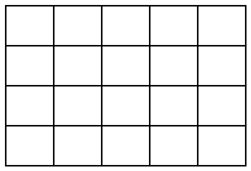
Table of Contents

Another shape we can have for a subquery is a multi-column, multi-row table



The subquery returns a virtual table that we can use in places where the query expects a table. We have seen this type of subquery in a CTE and in the Set operations. A Union join two subqueries.

The subquery could be used as a table expression in the From clause. In that case the subquery is sometimes called an in-line view. The subquery does not have to be the only table expression in the From clause; you can join the subquery result to a regular table to do a join.

1. Use a subquery in a From clause

When you embed a subquery in the From clause, it serves as a table expression.

- You should give the derived table an alias. Oracle does not always require a table alias but some other dbms do require this and Oracle allows the table alias.
- Each column needs a name; calculated columns need an alias
- The column names must be unique

Demo 01: This query is not very interesting but it shows using a subquery in the From clause; I am including a table alias- tbl.
I do not want to see you writing queries this simplistic. The subquery is providing no value.

```
select *
from (
     select *
     from emp_jobs
     ) tbl
;
```

This is also an inappropriate use of a subquery. Sometimes people get the idea that using subqueries makes your code more efficient- but that is not always the case.

```
select job_id, job_title, max_salary
from (
     select *
     from emp_jobs
     where max_salary is not null
     ) tbl
;
```

Demo 02: This is a query we did in unit 6 as a CTE because we could not use a column alias in the Select clause. We can also do this as an Inline view by taking the CTE subquery and using it as the From subquery.

```
With
ClNames as (
   select cl_state
   , cl_name_last || ' ' || cl_name_first as ClientName
   from vt_clients
)
select clientName || ' lives in ' || cl_state as "ClientInfo"
from ClNames
;
```

Using an inline view

```
select clientName || ' lives in ' || cl_state as "ClientInfo"
from (
   select cl_state
   , cl_name_last || ' ' || cl_name_first as ClientName
   from vt_clients
) ClNames
;
```

2. Counting in a subquery

You can use the COUNT (DISTINCT) feature to count the different numbers of shipping modes on all orders.

Demo 03: A standard COUNT DISTINCT- this does not count a null shipping_mode.

```
select COUNT (DISTINCT shipping_mode_id) as "num_diff_ship_modes"
from oe_orderHeaders;
```

```
num_diff_ship_modes
-------------------
                  6
```

Demo 04: If you wish to count nulls, you can use Coalesce to force a value to be counted.

```
select count (distinct coalesce (shipping_mode_id, 'none'))
    as "num_diff_ship_modes"
from oe_orderHeaders;
```

```
num_diff_ship_modes
-------------------
                  7
```

Demo 05: Another way to do this is to use a subquery in the FROM clause to return the distinct shipping methods- which does return a "null" group, and then use the parent query to count those rows. Oracle does not insist on a table alias; other dbms do require a table alias with a subquery in the From clause

```
select count (*) as "num_diff_ship_modes"
from (
   select distinct shipping_mode_id
   from oe_orderHeaders
   ) tbl;
```

```
num_diff_ship_modes
-------------------
                  7
```

Demo 06: Using a CTE (Give your CTE meaningful names)

```
With
ShipModes as (
   select distinct shipping_mode_id
   from oe_orderHeaders
   )
select count (*) "num_diff_ship_modes"
from ShipModes;
```

What if you want to count the number of distinct pairs of column values- for example, the number of pairs of shipping and order modes?

Demo 07: This approach yields an error :

```
select count (distinct shipping_mode_id, order_mode) as  "num_diff_modes"
from oe_orderHeaders;
ORA-00909: invalid number of arguments
```

Demo 08: You can do this with a subquery.

```
select count (*) as "num_diff_modes"
from (
   select distinct shipping_mode_id, order_mode
   from oe_orderHeaders) tbl;
```

```
num_diff_ship_modes
-------------------
                 13
```

3. Joining a regular table and a subquery

We want to display the number of employees in each department. We can start this as a simple query.

Demo 09: Since we have departments with no employees, we need an outer join. This is incorrect. Try to figure out the error before you go to the next demo.

```
select D.dept_id, count(*)as Row_Count
from emp_departments D
left join emp_employees E on D.dept_id = E.dept_id
group by D.dept_id;
```

```
   DEPT_ID   ROW_COUNT
----------- -----------
        10           1
        20           1
        30           8
        35           3
        80           3
        90           1
        95           1
       210           2
       215           4
```

Demo 10: The previous query used count(*) and counted the nulled-rows that the outer join generates. So every department row returned a value of at least 1. We want to count employees- not rows.

```
select D.dept_id, count(E.emp_id) as Emp_Count
from emp_departments D
left join emp_employees E on D.dept_id = E.dept_id
group by D.dept_id;
```

```
   DEPT_ID   EMP_COUNT
----------- -----------
        10           1
        20          1
        30          8
        35          3
        80          3
        90          0
        95          0
       210          2
       215          4
```

