Table of Contents

1. Using a single subquery 1
2. Using multiple subqueries 2
2.1. Nesting subqueries 4

A table expression determines a virtual table. We commonly see table expressions in the From clause of a query. We started with the simplest table expression- a single base table. (Remember a base table is a persistent table; we create the base table with a Create Table statement.).

Then we added Inner Joins and Outer Joins to connect two or more base tables to create a virtual table. That join is a table expression.

```
select an_name, cl_name_last
from vt_animals  an
join vt_clients cl  on an.cl_id = cl.cl_id;
```

Now we are going to discuss a technique that uses a subquery as a table expression. This is sometimes called an inline view.

The use of CTE to compartmentalize a complex query is elegant and fairly easy to read and write with a little practice. The use of a CTE is similar to the use of an inline view with the added advantage that you can use the CTE more than once in the query and it may seem cleaner to define the CTE at the top of the query rather than in the middle of the From clause.

1. Using a single subquery

Suppose you have a fairly complex query dealing with customer orders that you need to run only for a particular query. You would like to break the query down into smaller, more manageable chunks that you could test separately. One solution is to create a subquery that handles part of the query and then use that query in the From clause of the main query. Since we are using this subquery as a table expression, the subquery can have multiple columns and multiple rows. We also will provide a table alias to have a name for the subquery table expression.

This query does not work since you cannot use the column alias as a column name in the same Select clause,

```
select cl_name_last || ' '|| cl_name_first as ClientName
, ClientName || ' lives in ' || cl_state
from vt_clients;
```

Demo 01: Using a subquery in the From clause. The subquery exposes the alias ClientName which we can then use in the Where clause of the main query. The subquery table alias is ClientNames

```
select ClientName || ' lives in ' ||  cl_state as "Message"
from (
   select cl_name_first || ' ' ||  cl_name_last as ClientName
   , cl_state
   from vt_clients
) ClientNames;
```

```
Message
--------------------------------
Stanley Turrentine lives in CA
Wes Montgomery lives in OH
Theo Monk lives in NY
Coleman Hawkins lives in OH
.. .
Edger Boston lives in MA
Sue Biederbecke lives in IL
Sam Biederbecke lives in CA
 Biederbecke lives in IL
 Biederbecke lives in CA
```

The subquery is shown here. It is a Select that exposes the cl_state and an expression named ClientName

```
   select cl_name_first || ' ' ||  cl_name_last as ClientName
   , cl_state
   from vt_clients
```

The subquery is enclosed in parentheses, given a table alias, and placed in the From clause of the main query. The main query can use the exposed columns from the subquery. That allows us to use the calculated column by referencing its alias.

We could also focus on dealing with that space that shows up with a null first name in the subquery.

Demo 02: This is a more complex subquery that assembles the data for the orders and exposes three columns which are used in the main query.

```
select order_id
, order_date
, itemTotal
from (
   select
      order_id
   ,  order_date
   ,  customer_id
   ,  quoted_price * quantity_ordered as itemTotal
   from oe_orderHeaders
   join oe_orderDetails using(order_id)
   join prd_products using(prod_id)
   where quoted_price > 0 and quantity_ordered > 0
   Order by order_id
  )rpt_base
where order_date < '01-NOV-2015'
;
```

```
  ORDER_ID ORDER_DATE  ITEMTOTAL
---------- ---------- ----------
       105 01-OCT-15       155.4
       105 01-OCT-15         300
       105 01-OCT-15         750
       106 01-OCT-15      255.95
       107 02-OCT-15       49.99
       108 02-OCT-15        22.5
       109 12-OCT-15      149.99
       110 12-OCT-15      149.99
       110 12-OCT-15      149.99
       301 04-JUN-15         205
       302 04-JUN-15         120
. . .
```

2. Using multiple subqueries

Demo 03: This uses two subqueries and joins them. Each subquery has a name. The subqueries produce virtual tables and we are just joining the two virtual tables.

```
select
  customer_id
, customer_name
```

```
, prod_id
, ext_price
from (
  select
    customer_id
  , customer_name_first || ' ' || customer_name_last as customer_name
  from cust_customers
  where lower(customer_name_first) = 'william'
) t_cust
join (
  select
    order_id
  , order_date
  , customer_id
  , prod_id
  , quoted_price * quantity_ordered as ext_price
  from oe orderHeaders
  join oe_orderDetails using (order_id)
) t_ord using (customer_id)
;
```

CUSTOMER_ID	CUSTOMER_NAME	PROD_ID	EXT_PRICE
404950	William Morris	1090	149.99
404950	William Morris	1130	149.99
401890	William Northrep	1110	99.98
402100	William Morise	1130	625
402100	William Morise	1000	200
402100	William Morise	1120	1900
402100	William Morise	1080	25
402100	William Morise	1100	180
402100	William Morise	1150	19.96
402100	William Morise	1141	300

The From clause here is

```
From (subQuery1) t_cust
Join  (subQuery2) t_ord using (cust_id)
```

The demo has the Using (cust_id) syntax for the join. We could also code the join with the syntax

```
Select . . .
From (subQuery1)  t_cust
Join   (subQuery2)  t_ord on t_cust.cust_id = t_ord.cust_id;
```

Since we are joining the two virtual tables on the cust_id values, each subquery needs to expose that column. The first subquery contributes the cust_name and the second subquery contributes the prod_id and the ext_price.

Demo 04: Joining a subquery table expression to a base table

```
select customer_id
, customer_name_last
, prod_id
, ext_price
from cust_customers
join (select customer_id
       , prod_id
       , quoted_price * quantity_ordered as ext_price
       from oe_orderHeaders OH
       join oe_orderDetails OD on OH.order_id = OD.order_id
       )  t_ord using (customer_id)
;
```

2.1. Nesting subqueries

Demo 05: This nests two subqueries in the From clause. As it stands it is simply a complex way to get customers with the first name William, but it does show nested subqueries

```
select customer_name
from (
  select customer_name_first || ' ' || customer_name_last as customer_name
  from (
    select
      customer_id
    , customer_name_first
    , customer_name_last
    from cust_customers
    where lower(customer_name_first) = 'william'
  ) tbl_william
) tbl_concat
;
```

```
CUSTOMER_NAME
--------------------
William Northrep
William Morise
William Morris
William Morris
William Max
```

One difference we will see later where the CTE is even more of an advantage is that we can define a CTE once in the query and then use it multiple times in the query. If we need to do that logic with a subquery, we would have to repeat that subquery multiple times in the query.