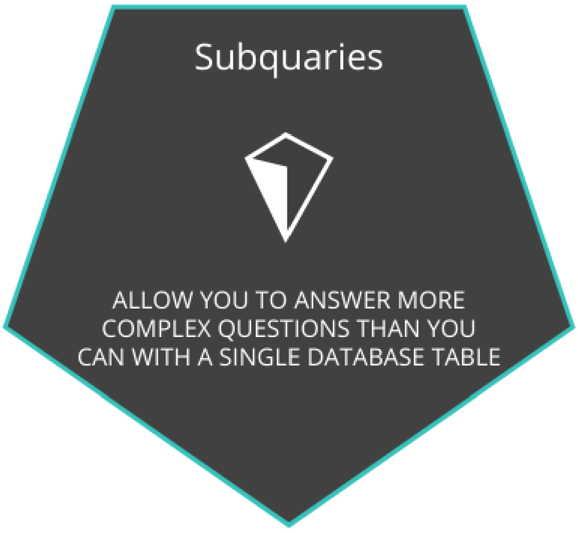
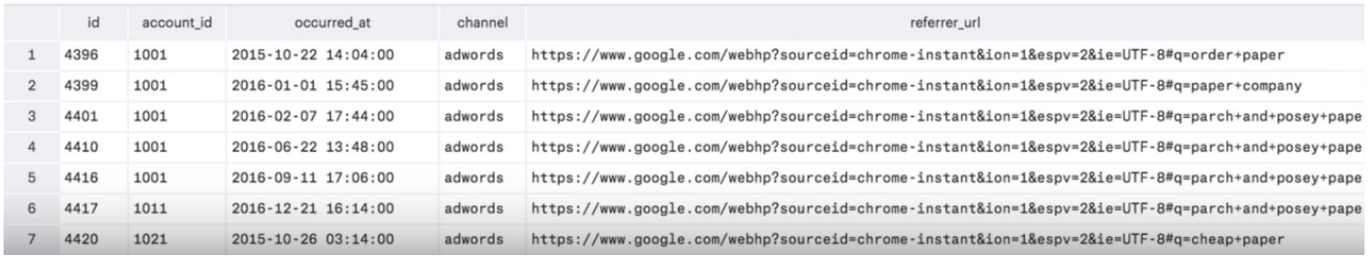
Subquaries:-


Subquaries
ALLOW YOU TO ANSWER MORE COMPLEX QUESTIONS THAN YOU CAN WITH A SINGLE DATABASE TABLE

Illustraion:-

- **First, we'll start by querying the underlying table to make sure that data makes sense for what we're trying to do.**

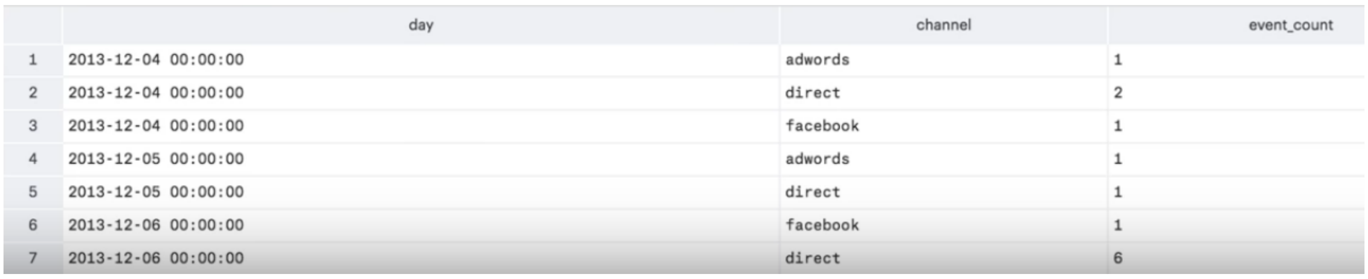
```
SELECT *
FROM demo.web_events_full
```



	id	account_id	occurred_at	channel	referrer_url
1	4396	1001	2015-10-22 14:04:00	adwords	https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=order+paper
2	4399	1001	2016-01-01 15:45:00	adwords	https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=paper+company
3	4401	1001	2016-02-07 17:44:00	adwords	https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=parch+and+posey+pape
4	4410	1001	2016-06-22 13:48:00	adwords	https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=parch+and+posey+pape
5	4416	1001	2016-09-11 17:06:00	adwords	https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=parch+and+posey+pape
6	4417	1011	2016-12-21 16:14:00	adwords	https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=parch+and+posey+pape
7	4420	1021	2015-10-26 03:14:00	adwords	https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=cheap+paper

