Subquery v/s CTE(with as)

In SQL, both **subqueries** and **Common Table Expressions (CTEs)** (using the `WITH` clause) allow you to define temporary result sets that can be referenced within a query. However, they have some differences in terms of syntax, readability, and usability. Here's a comparison between the two:

Aspect	Subquery	CTE (WITH AS)
Syntax	Nested inside another SQL statement.	Defined using the `WITH` clause before the main query.
Readability	Can be less readable, especially with multiple levels of nesting.	Generally more readable and organized for complex queries.
Reusability	Cannot be referenced multiple times within the same query.	Can be referenced multiple times in the main query.
Performance	May lead to performance issues if used extensively, as the subquery is executed for each row.	Generally optimized by the SQL engine; may improve performance when used with large datasets.
Scope	Limited to the statement in which it is defined.	Visible to the entire query that follows the `WITH` clause.
Use Cases	Useful for simple queries where temporary results are needed in one place.	Useful for complex queries, especially when multiple references to the same result set are needed.

Scalar subqueries

- A **scalar subquery** in SQL is a subquery that returns a single value (i.e., a scalar).
- **Scalar Subquery** are used when you need to compute a single value that can be used anywhere in the query (in the `SELECT`, `WHERE`, `HAVING`, or `FROM` clauses). It is particularly useful when you need to **compare the current row with an aggregate value across the entire dataset** (e.g., finding records greater than the average).
    - We cannot use `where,having etc.` clause with `windows function`

Example: We want to find those list price which are greater than avg(listprice)

- lets try this with both windows function and scalar sunquery
- Original dataset Image `without the ListPrice > Avg(ListPrice) condition`

Results | Messages

	ProductID	Name	StandardCost	ListPrice	Avg_over()	Avg_subquery	Avg_ListDiff_over()	Avg_ListDiff_Query
207	534	Top Tube	0.00	0.00	438.6662	438.6662	-438.6662	-438.6662
208	535	Tension Pulley	0.00	0.00	438.6662	438.6662	-438.6662	-438.6662
209	679	Rear Derailleur Cage	0.00	0.00	438.6662	438.6662	-438.6662	-438.6662
210	680	HL Road Frame - Bl...	1059.31	1431.50	438.6662	438.6662	992.8338	992.8338
211	706	HL Road Frame - Re...	1059.31	1431.50	438.6662	438.6662	992.8338	992.8338
212	707	Sport-100 Helmet, R...	13.0863	34.99	438.6662	438.6662	-403.6762	-403.6762
213	708	Sport-100 Helmet, Bl...	13.0863	34.99	438.6662	438.6662	-403.6762	-403.6762
214	709	Mountain Bike Sock...	3.3963	9.50	438.6662	438.6662	-429.1662	-429.1662
215	710	Mountain Bike Sock...	3.3963	9.50	438.6662	438.6662	-429.1662	-429.1662
216	711	Sport-100 Helmet, Bl...	13.0863	34.99	438.6662	438.6662	-403.6762	-403.6762
217	712	AWC Logo Cap	6.9223	8.99	438.6662	438.6662	-429.6762	-429.6762
218	713	Long-Sleeve Logo J...	38.4923	49.99	438.6662	438.6662	-388.6762	-388.6762
219	714	Long-Sleeve Logo J...	38.4923	49.99	438.6662	438.6662	-388.6762	-388.6762
220	715	Long-Sleeve Logo J...	38.4923	49.99	438.6662	438.6662	-388.6762	-388.6762
221	716	Long-Sleeve Logo J...	38.4923	49.99	438.6662	438.6662	-388.6762	-388.6762
222	717	HL Road Frame - Re...	868.6342	1431.50	438.6662	438.6662	992.8338	992.8338

```sql
select
    ProductID,
    pp.Name,
    StandardCost,
    ListPrice,
    avg(ListPrice) over() as 'Avg_over()',
    (select avg(ListPrice) from Production.Product) as 'Avg_subquery',

    ListPrice-avg(ListPrice) over() as 'Avg_ListDiff_over()',
    ListPrice-(select avg(ListPrice) from Production.Product) as 'Avg_ListDiff_Query'
from Production.Product pp
where ListPrice > (select avg(ListPrice) from Production.Product)
```

