Table of Contents

1. Any and All Operators 1
1.1. Some examples using the altgeld-mart tables. 6
2. Finding the best(?) 8

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; Oracle will also accept a list of literals. The words Any and Some are interchangeable.

Set up this table in a_testbed.

```
   create table TodaysSpecials (an_type varchar(15));
   insert into TodaysSpecials values ('fish');
   insert into TodaysSpecials values ('cat');
```

For reference, these are the rows in the zoo_ex table

```
   +----+---------+----------+
   | id | an_type | an_price |
   +----+---------+----------+
   |  1 | dog     |       80 |
   |  2 | turtle  |     NULL |
   |  3 | lizard  |     NULL |
   |  4 | bird    |      100 |
   |  5 | bird    |       50 |
   |  6 | fish    |       10 |
   |  7 | lizard  |       50 |
   |  8 | cat     |       10 |
   |  9 | snake   |       50 |
   | 10 | snake   |     NULL |
   | 11 | fish    |       10 |
   | 12 | lizard  |       50 |
   | 13 | fish    |       10 |
   | 14 | snake   |       25 |
   | 15 | bird    |       80 |
   | 16 | cat     |     NULL |
   | 17 | bird    |       80 |
   +----+---------+----------+
```

Demo 01: This uses ANY and says to return the rows from the zoo_ex table where the an_type has any of those in the todaysSpecial table. There are other ways- such as a join- to do this.

```
   select *
   from zoo_ex
   where an_type = ANY  (
       select an_type
       from todaysSpecials ) ;
   +----+---------+----------+
   | id | an_type | an_price |
   +----+---------+----------+
   |  6 | fish    |       10 |
   |  8 | cat     |       10 |
   | 11 | fish    |       10 |
   | 13 | fish    |       10 |
   | 16 | cat     |     NULL |
   +----+---------+----------+
```

Demo 02: If we do this with ALL then no rows are returned because no row in zoo_ex has an value for an_type that matches all of the values in the todaysSpecial table.

```
select *
from zoo_ex
where an_type = ALL  (
    select an_type
    from todaysSpecials
    ) ;
```

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

Demo 03: Now we can do 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.

```
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 |
|  5 | bird    |       50 |
|  7 | lizard  |       50 |
|  9 | snake   |       50 |
| 12 | lizard  |       50 |
|  1 | dog     |       80 |
| 15 | bird    |       80 |
| 17 | bird    |       80 |
|  4 | bird    |      100 |
+----+---------+----------+
```

Demo 04: This uses greater than or equal and still does not return all the rows.

```
select *
from zoo_ex
where  an_price >= ANY (select an_price from zoo_ex)
order by an_price;
+----+---------+----------+
| id | an_type | an_price |
+----+---------+----------+
| 13 | fish    |       10 |
| 11 | fish    |       10 |
|  8 | cat     |       10 |
|  6 | fish    |       10 |
| 14 | snake   |       25 |
|  7 | lizard  |       50 |
|  5 | bird    |       50 |
| 12 | lizard  |       50 |
|  9 | snake   |       50 |
| 15 | bird    |       80 |
|  1 | dog     |       80 |
| 17 | bird    |       80 |
|  4 | bird    |      100 |
+----+---------+----------+
```

Now test similar queries using the ALL operator

Demo 05: It makes sense that we have no rows with a price greater than all of the prices.

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

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

Demo 06: With MySQL we do get a result- this seems to be inconsistent since the table has a null price.

```
select *
from zoo_ex
where  an_price >= All (select an_price from zoo_ex)
order by an_price;
+----+---------+----------+
| id | an_type | an_price |
+----+---------+----------+
|  4 | bird    |      100 |
+----+---------+----------+
```

Demo 07: The table has some nulls in the price attribute so we need to handle that. I would suggest using this syntax if the table being tested might contain nulls

```
select *
from zoo_ex
where  an_price >= All (
    select an_price
    from zoo_ex
    where an_price is not null
    )
order by an_price;
+----+---------+----------+
| id | an_type | an_price |
+----+---------+----------+
|  4 | bird    |      100 |
+----+---------+----------+
```

Now we add another filter to the subquery

Demo 08: 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_price is not null
    and an_type ='bird'
    );
+----+---------+----------+
| id | an_type | an_price |
+----+---------+----------+
|  1 | dog     |       80 |
|  4 | bird    |      100 |
|  5 | bird    |       50 |
|  7 | lizard  |       50 |
|  9 | snake   |       50 |
| 12 | lizard  |       50 |
| 15 | bird    |       80 |
| 17 | bird    |       80 |
+----+---------+----------+
```

Demo 09: Which animals cost the same as a lizard- any lizard?