- **Next, we'll count up all the events in each channel, in each day.**

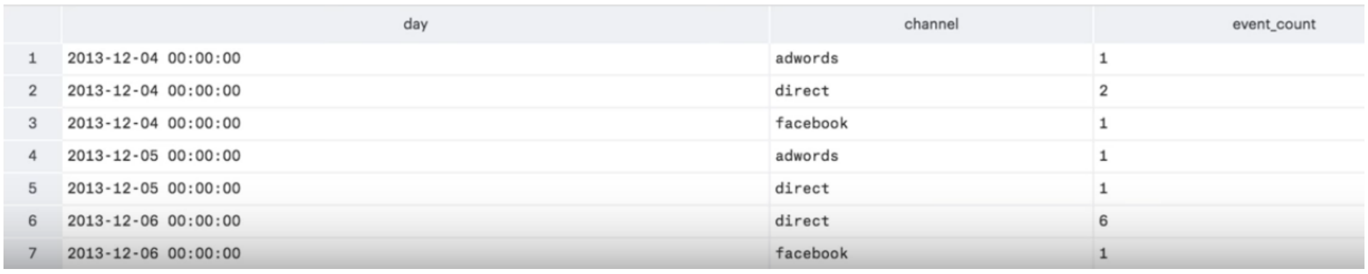
```
SELECT DATE_TRUNC('day', occurred_at) AS day,
       channel,
       COUNT(*) AS event_count
FROM demo.web_events_full
GROUP BY 1,2
ORDER BY 1
```



	day	channel	event_count
1	2013-12-04 00:00:00	adwords	1
2	2013-12-04 00:00:00	direct	2
3	2013-12-04 00:00:00	facebook	1
4	2013-12-05 00:00:00	adwords	1
5	2013-12-05 00:00:00	direct	1
6	2013-12-06 00:00:00	facebook	1
7	2013-12-06 00:00:00	direct	6

- **The last step is that we want to average across the events column we've created. In order to do that, we'll want to query against the results from this query. We can do that by wrapping the query in parentheses and using it in the FROM clause of the next query that we write. It is now a query within a query, also known as a subquery. Subqueries are required to have aliases, which are added after the parentheses, the same way you would add an alias to a table. Here, we're just selecting all of the data from the subquery.**

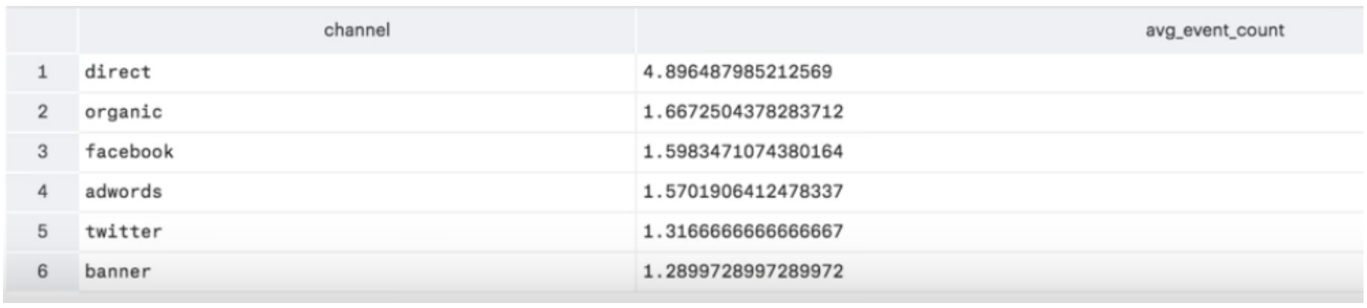
```
SELECT *
FROM (SELECT DATE_TRUNC('day', occurred_at) AS day,
       channel,
       COUNT(*) AS event_count
FROM demo.web_events_full
GROUP BY 1,2
ORDER BY 1) sub
```



