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 the Select. . . From a_oe.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 a_oe.order_headers is the **subquery**.

Demo 01: A simple subquery

```
Select credit_limit
From a_oe.customers
Where cust_id in (
    Select cust_id
    From a_oe.order_headers
   )
 order by credit_limit desc;
+--------------+
| credit_limit |
+--------------+
|         7500 |
|         6000 |
|         6000 |
|         6000 |
. . .
```

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 cust_name_last
, cust_name_first
, cust_id
From a_oe.customers
Where cust_id IN (
   Select cust_id
```

```
   From a_oe.order_headers
   Where ord_id  IN (
      Select ord_id
      From a_oe.order_details
      Where prod_id = 1020
      )
   )
order by cust_id
;
+----------------+-----------------+---------+
| cust_name_last | cust_name_first | cust_id |
+----------------+-----------------+---------+
| Northrep       | William         | 401890  |
| Williams       | Sally           | 403000  |
| 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 ord_id
From a_oe.order_details
Where prod_id = 1020;
```

These are the order id values for that product.

```
+--------+
| ord_id |
+--------+
|    105 |
|    405 |
|    516 |
|    519 |
|    520 |
|    529 |
|    716 |
+--------+
```

When we use this as a subquery with an IN list test, this output will be used as a list: (105,405,516,519,520,529, 716).

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

```
Select cust_id
From a_oe.order_headers
Where ord_id  IN (
     Select ord_id
     From a_oe.order_details
     Where prod_id = 1020)  ;
+---------+
| cust_id |
+---------+
|  401890 |
|  403000 |
|  403000 |
|  404900 |
|  408770 |
|  409150 |
|  409150 |
+---------+
```

So this is effectively what the query is doing.

```
Select cust_id
From a_oe.order_headers
Where ord_id  IN (105,405,516,519,520,529, 716);
```

The result is again used as an IN list (403000, 408770,409150, 401890,404900, 043000, 409150). Note that the 2-level subquery has the customer id 403000 appearing twice in the result set. 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 cust_name_last
, cust_name_first
, cust_id
From a_oe.customers
Where cust_id IN (403000, 408770, 409150, 401890, 404900, 403000, 409150);
```

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.cust_name_last
, CS.cust_name_first
, CS.cust_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
Where OD.prod_id = 1020
order by cust_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 cust_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 cust_id, cust_name_last, cust_name_first
From a_oe.customers
Where cust_id = (
   Select cust_id
   From a_oe.order_headers
   Where Ord_id  = 115)
;
+---------+----------------+-----------------+
| cust_id | cust_name_last | cust_name_first |
+---------+----------------+-----------------+
| 402100  | Morise         | William         |
+---------+----------------+-----------------+
```

We know that in the order table, the ord_id is the pk. So we can have only one row in the order table with ord_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

```
set @prod_id   := 5004;

Select cust_id, cust_name_last, cust_name_first
From a_oe.customers
Where cust_id = (
   Select cust_id
   From a_oe.order_headers  OH
   Join a_oe.order_details  OD on OH.ord_id = OD.ord_id
   Where prod_id  =@prod_id);
+---------+----------------+-----------------+
| cust_id | cust_name_last | cust_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 cust_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 a_emp.employees
Where emp_id  <> @empId
and emp_mng  = (
   Select emp_mng
   From a_emp.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 a_emp.employees
where emp_id  <> @empId
and emp_mng  = (
   Select emp_mng
   From a_emp.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 a_emp.employees
Where emp_id  <> @empId
and emp_mng  = (
   Select emp_mng
   From a_emp.employees
   Where emp_id  = @empId  );
Empty set (0.00 sec)
```

We cannot distinguish these two conditions from the output.

3.3. Subquery with an aggregate

Since aggregate queries return single values for large collections of data, they are useful in subqueries.

For now we are using an aggregate that returns only one value. We can have a subquery that uses an aggregate function and find all products that are priced more than the average price. The subquery is doing the average on the entire table and returns a single row.

With my current set of data the average price is 151.64.

Demo 10:

```
Select prod_id
, prod_name
, catg_id
, prod_list_price
From a_prd.products
Where prod_list_price >  (
   Select avg(prod_list_price)
   From a_prd.products  );
+---------+-----------------+---------+-----------------+
| prod_id | prod_name       | catg_id | prod_list_price |
+---------+-----------------+---------+-----------------+
|    1000 | Hand Mixer      | HW      |          125.00 |
|    1010 | Weights         | SPG     |          150.00 |
|    1040 | Treadmill       | SPG     |          349.95 |
|    1050 | Stationary bike | SPG     |          269.95 |
|    1060 | Mountain bike   | SPG     |          255.95 |
|    1090 | Gas grill       | HW      |          149.99 |
|    1120 | Washer          | APL     |          549.99 |
|    1125 | Dryer           | APL     |          500.00 |
|    1126 | WasherDryer     | APL     |          850.00 |
|    1130 | Mini Freezer    | APL     |          149.99 |
|    1160 | Mixer Deluxe    | HW      |          149.99 |
|    4567 | Deluxe Cat Tree | PET     |          549.99 |
|    4568 | Deluxe Cat Bed  | PET     |          549.99 |
|    4569 | Mini Dryer      | APL     |          349.95 |
+---------+-----------------+---------+-----------------+
```

Demo 11: Suppose we want to see products that are within $100 of the average price.

```
Select prod_id
, prod_name
, catg_id
, prod_list_price
From a_prd.products
Where prod_list_price
   between
         (Select avg(prod_list_price)
          From a_prd.products  ) - 100
   and
         (Select avg(prod_list_price)
          From  a_prd.products  ) + 100
;
+---------+--------------------+---------+-----------------+
| prod_id | prod_name          | catg_id | prod_list_price |
+---------+--------------------+---------+-----------------+
|    1000 | Hand Mixer         | HW      |          125.00 |
|    1010 | Weights            | SPG     |          150.00 |
|    1030 | Basketball         | SPG     |           29.95 |
```

```
|    1070 | Iron                | HW      |           25.50 |
|    1071 | Iron                | HW      |           25.50 |
|    1072 | Iron                | HW      |           25.50 |
|    1080 | Cornpopper          | HW      |           25.00 |
|    1090 | Gas grill           | HW      |          149.99 |
|    1100 | Blender             | HW      |           49.99 |
|    1110 | Pancake griddle     | HW      |           49.99 |
|    1130 | Mini Freezer        | APL     |          149.99 |
|    1141 | Bird cage- deluxe   | PET     |           99.99 |
|    1152 | Cat pillow Leather  | PET     |           55.28 |
|    1160 | Mixer Deluxe        | HW      |          149.99 |
|    4575 | Electric can opener | HW      |           49.95 |
|    4576 | Cosmo cat nip       | PET     |           29.95 |
|    4577 | Cat leash           | PET     |           29.95 |
|    5002 | Ball-Peen Hammer    | HD      |           23.00 |
|    5005 | Shingler Hammer     | HD      |           45.00 |
+---------+---------------------+---------+-----------------+
```

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 2013-05-12; 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 12: This query fails

```
Select cust_id, cust_name_last, cust_name_first
From a_oe.customers
Where cust_id  = (
    Select cust_id
    From a_oe.order_headers
    where ord_date  = '2013-05-12');
```

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 13: Changing to an In list for the subquery

```
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 ord_date  = '2013-05-12')
;
+---------+----------------+-----------------+
| cust_id | cust_name_last | cust_name_first |
+---------+----------------+-----------------+
|  403000 | Williams       | Sally           |
|  404950 | Morris         | William         |
+---------+----------------+-----------------+
```

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.

Demo 14:

```
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 ord_date  = '1888-08-08');
Empty set (0.00 sec)
```

Demo 15: If we change the query to a NOT IN query, then all customers are returned with our data set

