Table of Contents

1. Working up to Correlated Subqueries

We want to find out how many items (total quantity) are on each order we have. The first of these queries uses a scalar subquery in the Select but the output is not very interesting

Demo 01:

```
select customer_id, order_id
, (
    select sum(quantity_ordered)
    from oe_orderDetails  OD
   ) as "NumItems"
from oe_orderHeaders  OH
where rownum <=5;
```

```
  CUSTOMER_ID     ORDER_ID   NumItems
---------- ---------- ----------
    409150        218        950
    409150        223        950
    409160        224        950
    409160        225        950
    403050        227        950
```

This seems to say that every order has a total quantity of 950 items; that does not look correct. The scalar subquery calculated the total number of items purchased on **all** orders. Look at the subquery. It says:

```
select sum(quantity_ordered)
from oe_orderDetails  OD
```

What we want for each order is the total number of items **for this order**. We have to tie that subquery calculation to the Order id in the Order headers table. This is done via a correlation- the table in the subquery (order details) is tied to/joined to/correlated with the order header row we are looking at. This is different.

The subquery has a Where clause that correlates the subquery to the main query. OH is an alias for a table in the main query.

```
( select sum(quantity_ordered)
     from oe_orderDetails  OD
     where OH.order_id = OD.order_id)
```

Demo 02: Now our result set makes more sense and is more useful

```
select customer_id, order_id
, (  select sum(quantity_ordered)
     from oe_orderDetails  OD
     where OH.order_id = OD.order_id) as "NumItemsPerOrder"
from oe_orderHeaders  OH
where customer_id IN ( 404100, 903000)
order by customer_id, order_id;
```

```
  CUSTOMER_ID     ORDER_ID NumItemsPerOrder
------------- ------------ ----------------
       404100          303                1
       404100          605               43
       404100         2503                1
       404100         2504                1
       404100         2805               29
       903000          306                2
```

```
        903000          312              50
        903000          313               1
        903000          550              11
        903000          551              20
        903000          610               2
        903000         2120              10
        903000         2121              21
        903000         3810               2

14 rows selected.
```

The nested sub query has a Where clause that refers to the outer query. That way we get the total of the quantity for this particular order. **This is a correlated subquery**. The table reference in the subquery is joined to the table references in the outer query.

The main query uses the order headers table and we get one row per order header row. We are not doing a group by clause.

Demo 03: Grouping with an outer join. It is not unusual to have more than one way to do a query.

```
select customer_id, OH.order_id ,sum(quantity_ordered) AS NumPerOrder
from oe_orderHeaders  OH
left join oe_order_details  OD on OH.order_id = OD.order_id
where customer_id IN ( 404100, 903000)
group by Customer_id, OH.order_id
order by customer_id, OH.order_id
;
```

Demo 04: We might want to get the number of items per order and the cost of the order. This uses two subqueries.

```
select customer_id, order_id
, ( select SUM (quantity_ordered)
    from oe_orderDetails OD
    where OH.order_id = OD.order_id)  as "NumPerOrder"
, ( select SUM (quantity_ordered * quoted_price)
    from oe_orderDetails OD
    where OH.order_id = OD.order_id)  as "OrderCost"
from oe_orderHeaders OH
order by customer_id,order_id
;
```

```
CUSTOMER_ID   ORDER_ID NumPerOrder  OrderCost
----------- ---------- ----------- ----------
     400300        378          10       4500
     401250        106           1     255.95
     401250        113           1       22.5
     401250        119          10        225
     401250        301           1        205
     401250        552          10      157.3
     401250       2506
     401890        112           2      99.98
     401890        519           6     114.74
     402100        114           5        625
. . . .  ..
```

This query returns rows where we have an Order header row with no Detail rows. Why does it do that? How would you change the query to return only Order header row that have Detail rows?

2. Correlated Subqueries

With a non-correlated subquery, the inner query could work on its own. With a correlated subquery, the inner query refers to attributes found in the outer query. That means that the correlated subquery cannot be run independently. Logically, the outer query works on the first row and processes the subquery using the attributes in the first row; then the outer query works on the second row and then processes the subquery using the attributes in the second row; it then continues through the rest of the rows in the outer query reevaluating the subquery repeatedly.