Demo 11: If we want to see the department names, we have to add dept_name to the group by clause. That is not a big deal since it is only one extra attribute.

```
select D.dept_id, D.dept_name, count(E.emp_id) as Emp_Count
from emp_departments D
left join emp_employees E on D.dept_id = e.dept_id
group by D.dept_id, D.dept_name
order by D.dept_id;
```

```
    DEPT_ID DEPT_NAME             EMP_COUNT
----------- ------------------- -----------
         10 Administration                1
         20 Marketing                     1
         30 Development                   8
         35 Cloud Computing               3
         80 Sales                         3
         90 Shipping                      0
         95 Logistics                     0
        210 IT Support                    2
        215 IT Support                    4
```

Demo 12: Alternately we could write a query that counts employees grouping by the department ID. This does not give us departments with no employees.

```
select dept_id, count(*) as EmpCount
from emp_employees E
group by dept_id;
```

```
   DEPT_ID   EMPCOUNT
---------- ----------
        30          8
        20          1
       210          2
       215          4
        35          3
        80          3
        10          1
```

Demo 13: We could use that query and do an outer join to the department table. The subquery needs to supply an alias for the calculated (count) column because we want to refer to it.

Note that the subquery is enclosed in parentheses and given a table alias. Then the join is done as we usually do joins: `on D.dept_id = EC.dept_id`
We can use the subquery to contain the details of the aggregation.

```
select D.dept_id, D.dept_name, EC.EmpCount
from emp_departments    D
left join (
   select dept_id, count(*) as EmpCount
   from emp_employees
   group by dept_id) EC  on D.dept_id = EC.dept_id   ;
   DEPT_ID DEPT_NAME              EMPCOUNT
---------- -------------------- ----------
        10 Administration                1
        20 Marketing                     1
        30 Development                   8
        35 Cloud Computing               3
        80 Sales                         3
        90 Shipping
        95 Logistics
       210 IT Support                    2
       215 IT Support                    4
```

Demo 14: You could also write this as a CTE. In general you have a choice between using a CTE or an in-line view. If you need to use the subquery expression more than once in the same query- use the CTE. You can also take a CTE query and rewrite it using an in-line view.

```
With deptEmpCount as (
   select dept_id, count(*) as EmpCount
   from emp_employees
   group by dept_id
   )
select D.dept_id, D.dept_name, EC.EmpCount
from emp_departments D
left join  deptEmpCount  EC  on D.dept_id = EC.dept_id;
```

This demo does seem like a bit of over kill to avoid writing group by d.dept_id, d.dept_name. The following query wants to display three extra fields along with the aggregates. And you might just feel that it makes more sense to aggregate on the dept id only.

Demo 15:

```
select D.dept_id, D.dept_name
, l.loc_city, l.loc_state_province
, count(emp_id) as EmpCount
from emp_departments D
join emp_locations L on D.loc_id = l.loc_id
left join emp_employees E on d.dept_id = e.dept_id
group by D.dept_id, D.dept_name, l.loc_city, l.loc_state_province;
```

3.1. Joining subqueries

Demo 16: Using an outer join to a subquery- used as a table expression. And it has one grouping key

```
select D.dept_id, D.dept_name
, l.loc_city, l.loc_state_province
,  coalesce(EmpCount,0) as EmpCount
from emp_departments D
join emp_locations L on D.loc_id = l.loc_id
left join (
   select dept_id, count(emp_id) as EmpCount
   from emp_employees E
   group by dept_id) EC on D.dept_id = EC.dept_id;
```

This uses two subqueries as data sources.

Demo 17: How many employees do we have in each department and what percent is that of all employees? Use each of these as a virtual table in the FROM clause to get the percent of each department over the entire employee table. Notice that we do not need a joining clause since the overall count has only one return row.

```
select
  dept_id
, dept_count
, round((dept_count / Count_All),2) AS Percnt
from
    (select dept_id, count(1) AS dept_count  /* get count by department */
     from emp_employees
     group by dept_id)  vt1,   /* get total count for all employees */
    (select count(*) AS Count_All
     from emp_employees)   vt2
order by Dept_id;
```

```
   DEPT_ID DEPT_COUNT     PERCNT
---------- ---------- ----------
        10          1        .05
        20          1        .05
        30          8        .36
        35          3        .14
        80          3        .14
       210          2        .09
       215          4        .18
```

