Table of Contents

1. Any and All Operators 1
2. Using the All Operator 2
3. Finding the best(?) using the AltgeldMart tables 3
4. Using the Any Operator 5

1. Any and All Operators

The Any and All operators accept a list as an argument; you can compare the value returned by Any or All using the relational operators =, !=, >, <, >=, <=. The list is provided by a subquery.

Create a view that returns only the rows where the an_price is not null;

```
   Create view zoo_ex_notnull as (
       select id, an_type, an_price
       from zoo_ex
   where an_price is not null);
```

For reference, these are the rows in the zoo_ex table

```
        ID AN_TYPE           AN_PRICE
---------- --------------- ----------
         1 dog                     80
         2 turtle
         3 lizard
         4 bird                   100
         5 bird                    50
         6 fish                    10
         7 lizard                  50
         8 cat                     10
         9 snake                   50
        10 snake
        11 fish                    10
        12 lizard                  50
        13 fish                    10
        14 snake                   25
        15 bird                    80
        16 cat
        17 bird                    80
        18 dog                     80
        19 dog                     10
```

For reference, these are the rows in zoo_ex_notnull.

```
        ID AN_TYPE           AN_PRICE
---------- --------------- ----------
         1 dog                     80
         4 bird                   100
         5 bird                    50
         6 fish                    10
         7 lizard                  50
         8 cat                     10
         9 snake                   50
        11 fish                    10
        12 lizard                  50
        13 fish                    10
        14 snake                   25
        15 bird                    80
        17 bird                    80
        18 dog                     80
        19 dog                     10
```

2. Using the All Operator

The All operator is useful for finding the rows with largest value in a table including ties. We will start with a few examples using the view above to avoid issues with nulls.

Demo 01: We might want to find the most expensive animal.

```
select *
from zoo_ex_notnull
where an_price >=  ALL(
    select an_price
    from zoo_ex_notnull
   );
```

```
        ID AN_TYPE           AN_PRICE
---------- --------------- ----------
         4 bird                   100
```

If we tried this without the All operator we would get an error that the subquery returns more than one row. But for our query we want all of the rows since we want to find a row with a value for an_price that is larger than or equal to every row in the view.

Demo 02: Try this with > ALL

```
select *
from zoo_ex_notnull
where an_price >  ALL(
    select an_price
    from zoo_ex_notnull
   );
```

```
no rows selected
```

We get the empty set since there is no row where an_price is larger than every row since that would mean that there is a value for an_price that is larger than itself.

Demo 03: Now filter the two parts of the query for dog and when we filter for the most expensive dog, we get back both dog rows that were tied for the first place. This is probably the easiest way to code find the biggest with ties.

```
select *
from zoo_ex_notnull
where an_type = 'dog'
and an_price >=  ALL(
    select an_price
    from zoo_ex_notnull
    where an_type = 'dog'
   );
```

```
        ID AN_TYPE           AN_PRICE
---------- --------------- ----------
         1 dog                     80
        18 dog                     80
```

Demo 04: What if we try the same logic with the table which includes nulls; we get no rows returned because sql does not know if the null/missing prices are greater than 100 - the greatest actual value we have)

```
select *
from zoo_ex
```

```
where an_price >=  ALL(
    select an_price
    from zoo_ex
   );
```

```
no rows selected
```

Demo 05:

```
select *
from zoo_ex
where an_price >=  ALL(
    select an_price
    from zoo_ex
    where  an_price is not null
   );
```

```
        ID AN_TYPE           AN_PRICE
---------- --------------- ----------
         4 bird                   100
```

Demo 06: We could find the animal type where all of the animals of that type have the same price. For this we will exclude any nulls. This is a correlated subquery

```
select distinct an_type
from zoo_ex p1
where an_price is not null
and an_price = All (
    select an_price
    from zoo_ex  p2
    where an_price is not null
    and p1.an_type = p2.an_type
    ) ;
```

```
AN_TYPE
---------------
cat
fish
lizard
```

If you want to exclude any an_type (such as cat) where there is only one row of that type, then add a group by and a Having clause count(*) > 1.

3. Finding the best(?) using the AltgeldMart tables

Sometimes we need to analyze data and find the item that is- in some sense- the best among the data. For example we could be asked to find the best selling product. The first thing to do is to get a better definition of "best selling". We will get to this in a moment.

