The wonders of CTE's

Introduction to
Common Table Expressions

What is The “Common Table Expression”

The CTE can be seen as a collection of temp tables with data that you need for an overall larger query.

A CTE can also include Delete,Insert and Update operators opposed to only the Select statement.

It’s a more readable alternative to nested functions.

And a bonus to using a CTE is that it makes use of the query planer to create an optimal execution plan.

Well get back to Recursive CTE’s

Basic Syntax

The Basic Syntax is composed of 4 Elements:

- WITH : This shows the initialization of a CTE
- <alias> AS : the alias of the Expresion
- (expression): the actual expression that will be used in the final query
- The call statement

```
WITH
 eg_ AS
  (
    SELECT my_column FROM my_table
  )
SELECT * FROM eg_
```

Basic Example

```
WITH orders_ AS (
 SELECT * from
  orders
 LEFT JOIN
  order_details
 USING(orderid)
)

SELECT * from orders_
```

Comparing a CTE with a Nested Query

Return the items that sold more than the average of all sales

CTE

```
WITH orders_ AS (
 SELECT * from
  Orders
 LEFT JOIN
  Order_details
 USING(orderid)
)
,top_sellers AS (
 SELECT
  Productid
 FROM orders_
 GROUP BY productid
 HAVING
  SUM(unitprice*quantity) >=
   (select SUM(unitprice*quantity)/count(*) from orders_)
)
SELECT * FROM products
WHERE productid in (SELECT productid FROM top_sellers  )
```

Nested Query

```
select * from products
 where productid in
 (
  select productid
  from
   Orders
    left join
   Order_details
  using(orderid)
  group by productid
  having
   SUM(unitprice*quantity) >=
   (
    select SUM(unitprice*quantity)/count(*) from
     Orders
    left join
     Order_details
    using(orderid)
   )
  order by SUM(unitprice*quantity)/count(*)
 )
```

Look familiar

Look familiar

Selecting from a CTE

```
WITH suppliers_ AS(
 select * from suppliers
)
,products_ AS(
 select * from  products
)
,order_details_ AS(
 select * from  order_details
)
,orders_ AS(
 select * from orders
)
,the_join AS (
select * from
 orders_ o
 left join
 order_details_ od using(orderid)
 left join
 products_ p using(productid)
 left join
 suppliers_ s using(supplierid)
)
select * from the_join
```

```
WITH  suppliers_ AS (
 select * from suppliers
)
,products_ AS(
 select * from  products
)
,order_details_ AS(
 select * from  order_details
)
,orders_ AS(
 select * from orders
)
,the_join AS (
select * from
 orders_ o
 left join
 order_details_ od using(orderid)
 left join
 products_ p using(productid)
 left join
 suppliers_ s using(supplierid)
)
select * from orders_
```

Using an Update in a CTE

```
WITH orders_ AS (
 SELECT * from
  Orders
 LEFT JOIN
  Order_details
 USING(orderid)
)
,top_sellers AS(
 SELECT
  Productid
 FROM orders_
 GROUP BY productid
 HAVING
  SUM(unitprice*quantity) >=
   (select SUM(unitprice*quantity)/count(*) from orders_)
   order by   SUM(unitprice*quantity) limit 10
 )
,Update_top_sellers AS (
 update products
  set top_seller = TRUE
  WHERE productid in (SELECT productid FROM top_sellers  )
)
,Update_rest AS (
 update products
  set top_seller = FALSE
  WHERE productid not in (SELECT productid FROM top_sellers  )
)
Select * from products where top_seller = True
```

Using a Delete in a CTE

```
WITH moved_rows AS (
    DELETE FROM dummy_orders
    WHERE orderdate between '1996-07-01' AND '1996-08-01'
    RETURNING *
)
INSERT INTO products_log
SELECT * FROM moved_rows;
```

Using a Insert in a CTE

```
WITH moved_rows AS (
    DELETE FROM dummy_orders
    WHERE orderdate between '1996-07-01' AND '1996-08-01'
    RETURNING *
)
,Insert_rows AS(
 INSERT INTO products_log
 SELECT * FROM moved_rows
)
select * from products_log;
```

Recursive CTE’s Basic Syntax

```
WITH RECURSIVE my_recursive_cte(a_counter) AS (
 values(1)
  union all
 select a_counter+1 from my_recursive_cte
)
select * from my_recursive_cte WHERE a_counter< 5
```

Quick practical example of a recursive CTE

```
WITH  RECURSIVE fake_data (orderid, productid, unitprice, quantity, discount, total_) AS (
  SELECT
    (SELECT max(orderid)::integer FROM dummy_order_details) AS orderid
    , 0  as productid     , 0.00::real as unitprice     , 0  as quantity     , 0 as discount     , 0.00::real as total_
  UNION ALL
  SELECT
    fake_data.orderid     , products.productid         , products.unitprice     , fake_data.quantity     , fake_data.discount::integer
    , fake_data.total_ + (products.unitprice * fake_data.quantity)::real AS total_
  FROM
    (
      SELECT
        orderid + 1 AS orderid           ,round(random() * 77)::integer + 1 AS productid           , unitprice           ,(random() * 10)::integer + 1 AS quantity           , discount           , total_
      FROM fake_data
    ) fake_data
      LEFT JOIN products
              USING (productid)
  WHERE    total_ < 20000
)
SELECT * FROM fake_data where total_ <> 0
```

Thank You

I will now be taking Questions

- https://momjian.us/main/presentations/sql.html
- https://www.postgresql.org/docs/11/queries-with.html
- https://www.youtube.com/watch?v=VY5wdA8HIv0