```
select *
from zoo_ex
where an_price = ANY (
    select an_price
    from zoo_ex
    where an_type ='lizard'
    and an_price is not null
    );
+----+---------+----------+
| id | an_type | an_price |
+----+---------+----------+
|  5 | bird    |       50 |
|  7 | lizard  |       50 |
|  9 | snake   |       50 |
| 12 | lizard  |       50 |
+----+---------+----------+
```

Demo 10: Which animals cost the same as a bird- all of the birds? We get no rows returned because we have birds at different prices.

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

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

Demo 11: Which animals cost the same as a lizard- all of the lizards? This time we do get rows because all of our lizards have the same price.

```
select *
from zoo_ex
where an_price = All (
    select an_price
    from zoo_ex
    where an_price is not null
    and an_type ='lizard'
    );
+----+---------+----------+
| id | an_type | an_price |
+----+---------+----------+
|  5 | bird    |       50 |
|  7 | lizard  |       50 |
|  9 | snake   |       50 |
| 12 | lizard  |       50 |
+----+---------+----------+
```

Maybe we could see which categories of animals we have where all of the rows for that type of animal have the same price.

We need to consider where or not we want to ignore the nulls.

Demo 12:

```
select distinct an_type
from zoo_ex p1
where 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 |
+---------+
| dog     |
| turtle  |
| fish    |
| lizard  |
| cat     |
+---------+
```

Demo 13:

```
select distinct an_type
from zoo_ex p1
where an_price = All (
    select an_price
    from   zoo_ex  p2
    where  p1.an_type = p2.an_type
    )
;
+---------+
| an_type |
+---------+
| dog     |
| fish    |
+---------+
```

Maybe we do not want to include an types where there is only one animal of that type.

Demo 14:

```
select an_type
from zoo_ex p1
where an_price = All (
    select an_price
    from zoo_ex  p2
    where an_price is not null
    and p1.an_type = p2.an_type
    )
group by an_type
having count(*) > 1
;
+---------+
| an_type |
+---------+
| fish    |
| lizard  |
+---------+
```

Demo 15:

```
   select distinct an_type
   from  zoo_ex p1
   where an_price = All (
       select an_price
       from zoo_ex  p2
       where p1.an_type = p2.an_type
       )
   group by an_type
   having count(*) > 1
    ;
   +---------+
   | an_type |
   +---------+
   | fish    |
   +---------+
```

1.1. Some examples using the altgeld-mart tables.

We sometimes sell products at the current list price value and sometime the sale price is different. We could ask to see the items which are sold at their list price and those which are sold at less than their list price. We will limit this to HD items to make it easier to see the results.

Demo 16: Some intro queries to see the data we are working with.

-- these are the HD items we carry.

```
   select PR.prod_id, PR.prod_name
   from   a_prd.products  PR
   where  catg_id = 'HD'
   order by prod_id
   ;
   +---------+------------------+
   | prod_id | prod_name        |
   +---------+------------------+
   |    5002 | Ball-Peen Hammer |
   |    5004 | Dead Blow hammer |
   |    5005 | Shingler Hammer  |
   |    5008 | Claw Framing     |
   +---------+------------------+
```

-- these are the orders for the HD items

```
   select PR.prod_id
   , PR.prod_name
   , PR.prod_list_price
   , OD.quoted_price
   , PR.prod_list_price - OD.quoted_price as price_diff
   from a_prd.products  PR
   join a_oe.order_details  OD on PR.prod_id = OD.prod_id
   where PR.catg_id = 'HD'
   order by PR.prod_id;
```

