Table of Contents

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.)

We use Inner 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;
```

In this unit we added other table expressions using outer joins to create a virtual table. Those joins also define table expressions.

Now we are going to discuss a technique called a Common Table Expression which is available in Oracle and in SQL Server- but not yet in MySQL. This discussion is limited to the use of non-recursive CTEs.

1. Common table expression

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- perhaps the joining of the tables and give that subquery a name and then use that name in the From clause of the rest of the query

The Oracle Select statement has a clause called a With clause that lets us do that. First, a very simplistic example. If we want to see the ID and name of all of our customers with a first name of William, we would probably run the SQL statement shown here.

Demo 01: simple select

```
select
  customer_id
, customer_name_first || ' ' || customer_name_last As cust_name
from cust_customers
where customer_name_first = 'William';
```

```
CUSTOMER_ID CUST_NAME
----------- --------------------------------------------------
     401890 William Northrep
     402100 William Morise
     404950 William Morris
     409010 William Morris
     409020 William Max
```

Demo 02: We could rewrite this using a CTE.

```
With custnames as (
   select customer_id
   , customer_name_first || ' ' || customer_name_last as cust_name
   from cust_customers
   where customer_name_first = 'William'
)
select customer_id, cust_name
from custnames
;
```

We get the same result. At first it just looks like we made the select statement longer to no purpose. You should not use a CTE with a query this simple. But the demos always start simple.

What is happening is that the With clause defines a name (here **custnames**) and a subquery which can then be used by name in the From clause of the main Select. Oracle sometimes refers to this technique as a subquery-factoring clause; more often it is called a common table expression.

The select statement defined within the CTE is a subquery. It is enclosed in parentheses. It has a name.

The CTE does not exist after the query has finished executing.

Demo 03: We can use a CTE to encapsulate a Select and the column alias defined in the CTE can be used in the main query. We cannot define and use a column alias in the same Select/Where.

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

Demo 04: Using a more complex query. We can put the complexity of the join into the CTE and the main query body is much simpler.
Display the order date and line amount due for each detail line

```
With rpt_base as (
    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
    )
select
  order_id
, order_date
, itemTotal
from rpt_base
where order_date < '01-AUG-2015'
order by order_date;
```

```
  ORDER_ID ORDER_DATE  ITEMTOTAL
---------- ---------- ----------
       522 05-APR-15          45
       540 02-JUN-15       49.99
       540 02-JUN-15          45
       540 02-JUN-15       55.25
       307 04-JUN-15        2250
       307 04-JUN-15        2250
       306 04-JUN-15         500
       306 04-JUN-15         500
       302 04-JUN-15      349.95
. . .  rows omitted
```

Some people prefer this style of working out a query; they encapsulate part of the work in the CTE and then have a simpler query in the main Select.

2. Using multiple CTEs

You can have only one With clause in an SQL statement; but you can define more than one subquery. Give each one a name and separate them with commas.

Demo 05: Using a With clause to define two subqueries with calculated columns and column aliases

```
with  t_cust as
      ( select customer_id
       , customer_name_first || ' ' || customer_name_last as cust_name
       from cust_customers
       where customer_name_first = 'William'
)
,   t_ord  as
      (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) )
select
  customer_id
, cust_name
, prod_id
, ext_price
from t_cust
join t_ord Using (customer_id)
order By customer_id, prod_id;
```

```
CUSTOMER_ID CUST_NAME                                              PROD_ID  EXT_PRICE
----------- -------------------------------------------------- ---------- ----------
     401890 William Northrep                                         1020      64.75
     401890 William Northrep                                         1110      49.99
     401890 William Northrep                                         1110      99.98
     402100 William Morise                                           1000        200
     402100 William Morise                                           1030         27
     402100 William Morise                                           1080         25
     402100 William Morise                                           1100        180
     402100 William Morise                                           1120       1900
     402100 William Morise                                           1130        625

. . .  rows omitted.
```

In this demo I joined the two CTE using the regular syntax we use for joins. I could also join a CTE to a base table. The fact that a CTE has a name makes it easier to use when deriving more complex table expressions. The expression you use in the From clause to set up the result table can use base tables or more complex table expressions.

Demo 06: Joining the common table expression to a base table

```
With t_ord  as
      ( 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) )
```

```
select
  customer_id
, customer_name_last
, prod_id
, ext_price
from cust_customers
join t_ord Using (customer_id);
```

```
CUSTOMER_ID CUSTOMER_NAME_LAST           PROD_ID  EXT_PRICE
----------- ------------------------ ---------- ----------
     403000 Williams                       1030        300
     403000 Williams                       1020      155.4
     403000 Williams                       1010        750
     401250 Morse                          1060     255.95
     403050 Hamilton                       1110      49.99
     403000 Williams                       1080       22.5
     403000 Williams                       1130     149.99
     404950 Morris                         1090     149.99
     404950 Morris                         1130     149.99
     403000 Williams                       1150      249.5
. . .  rows omitted
```

For these types of queries it is a matter of preference where you write this as a simple multi-table join or as a series of CTEs. Often, people who have done more programming like the CTE style of writing queries.

In these examples, we are using the CTE as a way to move the complexity of a subquery away from the main query providing visual separation.

2.1. CTE order of definition

It is legal for one CTE to refer to another CTE as long at the expressions are defined in the proper order. This is a trivial example of such a CTE. The first expression filters for the customer name and the second expression uses the first to concatenate the name components. As we write more complex queries, you may prefer this style as a step-by-step approach to solving a problem. You might then decide to combine the CTE clauses into a single expression.

Demo 07: Using a CTE that refers to another CTE

```
With  cte1  as (
  select customer_id, customer_name_first, customer_name_last
  from cust_customers
  where customer_name_first = 'William'
)
, cte2 as (
  select customer_name_first || ' ' || customer_name_last as cust_name
  from cte1)
select cust_name
from cte2;
```

```
CUST_NAME
--------------------------------------------------
William Northrep
William Morise
William Morris
William Morris
William Max
```

What we are seeing here is more of the trend to modularize your code, even within a single SQL statement.

We will use CTE in some of the demos for the rest of the semester including using a CTE to create a subquery that can be referred to multiple times in the main query.