Table of Contents

The use of subqueries is a technique that we will develop over the semester. In this document we look at some queries that have a main query and a subquery that is used in a test in the Where clause. I am going to start with a demo and then develop the rules for the subqueries.

1. Demo

These use the Altgeld Mart database. Suppose we want to find all of the customers who have any orders. We could do this with a join.

Demo 01: Inner join- this gives 97 rows with my current data set

```
   select cs.customer_id, customer_name_last
   from cust_customers cs
   join oe_orderHeaders oh on  cs.customer_id = oh.customer_id;
```

If I add the Distinct keyword then I get one row per customer who has an order- 21 rows.

```
   select DISTINCT cs.customer_id, customer_name_last
   from cust_customers cs
   join oe_orderHeaders oh on  cs.customer_id = oh.customer_id;
```

Demo 02: I could also do this with a subquery. This returns 21 rows, one row per customer who has an order.

```
   select customer_id, customer_name_last
   from cust_customers
   where customer_id in (
      select customer_id
      from oe_orderHeaders);
```

The subquery is:

```
    (select cust_id from oe_ordeHeaders)
```

The subquery is executed and returns a list of all of the customer_id values from the order headers table- a list of customers with orders. That list is used by the IN filter so that we only get customers from the customer table who have an order. I do not have to use Distinct in either the main query or in the subquery. If a customer has more than one order, they still will appear only once in the result because the main query is pulling from the customers table.

I could display the order_date in the queries with the join, but I cannot display the order_date in the query using a subquery.

2. Terminology and some general rules

A subquery is a Select query that is enclosed in parentheses and is used within another statement. Subqueries can be used in Select, Insert, Update or Delete statements. For now, we will limit the discussion to subqueries in Select statements.

You can use the term **main query** for the top level Select. You can also refer to these two Selects as the **inner query**(the subquery) and the **outer query.** You can nest subqueries to many levels (255 levels in Oracle).

If a table appears only in a subquery and not in the outer query, then columns from the table in the inner query cannot be included in the output (the select list of the outer query).

A subquery can return 1 or more columns and 0 or more rows. But for certain uses of subqueries you need to restrict the subquery output.

.

3. Subquery using an IN list test

Suppose we want to find all of the orders for sporting goods items; we want a list of the order_id values for an order that included one or more sporting goods items.

We could do this with an inner join.

Demo 03: a query to find the order id of all orders that include sporting goods items.

```
select distinct ord_id
from oe_order_details OD
join prd_products  PR on OD.prod_id = PR.prod_id
where catg_id = 'SPG'
order by ord_id;
```

```
    ORD_ID
----------
       105
       106
       117
       120
       121
       128
. . .  rows omitted
```

Demo 04: We can also do this with a subquery that finds a list of product ids that are sporting goods from the products table and then delivers that list to the outer query which finds those product ids in the order_details table.

```
select DISTINCT order_id
from oe_orderDetails
where prod_id in (
  select prod_id
  from prd_products
  where catg_id = 'SPG'
)
order by order_id
;
```

Since there is more than one product id for sporting goods, we need to use the IN list syntax.

For this query we do want to use Distinct in the main query. We could have an order that includes more than one sporting goods items and our description said we wanted the order_id values for an order than included one or more sporting goods items. If we leave off the word Distinct in the outer select, then an order id will appear multiple times if a single order includes more than one sporting goods item line.

With the subquery approach, the only attributes that can be displayed are those found in the tables in the From clause of the main query. We could display attributes from the order detail tables but not the attributes from the products table.

If you compare the previous two query you can see that the attribute prod_id is used in the joins and also as the matching attribute in the subquery test. We are still "joining" the two tables based on the product id.

One of the nice thing about these subqueries is that they are stand alone queries that we could run separately to check our syntax and logic.