```
+---------+------------------+-----------------+--------------+------------+
| prod_id | prod_name        | prod_list_price | quoted_price | price_diff |
+---------+------------------+-----------------+--------------+------------+
|    5002 | Ball-Peen Hammer |           23.00 |        23.00 |       0.00 |
|    5002 | Ball-Peen Hammer |           23.00 |        23.00 |       0.00 |
|    5002 | Ball-Peen Hammer |           23.00 |        23.00 |       0.00 |
|    5004 | Dead Blow hammer |           15.00 |        15.00 |       0.00 |
|    5005 | Shingler Hammer  |           45.00 |        45.00 |       0.00 |
|    5005 | Shingler Hammer  |           45.00 |        42.15 |       2.85 |
|    5005 | Shingler Hammer  |           45.00 |        42.50 |       2.50 |
|    5008 | Claw Framing     |           12.50 |        10.00 |       2.50 |
|    5008 | Claw Framing     |           12.50 |         8.00 |       4.50 |
+---------+------------------+-----------------+--------------+------------+
```

The Claw Framing hammer (5008) was always sold at less than its list price.

The Shingler hammer (5005) was sometimes sold at its list price and sometimes less than its list price.

The Dead Blow hammer (5004) and the Ball-Peen hammer (5002) were always sold at their list price.

So we want to write queries to do this logic.

Demo 17: This uses > **ALL**

```
select distinct PR.prod_id, pr.prod_name
from a_prd.products  PR
join a_oe.order_details  OD on PR.prod_id = OD.prod_id
where  catg_id = 'HD'
and prod_list_price > ALL (
     select quoted_price
     from a_oe.order_details  OD2
     where OD2.prod_id =PR.prod_id
     );
+---------+--------------+
| prod_id | prod_name    |
+---------+--------------+
|    5008 | Claw Framing |
+---------+--------------+
```

Demo 18: This uses > **ANY**

```
select distinct PR.prod_id, pr.prod_name
from a_prd.products  PR
join a_oe.order_details  OD on PR.prod_id = OD.prod_id
where catg_id = 'HD'
and  prod_list_price > ANY (
        select quoted_price
        from a_oe.order_details  OD2
        where OD2.prod_id =PR.prod_id
        );
+---------+-----------------+
| prod_id | prod_name       |
+---------+-----------------+
|    5005 | Shingler Hammer |
|    5008 | Claw Framing    |
+---------+-----------------+
```

Demo 19: This uses = **ALL**

```
select distinct PR.prod_id, pr.prod_name
from a_prd.products  PR
```

```
join  a_oe.order_details  OD on PR.prod_id = OD.prod_id
where  catg_id = 'HD'
and  prod_list_price = ALL (
      select quoted_price
      from a_oe.order_details  OD2
      where  OD2.prod_id =PR.prod_id );
+---------+------------------+
| prod_id | prod_name        |
+---------+------------------+
|    5002 | Ball-Peen Hammer |
|    5004 | Dead Blow hammer |
+---------+------------------+
```

Demo 20: This uses = **ANY**

```
select distinct PR.prod_id, pr.prod_name
from a_prd.products  PR
join a_oe.order_details  OD on PR.prod_id = OD.prod_id
where  catg_id = 'HD'
and  prod_list_price = ANY (
        select quoted_price
        from a_oe.order_details  OD2
        where OD2.prod_id =PR.prod_id) ;
+---------+------------------+
| prod_id | prod_name        |
+---------+------------------+
|    5002 | Ball-Peen Hammer |
|    5004 | Dead Blow hammer |
|    5005 | Shingler Hammer  |
+---------+------------------+
```

2. Finding the best(?)

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.

Let's start with a count function; we are interested in sales of products so we should use the order details table.

```
select prod_id, count(*) as Cnt
from a_oe.order_details  OD
group by prod_id
order by 2;
+---------+-----+
| prod_id | Cnt |
+---------+-----+
|    1140 |   1 |
|    4576 |   1 |
|    5004 |   1 |
|    1151 |   2 |
. . .
|    1125 |   6 |
|    1010 |   7 |
|    1060 |   7 |
|    1080 |   7 |
|    1110 |   7 |
|    1130 |   9 |
+---------+-----+
```

What are we counting? We used count(*) so we are counting order detail rows. Is that the same as counting orders? Run the following query to look at order 312. Product 1060 appears on two lines in this order. So if we are counting orders for the product, the previous query is not correct.

