Subqueries

by Sophia

WHAT'S COVERED

This tutorial explores subqueries and using them in SELECT statements in three parts:

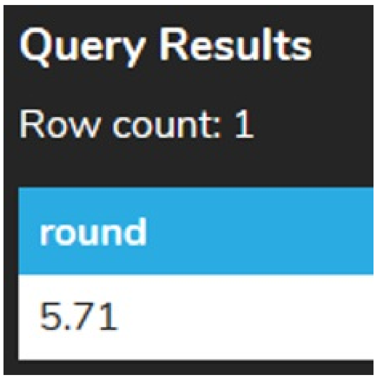
1. Subquery Basics
2. Considering Subquery Results
3. Adding Multiple Columns

1. Subquery Basics

There are many instances when we may want to use subqueries to create more complex types of queries. Very frequently, we will see this with aggregate calculations. For example, we may want to find all of the invoices that have the total amount larger than the average total amount.

We could do this in two separate steps, based on what we've learned so far. We could first calculate the average total:

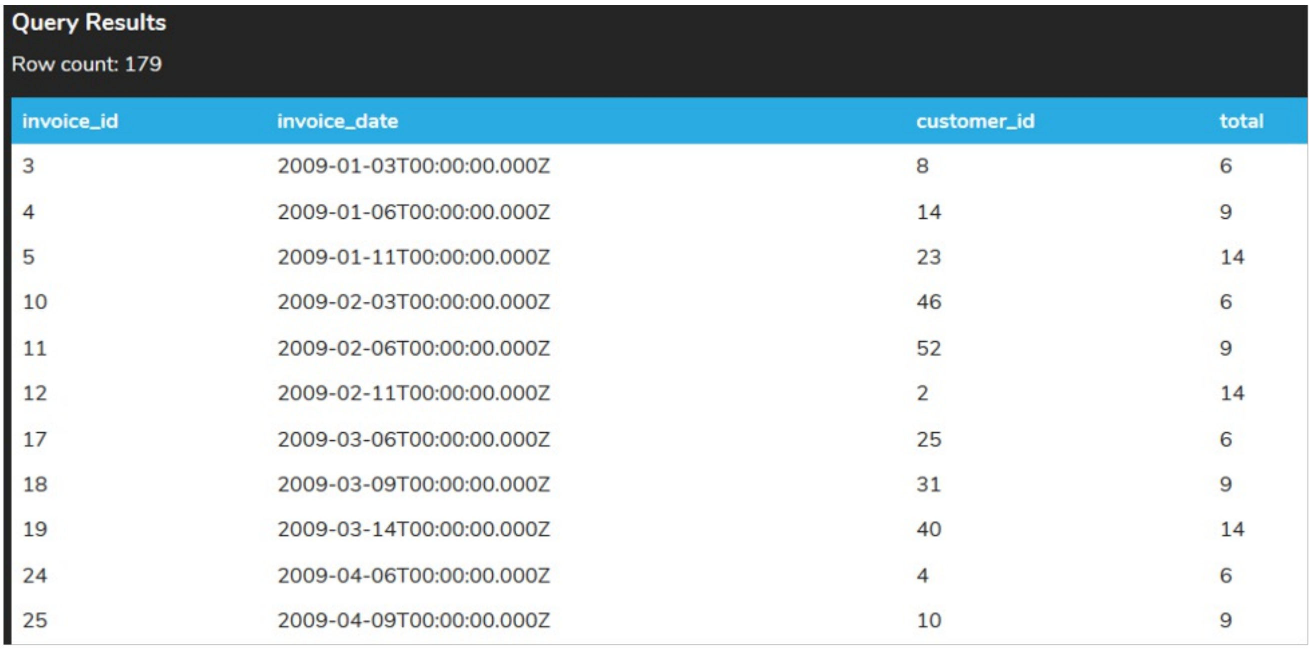
```
SELECT ROUND(AVG(total),2)
FROM invoice;
```



```
Query Results
Row count: 1
round
5.71
```

