## Table of Contents

## 1. Scalar Subquery

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

A scalar subquery returns a single value (one row and one column) . This is still a table, but it is sometimes called a **scalar subquery expression** to emphasize that the result has a single value. The value of a scalar subquery is the value in the Select clause. That means the Select clause in the subquery can have only one expression.   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:

```
select emp_id
, name_last as "Employee"
from emp_employees
where dept_id =  (
    select dept_id
    from 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 with an equality test.

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. The predicate dept_id = null is never true.

If you are using subqueries in this way, 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 emp_employees
where dept_id = (
    select dept_id
    from emp_employees
    )
;
```

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

```
SQL Error: ORA-01427: single-row subquery returns more than one row
```

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

```
select emp_id, name_last as "Employee"
from emp_employees
where dept_id = (
    select dept_id
    from 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 query 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.

( As you develop more skills, you could create procedures that will handle the possibility of these types of error and do something better than simply fail- but that is for another class- CS 151P)

## 2. Using a Scalar subquery

The previous examples show scalar subqueries being used in a Where clause predicate. We can use scalar subqueries 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 then finding products in that category within 10% of that average price.

Demo 04: This uses a variable for the category.

```
variable catg varchar2(5);
exec :catg := 'MUS';
```

You can see that this returns a single value. (Actually a very nice repeating decimal! SQL Developer)

```
select avg(prod_list_price)
from prd_products
where catg_id = :catg;
```

```
AVG(PROD_LIST_PRICE)
----------------------
13.878181818181818181818181818181818182
```

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 prd_products
where catg_id = :catg
and prod_list_price between (
     select 0.9 * avg(prod_list_price)
     from prd_products
     where catg_id = :catg)
and (
    select 1.1 * avg(prod_list_price)
    from 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 median salary of all of the employees.

```
select median(salary)
from emp_employees;
```

```
MEDIAN(SALARY)
--------------
         65000
```

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

```
select emp_id, name_last,
salary
, (
   select median(salary)
   from emp_employees
   ) as MedSalary
from emp_employees;
```

```
    EMP_ID NAME_LAST                    SALARY  MEDSALARY
---------- ------------------------- ---------- ----------
       100 King                          100000      65000
       201 Harts                          15000      65000
       101 Koch                           98005      65000
       108 Green                          62000      65000
       205 Higgs                          75000      65000
       102 D'Haa                          60300      65000
       103 Hunol                          69000      65000
       104 Ernst                          65000      65000
       145 Russ                           59000      65000
       150 Tuck                           20000      65000
       155 Hiller                         29000      65000
. . . rows omitted
```

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

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

```
select emp_id, name_last
, salary
, salary -(
      select median(salary)
      from emp_employees
      )  as "Over/Under"
from emp_employees;
```

```
   EMP_ID NAME_LAST                    SALARY Over/Under
--------- ------------------------- ---------- ----------
      100 King                         100000      35000
      201 Harts                         15000     -50000
      101 Koch                          98005      33005
      108 Green                         62000      -3000
      205 Higgs                         75000      10000
      102 D'Haa                         60300      -4700
      103 Hunol                          69000       4000
      104 Ernst                          65000          0
      145 Russ                           59000      -6000
      150 Tuck                           20000     -45000
. . . rows omitted
```

Do not try to put a non-scalar subquery in a Select clause. The following would produce an error (ORA-01427: single-row subquery returns more than one row).

```
Select emp_id, (select salary from emp_employees) as AllSalaries
from emp_employees;
```

## 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 emp_id
, name_last
, salary
, salary -(select median(salary) from emp_employees)  as "Over/Under"
from emp_employees
where dept_id in (30)
order by abs(salary -(select median(salary) from emp_employees) )
;
```

```
   DEPT_ID     EMP_ID NAME_LAST                     SALARY Over/Under
---------- ---------- ------------------------- ---------- ----------
        30        109 Fiet                           65000          0
        30        203 Mays                           64450       -550
        30        108 Green                          62000      -3000
        30        110 Chen                           60300      -4700
        30        205 Higgs                          75000      10000
        30        206 Geitz                          88954      23954
        30        101 Koch                           98005      33005
        30        204 King                           99090      34090
```