SELECT DATE_TRUNC('month', o.occurred_at) ord_date, SUM(o.gloss_amt_usd) tot_spen
t
FROM orders o
JOIN accounts a
ON a.id = o.account_id
WHERE a.name = 'Walmart'
GROUP BY 1
ORDER BY 2 DESC
LIMIT 1;
```

We would like to understand 3 different branches of customers based on the amount associated with their purchases. The top branch includes anyone with a Lifetime Value (total sales of all orders) `greater than 200,000` usd. The second branch is between `200,000 and 100,000` usd. The lowest branch is anyone `under 100,000` usd. Provide a table that includes the **level** associated with each **account**. You should provide the **account name**, the **total sales of all orders** for the customer, and the **level**. Order with the top spending customers listed first.

```
2. SELECT a.name, SUM(total_amt_usd) total_spent,
3.      CASE WHEN SUM(total_amt_usd) > 200000 THEN 'top'
4.      WHEN  SUM(total_amt_usd) > 100000 THEN 'middle'
5.      ELSE 'low' END AS customer_level
6. FROM orders o
7. JOIN accounts a
8. ON o.account_id = a.id
9. GROUP BY a.name
   ORDER BY 2 DESC;
```

SUB QUERY and NESTING

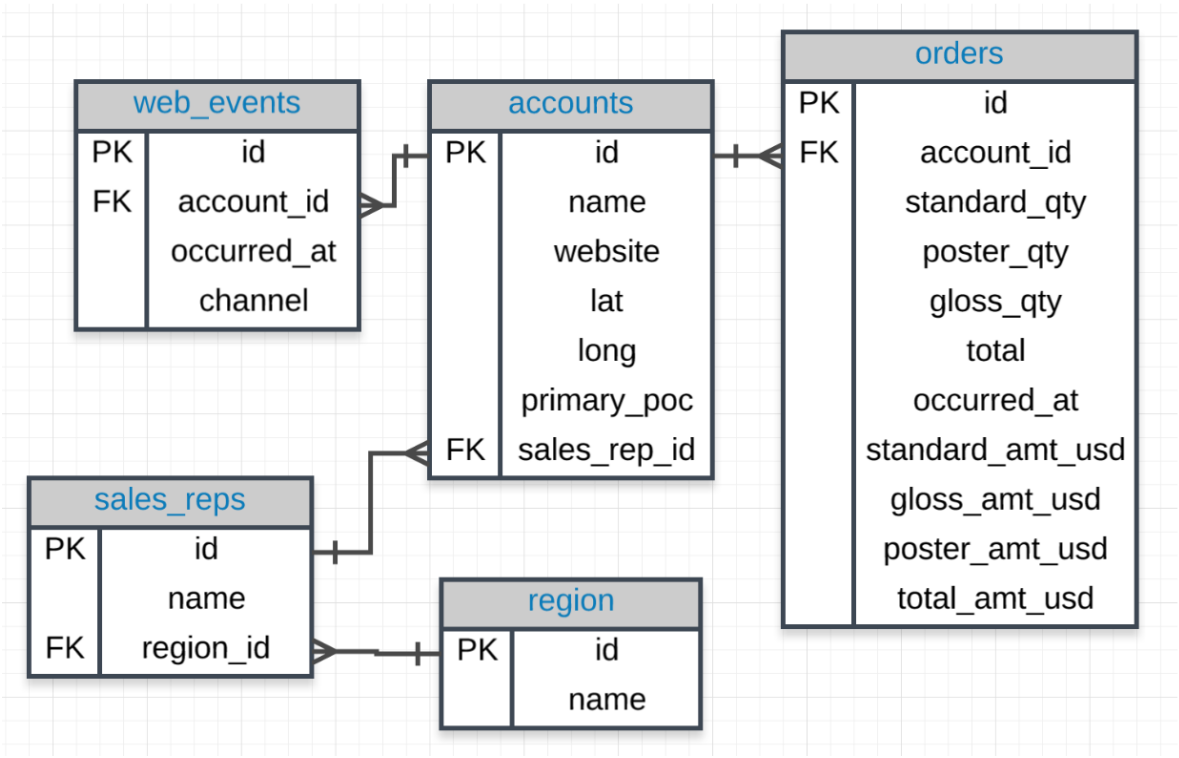
Finally, here we are able to get a table that shows the average number of events a day for each channel.

```
1. SELECT channel, AVG(events) AS average_events
2. FROM (SELECT DATE_TRUNC('day',occurred_at) AS day,
3.              channel, COUNT(*) as events
4.       FROM web_events
5.       GROUP BY 1,2) sub
6. GROUP BY channel
7. ORDER BY 2 DESC;
```

Then to pull the average for each, we could do this all in one query, but for readability, I provided two queries below to perform each separately.