Next, we can take that value and query the invoice table to find those that are larger than 5.71.

```
SELECT invoice_id, invoice_date,customer_id, total
FROM invoice
WHERE total > 5.71;
```


Query Results

Row count: 179

invoice_id	invoice_date	customer_id	total
3	2009-01-03T00:00:00.000Z	8	6
4	2009-01-06T00:00:00.000Z	14	9
5	2009-01-11T00:00:00.000Z	23	14
10	2009-02-03T00:00:00.000Z	46	6
11	2009-02-06T00:00:00.000Z	52	9
12	2009-02-11T00:00:00.000Z	2	14
17	2009-03-06T00:00:00.000Z	25	6
18	2009-03-09T00:00:00.000Z	31	9
19	2009-03-14T00:00:00.000Z	40	14
24	2009-04-06T00:00:00.000Z	4	6
25	2009-04-09T00:00:00.000Z	10	9

Although this approach works, we can also do it in a way that can pass the result from the first query (that calculated the average) into the second query by using a subquery. The subquery can be nested in a SELECT, INSERT, DELETE, or UPDATE statement, but we mostly see it in a SELECT statement.

To construct the subquery, we would enclose the second query in round brackets and place it in the WHERE clause as an expression. So, in our original second statement:

```
SELECT invoice_id, invoice_date,customer_id, total
FROM invoice
WHERE total > 5.71;
```

We would remove the 5.71 and replace it with round brackets enclosing the first statement, like this:

```
SELECT invoice_id, invoice_date,customer_id, total
FROM invoice
WHERE total > (SELECT ROUND(AVG(total),2) FROM invoice);
```

The query inside the round brackets is called the subquery, or inner query. The query that contains the subquery is called the outer query. Notice that we don’t include the semi-colon at the end of the inner statement in the subquery.

The database would execute the subquery first, then take the result from that subquery and pass it to the outer query. Then, the database would execute the outer query.

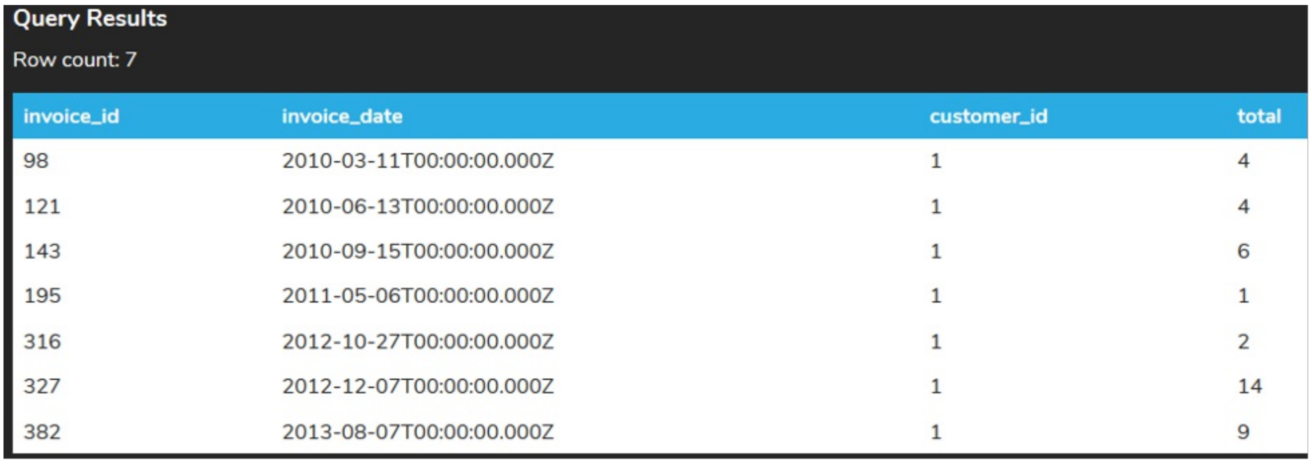
2. Considering Subquery Results

A subquery could potentially return 0 or more results, so it is important to think about the operator that is used to compare with the results. For example, if we use a = operator, we would expect that the subquery would return 0 or 1 row as a result.