```
   select ord_id, line_Item_id, prod_id
   from a_oe.order_details  OD
   where ord_id = 312;
   +--------+--------------+---------+
   | ord_id | line_Item_id | prod_id |
   +--------+--------------+---------+
   |    312 |            1 |    1040 |
   |    312 |            2 |    1050 |
   |    312 |            3 |    1060 |
   |    312 |            4 |    1060 |
   +--------+--------------+---------+
```

If we change the query to count distinct order_id values, then we get the proper counts for counting orders for a product (assuming we want to count order 312 as a single order for product 1060)

```
   select prod_id, count(distinct ord_id) as CntOrders
   from a_oe.order_details  OD
   group by prod_id
   order by 2;
   +---------+-----+
   | prod_id | Cnt |
   +---------+-----+
   |    1140 |   1 |
   |    4576 |   1 |
   |    5004 |   1 |
   |    1151 |   2 |
   . . .
   |    1125 |   6 |
   |    1060 |   6 |
   |    1010 |   7 |
   |    1080 |   7 |
   |    1110 |   7 |
   |    1130 |   9 |
   +---------+-----+
```

Now we can find the row with the largest value for CntOrders. We will need to consider the possibilities of ties so we cannot just sort and take the last row. When we say that product 1130 has the most orders we are saying that its count is bigger than the other counts; that means it is bigger than or equal to all of the counts.

```
   select prod_id
   from a_oe.order_details
   group by prod_id
   having count(distinct ord_id) >= All(
      select count(distinct  ord_id)
      from a_oe.order_details
      group by prod_id)
   ;
   +---------+
   | prod_id |
   +---------+
   |    1130 |
   +---------+
```

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

```
   select prod_id
   from   a_oe.order_details
   group by prod_id
   having sum(quantity_ordered) >=  All(
      select sum(quantity_ordered)
      from a_oe.order_details
      group by prod_id)
   ;
   +---------+
   | prod_id |
   +---------+
   |    1150 |
   +---------+
```

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

```
   select prod_id
   from   a_oe.order_details
   group by prod_id
   having sum(quantity_ordered*quoted_price) >= All(
      select sum(quantity_ordered*quoted_price)
      from a_oe.order_details
      group by prod_id)
   ;
   +---------+
   | prod_id |
   +---------+
   |    1010 |
   +---------+
```

If you want to test that tied values are returned you can temporarily add some rows. I would suggest using an order id of 1 and 2 so that is it easier to delete these. I am adding some rows for product 1010. To delete these rows delete the OD rows first and then the OH rows.

Deletes

```
   delete from  a_oe.order_details  where ord_id IN(1,2);
   delete from  a_oe.order_headers  where ord_id IN(1,2);
```

Inserts

```
   insert into a_oe.order_headers (ord_id, ord_date, ord_mode, cust_id,
   shipping_mode, ord_status, sales_rep_id)
          values ( 1, '2011-09-20', 'DIRECT', 404950, 'FEDEX1', 1, 155);
   insert into a_oe.order_details (ord_id, line_item_id, prod_id, quoted_price,
   quantity_ordered)
          values ( 1, 1, 1010,    1500.00,  1);
   insert into a_oe.order_details (ord_id, line_item_id, prod_id, quoted_price,
   quantity_ordered)
          values ( 1, 2, 1010,    1500.00,  1);

   insert into a_oe.order_headers (ord_id, ord_date, ord_mode, cust_id,
   shipping_mode, ord_status, sales_rep_id)
          values ( 2, '2011-09-20', 'DIRECT', 404950, 'FEDEX1', 1, 155);
   insert into a_oe.order_details (ord_id, line_item_id, prod_id, quoted_price,
   quantity_ordered)
          values ( 2, 1, 1010,    1500.00,  1);
```