	day	channel	event_count
1	2013-12-04 00:00:00	adwords	1
2	2013-12-04 00:00:00	direct	2
3	2013-12-04 00:00:00	facebook	1
4	2013-12-05 00:00:00	adwords	1
5	2013-12-05 00:00:00	direct	1
6	2013-12-06 00:00:00	direct	6
7	2013-12-06 00:00:00	facebook	1

- **Let's go the last mile and average events for each channel. Since the subquery acts like one table in the FROM clause, we'll put a GROUP BY clause after the subquery. Since we're now reordering based on this new aggregation, we no longer need the ORDER BY statement in the subquery. So let's take that out to keep things clean.**

```
SELECT channel,
       AVG(event_count) AS avg_event_count
FROM (SELECT DATE_TRUNC('day', occurred_at) AS day,
        channel,
        COUNT(*) AS event_count
        FROM demo.web_events_full
        GROUP BY 1,2) sub
GROUP BY 1
ORDER BY 2
```



	channel	avg_event_count
1	direct	4.896487985212569
2	organic	1.6672504378283712
3	facebook	1.5983471074380164
4	adwords	1.5701906412478337
5	twitter	1.3166666666666667
6	banner	1.2899728997289972

Examples:-

1. **First, we needed to group by the day and channel. Then ordering by the number of events (the third column) gave us a quick way to answer the first question.**

```
    SELECT DATE_TRUNC('day',occurred_at) AS day,
           channel, COUNT(*) as events
    FROM web_events
    GROUP BY 1,2
    ORDER BY 3 DESC;
```

2. **Here you can see that to get the entire table in question 1 back, we included an * in our SELECT statement. You will need to be sure to alias your table.**

```
    SELECT *
    FROM (SELECT DATE_TRUNC('day',occurred_at) AS day,
                 channel, COUNT(*) as events
            FROM web_events
            GROUP BY 1,2
            ORDER BY 3 DESC) sub;
```

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

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

Subquaries Formatting:-

- **When writing Subqueries, it is easy for your query to look incredibly complex. In order to assist your reader, which is often just yourself at a future date, formatting SQL will help with understanding your code.**
- **The important thing to remember when using subqueries is to provide some way for the reader to easily determine which parts of the query will be executed together. Most people do this by indenting the subquery in some way - you saw this with the solution blocks in the previous concept.**
- **The examples in this class are indented quite far—all the way to the parentheses. This isn't practical if you nest many subqueries, but in general, be thinking about how to write your queries in a readable way. Examples of the same query written multiple different ways is provided below. You will see that some are much easier to read than others.**

Badly Formatted Queries

- **Though these poorly formatted examples will execute the same way as the well formatted examples, they just aren't very friendly for understanding what is happening!**
- **Here is the first, where it is impossible to decipher what is going on:**

```
    SELECT * FROM (SELECT DATE_TRUNC('day',occurred_at) AS day, channel, COUNT(*) as events FROM web_events GROUP BY 1,2 ORDER BY 3 DESC) sub;
```

This second version, which includes some helpful line breaks, is easier to read than that previous version, but it is still not as easy to read as the queries in the Well Formatted Query section.

```
    SELECT *
    FROM (
    SELECT DATE_TRUNC('day',occurred_at) AS day,
    channel, COUNT(*) as events
    FROM web_events
    GROUP BY 1,2
    ORDER BY 3 DESC) sub;
```

Well Formatted Query

- **Now for a well formatted example, you can see the table we are pulling from much easier than in the previous queries.**

```
    SELECT *
    FROM (SELECT DATE_TRUNC('day',occurred_at) AS day,
                channel, COUNT(*) as events
          FROM web_events
```

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

Additionally, if we have a GROUP BY, ORDER BY, WHERE, HAVING, or any other statement following our subquery, we would then indent it at the same level as our outer query.

