Table of Contents

1. Intro

Much of this will be review but review is good as we are going into more complex subqueries. This discussion will also discuss techniques that people often have difficulties with as we use subqueries. We are focusing on the use of subqueries in the Where clause of a Selects query.

Terms: I will use the term **main query** for the **top level query**. In the following query(demo 01) the Select. . . From Customer.customers is the main query; this part of the query determines the columns that can be exposed by the query in the Select and used in the main Where clause and also in the main Order By clause.

The query Select … From OrderEntry.orderH eaders is the **subquery**.

Demo 01: A simple subquery

```
select customer_name_last
from Customer.customers
where customer_id in (
    select customer_id
    from OrderEntry.orderHeaders
   );
+--------------------+
| customer_name_last |
+--------------------+
| McGold             |
| Morse              |
| Northrep           |
| Morise             |
| Williams           |
| Otis               |
. . .
```

2. Nesting subqueries

For many of the demos I will use a single level of subquery. But you are not limited to a single level. You can nest subqueries.

Demo 02: This uses two subqueries to display customers who ordered a specific product- product id 1130

```
select customer_name_last
, customer_name_first
, customer_id
from Customer.customers
where customer_id IN (
   select customer_id
   from OrderEntry.orderHeaders
   where order_id  IN (
      select order_id
```

```
        from OrderEntry.orderDetails
        where prod_id = 1020
        )
   )
order by customer_id;
+--------------------+---------------------+-------------+
| customer_name_last | customer_name_first | customer_id |
+--------------------+---------------------+-------------+
| Northrep           | William             |      401890 |
| Williams           | Sally               |      403000 |
| Button             | D. K.               |      404100 |
| Williams           | Al                  |      404900 |
| Clay               | Clem                |      408770 |
| Martin             | Joan                |      409150 |
+--------------------+---------------------+-------------+
```

Let's take this query step by step. With the subqueries we are using now, we can start with the inner most query. This query uses the order details table and gets all of the order id for orders for a particular product

```
        select order_id
        from OrderEntry.orderDetails
        where prod_id = 1020;
```

These are the order id values for that product.

```
+----------+
| order_id |
+----------+
|      105 |
|      405 |
|      414 |
|      519 |
|      520 |
|      529 |
|      605 |
|      608 |
|      716 |
|     3516 |
|     3808 |
+----------+
```

When we use this as a subquery with an IN list test, this output will be used as a list: (105,405,414,519,520,529,605,608, 716, 3516, 3808).

So this is effectively what the query is doing.

```
        select customer_id
        from OrderEntry.orderHeaders
        where order_id  IN (105,405,414,519,520,529,605,608, 716,3516, 3808);
```

This is the two level subquery. Note that the subquery is enclosed within parentheses.

```
        select customer_id
        from OrderEntry.orderHeaders
        where order_id  IN (
             select order_id
             from OrderEntry.orderDetails
             where prod_id = 1020)  ;
+-------------+
| customer_id |
+-------------+
|      403000 |
|      408770 |
|      409150 |
|      401890 |
|      404900 |
|      403000 |
|      404100 |
|      403000 |
|      409150 |
|      409150 |
|      403000 |
+-------------+
```

The result is again used as an IN list (403000, 408770,409150, 409150, 401890,404900,40300, 404100, 403000, 409150, 403000). Note that the 2-level subquery has the customer id 403000 appearing several times in the result. This is not a problem. Some people use a Distinct in the subquery but that is not needed and in some systems can make the query less efficient since it would need to do extra work to remove duplicates.

Now we are up to the top level of the original query which is effectively doing

```
select customer_name_last
, customer_name_first
, customer_id
from Customer.customers
where customer_id IN (403000, 408770,409150, 409150, 401890,404900,40300,
404100, 403000, 409150, 403000);
```

You probably have realized that we could also do this as a three table join.

Demo 03: This is a three table join and we need distinct to remove duplicates. We did not need distinct in the subquery since the top level table expression is just the customer table and each customer appears only once in that table.

```
select Distinct
  CS.customer_name_last
, CS.customer_name_first
,customer_id
from Customer.customers CS
join OrderEntry.orderHeaders  OH using(customer_id)
join OrderEntry.orderDetails  OD using (order_id)
where OD.prod_id = 1020
order by customer_id;
```

Demo 04: Find customers who have ordered any appliance - using the category id APL

```
This is in the demo- but try to figure it out for yourself first. Reading
answers is not as helpful as discovering solutions.
```

3. Testing with equality tests

If the subquery is guaranteed to return a single row and a single column, then we can test the subquery result with the equality operators (and with the operators <>, >=, >, <=, <).

We sometimes call this a scalar subquery; but we know that every result set is a table- what we are saying is that this subquery will return a table with a single row and a single column.

We can ensure a single column by having only one column expression in the Select clause of the subquery. How do we guarantee that the subquery returns exactly one row? We can filter the subquery on a column that is declared as Unique in its table- often this will be the PK column(s). Sometimes we see examples of queries that return only one row for the current set of data- but that is not enough for robust code. We need to use a subquery that will **always** return a single row of data for any set of rows in the table.

