Table of Contents

1. Scalar Subquery

A subquery is a Select expression that is placed within a query. The subquery is generally placed within parentheses.

A scalar subquery returns a single value (one row and one column) . This is still a table, but is sometimes called a scalar subquery expression to emphasize that the return has a single value. The value of a scalar subquery is the value in the select. It can be used in places where the value of the expression can be used. You can test a scalar subquery with an equality test and you can use a scalar query in places where a single value is allowed. We have seen these starting in Unit 5; we used them as part of a Where clause filter. For example

Demo 01: A simple subquery- we want to find employees who work in the same department as employee with ID 162

```
select   emp_id, name_last as "Employee"
from     a_emp.employees
where    dept_id =
         (
          select  dept_id
          from    a_emp.employees
          where   emp_id = 162
);
+--------+----------+
| emp_id | Employee |
+--------+----------+
|    162 | Holme    |
|    200 | Whale    |
|    207 | Russ     |
+--------+----------+
```

Each employee has exactly one dept_id value so, if we have an employee 162 the subquery will return exactly one row and the select in the subquery has only one column so this subquery returns a single value. That value can then be used by the Where clause in the main query.

But what happens if we do not have an employee with id 162? In that case the subquery expression is null and no rows are returns by the main query.

If you are using subqueries in this ways, it is your responsibility to ensure that the subquery returns a single row and a single column. Suppose you change the query as shown here.

Demo 02:

```
select   emp_id
,        name_last as "Employee"
from     a_emp.employees
where    dept_id =
         (select  dept_id
          from    a_emp.employees);
```

Now the subquery might return more than one row and in that case we would get an error message

```
ERROR 1242 (21000): Subquery returns more than 1 row
```

Demo 03: Now consider the following query. Will this query run without error?

```
select   emp_id
,        name_last as "Employee"
from     a_emp.employees
where    dept_id =
         (select  dept_id
          from    a_emp.employees
          Where name_last = 'Green'
          );
```

If we have no employees with the last name of Green, the query will run; the subquery returns a null and the main query returns no rows. If we have exactly one employee with the last name of Green then the subquery returns the department id for that employee and the main queries returns all employees for that department. If we have more than one employee with the last name of Green, then the subquery is no longer a single-row subquery and you get an error message.

It is up to the person developing the query to ensure that in a query such as this that the subquery returns one or no rows. You don't get to say that when you set up the query we had only one employee with the name Green and it is not your fault that we now have two employees with that name. This query (demo3 and also demo 2) are poorly formed and it is your job to not write this type of query.

How can you ensure that a query returns exactly one row? The subquery could use a filter on a primary key; the subquery could return a single table aggregate; you could use where rownum <= 1- but that might be poor logic.

2. Using a Scalar subquery

The previous examples show scalar subqueries being use in a Where clause predicate. We can use them with tests for equality, inequality, between etc.

This is a demo of using a subquery in a between test; I am calculating the average price of products of a specific category and the finding products in that category within 10% of that average price.

Demo 04: This uses a variable for the category.

```
 Set @catg := 'MUS';
```

' You can see that this returns a single value.

```
select avg(prod_list_price)
from a_prd.products
where catg_id = @catg;
+----------------------+
| avg(prod_list_price) |
+----------------------+
|            13.878182 |
+----------------------+
```

Demo 05: The main query uses a between test with a subquery for the lower and upper range values.

```
select prod_id, prod_name, prod_list_price
from a_prd.products
where catg_id = @catg
and prod_list_price BETWEEN
    (select 0.9 * avg(prod_list_price)
     from a_prd.products
     where catg_id = @catg)
AND
   (select 1.1 * avg(prod_list_price)
    from a_prd.products
    where catg_id = @catg);
```

```
+---------+------------+-----------------+
| prod_id | prod_name  | prod_list_price |
+---------+------------+-----------------+
|    2746 | B00000JWCM |           14.50 |
|    2747 | B000002I4Q |           14.50 |
+---------+------------+-----------------+
```

3. Putting a subquery in the select.

Demo 06: We can determine the average salary of all of the employees.

```
select avg(salary)
from a_emp.employees;
+--------------+
| avg(salary)  |
+--------------+
| 42975.590909 |
+--------------+
```