Let's look at an obvious subquery that is querying on the customer_id to return the customer_id:

```
SELECT invoice_id, invoice_date,customer_id, total
FROM invoice
WHERE  customer_id =
    (SELECT customer_id
     FROM customer
     WHERE customer_id = 1);
```

Since the primary key of the table is the customer_id, querying on the customer_id would only return one value.



Query Results

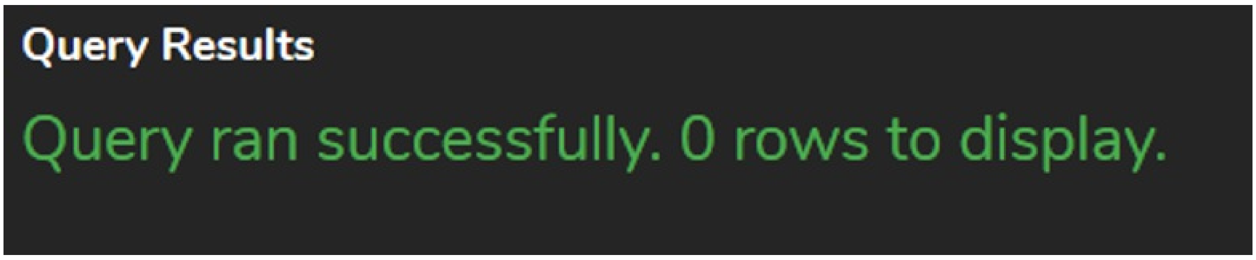
Row count: 7

invoice_id	invoice_date	customer_id	total
98	2010-03-11T00:00:00.000Z	1	4
121	2010-06-13T00:00:00.000Z	1	4
143	2010-09-15T00:00:00.000Z	1	6
195	2011-05-06T00:00:00.000Z	1	1
316	2012-10-27T00:00:00.000Z	1	2
327	2012-12-07T00:00:00.000Z	1	14
382	2013-08-07T00:00:00.000Z	1	9

If we tried to select a value that doesn't exist:

```
SELECT invoice_id, invoice_date,customer_id, total
FROM invoice
WHERE  customer_id =
    (SELECT customer_id
     FROM customer
     WHERE customer_id = 0);
```

The results would still run, but show that 0 rows were displayed:



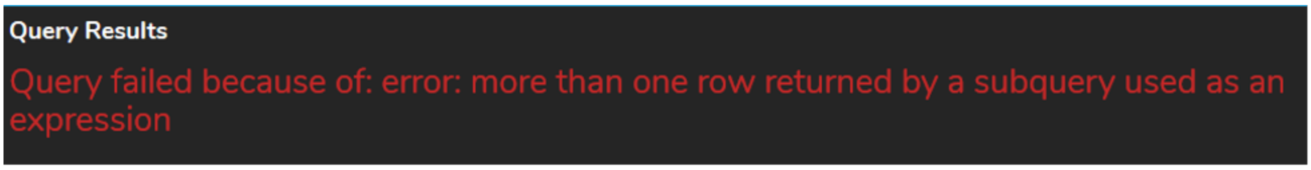
Query Results

```
Query ran successfully. 0 rows to display.
```

However, in the case that we have multiple rows being returned:

```
SELECT invoice_id, invoice_date,customer_id, total
FROM invoice
WHERE  customer_id =
    (SELECT customer_id
     FROM customer
     WHERE country = 'USA');
```

