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 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.

Demo 01: This uses an aggregate function to get the average price for all products.

```
Select AVG( prod_list_price)as avgprice
From a_prd.products  ;
+------------+
| avgprice   |
+------------+
| 117.670816 |
+------------+
```

Demo 02: 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 a_prd.products
Where prod_list_price >(
     Select AVG( prod_list_price)
     From a_prd.products
     );
+---------+-----------------+---------+
| prod_id | prod_list_price | catg_id |
+---------+-----------------+---------+
|    1000 |          125.00 | HW      |
|    1010 |          150.00 | SPG     |
|    1040 |          349.95 | SPG     |
|    1050 |          269.95 | SPG     |
|    1060 |          255.95 | SPG     |
|    1090 |          149.99 | HW      |
|    1120 |          549.99 | APL     |
|    1125 |          500.00 | APL     |
|    1126 |          850.00 | APL     |
|    1130 |          149.99 | APL     |
|    1160 |          149.99 | HW      |
|    4567 |          549.99 | PET     |
|    4568 |          549.99 | PET     |
|    4569 |          349.95 | APL     |
+---------+-----------------+---------+
```

Demo 03: This uses grouping and an aggregate function to get the average price for each product category.

```
Select AVG( prod_list_price)as avgPrice, catg_id
From a_prd.products
Group by catg_id;
+------------+---------+
| avgPrice   | catg_id |
+------------+---------+
| 479.986000 | APL     |
|   8.750000 | GFD     |
|  23.875000 | HD      |
|  67.641000 | HW      |
| 125.688182 | PET     |
| 178.125000 | SPG     |
+------------+---------+
```

Demo 04: 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 a_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 a_prd.products    Otr
Where prod_list_price >(
     Select AVG(prod_list_price)
     From a_prd.products    Inr
     Where Otr.catg_id = Inr.catg_id
     )
order by catg_id, prod_list_price;
+---------+---------+-----------------+
| prod_id | catg_id | 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 |
|    2747 | MUS     |           14.50 |
|    2746 | MUS     |           14.50 |
|    2987 | MUS     |           15.87 |
|    2337 | MUS     |           15.87 |
|    2984 | 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 |
+---------+---------+-----------------+
```

Consider the subquery:

```
     Select AVG(prod_list_price)
     From a_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.

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 will take several steps to develop.

Demo 05: Create a view that uses grouping and an aggregate function to get the total due for each order.

```
CREATE VIEW a_oe.OE_OrdExtTotal AS
    Select ord.cust_id
    , ord.ord_id
    , SUM(odt.Quantity_ordered * odt.quoted_price)AS ordertotal
    From a_oe.order_headers ord
    Join a_oe.order_details odt on ord.ord_id = odt.ord_id
    Group by ord.cust_id, ord.ord_id
;
```

```
View created.
```

Demo 06: We will need the average order size by customer.

```
Select cust_id
, round( AVG(ordertotal), 2)  frm_avg
, 1.25 * round(AVG(ordertotal),2 )  cut_off
From a_oe.OE_OrdExtTotal
Group by cust_id;
+---------+---------+-----------+
| cust_id | frm_avg | cut_off   |
+---------+---------+-----------+
|  400300 | 4500.00 | 5625.0000 |
|  401250 |  177.11 |  221.3875 |
|  401890 |  107.36 |  134.2000 |
|  402100 | 1092.32 | 1365.4000 |
|  403000 |  896.28 | 1120.3500 |
|  403010 | 1900.00 | 2375.0000 |
|  403050 |  212.15 |  265.1875 |
|  403100 |  218.84 |  273.5500 |
|  404000 |  469.95 |  587.4375 |
|  404100 |  141.66 |  177.0750 |
|  404900 |   38.85 |   48.5625 |
|  404950 |  212.89 |  266.1125 |
|  408770 | 1068.75 | 1335.9375 |
|  409030 |  751.97 |  939.9625 |
|  409150 |  181.82 |  227.2750 |
|  409160 |   65.24 |   81.5500 |
|  409190 |   49.99 |   62.4875 |
|  900300 | 4500.00 | 5625.0000 |
|  903000 | 3216.05 | 4020.0625 |
|  915001 |  212.48 |  265.6000 |
+---------+---------+-----------+
```

Demo 07: Now we need a **correlated subquery** to compare a particular order with average orders for that customer.

```
Select cust_id
, ord_id
, ordertotal
From a_oe.OE_OrdExtTotal  OTR
Where OrderTotal > 1.25 * (
     Select round( AVG(ordertotal), 2)
     From a_oe.OE_OrdExtTotal    INR
     Where OTR.Cust_id = INR.Cust_id
     Group by cust_id
     );
+---------+--------+------------+
| cust_id | ord_id | ordertotal |
+---------+--------+------------+
|  401250 |    106 |     255.95 |
|  401250 |    119 |     225.00 |
|  402100 |    115 |    2305.00 |
|  403000 |    105 |    1205.40 |
|  403000 |    390 |    1400.00 |
|  403000 |    395 |    2925.00 |
|  403000 |    528 |    2629.00 |
|  403050 |    527 |     440.47 |
|  403100 |    400 |     465.00 |
|  404950 |    110 |     299.98 |
|  404950 |    535 |     525.00 |
|  409030 |    130 |    1145.00 |
|  409150 |    518 |     727.97 |
|  409160 |    524 |     175.99 |
|  903000 |    312 |    9405.00 |
|  915001 |    129 |     299.97 |
+---------+--------+------------+
```

Demo 08: To get customers with more than one order, we can use the count function for each cust_id as it occurs in the outer query; we do not need to qualify cust_id with A_oe.order_headers in the inner query.

```
Select cust_id
, cust_name_last
From a_oe.customers  C
Where 1 < (
     Select COUNT( *)
     From a_oe.order_headers
     Where cust_id = C.cust_id
     );
+---------+----------------+
| cust_id | cust_name_last |
+---------+----------------+
|  401250 | Morse          |
|  401890 | Northrep       |
|  402100 | Morise         |
|  403000 | Williams       |
|  403050 | Hamilton       |
|  403100 | Stevenson      |
|  404100 | Button         |
|  404950 | Morris         |
|  409030 | Mazur          |
```

```
|  409150 | Martin          |
|  409160 | Martin          |
|  903000 | McGold          |
|  915001 | Adams           |
+---------+-----------------+
```

Demo 09: 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 cust_id, cust_name_last
 , Case (
        select count(*)
        from a_oe.order_headers OH
        where OH.cust_id = CS.cust_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 a_oe.customers  CS ;
+---------+-----------------+------------------+
| cust_id | cust_name_last  | NumberOfOrders   |
+---------+-----------------+------------------+
|  400300 | McGold          | 1 order          |
|  400801 | Washington      | . . . No orders  |
|  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        | 3 orders         |
```