Since that is a scalar subquery, we could put it in the select clause.

```
select emp_id,  name_last,
salary,
(select avg(salary) from a_emp.employees)  as MedSalary
from a_emp.employees;
+--------+-----------+----------+--------------+
| emp_id | name_last | salary   | MedSalary    |
+--------+-----------+----------+--------------+
|    100 | King      | 24000.00 | 42975.590909 |
|    101 | Koch      | 98005.00 | 42975.590909 |
|    102 | D'Haa     | 30300.00 | 42975.590909 |
|    103 | Hunol     |  9000.00 | 42975.590909 |
|    104 | Ernst     | 50000.00 | 42975.590909 |
|    108 | Green     | 12000.00 | 42975.590909 |
|    109 | Fiet      | 15000.00 | 42975.590909 |
|    110 | Chen      | 30300.00 | 42975.590909 |
```

That is not too interesting since the subquery value is always the same. But we might want to know how far people salaries are from the average .

Demo 07: Using the scalar subquery in an calculation in the select

```
select emp_id,name_last,
salary,
round(salary -(select avg(salary) from a_emp.employees),0)  as "Over/Under"
from a_emp.employees;
+--------+-----------+----------+------------+
| emp_id | name_last | salary   | Over/Under |
+--------+-----------+----------+------------+
|    100 | King      | 24000.00 |     -18976 |
|    101 | Koch      | 98005.00 |      55029 |
|    102 | D'Haa     | 30300.00 |     -12676 |
|    103 | Hunol     |  9000.00 |     -33976 |
|    104 | Ernst     | 50000.00 |       7024 |
|    108 | Green     | 12000.00 |     -30976 |
|    109 | Fiet      | 15000.00 |     -27976 |
|    110 | Chen      | 30300.00 |     -12676 |
```

4. Putting a scalar subquery in an Order by clause

This is pushing the idea a bit but we can put a scalar subquery in an Order by clause- not common but it works.

We want to sort employees by how far away they are from the median salary- either above or below (that is what abs does for us. I filtered for dept 30 simply to reduce the output display.

Demo 08:

```
select dept_id, emp_id,
  name_last,
  salary,
  round(salary -(select avg(salary) from a_emp.employees),0)  as "Over/Under"
from a_emp.employees
where dept_id in (30)
order by abs(salary -(select avg(salary) from a_emp.employees) )
;
+---------+--------+-----------+----------+------------+
| dept_id | emp_id | name_last | salary   | Over/Under |
+---------+--------+-----------+----------+------------+
|      30 |    203 | Mays      | 44450.00 |       1474 |
|      30 |    110 | Chen      | 30300.00 |     -12676 |
|      30 |    109 | Fiet      | 15000.00 |     -27976 |
|      30 |    204 | King      | 15000.00 |     -27976 |
|      30 |    205 | Higgs     | 15000.00 |     -27976 |
|      30 |    108 | Green     | 12000.00 |     -30976 |
|      30 |    206 | Geitz     | 88954.00 |      45978 |
|      30 |    101 | Koch      | 98005.00 |      55029 |
+---------+--------+-----------+----------+------------+
```

We can also nest a scalar subquery in the select clause since the scalar subquery returns a single value. This gives us the ability to display simple and aggregated data on the same output line

Demo 09: This shows how this item's price compares with the average price for all items

```
select   prod_id, prod_name, catg_id
,        prod_list_price - (
           select Round(avg(prod_list_price), 2)
           from a_prd.products
           )
           as "Over/Under Avg Price"
from   a_prd.products
order by catg_id
;
Selected rows
+---------+---------------------+---------+----------------------+
| prod_id | prod_name           | catg_id | Over/Under Avg Price |
+---------+---------------------+---------+----------------------+
|    1120 | Washer              | APL     |               432.32 |
|    1130 | Mini Freezer        | APL     |                32.32 |
|    4569 | Mini Dryer          | APL     |               232.28 |
|    1125 | Dryer               | APL     |               382.33 |
|    1126 | WasherDryer         | APL     |               732.33 |
|    2412 | B00005UF3I          | MUS     |              -107.80 |
|    2746 | B00000JWCM          | MUS     |              -103.17 |
|    2747 | B000002I4Q          | MUS     |              -103.17 |
|    1152 | Cat pillow Leather  | PET     |               -62.39 |
|    4576 | Cosmo cat nip       | PET     |               -87.72 |
```

```
|    4568 | Deluxe Cat Bed      | PET     |               432.32 |
|    4567 | Deluxe Cat Tree     | PET     |               432.32 |
|    1141 | Bird cage- deluxe   | PET     |               -17.68 |
|    1142 | Bird seed           | PET     |              -115.17 |
|    4577 | Cat leash           | PET     |               -87.72 |
|    1140 | Bird cage- simple   | PET     |              -102.68 |
|    1151 | Cat pillow          | PET     |              -102.68 |
|    1150 | Cat exerciser       | PET     |              -112.68 |
|    1143 | Bird seed deluxe    | PET     |              -115.17 |
+---------+---------------------+---------+----------------------+
```

