1. Table of Contents

This is mostly review; it can help to go back over material several times. Something that did not make sense two weeks ago might click now.

1. Subqueries versus Joins and Distinct

We want to know the names of all of our customers who currently have an order in our order headers table. Since we need the customer name we need the customer table. We only want customers with orders so we join to the order headers table. We were not asked to check that the order has any detail lines so we do not need the order details table.

Demo 01: Using an Inner join

```
select 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
order by CS.cust_name_last, CS.cust_name_first
;
```

-- sample rows-

but we do have 67 rows and only 33 customers. It is pretty clear that we have some duplicates here.

```
+----------------+-----------------+
| cust_name_last | cust_name_first |
+----------------+-----------------+
| Adams          | Abigail         |
| Adams          | Abigail         |
| Button         | D. K.           |
| Button         | D. K.           |
| Button         | D. K.           |
| Clay           | Clem            |
| Day            | David           |
| Hamilton       | Alexis          |
| Hamilton       | Alexis          |
| Hamilton       | Alexis          |
. . .
67 rows in set (0.02 sec)
```

What we commonly do when we have duplicate output is add the Distinct modifier.

Demo 02: Using distinct and the inner join

```
select distinct 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
order by cust_name_last, cust_name_first
;
+----------------+-----------------+
| cust_name_last | cust_name_first |
+----------------+-----------------+
| Adams          | Abigail         |
| Button          | D. K.            |
| Clay            | Clem             |
| Day             | David            |
| Hamilton        | Alexis           |
| Martin          | Jane             |
| Martin          | Joan             |
| Mazur           | Barry            |
| McGold          | Arnold           |
| Morise          | William          |
| Morris          | William          |
| Morse           | Samuel           |
| Northrep        | William          |
| Olmsted         | Frederick        |
| Otis            | Elisha           |
| Prince          | NULL             |
| Stevenson       | James            |
| Williams        | Al               |
| Williams        | Sally            |
+-----------------+------------------+
19 rows in set (0.00 sec)
```

That looks better- but this is hiding some data. We know that we could have customers with the same name.

Demo 03: Adding the customer id to the Select. Using distinct and the inner join

```
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
order by cust_name_last, cust_name_first
;
+-----------------+------------------+---------+
| cust_name_last  | cust_name_first  | cust_id |
+-----------------+------------------+---------+
| Adams           | Abigail          |  915001 |
| Button          | D. K.            |  404100 |
| Clay            | Clem             |  408770 |
| Day             | David            |  405000 |
| Hamilton        | Alexis           |  403050 |
| Martin          | Jane             |  409160 |
| Martin          | Joan             |  409150 |
| Mazur           | Barry            |  409030 |
| McGold          | Arnold           |  400300 |
| McGold          | Arnold           |  900300 |
| McGold          | Arnold           |  903000 |
| Morise          | William          |  402100 |
| Morris          | William          |  404950 |
| Morse           | Samuel           |  401250 |
| Northrep        | William          |  401890 |
| Olmsted         | Frederick        |  404000 |
| Otis            | Elisha           |  403010 |
| Prince          | NULL             |  409190 |
| Stevenson       | James            |  403100 |
| Williams        | Al               |  404900 |
| Williams        | Sally            |  403000 |
+-----------------+------------------+---------+
```

Now we get 2 more rows. In this case we could find them by inspection. We have three customers with the name Arnold. McGold . We do not know if these are three different people or one or two people who registered with us more than one time. The SQL developer cannot make that decision.

But we did expose the customer ID- maybe we should not do that. We are to display the name only.

Demo 04: We could hide the query with the custID inside a subquery and use it as the data source. This will give use the rows we expect with Arnold McGold showing up three times. .

This will give

```
Select cust_name_last, cust_name_first
 from (
   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
) CS
;
+----------------+-----------------+
| cust_name_last | cust_name_first |
+----------------+-----------------+
| McGold         | Arnold          |
| Morse          | Samuel          |
| Northrep       | William         |
| Morise         | William         |
| Williams       | Sally           |
| Otis           | Elisha          |
| Hamilton       | Alexis          |
| Stevenson      | James           |
| Olmsted        | Frederick       |
| Button         | D. K.           |
| Williams       | Al              |
| Morris         | William         |
| Day            | David           |
| Clay           | Clem            |
| Mazur          | Barry           |
| Martin         | Joan            |
| Martin         | Jane            |
| Prince         | NULL            |
| McGold         | Arnold          |
| McGold         | Arnold          |
| Adams          | Abigail         |
+----------------+-----------------+
```

Demo 05: Another solution would be to have a subquery that gets us the customer ID from the order headers table- that gives us customer with orders and then use that subquery as a filter for the outer query which uses the customers table to get the names.:

```
select cust_name_last, cust_name_first
from a_oe.customers CS
where cust_id in
   (select cust_id
    from a_oe.order_headers );
```

Which of these two approaches is better? That depends on several things. Optimizers may make different plans based on the size of the tables, the presence or absence of indexes, and the distribution of data. So having a rule such as technique A is always better than technique B is not generally true.

If your dbms always does a sort if you use Distinct, then the second version might be more efficient. A sort , including a hidden sort, is generally expensive and you should avoid it when possible.

Of course there is another question- what are the business needs of the person who asked for the result set (Remember users never want queries- they want results). Maybe the user really wants to see the different names of people who have orders- marketing people do odd types of sales analysis and maybe they are looking for name patterns!

