Table of Contents

1. Subqueries with the Exists Predicate 1
2. Correlation and the Exists Predicate 5
3. Subqueries that produce report style output 10
4. For all query 12

Exists is a predicate that accepts a subquery as an argument and returns either True or False. Exists is a test for existence- does the subquery return any rows. Sometimes you do not care what is in the rows returned by a subquery- only if there are any such rows. For example- does this customer have any orders? We might not care about how many orders the customer might have or what is on those orders- we only want to know if there are any orders for the customer. Suppose the customer does have one or more orders- as soon as we find one order we have the answer and we do not need to look any further. So using Exists can be a more efficient way to return data.

Exists is commonly used with correlated subqueries but we will start with some non-correlated subqueries to see how this predicate works.

1. Subqueries with the Exists Predicate

This is another version of the zoo table (zoo_ex). The SQL for this is in the posted SQL. I have inserted blank lines between the animal types in the display.

```
+----+---------+----------+
| id | an_type | an_price |
+----+---------+----------+
|  4 | bird    |      100 |
|  5 | bird    |       50 |
| 15 | bird    |       80 |
| 17 | bird    |       80 |

|  8 | cat     |       10 |
| 16 | cat     |     NULL |

|  1 | dog     |       80 |

|  6 | fish    |       10 |
| 11 | fish    |       10 |
| 13 | fish    |       10 |

|  3 | lizard  |     NULL |
|  7 | lizard  |       50 |
| 12 | lizard  |       50 |

|  9 | snake   |       50 |
| 10 | snake   |     NULL |
| 14 | snake   |       25 |

|  2 | turtle  |     NULL |
+----+---------+----------+
17 rows in set (0.00 sec)
```

EXISTS is an operator that is used with subqueries. If the subquery brings back at least one row, then the Exists operator has the value True. If the subquery does not bring back any rows, then the Exists has the value False. Since we do not need any specific values brought back from the inner query, we can use Select 'X'- or any other literal for efficiency. All we need to know is if the subquery brings back any rows or not. Even if the subquery brings back rows that have null values, those values still exist. The Exists operator can be negated.

For this first few demos I need a table in the from clause of the select. MySQL allows the use of the syntax Select * from dual in a situation where you need a table name. In MySQL dual is not an actual table.

Demo 01: What does Exists do?

```
select 'Found'
from dual
where exists (
  Select 1
  from zoo_ex) ;
```

This first query might look strange since we are using the dual table in the outer query- but in MySQL you can use this technique to write a valid query without worrying about the table. And I want to emphasize what Exists actually does. If you run this query after you have created the table but before you have entered any rows, the result is no rows.

The subquery does not return any rows since the table is empty. Therefore Exists evaluates as False. The outer query is now: select 'Found' from dual Where False. So the outer query returns no rows.

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

If you have entered even a single row, then the query returns the following result. Exists lets the query know if there were any rows at all in the subquery's result set.

```
+-------+
| Found |
+-------+
| Found |
+-------+
```

The subquery returns one or more rows. Therefore Exists evaluates as True. The outer query is now: select 'Found' from dual Where True. So the outer query returns all the rows in its data source.

Demo 02: Now we make the subquery slightly more interesting by adding a filter. We do have a t least one row with a snake returned by the subquery.

```
select 'Found'
from dual
where exists (
  Select 1
  from zoo_ex
  where an_type='snake'
  )
;
+-------+
| Found |
+-------+
| Found |
+-------+
```

Demo 03: But if we filter for Penguin- we get no rows in the subquery and Exists returns False so the outer query returns no rows.

```
select 'Found'
from dual
where exists (
  Select 1
  from zoo_ex
  where an_type='penguin'
  )
;
Empty set (0.00 sec)
```

If we looks at the rows for an_type = 'snake' we will see that there is a row where the price is null. Suppose our subquery filters for rows for snakes with a null price and the subquery returns the price attribute. Even though that price is null we do get a row returned.

Demo 04: Snakes with a null price

```
  Select an_price
  from zoo_ex
  where an_type='snake'
  and an_price is null ;
+----------+
| an_price |
+----------+
|     NULL |
+----------+
```

Demo 05: So if we use that subquery with Exists, the outer query returns its rows

```
select 'Found'
from dual
where exists (
  select an_price
  from zoo_ex
  where an_type='snake'
  and an_price is null
  )
;
+-------+
| Found |
+-------+
| Found |
+-------+
```

We really do not care what attribute is used in the select in the subquery and we commonly use a literal- such as 1 or 'x'. You may see people use select * in the subquery- and most dbms will rewrite that for you. But using Select an_price or Select * suggests that you want all of the attributes or a specific attribute value and that is not the way that the Exists operator works.