5. Correlated subquery

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 10:

```
select  cust_id,  ord_id
,  ( select sum(quantity_ordered)
     from   a_oe.order_details  OD
   ) as "NumItems"
from  a_oe.order_headers  OH
limit 5;
+---------+--------+----------+
| cust_id | ord_id | NumItems |
+---------+--------+----------+
|  400300 |    378 |      513 |
|  401250 |    106 |      513 |
|  401250 |    113 |      513 |
|  401250 |    119 |      513 |
|  401250 |    301 |      513 |
+---------+--------+----------+
```

This seems to say that every order has a total quantity of 513 items; that does not look correct. The scalar subquery calculated the total number of items purchased on all orders. 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.

Demo 11: Now our result set makes more sense and is more useful

```
select
   cust_id
,  ord_id
,  ( select sum(quantity_ordered)
     from   a_oe.order_details  OD
     where  OH.ord_id = OD.ord_id) as "NumItemsPerOrder"
from  a_oe.order_headers  OH
limit 10;
+---------+--------+------------------+
| cust_id | ord_id | NumItemsPerOrder |
+---------+--------+------------------+
|  400300 |    378 |               10 |
|  401250 |    106 |                1 |
|  401250 |    113 |                1 |
```

```
|  401250 |    119 |               10 |
|  401250 |    301 |                1 |
|  401250 |    506 |             NULL |
|  401890 |    112 |                2 |
|  401890 |    519 |                6 |
|  402100 |    114 |                5 |
|  402100 |    115 |               11 |
+---------+--------+------------------+
```

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 12: If you want to display a zero value instead of a blank for Orders with no items, use the Coalesce function. You need to do this with some of the aggregates- sum, avg.

```
select cust_id,ord_id
,      coalesce(
         ( select  SUM(quantity_ordered)
           from    a_oe.order_details
           where   a_oe.order_headers.ord_id=
                   a_oe.order_details.ord_id)
           , 0)
           as "NumitemsPerOrder"
from   a_oe.order_headers
order by NumItemsPerOrder;
```

Demo 13: Grouping with an outer join. It is not unusual to have more than one way to do a query. This is a group by query where it is safe to group by only the order id, since there is only one customer id per order id

```
select   cust_id, ord_id
,        SUM(quantity_ordered) AS NumItemsPerOrder
from     a_oe.order_headers  ORD
join     a_oe.order_details  ORDT using(ord_id)
group by cust_id, ord_id
order by NumItemsPerOrder;
```

Demo 14: We might want to get the number of items per order and the cost of the order.

```
select  cust_id, ord_id
,      ( select   SUM(quantity_ordered)
         from     a_oe.order_details
         where    a_oe.order_headers.ord_id =
                    a_oe.order_details.ord_id
       ) as "NumPerOrder"
,      ( select   SUM(quantity_ordered * quoted_price)
         from     a_oe.order_details
         where    a_oe.order_headers.ord_id =
                  a_oe.order_details.ord_id
       ) as "OrderCost"
from    a_oe.order_headers
limit 8;
```

cust_id	ord_id	NumPerOrder	OrderCost
400300	378	10	4500.00
401250	106	1	255.95
401250	113	1	22.50
401250	119	10	225.00
401250	301	1	205.00
401250	506	NULL	NULL
401890	112	2	99.98
401890	519	6	114.74