Demo 05: Suppose we ran this query first; we would get a row for each sporting goods item in the products table.

```
select prod_id, catg_id
from prd_products
where catg_id = 'SPG'
```

```
   PROD_ID CATG_I
---------- ------
      1010 SPG
      1020 SPG
      1030 SPG
      1040 SPG
      1050 SPG
      1060 SPG

6 rows selected.
```

Demo 06: But if we used that Select directly as a subquery we get an error.(Not one of Oracle's best error messages!)

```
select DISTINCT order_id
from oe_orderDetails
where prod_id in (
  select prod_id, catg_id
  from prd_products
  where catg_id = 'SPG'
)
order by order_id
;
```

```
ORA-00913: too many values
```

Demo 07: This finds sporting good products that were sold at their list price.

```
select order_id, prod_id
from oe_orderDetails
where (prod_id, quoted_price) in (
  select prod_id, prod_list_price
  from prd_products
  where catg_id = 'SPG'
)
order by order_id;
```

We are not limited in a subquery to a single table.

Demo 08: Suppose we want to see the customers who purchased a sporting goods item in Nov 2015. We could do this with a subquery.

```
select customer_id, customer_name_last
from cust_customers CS
where customer_id in (
   select customer_id
   from oe_orderHeaders OH
   join oe_orderDetails OD on OH.order_id  = OD.order_id
   join prd_products PR on OD.prod_id = PR.prod_id
   where to_char(order_date, 'YYYY-MM') = '2015-11'
   and catg_id = 'SPG');
```

Demo 09: first version using multiple subqueries(no joins). This has three subqueries. It does not yet filter on sporting goods or on the order date

```
select customer_id, customer_name_last
from cust_customers CS
where customer_id in (
   select customer_id
   from oe_orderHeaders OH
   where order_id IN (
      select order_id
      from oe_orderDetails OD
      where prod_id in (
         select prod_id
         from prd_products PR
      )
   )
);
```

Now we need to think about where to place the filters on sporting goods and on the order date.
The test on catg_id is simple; that attribute is in the products table so we put that filter in the last of the subqueries. `(select prod_id from prd_products PR where catg_id = 'SPG').`

The attribute for the order date is in the order headers table so we need the test at that level. I find it much easier to deal with the parentheses if I keep the simple filter before the In (subquery filter).

Demo 10: The test on order date is moved into the first subquery with a AND test

```
select customer_id, customer_name_last
from cust_customers CS
where customer_id in (
   select customer_id
   from oe_orderHeaders OH
   where to_char(order_date, 'YYYY-MM') = '2015-11'
   and order_id IN (
      select order_id
    from oe_orderDetails OD
    where prod_id in (
       select prod_id
       from prd_products PR
       where catg_id = 'SPG'
       )
    )
);
```

It is possible to run this query with the order_date filters in the innermost subquery. The inner subquery can reference columns in the outer queries. This can cause confusion when an identifier appears at more than one level (SQL does not get confused- but you might). Generally avoid these types of tests unless you are very careful.

These examples have use the filter IN (list) . You should also consider using filters for NOT IN (list)'

Remember with the subquery approach, take care to consider if any of the subqueries would return a null. That requires special handling.

4. The ANY operator

This is another way to write the In list subquery test. It is not as common, but you may find it easier to read.

Demo 11: Compare to demo 04. We are finding order id values for orders that include a sporting goods item.

```
select distinct order_id
from oe_orderDetails
where prod_id = ANY (
   select prod_id
   from prd_products
   where catg_id = 'SPG'
   )
order by order_id;
```

Demo 12: Oracle also allows this operator with a simple list of values

```
   select prod_id , catg_id
   from prd_products
   where catg_id = ANY('SPG', 'MUS');
```

5. The comparison operators

5.1. subquery with an equality test

Demo 13: Suppose we want to find order id values for orders that include a sporting goods item and we use the following where we use an equality test with the subquery. This gives us an error message.

```
select distinct order_id
from oe_orderDetails
where prod_id = (
   select prod_id
   from prd_products
   where catg_id = 'SPG'
   )
order by order_id;
```

```
ORA-01427: single-row subquery returns more than one row
```

This is not allowed when the subquery follows a comparison operator: =, !=, <, <= , >, >= .