Demo 06: If we run the previous query with a filter for 'bird' we get norows returned since we do not have any birds with a null price. This is not specific to nulls; you could change the filter to an_price - 80 or to an_price = 1250 to see if we have any rows matching that test. The important thing to remember with exists is that it returns True or False and not the sets of rows from the subquery.

```
select 'Found'
from dual
where exists (
  select an_price
  from zoo_ex
  where an_type='bird'
  and an_price is null
  )
;
Empty set (0.00 sec)
```

Now we will start to use the zoo_ex table in the outer query also.

Demo 07: If we have any birds that cost less than 75.00 then display all of the birds.
Since we have at least one row that meets that test, all of the rows for birds are returned by the outer query.

```
select *
from  zoo_ex
where an_type='bird'
and   exists (
  select 1
  from zoo_ex
  where an_type='bird'
  and an_price < 75
  )
;
+----+---------+----------+
| id | an_type | an_price |
+----+---------+----------+
|  4 | bird    |      100 |
|  5 | bird    |       50 |
| 15 | bird    |       80 |
| 17 | bird    |       80 |
+----+---------+----------+
```

Change the filter test to an_price < 15 and no rows are returned.

This might seem rather impractical but suppose your tables dealt with a manufacturing process for medicines and these was a 'quality' attribute which indicated how well this batch passed its quality tests. You might want to run a query that says if any batch run on a particular day failed its tests then display all of the batches run that day.

You can negate the exists test- but the logic is harder to follow

Demo 08: Negating the Exists; try this and then try changing the filter to an_price < 15 and run that one also.

```
select *
from zoo_ex
where an_type='bird'
and  NOT exists (
  Select 1
  from zoo_ex
  where an_type='bird'
  and an_price < 75
  )
;
```

Demo 09: Compare the Exists to an IN test- this returns the birds which cost less than 75 and does not need the subquery approach.

```
select *
from zoo_ex
where an_type='bird'
and  id in (
  Select id
  from zoo_ex
  where an_type='bird'
  and an_price < 75  );
+----+---------+----------+
| id | an_type | an_price |
+----+---------+----------+
|  5 | bird    |       50 |
+----+---------+----------+
```

Now take queries 9 and 10 and run them testing for snake- which has a null price. How does this affect the output? It is always a good idea to try to figure out what these will do before you try them. Just running queries is not as helpful. And you can run the subquery part by itself to help understand what happens with nulls.

Demo 10: Snakes less than 75 and then snakes less than 15

```
select *
from zoo_ex
where an_type='snake'
and  exists (
  Select 1
  from zoo_ex
  where an_type='snake'
  and an_price < 75
  )
;

-- demo 10B:
select *
from zoo_ex
where an_type='snake'
and  exists (
  Select 1
  from zoo_ex
  where an_type='snake'
  and an_price < 15
  );
```

Demo 11: Negating these : Snakes less than 75 and then snakes less than 15

```
select *
from zoo_ex
where an_type='snake'
and  NOT exists (
  Select 1
  from zoo_ex
  where an_type='snake'
  and an_price < 75
  );

-- Demo 11B:
select *
from zoo_ex
where an_type='snake'
and  NOT exists (
  Select 1
  from zoo_ex
  where an_type='snake'
  and an_price < 15
  );
```

2. Correlation and the Exists Predicate

When we add correlation then these can be used to solve more problems. The following queries are correlated queries so that we are checking if there are any rows in the subquery that are related to the current row in the parent query. Remember that if you want to find subquery information for **this** customer or for **this** order, you generally will need to correlate.

We are now using the tables in the a_altgeld_mart database.

Which customers have an order with a date in Oct 2012? We don't care how many orders they had or what they bought- we only want a list of the customers with an order in that month.