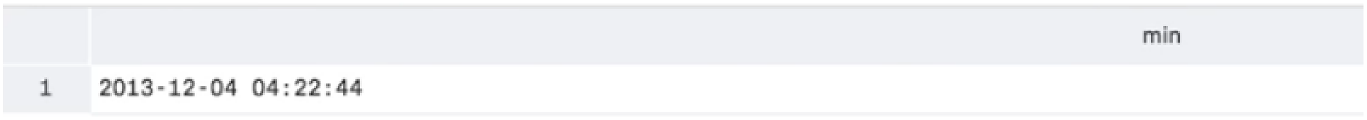
The query below is similar to the above, but it is applying additional statements to the outer query, so you can see there are GROUP BY and ORDER BY statements used on the output are not tabbed. The inner query GROUP BY and ORDER BY statements are indented to match the inner table.

```
SELECT *
FROM (SELECT DATE_TRUNC('day',occurred_at) AS day,
            channel, COUNT(*) as events
      FROM web_events
      GROUP BY 1,2
      ORDER BY 3 DESC) sub
GROUP BY day, channel, events
ORDER BY 2 DESC;
```

More Subquaries:-

- **Subqueries can be used in several places within a query. It can really be used anywhere you might use a table name or even a column name or an individual value. They're especially useful in conditional logic, in conjunction with where or Join clauses, or in the when portion of a case statement.**

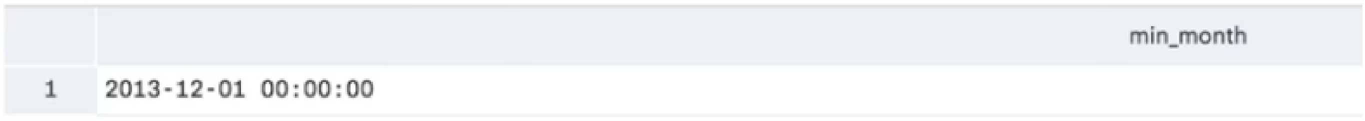
```
SELECT MIN(occurred_at) AS min
FROM demo.orders
```



	min
1	2013-12-04 04:22:44

- **For example, you might want to return only orders that occurred in the same month as Parch and Posies first order ever. To get the date of the first order, you can write a subquery with a min function. Let's add a date_trunc function to get the month.**

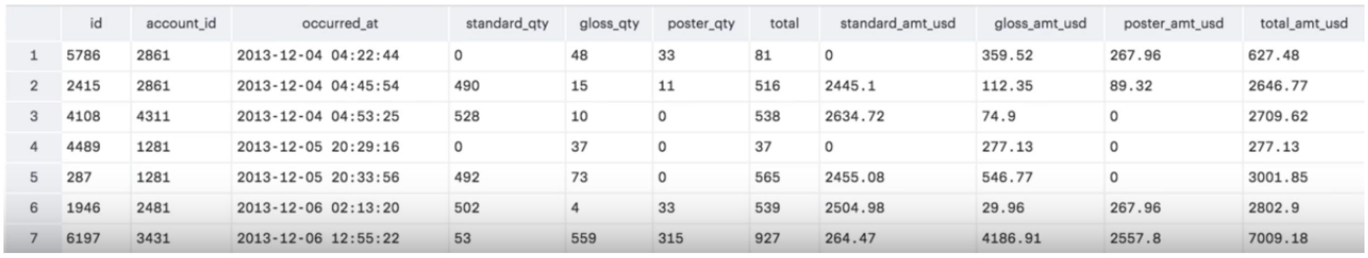
```
SELECT DATE_TRUNC('month',MIN(occurred_at)) AS min_month
FROM demo.orders
```



	min_month
1	2013-12-01 00:00:00

- **Finally, let's write an outer query that uses this to filter the orders table and sorts by the occurred at column. You can see that all of these orders took place in December 2013, the same month as the first order. This query works because the result of the subquery is only one cell. Most conditional logic will work with subqueries containing one-cell results. But IN is the only type of conditional logic that will work when the inner query contains multiple results.**

```
    SELECT *
    FROM demo.orders
    WHERE DATE_TRUNC('month', occurred_at) = (SELECT
DATE_TRUNC('month',MIN(occurred_at)) AS min_month
                                              FROM demo.orders)
    ORDER BY occurred_at
```