If you run the subquery by itself, you get 6 rows returned. So this query is trying to test if the prod_id value in the order details table equals 6 different values at the same time. An IN list test is actually an OR test so demo 4 tests if the prod_id value in the order details table equals the first value brought back by the subquery or the second value brought back by the subquery, etc. (This is also why you run into troubles when the subquery could contain a null.)

We do not want an equality test in the outer query if the subquery could return multiple rows.

Demo 14: What if the subquery returns no rows? The subquery returns an empty virtual table and the outer query finds no matches and also returns an empty table. This is not considered an error.

```
select distinct order_id
from oe_orderDetails
where prod_id = (
   select prod_id
   from prd_products
   where catg_id = 'QWE'
   )
order by order_id;
```

```
no rows selected
```

Suppose we want to find all of the people who work in the same department as employee 162. We could write a query to find that department id.

```
select dept_id
from emp_employees
where emp_id = 162
```

And then write a query to find employees in that department.

```
select emp_id
, name_last as "Employee"
, dept_id
from emp_employees
where dept_id =    <--  put the department number here
```

But we can also build these two queries into one query by using a subquery.

Demo 15: A simple subquery- we want to find employees who work in the same department as employee with ID 162 . Since we are looking at the data in the employee table in two different ways, both the main and the subquery use the employees table.

```
select emp_id, name_last as "Employee"
from emp_employees
where dept_id = (
    select dept_id
    from emp_employees
    where emp_id = 162
    )
;
```

```
     EMP_ID EMPLOYEE
----------- -------------------------
        162 Holme
        200 Whale
        207 Russ
```

The inner query, the subquery, is a complete query that could stand by itself. It returns a single value- the department id where this employee works. The subquery is enclosed in parentheses and the result of the subquery is used by the outer query to return the other employees from that department.

Demo 16: If we really want to return the **other** people from that department, we can eliminate employee 162 from the final result set. Note that the filter emp_id <> 162 is written outside the subquery so that it filters the final result set.

```
select emp_id, name_last as "Employee"
from emp_employees
where dept_id = (
   select dept_id
   from emp_employees
   where emp_id = 162
   )
and emp_id <> 162
;
```

```
    EMP_ID  EMPLOYEE
----------- ------------------------
       200 Whale
       207 Russ
```

With this filter we have to take care that the subquery returns a single value- one row and one column, because we are using it in an = test.

5.1. subquery with other comparison tests

We could use this technique to find employees who earn more than employee 162.

The subquery to find the salary for employee 162 would be

```
             select salary
             from emp_employees
             where emp_id = 162
```

Each employee has exactly one salary value and we have only one employee 162 since emp_id is the primary key. This subquery returns one value.

Demo 17: Using the subquery with an > test

```
   select emp_id, name_last as "Employee", salary
   from emp_employees
   where salary > (
      select salary
      from emp_employees
      where emp_id = 162
      );
```

```
    EMP_ID Employee                     SALARY
---------- ------------------------ ----------
       100 King                         100000
       101 Koch                          98005
       161 Dewal                        120000
       204 King                          99090
```

Try this query with some different employee id value- who earns more than employee 104? More than employee 161?

Demo 18: Now try this query where the subquery filters on the department id. You will get an error message

```
select emp_id, name_last as "Employee", salary
from emp_employees
where salary > (
   select salary
   from emp_employees
   where dept_id = 210
   )  ;
```

```
ORA-01427: single-row subquery returns more than one row
```

This is a good error message. The subquery is

```
select salary
from emp_employees
where dept_id = 210
```

We have two employees in dept 210, so the result of that subquery is

```
     SALARY
-----------
   69000.00
   55000.00

2 rows selected
```

So are we asking who earns more than 69000 or more than 5500? This is an invalid query; you cannot use an = or a > test against a subquery that returns more than one row.