Demo 12: We can solve this with an Exists and a correlated subquery. We use a corrected subquery because- for each customer we want to know if there are any orders **for this** customer.

```
select cust_id, cust_name_last, cust_name_first
from a_oe.customers
where EXISTS (
     select 'X'
     from a_oe.order_headers
     where a_oe.order_headers.cust_id = a_oe.customers.cust_id
     and extract( MONTH from ord_date) = 10
     and extract(Year from ord_date) = 2012
     )
;
+---------+----------------+-----------------+
| cust_id | cust_name_last | cust_name_first |
+---------+----------------+-----------------+
|  401250 | Morse          | Samuel          |
|  403000 | Williams       | Sally           |
|  403050 | Hamilton       | Alexis          |
|  403100 | Stevenson      | James           |
|  404950 | Morris         | William         |
+---------+----------------+-----------------+
```

Demo 13: We can also solve this with an In subquery and with an inner join.

```
select cust_id, cust_name_last, cust_name_first
from a_oe.customers
where cust_id in  (
     select cust_id
     from a_oe.order_headers
     where extract(month from ord_date) = 10
     and extract(year from ord_date) = 2012
     )
;
```

Demo 14: This query may be less efficient if there is a sort done to handle the distinct.

```
select Distinct CS.cust_id, CS.cust_name_last, CS.cust_name_first
from a_oe.customers CS
join a_oe.order_headers  OH on  CS.cust_id =OH.cust_id
where extract(month from ord_date) = 10
and extract(year from ord_date) = 2012
;
```

Demo 15: This query can return multiple rows for the same customer.

```
select CS.cust_id, CS.cust_name_last, CS.cust_name_first
from a_oe.customers CS
join a_oe.order_headers  OH on  CS.cust_id =OH.cust_id
where extract(month from ord_date) = 10
and extract(year from ord_date) = 2012
;
```

```
+---------+---------------+-----------------+
| cust_id | cust_name_last | cust_name_first |
+---------+---------------+-----------------+
|  401250 | Morse          | Samuel          |
|  403000 | Williams       | Sally           |
|  403000 | Williams       | Sally           |
|  403000 | Williams       | Sally           |
|  403050 | Hamilton       | Alexis          |
|  403100 | Stevenson      | James           |
|  403100 | Stevenson      | James           |
|  403100 | Stevenson      | James           |
|  404950 | Morris         | William         |
+---------+---------------+-----------------+
```

The following requires a correlated query, since we need to check not if there is any order at the discount- but if **this customer** has an order with this discount.

Demo 16: Display customers who purchased at item at a discount greater than 10% of the list price .

```
select cust_id
, cust_name_last
, cust_name_first
from a_oe.customers
where EXISTS (
     select 1
     from a_oe.order_headers OH
     join a_oe.order_details OD on OH.ord_id =  OD.ord_id
     join a_prd.products     PR on OD.prod_id = PR.prod_id
     where (prod_list_price - quoted_price)/prod_list_price > 0.1
     and a_oe.customers.cust_id = OH.cust_id
     )
order by cust_id
;
+---------+---------------+-----------------+
| cust_id | cust_name_last | cust_name_first |
+---------+---------------+-----------------+
|  400300 | McGold         | Arnold          |
|  401250 | Morse          | Samuel          |
|  402100 | Morise         | William         |
|  403000 | Williams       | Sally           |
|  403050 | Hamilton       | Alexis          |
|  403100 | Stevenson      | James           |
|  404000 | Olmsted        | Frederick       |
|  404950 | Morris         | William         |
|  409150 | Martin         | Joan            |
|  409160 | Martin         | Jane            |
|  900300 | McGold         | Arnold          |
|  903000 | McGold         | Arnold          |
+---------+---------------+-----------------+
```

What do we get when we negate the Exists? Remember that negative logic requires extra thought. Do we want customers who have never gotten the discount or do we want customers who have a purchase which was not at the discount.

Demo 17:

```
select cust_id
, cust_name_last
, cust_name_first
from a_oe.customers
where NOT EXISTS (
     select 1
     from a_oe.order_headers OH
     join a_oe.order_details OD on OH.ord_id =  OD.ord_id
     join a_prd.products     PR on OD.prod_id = PR.prod_id
     where (prod_list_price - quoted_price)/prod_list_price > 0.1
     and a_oe.customers.cust_id = OH.cust_id)
order by cust_id;
+---------+---------------+-----------------+
| cust_id | cust_name_last | cust_name_first |
+---------+---------------+-----------------+
|  400801 | Washington     | Geo             |
|  401890 | Northrep       | William         |
|  402110 | Coltrane       | John            |
|  402120 | McCoy          | Tyner           |
|  402500 | Jones          | Elton John      |
|  403010 | Otis           | Elisha          |
|  403500 | Stevenson      | JAMES           |
|  403750 | O'Leary        | Mary            |
|  403760 | O'Leary        | Mary            |
|  404100 | Button         | D. K.           |
|  404150 | Dancer         | Tom             |
|  404180 | Shay           | Danielle        |
|  404890 | Kelley         | Florence        |
|  404900 | Williams       | Al              |
|  405000 | Day            | David           |
|  408770 | Clay           | Clem            |
|  409010 | Morris         | William         |
|  409020 | Max            | William         |
|  409030 | Mazur          | Barry           |
|  409190 | Prince         | NULL            |
|  915001 | Adams          | Abigail         |
+---------+---------------+-----------------+
```