```
Select cust_id, cust_name_last, cust_name_first
From a_oe.customers
Where cust_id  NOT IN (
   Select cust_id
   From a_oe.order_headers
   Where ord_date  = '1888-08-08');
+---------+----------------+-----------------+
| cust_id | cust_name_last | cust_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 2012- 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 16: 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 a_oe.customers.cust_id
, cust_name_last, cust_name_first
From a_oe.customers
join a_oe.order_headers on a_oe.customers.cust_id= a_oe.order_headers.cust_id
Where month (ord_date) = 12 and year(ord_date) = 2012;
+---------+----------------+-----------------+
| cust_id | cust_name_last | cust_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 17: This uses a subquery. Use cust_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 2012. This query cannot show the ord_date since it is not a column in the parent query.

```
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 month (ord_date) = 12 and year(ord_date) = 2012);
+---------+----------------+-----------------+
| cust_id | cust_name_last | cust_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 18: This is an outer join query that returns customers who have no orders.

```
Select CS.cust_id, CS.cust_name_last
From a_oe.customers  CS
left join a_oe.order_headers OH on CS.cust_id = OH.cust_id
Where OH.ord_id is null;
+---------+----------------+
| cust_id | cust_name_last |
+---------+----------------+
|  400801 | Washington     |
|  402110 | Coltrane       |
|  402120 | McCoy          |
|  402500 | Jones          |
|  403500 | Stevenson      |
|  403750 | O'Leary        |
|  403760 | O'Leary        |
|  404150 | Dancer         |
|  404180 | Shay           |
|  404890 | Kelley         |
|  409010 | Morris         |
|  409020 | Max            |
+---------+----------------+
```

Demo 19: This is a subquery to accomplish the same task.

```
Select a_oe.customers.cust_id, cust_name_last
From a_oe.customers
where cust_id not in (
   Select cust_id
   From a_oe.order_headers );
```

Demo 20: Do we have any order rows for which there are no associated order detail rows?

```
Select Ord_id, ord_date
```

```
From a_oe.order_headers
Where ord_id not in (
    Select ord_id
    From a_oe.order_details );
+--------+---------------------+
| Ord_id | ord_date            |
+--------+---------------------+
|    116 | 2012-11-12 00:00:00 |
|    123 | 2012-12-05 00:00:00 |
|    506 | 2013-01-12 00:00:00 |
+--------+---------------------+
```

7. More subqueries using aggregates

Using aggregates in subqueries allow you to find the answers to questions such as: Who bought the most expensive book? Which customers in California have the highest credit limit? There may be ties so you may get multiple rows returned.

When we talked about aggregate queries, we said that we could not return the description of the most expensive item of a particular category. We can do this with a subquery. The subquery can find the max price for pet supplies (catg_id PET). This will be a single value. We can then pass that value from the subquery to the parent query which will find all pet supplies items that match that price.

I am going to use a variable for the category id so that it would be easy to change it and test another category.

Demo 21: Note that this returns pet supplies that are priced at the max price for pet supplies, but also items that are not pet supplies but that have a list price that equals the max price for pet supplies. (I don't know about your cats but my cats do not consider a Washer to be a pet supply!)

```
set @catg = 'PET' ;

Select prod_id
, prod_name
, catg_id
, prod_list_price
From a_prd.products
Where prod_list_price =  (
   Select max(prod_list_price)
   From a_prd.products
   Where catg_id = @catg);
+---------+-----------------+---------+-----------------+
| prod_id | prod_name       | catg_id | prod_list_price |
+---------+-----------------+---------+-----------------+
|    1120 | Washer          | APL     |          549.99 |
|    4567 | Deluxe Cat Tree | PET     |          549.99 |
|    4568 | Deluxe Cat Bed  | PET     |          549.99 |
+---------+-----------------+---------+-----------------+
```

Demo 22: Note that outer query now also checks that the category is the category we want.