Demo 18: To get a percent value, multiply by 100 .You can also do some formatting to get a value that looks more like a percent.

```
select
  dept_id
, dept_count
, round((1.00 * dept_count / Count_All),2) AS Percnt
, to_char ((round((dept_count / Count_All),2)  * 100), 99.9) || '%' AS Percnt
from
    (select dept_id, count(1) AS dept_count
     from emp_employees
     group by dept_id)  vt1,
    (select count(*) AS Count_All
     from emp_employees)   vt2
order by Dept_id
;
```

```
   DEPT_ID DEPT_COUNT     PERCNT PERCNT
---------- ---------- ---------- ------
        10          1        .05   5.0%
        20          1        .05   5.0%
        30          8        .36  36.0%
        35          3        .14  14.0%
        80          3        .14  14.0%
       210          2        .09   9.0%
       215          4        .18  18.0%
```

Demo 19: Using a CTE

```
With DeptCount as
    (select dept_id, count(1) AS dept_count
     from emp_employees
     group by dept_id)
,
AllCount as
   (select count(*) AS Count_All
    from emp_employees)
select
   dept_id
,  dept_count
,  to_char ((Round((dept_count / Count_All),2)  * 100), 99.9) || '%' AS Percnt
from DeptCount
cross join AllCount
order by Dept_id
;
```

Demo 20: One way to find customers who bought both an appliance and a houseware item.
(oe_customer_orders is a view you should have created earlier)
The first subquery picks up the appliances
The second subquery picks up the houseware items
The join of the two subqueries checks that we are looking at the same customer id

```
select distinct tblapl.custid
from ( select CustID
       from oe_customer_orders
       where category ='APL' )  tblapl
join ( select CustID
       from oe_customer_orders
       where Category ='HW' )  tblhw
on tblapl.custid = tblhw.custid;
```

4. Generating data in the subquery

Demo 21: We decide we want to display descriptive literals for various salary ranges. We could do this with a case expression. But we can also generate the rating data with a union

```
     select 'under paid' as catg, 0 as low, 49999.99 as high from dual
  Union all
     select 'medium paid' as catg, 50000.00 as low, 79999.99 as high from dual
  Union all
     select 'over paid' as catg, 80000.00 as low, 9999999.99 as high from dual;
```

```
CATG                 LOW           HIGH
----------- ------------ --------------
under paid             0       49999.99
medium paid        50000       79999.99
over paid          80000     9999999.99
```

Demo 22: Now join that union to the employees table

```
select emp_id, name_last,salary, catg
from emp_employees E
join (
     select 'under paid' as catg, 0 as low, 49999.99 as high from dual
   Union all
     select 'medium paid' as catg, 50000.00 as low, 79999.99 as high from dual
   Union all
     select 'over paid' as catg, 80000.00 as low, 9999999.99 as high from dual
  ) Ratings on E.salary between Ratings.low and Ratings.high
order by salary;
```

Selected rows

```
    EMP_ID NAME_LAST                     SALARY CATG
---------- ------------------------- ---------- -----------
       201 Harts                          15000 under paid
       150 Tuck                           20000 under paid
       155 Hiller                         29000 under paid
       207 Russ                           30000 under paid
       145 Russ                           59000 medium paid
       110 Chen                           60300 medium paid
       205 Higgs                          75000 medium paid
       146 Partne                         88954 over paid
       206 Geitz                          88954 over paid
       162 Holme                          98000 over paid
```

Demo 23: Again- you can use a CTE instead of a subquery

```
With ratings as (
  select 'under paid' as catg, 0 as low , 49999.99 as high from dual
  Union all
  select 'medium paid' , 50000.00 , 79999.99 from dual
  Union all
  select 'over paid' , 80000.00 , 9999999.99 from dual  )
select emp_id, name_last,salary, catg
from emp_employees E
join ratings R on E.salary between R.low and R.high
order by salary;
```

Demo 24: You could do this with a case

```
select emp_id, name_last,salary
,  case
     when salary between 0 and 49999.99 then 'under paid'
     when salary between 50000.00 and 79999.99 then 'medium paid'
     when salary between 80000.00 and 9999999.99 then 'over paid'
   end as catg
from emp_employees E
order by salary;
```

Demo 25: we may want just the aggregates

```
select catg, count(*)as NumEmployees
from emp_employees E
join (
  select 'under paid' as catg, 0 as low, 49999.99 as high from dual
  Union all
  select 'medium paid' as catg, 50000.00 as low, 79999.99 as high from dual
  Union all
  select 'over paid' as catg, 80000.00 as low, 9999999.99 as high from dual
  ) Ratings on E.salary between Ratings.low and Ratings.high
group by catg;
```

```
CATG        NUMEMPLOYEES
----------- ------------
under paid             4
medium paid           11
over paid              7
```

The emp_employees table has the attribute salary defined as nullable. Which of the above queries will report back employees with no salary value? How would we get those employees into the reports?