Demo 18:

```
select cust_id
, cust_name_last
, cust_name_first
from a_oe.customers
where EXISTS (
     select  1
     from a_oe.order_headers OH
     join a_oe.order_details OD on OH.ord_id =  OD.ord_id
     join a_prd.products     PR on OD.prod_id = PR.prod_id
     where not((prod_list_price - quoted_price)/prod_list_price > 0.1 )
     and a_oe.customers.cust_id = OH.cust_id);
+---------+---------------+-----------------+
| cust_id | cust_name_last | cust_name_first |
+---------+---------------+-----------------+
|  400300 | McGold         | Arnold          |
|  401250 | Morse          | Samuel          |
|  401890 | Northrep       | William         |
|  402100 | Morise         | William         |
|  403000 | Williams       | Sally           |
|  403010 | Otis           | Elisha          |
|  403050 | Hamilton       | Alexis          |
```

```
|  403100 | Stevenson      | James           |
|  404000 | Olmsted        | Frederick       |
|  404100 | Button         | D. K.           |
|  404900 | Williams       | Al              |
|  404950 | Morris         | William         |
|  408770 | Clay           | Clem            |
|  409030 | Mazur          | Barry           |
|  409150 | Martin         | Joan            |
|  409160 | Martin         | Jane            |
|  409190 | Prince         | NULL            |
|  900300 | McGold         | Arnold          |
|  903000 | McGold         | Arnold          |
|  915001 | Adams          | Abigail         |
+---------+----------------+-----------------+
```

Demo 19: Display any products for which we have an order with a quantity order of 10 or more. Note that with this query we do not care how many such orders we have only whether or not we have any such orders **for this product**.

```
select prod_id, prod_name
from a_prd.products PR
where EXISTS (
     select 'X'
     from a_oe.order_details OD
     where PR.prod_id = OD.prod_id
     and quantity_ordered >= 10
     )
;
+---------+------------------+
| prod_id | prod_name        |
+---------+------------------+
|    1010 | Weights          |
|    1020 | Dartboard        |
|    1030 | Basketball       |
|    1040 | Treadmill        |
|    1050 | Stationary bike  |
|    1060 | Mountain bike    |
|    1070 | Iron             |
|    1140 | Bird cage- simple |
|    1150 | Cat exerciser    |
|    4576 | Cosmo cat nip    |
+---------+------------------+
10 rows in set (0.00 sec)
```

Demo 20: Which customers bought both an APL product and a HW product? We solved this problem earlier with two IN tests and again with a join.

```
select a_oe.customers.cust_id
, cust_name_last
from a_oe.customers
where EXISTS (
     select 'X'
     from a_oe.cust_orders
     where a_oe.cust_orders.custID = A_oe.customers.cust_id
     and category ='APL'
     )
and  EXISTS (
     select 'X'
     from a_oe.cust_orders
```

```
      where a_oe.cust_orders.custID = A_oe.customers.cust_id
      and category ='HW'
      )
;
+---------+---------------+
| cust_id | cust_name_last |
+---------+---------------+
|  402100 | Morise         |
|  403000 | Williams       |
|  404100 | Button         |
|  404950 | Morris         |
|  409030 | Mazur          |
|  409150 | Martin         |
|  903000 | McGold         |
+---------+---------------+
```

How would you use exists to find

- customers who bought either an APL product or a HW product
- customers who bought an APL product but not a HW product?

3. Subqueries that produce report style output

These queries produce tables- that is always the case but these look like very simple reports