I am going to add another order for a sporting goods item so that we will have a tie for this category in terms of orders. I will use order id 1 since that will be easier to delete later.

```
insert into oe_orderHeaders (order_id, order_date, order_mode, customer_id, 
shipping_mode_id, order_status, sales_rep_id)
       values ( 1, date '2014-06-20', 'DIRECT', 404950, 'FEDEX1', 1, 155);
insert into oe_orderDetails (order_id, line_item_id, prod_id, quoted_price, 
quantity_ordered)
       values ( 1, 1, 1020, 2200.00, 10);
```

To remove these later use

```
delete from oe_orderDetails  where order_id IN(1);
delete from oe_orderHeaders  where order_id IN(1);
```

Demo 07: Let's start with a count function; we are interested in sales of products so we should use the order details table and I will limit this to the SPG category to keep the results short.

```
select prod_id, catg_id, count(distinct order_id ) as Cnt
from oe_orderDetails  OD
join prd_products PR using(prod_id)
where catg_id = 'SPG'
group by prod_id, catg_id
order by Cnt;
```

```
   PROD_ID CATG_I        CNT
---------- ------ ----------
      1030 SPG             4
      1050 SPG             4
      1040 SPG             8
      1060 SPG             9
      1020 SPG            12
      1010 SPG            12
```

We want to count distinct order id in case some product was ordered twice on the same order. (that is a business decision.)

```
select order_id, line_item_id, prod_id
from oe_orderDetails
where order_id = 312
;
```

```
  ORDER_ID LINE_ITEM_ID    PROD_ID
---------- ------------ ----------
       312            1       1040
       312            2       1050
       312            3       1060
       312            4       1060
```

Demo 08: Now we can find the row with the largest value for CntOrders for the SPG category. We will need to consider the possibilities of ties so we cannot just sort and take the last row

```
select prod_id, prod_name, prod_desc
from oe_orderDetails
join prd_products PR using(prod_id)
where catg_id = 'SPG'
group by prod_id, prod_name, prod_desc
having count(distinct order_id) >= All(
   select count(distinct  order_id)
   from oe_orderDetails
   join prd_products PR using(prod_id)
   where catg_id = 'SPG'
   group by prod_id)
;
```

```
   PROD_ID PROD_NAME       PROD_DESC
---------- --------------- ---------------------------------
      1020 Dartboard       Cork-backed dartboard with hanger
      1010 Weights         Set of 12 barbells 15 pounds
```

Demo 09: What if our definition of "best selling" should be based on the quantity of items sold?

```
select prod_id, prod_name, prod_desc
from oe_orderDetails
join prd_products PR using(prod_id)
where catg_id = 'SPG'
group by prod_id, prod_name, prod_desc
```

```
having sum(quantity_ordered) >= All(
   select sum(quantity_ordered)
   from oe_orderDetails
   join prd_products PR using(prod_id)
   where catg_id = 'SPG'
   group by prod_id);
```

```
  PROD_ID PROD_NAME       PROD_DESC
---------- --------------- --------------------------------
      1010 Weights         Set of 12 barbells 15 pounds
```

Demo 10: What if our definition of "best selling" should be based on the sales amount (total of price * quantity)?

```
select prod_id, prod_name, prod_desc
from oe_orderDetails
join prd_products PR using(prod_id)
where catg_id = 'SPG'
group by prod_id, prod_name, prod_desc
having sum(quantity_ordered * quoted_price) >= All(
   select sum(quantity_ordered*quoted_price)
   from oe_orderDetails
   join prd_products PR using(prod_id)
   where catg_id = 'SPG'
   group by prod_id)
;
```

```
  PROD_ID PROD_NAME       PROD_DESC
---------- --------------- --------------------------------
      1020 Dartboard       Cork-backed dartboard with hanger
```

4. Using the Any Operator

The Any operator is similar to All. The words Any and Some are interchangeable. I do not find this operator as useful as the ALL operator. In some cases you can use Any instead of an In list.

Demo 11: This is an ANY test on price. If we ask to see all of the rows with a price greater than any of the prices we get rows returned. This means we want prices greater than any of the other prices- essentially all prices greater than the smallest price in the table(in our case the vaue 10.); it does not return rows with nulls.

```
select *
from zoo_ex
where an_price > ANY (select an_price from zoo_ex)
order by an_price
;
```

```
        ID AN_TYPE           AN_PRICE
---------- --------------- ----------
        14 snake                   25
        12 lizard                  50
         9 snake                   50
         5 bird                    50
         7 lizard                  50
        15 bird                    80
        18 dog                     80
        17 bird                    80
         1 dog                     80
         4 bird                   100
```

Demo 12: Which animals cost the same as a bird- any bird?

```
select *
from zoo_ex
where an_price = ANY (
    select an_price
    from zoo_ex
    where an_type ='bird'
    and an_price is not null
    )
order by an_price;
```

```
        ID AN_TYPE           AN_PRICE
---------- --------------- ----------
         5 bird                    50
         7 lizard                  50
         9 snake                   50
        12 lizard                  50
         1 dog                     80
        15 bird                    80
        17 bird                    80
        18 dog                     80
         4 bird                   100
```