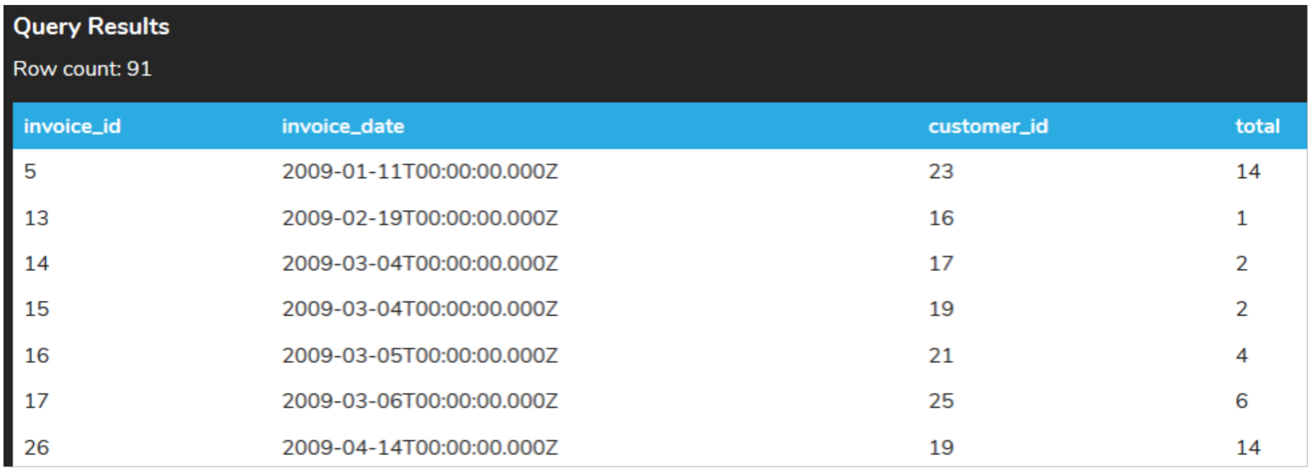
Since there are 13 customers that live in the country USA, we will end up getting this error:



```
Query Results
Query failed because of: error: more than one row returned by a subquery used as an expression
```

In order to avoid this error, we have to use the IN operator instead of the equal sign. This will allow 0, 1, or many results to be returned:

```
SELECT invoice_id, invoice_date, customer_id, total
FROM invoice
WHERE customer_id IN
    (SELECT customer_id
     FROM customer
     WHERE country = 'USA');
```



Query Results

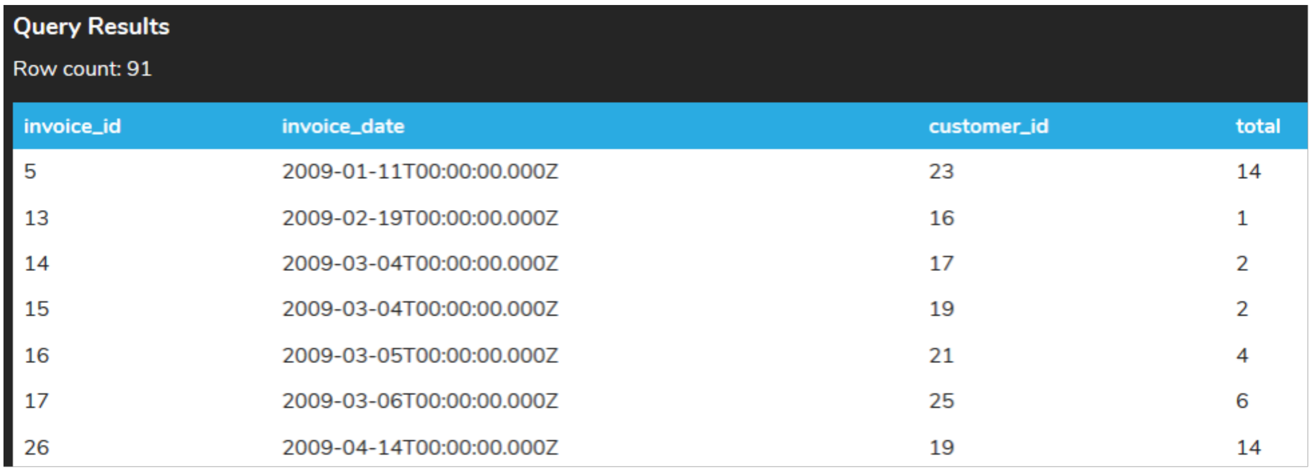
Row count: 91

invoice_id	invoice_date	customer_id	total
5	2009-01-11T00:00:00.000Z	23	14
13	2009-02-19T00:00:00.000Z	16	1
14	2009-03-04T00:00:00.000Z	17	2
15	2009-03-04T00:00:00.000Z	19	2
16	2009-03-05T00:00:00.000Z	21	4
17	2009-03-06T00:00:00.000Z	25	6
26	2009-04-14T00:00:00.000Z	19	14