Demo 21: This query produces a simple report as to customer purchase in three categories

```
select CS.cust_id
, CS.cust_name_last
, case when EXISTS (
    select 'X'
    from a_oe.cust_orders
    where a_oe.cust_orders.custID = CS.cust_id
    and category ='APL'
    )  then 'Appliances  '
  else '    ---    '
  End as " "
, case when EXISTS (
    select 'X'
    from a_oe.cust_orders
    where a_oe.cust_orders.custID = CS.cust_id
    and category ='HW'
    )  then 'Housewares  '
  else '    ---    '
  End   as " "
, case when EXISTS (
    select 'X'
    from a_oe.cust_orders
    where a_oe.cust_orders.custID = CS.cust_id
    and category ='PET'
    )  then 'PetSupplies '
  else '    ---    '
  End   as " "
 From a_oe.customers  CS
;
+---------+---------------+-------------+-------------+-------------+
| cust_id | cust_name_last |             |             |             |
+---------+---------------+-------------+-------------+-------------+
|  400300 | McGold         | Appliances  |     ---     |     ---     |
|  400801 | Washington     |     ---     |     ---     |     ---     |
```

```
|  401250 | Morse          |     ---      | Housewares   |     ---      |
|  401890 | Northrep       |     ---      | Housewares   |     ---      |
|  402100 | Morise         | Appliances   | Housewares   | PetSupplies  |
|  402110 | Coltrane       |     ---      |     ---      |     ---      |
|  402120 | McCoy          |     ---      |     ---      |     ---      |
|  402500 | Jones          |     ---      |     ---      |     ---      |
|  403000 | Williams       | Appliances   | Housewares   | PetSupplies  |
|  403010 | Otis           | Appliances   |     ---      |     ---      |
|  403050 | Hamilton       |     ---      | Housewares   | PetSupplies  |
|  403100 | Stevenson      |     ---      |     ---      | PetSupplies  |
```

Demo 22: A minor change in the query gives you a single column of purchase categories.

```
select CS.cust_id
,      CS.cust_name_last
,    CONCAT(
  case when EXISTS (
    select 'X'
    from a_oe.cust_orders
    where a_oe.cust_orders.custID = CS.cust_id
    and category ='APL'
    )  then 'Appliances  '
  else ''
  End
, case when EXISTS (
   select 'X'
   from a_oe.cust_orders
   where a_oe.cust_orders.custID = CS.cust_id
   and category ='HW'
   )  then 'Housewares  '
  else ''
  End
, case when EXISTS (
   select 'X'
   from a_oe.cust_orders
   where a_oe.cust_orders.custID = CS.cust_id
   and category ='PET'
   )  then 'PetSupplies'
  else ''
  End
  ) as PurchaseAreas
 From a_oe.customers  CS
;
+---------+---------------+--------------------------------------+
| cust_id | cust_name_last | PurchaseAreas                       |
+---------+---------------+--------------------------------------+
|  400300 | McGold         | Appliances                          |
|  400801 | Washington     |                                     |
|  401250 | Morse          | Housewares                          |
|  401890 | Northrep       | Housewares                          |
|  402100 | Morise         | Appliances  Housewares  PetSupplies |
|  402110 | Coltrane       |                                     |
|  402120 | McCoy          |                                     |
|  402500 | Jones          |                                     |
|  403000 | Williams       | Appliances  Housewares  PetSupplies |
|  403010 | Otis           | Appliances                          |
|  403050 | Hamilton       | Housewares  PetSupplies             |
|  403100 | Stevenson      | PetSupplies                         |
```

4. For all query

Warning: these take some time to work though; They are not on the assignment nor on the final exam

Suppose we want to find customers who have bought all of our products. That is a bit too much to expect- so we will limit this to customers who have bought all of our SPG items. What we want is the customer(s) who have purchased every product with a SPG category. This is sometimes call a for all query.

This can be converted into a double not exists query.

Logic- level 1

find customers
where there is no SPG product
which is not on an order for that customer

Logic- level 2

```
select cust_id from customers
where NOT EXISTS
    ( select * from SPG product
      where NOT EXISTS
          ( select * from Orders
            for these products
            for these customers) )
```

Demo 23: For this, it will help to have some views set up. I did one for orders and another for SPG products. This makes the logic in the main query easier to write

```
create or replace  view a_oe.Ord as (
   select cust_id as CustID, prod_id  as prodid
   from a_oe.order_headers OH
   join a_oe.order_details OD  on OH.ord_id = OD.ord_id );

create or replace  view a_oe.SPG as (
   select prod_id  as spg_prod_id
   from a_prd.products
   where catg_id = 'SPG' );
```

Demo 24:

```
select Cust_id, cust_name_last
from a_oe.customers CS
where not exists
   (select *
    from a_oe.SPG
    where not exists
       (select *
       from a_oe.ORD
        where a_oe.SPG.spg_prod_id = a_oe.Ord.prodid
        and a_oe.ord.custid = CS.cust_id) )
;
+---------+----------------+
| Cust_id | cust_name_last |
+---------+----------------+
| 403000  | Williams       |
| 408770  | Clay           |
+---------+----------------+
```