```
Select prod_id
, prod_name
, catg_id
, prod_list_price
From a_prd.products
Where catg_id = @catg
and prod_list_price =  (
   Select max(prod_list_price)
```

```
   From a_prd.products
   Where catg_id = @catg);
+---------+-----------------+---------+-----------------+
| prod_id | prod_name       | catg_id | prod_list_price |
+---------+-----------------+---------+-----------------+
|    4567 | Deluxe Cat Tree | PET     |          549.99 |
|    4568 | Deluxe Cat Bed  | PET     |          549.99 |
+---------+-----------------+---------+-----------------+
```

Demo 23: Suppose we wanted to find out which customers bought the most expensive pet supplies items. This definition of "most expensive" relies on the list price- this does not say who paid the most for a pet supplies item.

```
Select distinct cust_id, prod_id
From a_oe.order_headers OH
join a_oe.order_details OD on OH.ord_id = OD.ord_id
Where prod_id in (
   Select prod_id
   From a_prd.products
   Where catg_id = @catg
   and prod_list_price =  (
     Select max(prod_list_price)
     From a_prd.products
     Where catg_id = @catg)
     );
```

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

It looks like no one has bought those very expensive pet supply items. Since we did this with a variable, change the variable to SPG and rerun the query and you can see that the query works- people did buy the most expensive sporting good item.

Notice that with the subquery technique, the only attributes you can display are those in the outer query's tables. In the example above we could display data from the order headers and order details table only. If you tried to display the product name, you would get an error even though the product table is referenced in the query. The product table is inside the subquery and its fields are not exposed to the outer most query.

8. More demos

For the next few demos I want to have a way to refer to the result set of the following query. I could do this with by creating a view and using that view as a data source in the From clause.

Demo 24: Create a view oe_cust_orders which joins the products, order details, order header tables. Several of the rest of the queries will use this view.

```
Create or replace view  a_oe.oe_cust_orders as (
Select
   OH.ord_id  as Invoice
,  OH.ord_date as OrderDate
,  OH.cust_id  as CustID
,  PR.catg_id  as Category
,  OD.prod_id  as ItemPurchased
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 OD.quoted_price > 0
and OD.quantity_ordered > 0 );
```

Demo 25: We can use the view as a table expression in the From clause of a query.

```
Select *
From a_oe.oe_cust_orders
order by invoice, category
;
```

Invoice	OrderDate	CustID	Category	ItemPurchased
105	2012-10-01 00:00:00	403000	SPG	1030
105	2012-10-01 00:00:00	403000	SPG	1020
105	2012-10-01 00:00:00	403000	SPG	1010
106	2012-10-01 00:00:00	401250	SPG	1060
107	2012-10-02 00:00:00	403050	HW	1110
108	2012-10-02 00:00:00	403000	HW	1080
109	2012-10-12 00:00:00	403000	APL	1130
110	2012-10-12 00:00:00	404950	APL	1130
110	2012-10-12 00:00:00	404950	HW	1090
111	2012-11-01 00:00:00	403000	PET	1150
111	2012-11-01 00:00:00	403000	PET	1141

. . . .

Demo 26: If someone bought an appliance, what else (besides an appliance) did they buy on the same order?

```
Select distinct
  custid
, invoice
, category
From a_oe.oe_cust_orders
Where category <> 'APL'
and oe_cust_orders.invoice in (
   Select invoice
   From a_oe.oe_cust_orders
   Where category = 'APL'  )
order by invoice
;
```

CustID	Invoice	Category
404950	110	HW
402100	115	HW
409030	130	HW
409150	415	PET
403000	509	HW
403000	511	SPG
409150	518	PET
409150	518	MUS
403000	529	SPG
409150	718	PET

Demo 27: If someone bought an appliance, what else (besides an appliance) did they buy, not necessarily on the same order?