Demo 05: We can use a subquery with customer_id = since the inner query returns one row and one column. This is filtering on the PK of the order headers table so we know we get one customer id.

```
select customer_id, customer_name_last, customer_name_first
from Customer.customers
where customer_id = (
   select customer_id
   from OrderEntry.orderHeaders
   where order_id  = 115)
;
+-------------+--------------------+---------------------+
| customer_id | customer_name_last | customer_name_first |
+-------------+--------------------+---------------------+
|      402100 | Morise             | William             |
+-------------+--------------------+---------------------+
```

We know that in the order table, the order_id is the pk. So we can have only one row in the order table with order_id 115. The subquery will return the single customer _id for that order. We can use that returned value in the Where clause of the parent query to find that customer's information in the customer table.

3.1. Guaranteeing a scalar result

This is an area where there is some confusion, particularly in assignments where your subquery runs and gives what seems to be a correct answer but the query is still incorrect.

Suppose we want to find all customers who ordered a particular product. We decide to set a variable for the product id. Then we are going to use a subquery and do a join of two tables in the subquery.

Demo 06: This runs and produces a single row of output with my current set of data

```
set @prod_id   := 5004;

select customer_id, customer_name_last, customer_name_first
from Customer.customers
where customer_id = (
   select customer_id
   from OrderEntry.orderHeaders  OH
   join OrderEntry.orderDetails  OD on OH.order_id = OD.order_id
   where prod_id  =@prod_id);
+-------------+--------------------+---------------------+
| customer_id | customer_name_last | customer_name_first |
+-------------+--------------------+---------------------+
|      403100 | Stevenson          | James               |
+-------------+--------------------+---------------------+
```

But if you change the value of the variable to 1080 and rerun the query you get an error.

ERROR 1242 (21000): Subquery returns more than 1 row

Why does this happen? With the product id 5004 we have only one order so the subquery returns one customer id. With the product id 1080 we have several orders so the subquery returns several customer ids. That causes the test customer_id = filter to crash, The query is not logically correct since it assumes a single order and that assumption is not supported by the table design.

3.2. Subquery that returns no rows

Suppose we want to ask the following question about employees and managers. We want a list of all of the employees who have the same manager as a specific employee.

First we will use a variable for the employee id we are using. The subquery gets the manager ID for that employee. This subquery returns only a single value since one employee has at most one manager in the definition of our tables. That id is passed up to the outer query which lists all people managed by that person and skips the employee with the originally specified id.

Demo 07: Who is managed by the same person who manages employee 145?

```
set @empId  := 145;

select emp_id
from employee.employees
where emp_id  <> @empId
and emp_mng  = (
   select emp_mng
   from Employee.employees
   where emp_id  = @empId );
+--------+
| emp_id |
+--------+
|    101 |
|    102 |
|    146 |
|    201 |
+--------+
```

Demo 08: Now we change the employee ID and we do not get any rows returned. We do have an employee 100 but he does not have a manager.

```
set @empId  = 100;

select emp_id
from employee.employees
where emp_id  <> @empId
and emp_mng  = (
   select emp_mng
   from Employee.employees
   where emp_id  = @empId  );
```

```
Empty set (0.00 sec)
```

Demo 09: This ID value also does not return any rows. We do not have an employee with ID 408

```
set @empId  := 408;
select emp_id
from Employee.employees
where emp_id  <> @empId
and emp_mng  = (
   select emp_mng
   from Employee.employees
   where emp_id  = @empId  );
```

```
Empty set (0.00 sec)
```

We cannot distinguish these two conditions from the output.

```
              from Product.products  ) - 100
```

4. Testing with the In List filter

Suppose we try to run the following query. The subquery can return multiple rows and the query fails. So we should not test this with an equals test, but we can use an IN list test. We are not saying that the subquery must return multiple rows. We cannot determine that by looking at the query. It is possible that our current set of data has no orders dated 2015-10-01; there could be exactly one such order, or there could be many such orders and the subquery returns multiple rows. We need to write queries that work with any valid set of data in the table- not with a particular collection of rows.