You can run some additional queries to help check your logic

```
Select prod_id, prod_name, catg_id
from a_prd.products
where catg_id  = 'SPG';

select OH.cust_id, OD.prod_id, PR.prod_name, PR.catg_id
from a_oe.order_headers OH
join a_oe.order_details OD  on OH.ord_id = OD.ord_id
join a_prd.products     PR on OD.prod_id = PR.prod_id
where cust_id in (403000, 408770)
and PR.catg_id  = 'SPG'
order by OH.cust_id, OD.prod_id
;
```

Demo 25: If you do not want the views, then you can use subqueries for the result sets returned by the view. Sometimes it is helpful to create a view to break down the query into steps and then replace the view reference with a subquery that encapsulates that view

```
select Cust_id, cust_name_last
from a_oe.customers CS
where not exists (
    select *
    from (
       select prod_id  as spg_prod_id
       from a_prd.products
       where catg_id = 'SPG' ) tbl_S
    where not exists
       (select *
        from (
              select cust_id as CustID, prod_id  as prodid
              from a_oe.order_headers OH
              join a_oe.order_details OD  on OH.ord_id = OD.ord_id
              ) tbl_o
         where tbl_S.spg_prod_id = tbl_o.prodid
         and tbl_o.custid = CS.cust_id
      )
   )
;
```

This is another approach. How many products do we have in the products table that are SPG items.(With my current data set I have 6 such items) We can also find all customers who bough that many different SPG products. This uses two queries and checks that the counts are the same.

I tossed the major joins into a subquery retrieving the customer id (to be displayed and needed for grouping) , the customer name (to be displayed) and the product id(to be counted). The first part of the main query groups the sales by the customer id and counts the distinct product ids- this needs to be distinct since a person might have bought the same item more than once. The second part of the query counts the products id in the product table.

Demo 26: Another for each

```
select cust_id, cust_name_last
from (
   select CS.cust_id, CS.cust_name_last,PR.prod_id
   from a_oe.customers CS
   join a_oe.order_headers OH on CS.cust_id = OH.cust_id
```

```
   join a_oe.order_details OD  on OH.ord_id = OD.ord_id
   join a_prd.products PR  on OD.prod_id = PR.prod_id
   where catg_id = 'SPG') sales
group by cust_id,cust_name_last
having count(distinct prod_id)
   =
( select count(prod_id)
  from a_prd.products PR
  where  catg_id = 'SPG'
)
;
+---------+----------------+
| Cust_id | cust_name_last |
+---------+----------------+
| 403000  | Williams       |
| 408770  | Clay           |
+---------+----------------+
2 rows in set (0.00 sec)
```

We can write a variation on this query where we want to find customers who have purchased all of our SPG items but has not purchased anything else.

Demo 27:

```
select AllSales.cust_id, AllSales.cust_name_last
from   (
   select CS.cust_id, CS.cust_name_last,PR.prod_id, pr.catg_id
   from a_oe.customers CS
   join a_oe.order_headers OH on CS.cust_id = OH.cust_id
   join a_oe.order_details OD on OH.ord_id = OD.ord_id
   join a_prd.products PR  on OD.prod_id = PR.prod_id
   ) AllSales
join
(
   select CS.cust_id, CS.cust_name_last,PR.prod_id, pr.catg_id
   from a_oe.customers CS
   join a_oe.order_headers OH on CS.cust_id = OH.cust_id
   join a_oe.order_details OD on OH.ord_id = OD.ord_id
   join a_prd.products PR  on OD.prod_id = PR.prod_id
   where  catg_id = 'SPG'
   ) spgSales on AllSales.cust_id = SPGSales.cust_id
cross join
(
  select count(prod_id) as NumToys
  from a_prd.products PR
  where  catg_id = 'SPG'
)ToyCount
group by AllSales.cust_id,AllSales.cust_name_last,ToyCount.NumToys
having count(distinct spgSales.prod_id)
  = ToyCount.NumToys
and count(distinct allsales.prod_id)
  = ToyCount.NumToys
;
```

```
CUST_ID CUST_NAME_LAST
------- ------------------------
408770  Clay
```

Breaking this down into pieces: we have a subquery which calculates all sales; we have a subquery that gets the spg sales. The third subquery returns a single row, single column. Remember that when you join a single row table you can do this with a cross join.

We want the count of the distinct sporting goods items to equal the number of items the customer has purchased and we also want the number of distinct all items to equal the number of items the customer has purchased- which means the customer did not purchased any non-spg items.