Table of Contents

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. This use of Exists is only to start with a simpler syntax. **Almost all uses of Exists in realistic queries use correlated subqueries.**

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

         1 dog                80
        18 dog                80
        19 dog                10

         6 fish               10
        11 fish               10
        13 fish               10

         3 lizard
         7 lizard             50
        12 lizard             50

         9 snake              50
        10 snake
        14 snake              25

         2 turtle
```

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 rows still exist. The Exists operator can be negated.

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 Oracle 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.

```
---- no rows selected-
```

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.

```
'FOUN
-----
Found

1 row selected.
```

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 at 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')
  ;
```

```
'FOUN
-----
Found

1 row selected.
```

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')
  ;
```

```
---- no rows selected-
```

If we look 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
-----------

1 row selected.
```

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)  ;
```

```
'FOUN
-----
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 no rows 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)
  ;
```

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

1 row selected.
```

Now take queries 7 and 8 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: A: Snakes less than 75 and B: snakes less than 15

```
-- demo 10A:
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 : A: Snakes less than 75 and then B: snakes less than 15

```
-- demo 11A:
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.

This is a much more common use of Exists.

We are now using the tables in the altgeld_mart database.

Which customers have an order with a date in October 2015? 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 customer_id, customer_name_last, customer_name_first
from cust_customers
where  EXISTS (
     select 'X'
     from oe_orderHeaders
     where oe_orderHeaders.customer_id = cust_customers.customer_id
     and extract( MONTH from order_date) = 10
     and extract(Year from order_date) = 2015)
;
```

```
  CUSTOMER_ID CUSTOMER_NAME_LAST            CUSTOMER_NAME_FIRST
---------- ------------------------ ------------------------
    401250 Morse                    Samuel
    403000 Williams                 Sally
    403050 Hamilton                 Alexis
    403100 Stevenson                James
    404950 Morris                   William
```

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

-- demo 12B

```
select customer_id, customer_name_last, customer_name_first
from cust_customers
where customer_id in  (
     select customer_id
     from oe_orderHeaders
     where extract( MONTH from order_date) = 10
     and extract(Year from order_date) = 2015);
```

-- demo 12C This query may be less efficient if there is a sort done to handle the distinct.

```
select distinct customer_id, CS.customer_name_last, CS.customer_name_first
from cust_customers CS
join oe_orderHeaders OH using (customer_id)
where extract( MONTH from order_date) = 10
and extract(Year from order_date) = 2015;
```

-- demo 12D This query can return multiple rows for the same customer.

```
select customer_id, CS.customer_name_last, CS.customer_name_first
from cust_customers CS
join oe_orderHeaders  OH using (customer_id)
where extract( MONTH from order_date) = 10
and extract(Year from order_date) = 2015;
```

Demo 13: 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. Display customers who purchased at item at a discount greater than 10% of the list price .

```
select customer_id
, customer_name_last
, customer_name_first
from cust_customers
where EXISTS (
     select 1
     from oe_orderHeaders OH
     join oe_orderDetails OD using(order_id)
     join prd_products PR using(prod_id)
     where (prod_list_price - quoted_price)/prod_list_price > 0.1
     and cust_customers.customer_id = OH.customer_id)
order by customer_id;
```

```
CUSTOMER_ID CUSTOMER_NAME_LAST        CUSTOMER_NAME_FIRST
----------- ------------------------- ------------------------
     400300 McGold                    Arnold
     401250 Morse                     Samuel
     402100 Morise                    William
     403000 Williams                  Sally
     403050 Hamilton                  Alexis
     403100 Stevenson                 James
     404000 Olmsted                   Frederick
. . . rows omitted
```

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 14:

```
select customer_id
, customer_name_last
, customer_name_first
from cust_customers
where NOT EXISTS (
     select 1
     from oe_orderHeaders OH
     join oe_orderDetails OD using(order_id)
     join prd_products PR using(prod_id)
     where (prod_list_price - quoted_price)/prod_list_price > 0.1
     and cust_customers.customer_id = OH.customer_id)
order by customer_id;
```

Demo 15:

```
select customer_id
, customer_name_last
, customer_name_first
from cust_customers
where EXISTS (
     select 1
     from oe_orderHeaders OH
     join oe_orderDetails OD using(order_id)
     join prd_products PR using(prod_id)
     where not((prod_list_price - quoted_price)/prod_list_price > 0.1 )
     and cust_customers.customer_id = OH.customer_id)
;
```

Demo 16: 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 Prd_products PR
where EXISTS (
     select 'X'
     from oe_orderDetails 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
```

Demo 17: 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 cust_customers.customer_id
, customer_name_last
from cust_customers
where EXISTS (
     select 'X'
     from oe_customer_orders
     where oe_customer_orders.custID = Cust_customers.customer_id
     and category ='APL')
and  EXISTS (
     select 'X'
     from oe_customer_orders
     where oe_customer_orders.custID = Cust_customers.customer_id
     and category ='HW')
;
```

```
 CUSTOMER_ID CUSTOMER_NAME_LAST
----------- -------------------------
     404950 Morris
     404100 Button
     903000 McGold
     409150 Martin
     403000 Williams
     900300 McGold
     402100 Morise
     409030 Mazur
```

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

```
select CS.customer_id
, CS.customer_name_last
, case when EXISTS (
     select 'X'
     from oe_customer_orders
     where oe_customer_orders.custID = CS.customer_id
     and category ='APL')
  then 'Appliances  '
  else '    ---    '
  End as " "
, case when EXISTS (
    select 'X'
    from oe_customer_orders
    where oe_customer_orders.custID = CS.customer_id
    and category ='HW')
  then 'Housewares  '
  else '    ---    '
  End   as " "
, case when EXISTS (
     select 'X'
```

```
       from oe_customer_orders
       where oe_customer_orders.custID = CS.customer_id
       and category ='PET')
    then 'PetSupplies '
    else '    ---    '
    End   as " "
   from cust_customers  CS;
  -- Selected rows
```

```
CUSTOMER_ID CUSTOMER_NAME_LAST
----------- ------------------------ ------------ ------------ ------------
     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
     403500 Stevenson                    ---          ---          ---
     403750 O'Leary                      ---          ---          ---
     403760 O'Leary                      ---          ---          ---
     903000 McGold                   Appliances   Housewares   PetSupplies
```

Demo 19: A minor change in the query gives you a single column of purchase categories.

```
  select CS.customer_id
  , CS.customer_name_last
  , case when EXISTS (
       select 'X'
       from oe_customer_orders
       where oe_customer_orders.custID = CS.customer_id
       and category ='APL')
     then 'Appliances  '
     else ''
     End
  || case when EXISTS (
        select 'X'
        from oe_customer_orders
        where oe_customer_orders.custID = CS.customer_id
        and category ='HW')
     then 'Housewares  '
     else ''
     End
  || case when EXISTS (
        select 'X'
        from oe_customer_orders
        where oe_customer_orders.custID = CS.customer_id
        and category ='PET')
     then 'PetSupplies'
     else ''
     End   as PurchaseAreas
   from cust_customers  CS;
  -- Selected rows
```

```
CUSTOMER_ID CUSTOMER_NAME_LAST       PURCHASEAREAS
----------- ------------------------ -----------------------------------
     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
      403500 Stevenson
      403750 O'Leary
      403760 O'Leary
      903000 McGold                  Appliances  Housewares  PetSupplies
      404000 Olmsted                 PetSupplies
```