	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	5786	2861	2013-12-04 04:22:44	0	48	33	81	0	359.52	267.96	627.48
2	2415	2861	2013-12-04 04:45:54	490	15	11	516	2445.1	112.35	89.32	2646.77
3	4108	4311	2013-12-04 04:53:25	528	10	0	538	2634.72	74.9	0	2709.62
4	4489	1281	2013-12-05 20:29:16	0	37	0	37	0	277.13	0	277.13
5	287	1281	2013-12-05 20:33:56	492	73	0	565	2455.08	546.77	0	3001.85
6	1946	2481	2013-12-06 02:13:20	502	4	33	539	2504.98	29.96	267.96	2802.9
7	6197	3431	2013-12-06 12:55:22	53	559	315	927	264.47	4186.91	2557.8	7009.18

Expert Tip:-

- **Note that you should not include an alias when you write a subquery in a conditional statement. This is because the subquery is treated as an individual value (or set of values in the IN case) rather than as a table.**

- **Also, notice the query here compared a single value. If we returned an entire column IN would need to be used to perform a logical argument. If we are returning an entire table, then we must use an ALIAS for the table, and perform additional logic on the entire table.**

Examples:-

1. **Here is the necessary quiz to pull the first month/year combo from the orders table.**

```
    SELECT DATE_TRUNC('month', MIN(occurred_at))
    FROM orders;
```

2. **Then to pull the average for each, we could do this all in**

```
    SELECT AVG(standard_qty) avg_std, AVG(gloss_qty) avg_gls,
AVG(poster_qty) avg_pst
    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);
```

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

```
SELECT t3.rep_name, t3.region_name, t3.total_amt
FROM(SELECT region_name, MAX(total_amt) total_amt
     FROM(SELECT s.name rep_name, r.name region_name, SUM(o.total_amt_usd)
total_amt
             FROM sales_reps s
             JOIN accounts a
             ON a.sales_rep_id = s.id
             JOIN orders o
             ON o.account_id = a.id
             JOIN region r
             ON r.id = s.region_id
             GROUP BY 1, 2) t1
     GROUP BY 1) t2
JOIN (SELECT s.name rep_name, r.name region_name, SUM(o.total_amt_usd)
total_amt
     FROM sales_reps s
     JOIN accounts a
     ON a.sales_rep_id = s.id
     JOIN orders o
     ON o.account_id = a.id
     JOIN region r
     ON r.id = s.region_id
     GROUP BY 1,2
     ORDER BY 3 DESC) t3
ON t3.region_name = t2.region_name AND t3.total_amt = t2.total_amt;
```

4. **For the region with the largest sales total_amt_usd, how many total orders were placed?**

```
SELECT r.name, COUNT(o.total) total_orders
FROM sales_reps s
JOIN accounts a
ON a.sales_rep_id = s.id
JOIN orders o
ON o.account_id = a.id
JOIN region r
ON r.id = s.region_id
GROUP BY r.name
HAVING SUM(o.total_amt_usd) = (
      SELECT MAX(total_amt)
      FROM (SELECT r.name region_name, SUM(o.total_amt_usd) total_amt
              FROM sales_reps s
              JOIN accounts a
              ON a.sales_rep_id = s.id
              JOIN orders o
              ON o.account_id = a.id
              JOIN region r
              ON r.id = s.region_id
              GROUP BY r.name) sub);
```

5. **For the account that purchased the most (in total over their lifetime as a customer) standard_qty paper, how many accounts still had more in total purchases?**

```
SELECT a.name
FROM orders o
JOIN accounts a
ON a.id = o.account_id
GROUP BY 1
HAVING SUM(o.total) > (SELECT total
                  FROM (SELECT a.name act_name, SUM(o.standard_qty)
tot_std, SUM(o.total) total
                        FROM accounts a
                        JOIN orders o
                        ON o.account_id = a.id
                        GROUP BY 1
                        ORDER BY 2 DESC
                        LIMIT 1) sub);
```

