ADVANCED QUERY TECHNIQUES

SUBQUERIES

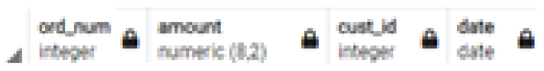
In practice, when doing data analysis one needs to use advanced SQL techniques that go beyond basic SELECT and JOIN commands.

Example:

An example of an advanced query is when you want to find the frequency distribution of the number of orders per customer. That requires an intermediate aggregation step that can be solved using the result of another query as an input.

First, I created a table:

```
CREATE TABLE  orders2(

ord_num int NOT NULL,

amount numeric(8,2),

date date,

cust_id int NOT NULL);
```



ord_num	amount	cust_id	date
integer	numeric (8,2)	integer	date

Then, I insert values into **orders2** that consist of records of order:

```
INSERT INTO orders2 (ord_num, amount, date, cust_id)

VALUES (1002,65.26, '2021-02-05',3002),

(1004,110.5, '2021-02-17',3009),

(1007,948.5, '2012-02-10',3005),

(1005,400.6, '2012-03-27',3007),
(1008,760,'2012-04-10',3002),

(1010,983.43,'2012-04-10',3004),

(1003,440.4, '2012-05-10',3009),

(1012,240.45, '2012-05-27',3008),

(1011,349.19, '2012-06-17',3003),

(1013,2045.6, '2012-07-25',3002);
```

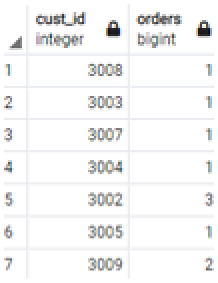
Alternatively, you can insert values into orders2 table from the attached csv file using the following SQL query:

(ps: I’m using PostgreSQL 12)

```
COPY orders2(ord_num, amount, date, cust_id)

FROM 'C:\Program Files\PostgreSQL\12\data\file.csv'

DELIMITER ',' CSV HEADER;
```

Next, to find the frequency distribution of the number of orders per customer, it requires an intermediate aggregation step that can be solved using **the following subquery that counts the number of orders placed by each customer:**

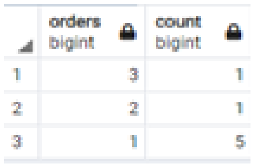
```
SELECT cust_id, COUNT(ord_num) as orders

FROM orders2

GROUP BY 1;
```



	cust_id integer	orders bigint
1	3008	1
2	3003	1
3	3007	1
4	3004	1
5	3002	3
6	3005	1
7	3009	2

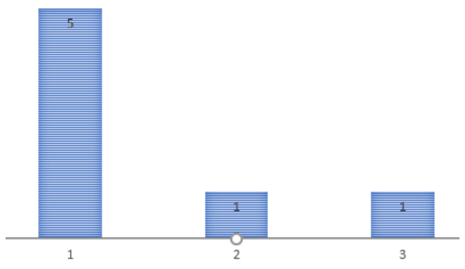
The outer query uses orders (as category) and counts the number of customers. The result is the following frequency distribution table:

```
SELECT orders, count(*)

FROM (select cust_id, count(ord_num) as orders

FROM orders2

GROUP by 1) a

GROUP BY 1;
```



	orders bigint	count bigint
1	3	1
2	2	1
3	1	5

To create a histogram of the number of orders per customer using the frequency distribution table, you can use any data visualization tool (Tableau, Excel):

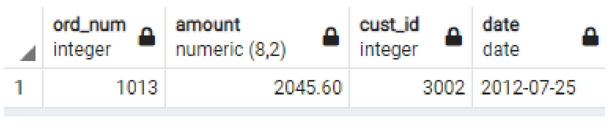


Subqueries are useful is filter data when you don't know the value for comparison.

Example:

To get the top 10% of customers according to the amount spent, you can use a subquery to generate the values in the WHERE clause.

```
SELECT * FROM orders2

WHERE amount > (

SELECT percentile_cont(.9) WITHIN GROUP (ORDER BY amount)

FROM orders2)

  ORDER BY amount DESC;
```

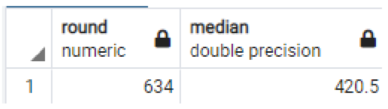


	ord_num integer	amount numeric (8,2)	cust_id integer	date date
1	1013	2045.60	3002	2012-07-25

Subqueries can be used to generate new columns:

Example:

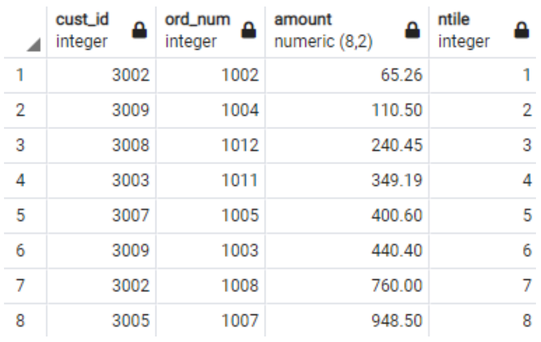
```
SELECT round(avg(amount),0),

(SELECT percentile_cont(.5) WITHIN GROUP (ORDER BY amount) as median FROM orders2)

FROM orders2;
```



	round numeric	median double precision
1	634	420.5

Another example:

```
SELECT cust_id, ord_num, amount, ntile(10) over (order by amount) as ntile

   FROM orders2;
```



	cust_id integer	ord_num integer	amount numeric (8,2)	ntile integer
1	3002	1002	65.26	1
2	3009	1004	110.50	2
3	3008	1012	240.45	3
4	3003	1011	349.19	4
5	3007	1005	400.60	5
6	3009	1003	440.40	6
7	3002	1008	760.00	7
8	3005	1007	948.50	8

In practice ntiles()are used to bin records. Ntiles distribute rows into equal groups assigning the bucket number in an **“equiheight” histogram**.

Example:

```
SELECT ntile, min(amount),count(ord_num) as orders

FROM

(SELECT cust_id, ord_num, amount, ntile(10) over (order by amount) as ntile

   FROM orders2) a

GROUP BY 1;
```

As you can see, subqueries are a useful tool in data analysis with SQL. You can even join subqueries as ‘derived’ tables to perform several preprocessing steps that can used in an outer query.