Some of the following are correlated subqueries. Although a correlated subquery may seem inefficient, the efficiency depends on the optimizer for the database engine.

If you have the choice of solving a task with a correlated query or with non-correlated query, you should generally choose the non-correlated version.

Demo 05: This uses an aggregate function to get the average price for all products.

```
select round( Avg ( prod_list_price),2)
from prd_products;
```

```
AVG(PROD_LIST_PRICE)
--------------------
              117.67
```

Demo 06: This uses a subquery to get products that cost more than the average price for all products.

```
select prod_id, prod_list_price, catg_id
from prd_products
where prod_list_price > (
    select round( Avg ( prod_list_price),2)
    from prd_products
    );
```

```
   PROD_ID PROD_LIST_PRICE CATG_I
---------- --------------- ------
      1000          125.00 HW
      1010          150.00 SPG
      1090          149.99 HW
      1040          349.95 SPG
      1050          269.95 SPG
      1060          255.95 SPG
      1160          149.99 HW
      4567          549.99 PET
      4568          549.99 PET
      4569          349.95 APL
      1120          549.99 APL
      1125          500.00 APL
      1126          850.00 APL
      1130          149.99 APL
```

Demo 07: This uses grouping and an aggregate function to get the average price for each product category.

```
select round( Avg ( prod_list_price),2) as AvgPrice, catg_id
from prd_products
group by catg_id;
```

```
AVGPRICE  CATG_ID
--------- -------
   123.17 PET
    23.88 HD
    13.88 MUS
```

```
    67.64 HW
   178.13 SPG
   479.99 APL
     8.75 GFD
```

Demo 08: We can use a **correlated subquery** to get the products that cost more than the average price for the **same type of products**. Notice that prd_products occurs in both the parent and the child query so we need to use table aliases in the join.

```
select prod_id, catg_id, prod_list_price
from prd_products  Otr
where prod_list_price > (
     select Round( Avg ( prod_list_price),2)
     from prd_products  Inr
     where Otr.catg_id = Inr.catg_id )
order by catg_id, prod_list_price
;
```

```
   PROD_ID CATG_I PROD_LIST_PRICE
---------- ------ ---------------
      1125 APL             500.00
      1120 APL             549.99
      1126 APL             850.00
      5000 GFD              12.50
      5005 HD               45.00
      1000 HW              125.00
      1090 HW              149.99
      1160 HW              149.99
      2746 MUS              14.50
      2747 MUS              14.50
      2987 MUS              15.87
      2984 MUS              15.87
      2337 MUS              15.87
      2234 MUS              15.88
      2014 MUS              15.95
      4567 PET             549.99
      4568 PET             549.99
      1060 SPG             255.95
      1050 SPG             269.95
      1040 SPG             349.95

20 rows selected.
```

Consider the subquery:

```
select AVG(prod_list_price)
from prd_products    Inr
where Otr.catg_id = Inr.catg_id
```

This does not include a Group By clause. The subquery looks at only one value for catg_id - the one that matches the value for catg_id in the outer query for the current row being considered.

Since it is looking at only one category id, it will find only one average and so we can use the average in a filter of the type Price > average.

The one thing that should look odd about the subquery it that is refers to a table with an alias Otr that is not part of the subquery. That is where the "correlated" part of the correlated subquery comes in.

So let's go back to the main query. It gets one row from the product table and then tries to compare the price of that row to the average- what average; the average calculated by the subquery- which is the average for the same category id as that on the products table row we are looking at.

So the way to think of this query is

-- working one row at a time through the products table

-- for each row in the products table (one at a time) calculate the average price for that product id

-- if the price for that product row is > average price for that category id, then return it.

This makes it sound like a very inefficient way to do this. Imagine we have a product tables of 50,000 rows of 10 different category ids. If the dbms actually carried the query out as I just described, that would mean calculated the average price 50,000 times. The dbms generally has a more efficient way - internally- to do this type of query. But logically you should think of the query as working with one row from the outer query processed against the subquery.

We want to find orders that are unusually high for a customer. Because we don't have a lot of rows, I defined this as an order that is more than 1.25 times the average order cost for that customer. This uses a CTE to assemble the data being used.