6. **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;
```

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

```
SELECT AVG(tot_spent)
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 10) temp;
```

8. **What is the lifetime average amount spent in terms of total_amt_usd for only the companies that spent more than the average of all orders.**

```
SELECT AVG(avg_amt)
FROM (SELECT o.account_id, AVG(o.total_amt_usd) avg_amt
    FROM orders o
    GROUP BY 1
    HAVING AVG(o.total_amt_usd) > (SELECT AVG(o.total_amt_usd) avg_all
                                   FROM orders o)) temp_table;
```

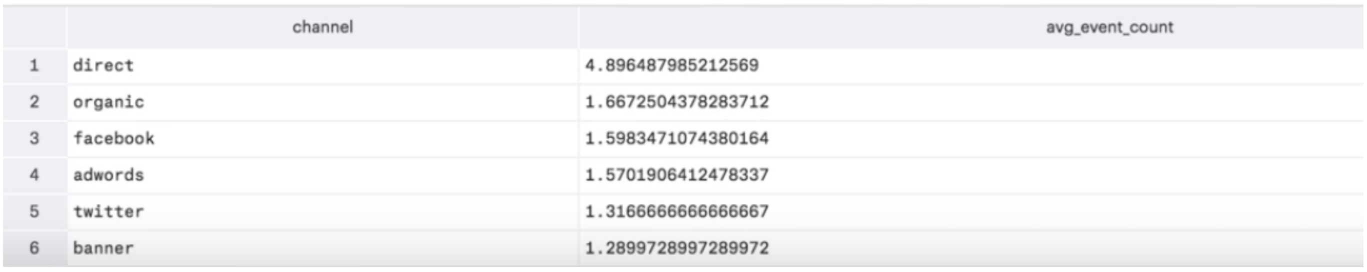
WITH:-

The WITH statement is often called a Common Table Expression or CTE. Though these expressions serve the exact same purpose as subqueries, they are more common in practice, as they tend to be cleaner for a future reader to follow the logic.

Illustration:-

- **One problem with subqueries is that they can make your queries lengthy and difficult to read. Common Table expressions or CTEs can help break your query into separate components so that your query logic is more easily readable. You can read the subqueries logic on its own and then read the final queries logic easily as well. Let's look back over this example in which we found the average number of events per day in each marketing channel.**

```
SELECT channel,
        AVG(events_counts) AS avg_event_count
FROM
        (SELECT DATE_TRUNC('day', occurred_at) AS day,
                channel,
                COUNT(*) AS event_count
        FROM demo.web_events
        GROUP BY 1,2)sub
GROUP BY 1
ORDER BY 2 DESC
```

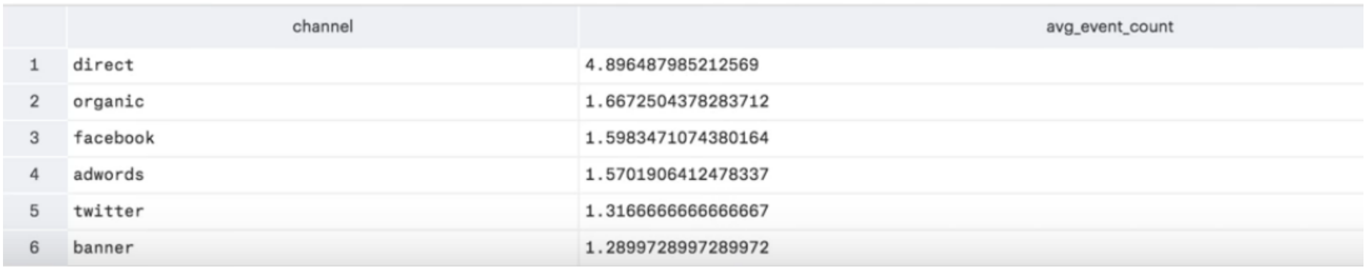


	channel	avg_event_count
1	direct	4.896487985212569
2	organic	1.6672504378283712
3	facebook	1.5983471074380164
4	adwords	1.5701906412478337
5	twitter	1.3166666666666667
6	banner	1.2899728997289972

- **Let's break the subquery out into his own common table expression which we'll create using the WITH command. Here, we're only defining this one common table expression but we could theoretically write as many as we want. We need to define any common table expressions at the beginning of the query in order to use them in our final query at the bottom. Each common table expression gets an alias just like a subquery.**