```
Select distinct
  custid
, invoice,category
From a_oe.oe_cust_orders
Where Category <> 'APL'
and CustID  IN (
   Select custid
   From a_oe.oe_cust_orders
   Where category = 'APL'  )
order by custid
;
+--------+---------+----------+
| CustID | Invoice | Category |
+--------+---------+----------+
| 402100 |     115 | HW       |
| 402100 |     117 | PET      |
| 402100 |     117 | SPG      |
| 403000 |     509 | HW       |
| 403000 |     505 | SPG      |
| 403000 |     511 | SPG      |
| 403000 |     390 | SPG      |
| 403000 |     395 | SPG      |
. . .
```

Demo 28: Which customers bought both an appliance and a houseware item? The first IN tests that this customer bought an appliance and the second that they bought a houseware item.

```
Select cust_id
, cust_name_last
From a_oe.customers
Where cust_id IN (
   Select custID
   From a_oe.oe_cust_orders
   Where category = 'APL')
and cust_id IN (
   Select CustID
   From a_oe.oe_cust_orders
   Where Category = 'HW')
;
+---------+----------------+
| cust_id | cust_name_last |
+---------+----------------+
|  402100 | Morise         |
|  403000 | Williams       |
|  404100 | Button         |
|  404950 | Morris         |
|  409030 | Mazur          |
|  409150 | Martin         |
|  903000 | McGold         |
+---------+----------------+
```

The next demos are examples of queries that people try to use to solve these problems but they do not work properly.

Demo 29: This tries to match rows-- testing on a single row that the category is both APL and HW. This can never happen and the query returns no rows. Remember the From clause produces a tentative result set and the Where clause tests each row in the result set one row at a time.

```
Select distinct cust_id
, cust_name_last
From a_oe.customers  CS
Join a_oe.oe_cust_orders  CO on CS.cust_id = CO.CustID
Where category = 'APL' and category = 'HW';
```

Demo 30: People then try to use an OR test - or the equivalent IN list test. This returns rows for customers who bought either an appliance or a houseware or possibly both. But there is nothing that tests that they bought both.

```
Select distinct cust_id
, cust_name_last
From a_oe.customers  CS
Join a_oe.oe_cust_orders  CO on CS.cust_id = CO.CustID
Where category = 'APL' or category = 'HW';
+---------+----------------+
| cust_id | cust_name_last |
+---------+----------------+
| 402100  | Morise         |
| 409030  | Mazur          |
| 903000  | McGold         |
| 900300  | McGold         |
| 400300  | McGold         |
| 403010  | Otis           |
| 409150  | Martin         |
| 403000  | Williams       |
| 404950  | Morris         |
| 404100  | Button         |
| 401250  | Morse          |
| 409160  | Martin         |
| 915001  | Adams          |
| 403050  | Hamilton       |
| 409190  | Prince         |
| 401890  | Northrep       |
+---------+----------------+
16 rows in set (0.00 sec)
```

If we look at just the orders for client 401250, that client bought only housewares and sporting goods- no appliances.

```
+---------+----------------+----------+---------------+
| cust_id | cust_name_last | Category | ItemPurchased |
+---------+----------------+----------+---------------+
| 401250  | Morse          | SPG      |          1060 |
| 401250  | Morse          | HW       |          1080 |
| 401250  | Morse          | HW       |          1070 |
| 401250  | Morse          | HW       |          1100 |
+---------+----------------+----------+---------------+
```

If we look at just the orders for client 403000, that client bought housewares and appliances and sporting goods and pet supplies

```
+---------+---------------+----------+---------------+
| cust_id | cust_name_last | Category | ItemPurchased |
+---------+---------------+----------+---------------+
| 403000  | Williams       | SPG      |          1030 |
| 403000  | Williams       | SPG      |          1020 |
| 403000  | Williams       | SPG      |          1010 |
| 403000  | Williams       | HW       |          1080 |
| 403000  | Williams       | APL      |          1130 |
| 403000  | Williams       | PET      |          1150 |
| 403000  | Williams       | PET      |          1141 |
. . .
```