```
SELECT AVG(standard_qty) avg_std, AVG(gloss_qty) avg_gls, AVG(poster_qty) avg_ps
t
FROM orders
WHERE DATE_TRUNC('month', occurred_at) =
     (SELECT DATE_TRUNC('month', MIN(occurred_at)) FROM orders);

SELECT SUM(total_amt_usd)
FROM orders
WHERE DATE_TRUNC('month', occurred_at) =
      (SELECT DATE_TRUNC('month', MIN(occurred_at)) FROM orders);
```



Provide the **name** of the **sales_rep** in each **region** with the largest amount of **total_amt_usd** sales. →

SELECT S_Name, R_Name,MAX(Total)

FROM

(SELECT s.name AS S_Name,

r.name AS R_Name,

SUM(o.total_amt_usd) AS Total

FROM sales_reps s

JOIN region r

ON s.region_id = r.id

JOIN accounts a

ON s.id=a.sales_rep_id

Join orders o

ON a.id = o.account_id

GROUP BY 1,2) sub

GROUP BY 1,2

For the customer that spent the most (in total over their lifetime as a customer) **total_amt_usd**, how many **web_events** did they have for each channel?

```
SELECT a.name, w.channel, COUNT(*)
FROM accounts a
JOIN web_events w
ON a.id = w.account_id AND a.id =  (SELECT id
                     FROM (SELECT a.id, a.name, SUM(o.total_amt_usd) tot_spent
                           FROM orders o
                           JOIN accounts a
                           ON a.id = o.account_id
                           GROUP BY a.id, a.name
                           ORDER BY 3 DESC
                           LIMIT 1) inner_table)
GROUP BY 1, 2
ORDER BY 3 DESC;
```

Provide the **name** of the **sales_rep** in each **region** with the largest amount of **total_amt_usd** sales.

WITH Table1 AS (SELECT s.name AS Rep_name,

r.name AS Region,

SUM(o.total_amt_usd) AS Total_Usd

FROM sales_reps s

JOIN region r

ON s.region_id=r.id

JOIN accounts a

ON s.id=a.sales_rep_id

JOIN orders o

ON a.id=o.account_id

```
        GROUP BY 1,2
                                ORDER BY 3 DESC),

Table2 AS (SELECT       Region,
                        MAX(Total_Usd) AS Max_Total
      FROM Table1
      GROUP BY 1)
SELECT Table1.Rep_name, Table1.Region, Table1.Total_USD
FROM Table1
JOIN Table2
ON Table1.Region=Table2.Region AND Table1.Total_USD=Table2.Max_Total
```

What is the lifetime average amount spent in terms of **total_amt_usd** for the top 10 total spending **accounts**?

```
WITH t1 AS (
   SELECT a.id, a.name, SUM(o.total_amt_usd) tot_spent
   FROM orders o
   JOIN accounts a
   ON a.id = o.account_id
   GROUP BY a.id, a.name
   ORDER BY 3 DESC
   LIMIT 10)
SELECT AVG(tot_spent)
FROM t1;
```

CLEANING DATA

Use the **accounts** table and a **CASE** statement to create two groups: one group of company names that start with a number and a second group of those company names that start with a letter. What proportion of company names start with a letter?

```
SELECT SUM(num) nums, SUM(letter) letters
FROM (SELECT name, CASE WHEN LEFT(UPPER(name), 1) IN ('0','1','2','3','4','5'
,'6','7','8','9')
                        THEN 1 ELSE 0 END AS num,
         CASE WHEN LEFT(UPPER(name), 1) IN ('0','1','2','3','4','5','6','7','8'
,'9')
                        THEN 0 ELSE 1 END AS letter
      FROM accounts) t1;
```

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. A list of extensions (and pricing) is provided here. Pull these extensions and provide how many of each website type exist in the **accounts** table.

```
SELECT RIGHT(website, 3) AS domain, COUNT(*) num_companies
FROM accounts
GROUP BY 1
ORDER BY 2 DESC;
```

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

```
SELECT LEFT(primary_poc, STRPOS(primary_poc, ' ') -1 ) first_name,
RIGHT(primary_poc, LENGTH(primary_poc) - STRPOS(primary_poc, ' ')) last_name
FROM accounts;
```

---

1. Run the query entered below in the SQL workspace to notice the row with missing data.
2. Use **COALESCE** to fill in the `accounts.id` column with the `account.id` for the NULL value for the table in `1`.
3. Use **COALESCE** to fill in the `orders.account_id` column with the `account.id` for the NULL value for the table in `1`.
4. Use **COALESCE** to fill in each of the **qty** and **usd** columns with 0 for the table in `1`.
5. Run the query in `1` with the **WHERE** removed and **COUNT** the number of `id`s .
6. Run the query in `5`, but with the **COALESCE** function used in questions `2` through `4`.

```
SELECT COALESCE(o.id, a.id) filled_id, a.name, a.website, a.lat, a.long, a.
primary_poc, a.sales_rep_id, o.*
FROM accounts a
LEFT JOIN orders o
ON a.id = o.account_id
WHERE o.total IS NULL;
```

Ranking Total Paper Ordered by Account

Select the `id`, `account_id`, and `total` variable from the `orders` table, then create a column called `total_rank` that ranks this total amount of paper ordered (from highest to lowest) *for each account* using a partition. Your final table should have these four columns.

SELECT id,

account_id,

total,

RANK() OVER (PARTITION BY account_id ORDER BY total DESC) AS total_rank

FROM orders

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
       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
```