```
WITH events AS (SELECT DATE_TRUNC('day', occurred_at) AS day,
                channel,
                COUNT(*) AS event_count
        FROM demo.web_events
        GROUP BY 1,2)


SELECT channel,
        AVG(events_counts) AS avg_event_count
FROM events
GROUP BY 1
ORDER BY 2 DESC
```



	channel	avg_event_count
1	direct	4.896487985212569
2	organic	1.6672504378283712
3	facebook	1.5983471074380164
4	adwords	1.5701906412478337
5	twitter	1.3166666666666667
6	banner	1.2899728997289972

Examples:-

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

```
WITH t1 AS (
  SELECT s.name rep_name, r.name region_name, SUM(o.total_amt_usd)
total_amt
   FROM sales_reps s
   JOIN accounts a
   ON a.sales_rep_id = s.id
   JOIN orders o
   ON o.account_id = a.id
   JOIN region r
   ON r.id = s.region_id
   GROUP BY 1,2
   ORDER BY 3 DESC),
t2 AS (
   SELECT region_name, MAX(total_amt) total_amt
   FROM t1
   GROUP BY 1)
SELECT t1.rep_name, t1.region_name, t1.total_amt
FROM t1
JOIN t2
ON t1.region_name = t2.region_name AND t1.total_amt = t2.total_amt;
```

2. **For the region with the largest sales total_amt_usd, how many total orders were placed?**

```
WITH t1 AS (
   SELECT r.name region_name, SUM(o.total_amt_usd) total_amt
   FROM sales_reps s
   JOIN accounts a
   ON a.sales_rep_id = s.id
   JOIN orders o
   ON o.account_id = a.id
   JOIN region r
   ON r.id = s.region_id
   GROUP BY r.name),
t2 AS (
   SELECT MAX(total_amt)
   FROM t1)
SELECT r.name, COUNT(o.total) total_orders
FROM sales_reps s
JOIN accounts a
ON a.sales_rep_id = s.id
JOIN orders o
ON o.account_id = a.id
JOIN region r
ON r.id = s.region_id
GROUP BY r.name
HAVING SUM(o.total_amt_usd) = (SELECT * FROM t2);
```

3. **For the account that purchased the most (in total over their lifetime as a customer) standard_qty paper, how many accounts still had more in total purchases?**

```
WITH t1 AS (
  SELECT a.name account_name, SUM(o.standard_qty) total_std, SUM(o.total)
total
  FROM accounts a
  JOIN orders o
  ON o.account_id = a.id
  GROUP BY 1
  ORDER BY 2 DESC
  LIMIT 1),
t2 AS (
  SELECT a.name
  FROM orders o
  JOIN accounts a
  ON a.id = o.account_id
  GROUP BY 1
  HAVING SUM(o.total) > (SELECT total FROM t1))
SELECT COUNT(*)
FROM t2;
```

4. **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?**

```
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 1)
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 t1)
GROUP BY 1, 2
ORDER BY 3 DESC;
```

5. **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;
```

6. **What is the lifetime average amount spent in terms of total_amt_usd for only the companies that spent more than the average of all accounts.**

```
WITH t1 AS (
   SELECT AVG(o.total_amt_usd) avg_all
   FROM orders o
   JOIN accounts a
   ON a.id = o.account_id),
t2 AS (
   SELECT o.account_id, AVG(o.total_amt_usd) avg_amt
   FROM orders o
   GROUP BY 1
   HAVING AVG(o.total_amt_usd) > (SELECT * FROM t1))
SELECT AVG(avg_amt)
FROM t2;
```