2. OUTER JOINS- what does that null mean?

Suppose we write a query that shows the various locations for our employee departments. We want to include any department that might not have an assigned location.

Demo 06: An outer join from department to locations.

```
select
  dept_id
, dept_name
, loc_type  as LocationType
from a_emp.departments D
left join a_emp.locations   L on D.loc_id = L.loc_id
order by dept_id
;
+---------+-----------------+--------------+
| dept_id | dept_name       | LocationType |
+---------+-----------------+--------------+
|      10 | Administration  | office       |
|      20 | Marketing       | warehouse    |
|      30 | Development     | warehouse    |
|      35 | Cloud Computing | NULL         |
|      80 | Sales           | warehouse    |
|      90 | Shipping        | NULL         |
|      95 | Logistics       | NULL         |
|     210 | IT Support      | office       |
|     215 | IT Support      | NULL         |
+---------+-----------------+--------------+
```

This has several rows with a null for the location type - what can that null mean in terms of the query. Try to think of the possibilities; you may have to look at the table create statements to see how this oculd happen.

One possibility is that the department does not have an assigned location and the dbms generated a null rocord due to the outer join.

If you look at the definition of the location table, the loc_type was defined as nullable- so it might be that the department has a location but the location type is null. The result set does not differentiate between these two.

If you look at the current data set you can see that dept 35 has no assigned location; dept 215 has an assigned location, but that location has a null for the location type.

Demo 07: Let's show more columns so that we can work on this.

```
select
  dept_id
, l.loc_id
, loc_type as LocationType
, loc_city as City
, loc_state_province as "State/Province"
```

```
from a_emp.departments D
left join a_emp.locations   L on D.loc_id = L.loc_id
order by dept_id
;
+---------+--------+--------------+---------------------+-----------------+
| dept_id | loc_id | LocationType | City                | State/Province  |
+---------+--------+--------------+---------------------+-----------------+
|      10 |   1560 | office       | San Francisco       | California      |
|      20 |   1400 | warehouse    | Southlake           | Texas           |
|      30 |   1500 | warehouse    | South San Francisco | California      |
|      35 |   NULL | NULL         | NULL                | NULL            |
|      80 |   1400 | warehouse    | Southlake           | Texas           |
|      90 |   NULL | NULL         | NULL                | NULL            |
|      95 |   NULL | NULL         | NULL                | NULL            |
|     210 |   1800 | office       | Toronto             | Ontario         |
|     215 |   2700 | NULL         | Munich              | Bavaria         |
+---------+--------+--------------+---------------------+-----------------+
```

What we want as a result is the following style:

```
 dept_id LocationType   City                     State
      10 office         San Francisco            California
      35 Unknown type   No Site                  No Site
      80 warehouse      Southlake                Texas
     215 Unknown type   Munich                   Bavaria
```

Demo 08: We commonly use coalesce in these situations.

```
select
  dept_id
, coalesce(loc_type, 'Unknown type') as LocationType
, coalesce(loc_city, 'No Site') as City
, coalesce(loc_state_province, 'No Site') as "State/Province"
from a_emp.departments D
left join a_emp.locations   L on D.loc_id = L.loc_id
order by dept_id
;
+---------+--------------+---------------------+-----------------+
| dept_id | LocationType | City                | State/Province  |
+---------+--------------+---------------------+-----------------+
|      10 | office       | San Francisco       | California      |
|      20 | warehouse    | Southlake           | Texas           |
|      30 | warehouse    | South San Francisco | California      |
|      35 | Unknown type | No Site             | No Site         |
|      80 | warehouse    | Southlake           | Texas           |
|      90 | Unknown type | No Site             | No Site         |
|      95 | Unknown type | No Site             | No Site         |
|     210 | office       | Toronto             | Ontario         |
|     215 | Unknown type | Munich              | Bavaria         |
+---------+--------------+---------------------+-----------------+
```

This works for the City and State- these are not null attributes in the location table definition so they only way there are null is if the department does not have a location. But the location type has two ways to be null. We want to handle that circumstance on the location table level. If the field in the location table is null then use the alternate display.

```
select loc_id
,  coalesce(loc_type, 'UnknownType') as loc_type
, loc_city, loc_state_province
from a_emp.locations
;
```

If we use this as the data source- rather than using the location table directly, then in the outer query location type can be treated the same as the City and state. So just substitute the above query as a subquery in our outer join.

Demo 09: Using the subquery as a data source

```
select
  dept_id
, coalesce(loc_type, 'No Site') as LocationType
, coalesce(loc_city, 'No Site') as City
, coalesce(loc_state_province, 'No Site') as "State/Province"
from  a_emp.departments D
left join (
   select loc_id
   ,  coalesce(loc_type, 'UnknownType') as loc_type
   , loc_city, loc_state_province
   from a_emp.locations) L  on  D.loc_id = L.loc_id
  order by dept_id
;
+---------+--------------+---------------------+----------------+
| dept_id | LocationType | City                | State/Province |
+---------+--------------+---------------------+----------------+
|      10 | office       | San Francisco       | California     |
|      20 | warehouse    | Southlake           | Texas          |
|      30 | warehouse    | South San Francisco | California     |
|      35 | No Site      | No Site             | No Site        |
|      80 | warehouse    | Southlake           | Texas          |
|      90 | No Site      | No Site             | No Site        |
|      95 | No Site      | No Site             | No Site        |
|     210 | office       | Toronto             | Ontario        |
|     215 | UnknownType  | Munich              | Bavaria        |
+---------+--------------+---------------------+----------------+
```