Demo 10: This query fails

```
select customer_id, customer_name_last, customer_name_first
from Customer.customers
where customer_id  = (
    select customer_id
    from OrderEntry.orderHeaders
    where order_date  = '2015-10-01');
```

If you run a query with a subquery that returns multiple rows and you use an equality test, you get an error message.

```
ERROR 1242 (21000): Subquery returns more than 1 row
```

Demo 11: Changing to an In list for the subquery

```
select customer_id, customer_name_last, customer_name_first
from Customer.customers
where customer_id IN (
    select customer_id
    from OrderEntry.orderHeaders
    where order_date  = '2015-10-01');
+-------------+--------------------+---------------------+
| customer_id | customer_name_last | customer_name_first |
+-------------+--------------------+---------------------+
|      401250 | Morse              | Samuel              |
|      403000 | Williams           | Sally               |
+-------------+--------------------+---------------------+
```

Demo 12: Suppose we run the following query with a date that does not match any orders; then we do not get any rows in the result set. Remember this is not an error- we simply do not have any such data.

```
select customer_id, customer_name_last, customer_name_first
from Customer.customers
where customer_id IN (
   select customer_id
   from OrderEntry.orderHeaders
   where order_date  = '1888-08-08');
```

```
Empty set (0.00 sec)
```

Demo 13: If we change the query to a NOT IN query, then all customers are returned with our data set

```
select customer_id, customer_name_last, customer_name_first
from Customer.customers
where customer_id  NOT IN (
   select customer_id
   from OrderEntry.orderHeaders
   where order_date  = '1888-08-08');
+-------------+--------------------+---------------------+
| customer_id | customer_name_last | customer_name_first |
+-------------+--------------------+---------------------+
|      400300 | McGold             | Arnold              |
|      400801 | Washington         | Geo                 |
|      401250 | Morse              | Samuel              |
|      401890 | Northrep           | William             |
|      402100 | Morise             | William             |
|      402110 | Coltrane           | John                |
```

5. Subqueries versus Joins

The next two queries both filter for orders in Dec 2015- one returns 8 rows and the other query returns 5. You need to know the business needs for the query to determine which is the correct query.

Demo 14: This does not use a subquery. It uses a join. Depending on our needs, we might want to include a Distinct in the select clause.

```
select Customer.customers.customer_id
, customer_name_last, customer_name_first
from Customer.customers
join OrderEntry.orderHeaders on Customer.customers.customer_id=
OrderEntry.orderHeaders.customer_id
where month (order_date) = 12 and year(order_date) = 2015
;
+-------------+--------------------+---------------------+
| customer_id | customer_name_last | customer_name_first |
+-------------+--------------------+---------------------+
|      409030 | Mazur              | Barry               |
|      409030 | Mazur              | Barry               |
|      409150 | Martin             | Joan                |
|      409160 | Martin             | Jane                |
|      409160 | Martin             | Jane                |
|      409190 | Prince             | NULL                |
|      915001 | Adams              | Abigail             |
|      915001 | Adams              | Abigail             |
+-------------+--------------------+---------------------+
```

Demo 15: This uses a subquery. Use customer_id IN since the inner query can return multiple rows. This will bring back only one row per customer even if they have multiple orders in Dec 2015. This query cannot show the order_date since it is not a column in the parent query.

```
select customer_id, customer_name_last, customer_name_first
from Customer.customers
where customer_id IN (
   select customer_id
   from OrderEntry.orderHeaders
   where month (order_date) = 12 and year(order_date) = 2015);
+-------------+--------------------+---------------------+
| customer_id | customer_name_last | customer_name_first |
+-------------+--------------------+---------------------+
|      409030 | Mazur              | Barry               |
|      409150 | Martin             | Joan                |
|      409160 | Martin             | Jane                |
|      409190 | Prince             | NULL                |
|      915001 | Adams              | Abigail             |
+-------------+--------------------+---------------------+
```

6. Subqueries for unmatched rows

We used outer joins and tested for null values to find unmatched rows- such as customers with no orders. We can also do this with subqueries. Many people find the Not In subquery approach easier to understand than the outer join.

Demo 16: This is an outer join query that returns customers who have no orders.

```
select CS.customer_id, CS.customer_name_last
from Customer.customers  CS
left join OrderEntry.orderHeaders OH on CS.customer_id = OH.customer_id
where OH.order_id is null;
+-------------+--------------------+
| customer_id | customer_name_last |
+-------------+--------------------+
|      400801 | Washington         |
|      402110 | Coltrane           |
|      402120 | McCoy              |
|      402500 | Jones              |
|      403500 | Stevenson          |
|      403750 | O'Leary            |
|      403760 | O'Leary            |
|      404150 | Dancer             |
|      404180 | Shay               |
|      404890 | Kelley             |
|      408777 | Morise             |
|      409010 | Morris             |
|      409020 | Max                |
+-------------+--------------------+
13 rows in set (0.01 sec)
```

Demo 17: This is a subquery to accomplish the same task.

```
select Customer.customers.customer_id, customer_name_last
from Customer.customers
where customer_id not in (
   select customer_id
   from OrderEntry.orderHeaders );
```

Demo 18: Do we have any order rows for which there are no associated order detail rows?

```
select Order_id, order_date
from OrderEntry.orderHeaders
where order_id not in (
    select order_id
    from OrderEntry.orderDetails );
+----------+---------------------+
| Order_id | order_date          |
+----------+---------------------+
|      116 | 2015-11-12 00:00:00 |
|      123 | 2015-12-05 00:00:00 |
|     2506 | 2016-01-12 00:00:00 |
+----------+---------------------+
3 rows in set (0.00 sec)
```