3. Adding Multiple Columns

We can also add in multiple columns as criteria to compare with the subquery. In order to do so, we must use the round brackets around the columns within the WHERE clause and have them match up with the columns that we want to compare within the subquery. For example, if we wanted to compare the customer_id and the billing_country in the invoice table with the customer_id and country in the customer table, we could do the following:

```
SELECT invoice_id, invoice_date,customer_id, total
FROM invoice
WHERE  (customer_id,billing_country) IN
    (SELECT customer_id, country
     FROM customer
     WHERE country = 'USA');
```

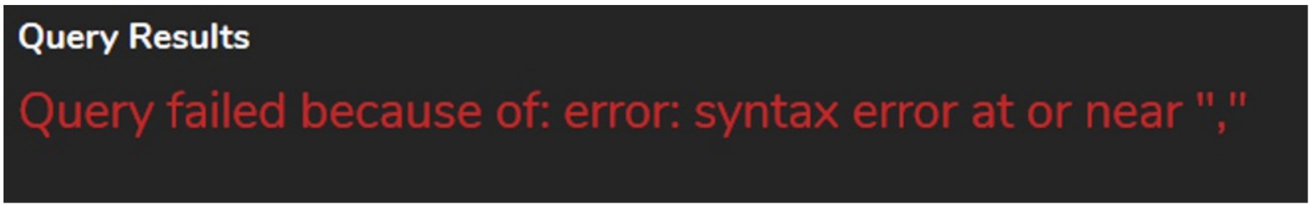

Query Results

Row count: 91

invoice_id	invoice_date	customer_id	total
5	2009-01-11T00:00:00.000Z	23	14
13	2009-02-19T00:00:00.000Z	16	1
14	2009-03-04T00:00:00.000Z	17	2
15	2009-03-04T00:00:00.000Z	19	2
16	2009-03-05T00:00:00.000Z	21	4
17	2009-03-06T00:00:00.000Z	25	6
26	2009-04-14T00:00:00.000Z	19	14

If we did not include the round brackets around the customers to be compared to, we would get an error:

```
SELECT invoice_id, invoice_date,customer_id, total
FROM invoice
WHERE  customer_id, billing_country IN
   (SELECT customer_id, country
    FROM customer
    WHERE country = 'USA');
```


Query Results

```
Query failed because of: error: syntax error at or near ","
```

Video Transcription

[MUSIC PLAYING] There are many instances in which we can make use of sub-queries. The use of sub-queries can be quite helpful to find information where you have multiple different queries combined together, and then you combine those queries together to simplify what you're looking for. So in this example here, what we want to try to find are all the invoices that have a total that's greater than the average invoice. So in order to do this with separate queries, you might want to first identify what the average amount is.

In this case here, we're going to look to see that it's going to be 5.71. We take note of that, and then we can go ahead and make a modification. To have a separate query in this case here, that's going to be taking that 5.71, and then checking, getting the invoice ID, the invoice data, customer ID as well as total from the invoice table where the total is greater than 5.71. Go ahead and execute that, and I'll provide us with that information.

However, utilizing sub-queries to combine the two queries together so that the information can derive directly from the other piece. The only thing we need to do is take the two queries and what we're going to do is replace that value inside of round brackets with the other query. Now if we just sit like that and run that query, we'll find the exact same amounts. However, we don't have to run those two separate queries separately, rather together and then run them as sub-queries.

[MUSIC PLAYING]

TRY IT

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

SUMMARY

Subqueries can be used to create more complex queries that are combined together.

Source: Authored by Vincent Tran