Demo 09: We will need the average order size by customer.

```
With OE_OrdExtTotal as (
    select OH.customer_id
    , order_id
    , sum ( OD.Quantity_ordered * OD.quoted_price) AS ordertotal
    from oe_orderHeaders OH
    join oe_orderDetails OD  using(order_id)
    group by OH.customer_id, order_id
    )
select customer_id as "custid"
, to_char(avg(ordertotal), 9999.99) as  "AvgPrice"
, to_char(1.25 * AVG(ordertotal), 9999.99)  as "Cut_off"
from OE_OrdExtTotal
group by customer_id
order by customer_id;
```

```
    custid AvgPrice Cut_off
---------- -------- --------
    400300  4500.00  5625.00
    401250   173.15   216.44
    401890   107.36   134.20
    402100  1092.32  1365.40
    403000   727.30   909.12
    403010  1900.00  2375.00
    403050   269.23   336.54
    403100   218.84   273.55
. . . rows omitted
```

Demo 10: Now we need a **correlated subquery** to compare a particular order with average orders for that customer.

```
With OE_OrdExtTotal as (
    select OH.customer_id
    , order_id
    , SUM ( OD.Quantity_ordered * OD.quoted_price) AS ordertotal
    from Oe_orderHeaders OH
    join Oe_orderDetails OD  using(order_id)
    group by OH.customer_id, order_id
    )
```

```
select customer_id as "custid"
, order_id as "ordid"
, to_char( ordertotal, 9999.99) as  "OrderCost"
from OE_OrdExtTotal  OTR
where OrderTotal > 1.25 *  (
     select AVG (ordertotal)
     from OE_OrdExtTotal    INR
     where OTR.Customer_id = INR.Customer_id
     group by customer_id
     )
order by customer_id;
```

```
custid       ordid                OrdTotal
------------ -------------------- --------
401250       106                    255.95
401250       119                    225.00
402100       115                   2305.00
403000       390                   1400.00
403000       528                   2629.00
403000       105                   1205.40
403000       395                   2925.00
403050       527                    440.47

. . . rows omitted
```

Let's start with the first customer shown here- customer_id 401250. This customer has 5 orders:

```
   CUSTOMER_ID     ORDER_ID ORDERTOTAL
---------- ---------- ----------
    401250        113      22.50
    401250        552     157.30
    401250        301     205.00
    401250        119     225.00
    401250        106     255.95
```

The cutoff for this customer is 216.44. Our query returns only two of this customers's orders- the ones where the order total is over this customer's cutoff.

```
401250       106                    255.95
401250       119                    225.00
```

Demo 11: To get customers with more than one order, we can use the count function for each customer_id as it occurs in the outer query; we do not need to qualify customer_id with Oe_order_headers in the inner query.

```
select customer_id
, customer_name_last
from cust_customers
where 1 <  (
     select count ( *)
     from oe_orderHeaders
     where customer_id = cust_customers.customer_id )
;
```

```
CUSTOMER_ID CUSTOMER_NAME_LAST
----------- --------------------
     401250 Morse
     401890 Northrep
     402100 Morise
     403000 Williams
     403050 Hamilton
     403100 Stevenson

. . . rows omitted
```

Demo 12: Here the correlated subquery returns a number which is used as a parameter to a case expression. We want to use the number of orders for this customer- not for all customers.

```
select customer_id, customer_name_last
 , Case (
        select count(*)
        from oe_orderHeaders OH
        where OH.customer_id = CS.customer_id
        )
      when 0 then '. . . No orders'
      when 1 then '1 order'
      when 2 then '2 orders'
      when 3 then '3 orders'
   else  '4+ orders'
   end as NumberOfOrders
from cust_customers  CS
;
```

```
CUSTOMER_ID CUSTOMER_NAME_LAST        NUMBEROFORDERS
----------- ------------------------- ---------------
     401250 Morse                     4+ orders
     401890 Northrep                  2 orders
     402100 Morise                    3 orders
     402110 Coltrane                  . . . No orders
     402120 McCoy                     . . . No orders
     402500 Jones                     . . . No orders
     403000 Williams                  4+ orders
     403010 Otis                      1 order
     403050 Hamilton                  4+ orders
     403100 Stevenson                 4+ orders
. . . rows omitted
```