Results | Messages

	ProductID	Name	StandardCost	ListPrice	Avg_over()	Avg_subquery	Avg_ListDiff_over()	Avg_ListDiff_Query
1	680	HL Road Frame - Black, 58	1059.31	1431.50	1450.074	438.6662	-18.574	992.8338
2	706	HL Road Frame - Red, 58	1059.31	1431.50	1450.074	438.6662	-18.574	992.8338
3	717	HL Road Frame - Red, 62	868.6342	1431.50	1450.074	438.6662	-18.574	992.8338
4	718	HL Road Frame - Red, 44	868.6342	1431.50	1450.074	438.6662	-18.574	992.8338
5	719	HL Road Frame - Red, 48	868.6342	1431.50	1450.074	438.6662	-18.574	992.8338
6	720	HL Road Frame - Red, 52	868.6342	1431.50	1450.074	438.6662	-18.574	992.8338
7	721	HL Road Frame - Red, 56	868.6342	1431.50	1450.074	438.6662	-18.574	992.8338
8	731	ML Road Frame - Red, 44	352.1394	594.83	1450.074	438.6662	-855.244	156.1638
9	732	ML Road Frame - Red, 48	352.1394	594.83	1450.074	438.6662	-855.244	156.1638
10	733	ML Road Frame - Red, 52	352.1394	594.83	1450.074	438.6662	-855.244	156.1638
11	734	ML Road Frame - Red, 58	352.1394	594.83	1450.074	438.6662	-855.244	156.1638
12	735	ML Road Frame - Red, 60	352.1394	594.83	1450.074	438.6662	-855.244	156.1638
13	739	HL Mountain Frame - Silver, 42	747.2002	1364.50	1450.074	438.6662	-85.574	925.8338
14	740	HL Mountain Frame - Silver, 44	706.811	1364.50	1450.074	438.6662	-85.574	925.8338
15	741	HL Mountain Frame - Silver, 48	706.811	1364.50	1450.074	438.6662	-85.574	925.8338
16	742	HL Mountain Frame - Silver, 46	747.2002	1364.50	1450.074	438.6662	-85.574	925.8338

- As we can see the Windows function is givng wrong output while the scalar subquery are correct

Correlated Subqueries

Correlated Subqueries are subqueries that depend on the outer query for their values. Unlike regular subqueries (which are independent and executed once), correlated subqueries execute once for every row in the outer query. Each time a row in the outer query is processed, the correlated subquery is reevaluated with reference to that specific row.

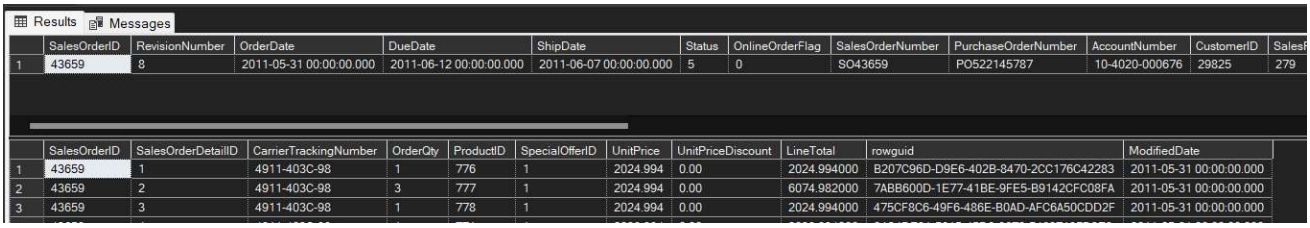
- They are run once for each record in outer query

```
select
    PurchaseOrderID,
    VendorID,
    OrderDate,
    TotalDue,
    (
    select count(*)
    from Purchasing.PurchaseOrderDetail ppod
    where ppod.PurchaseOrderID = ppoh.PurchaseOrderId
    and RejectedQty = 0
    )
    as NonRejectedItems
from Purchasing.PurchaseOrderHeader ppoh
```

Exists()

In SQL, the `EXISTS` keyword is used to test for the existence of any records in a subquery. It returns `TRUE` if the subquery returns one or more records and `FALSE` if the subquery returns no records. This is typically used in `WHERE` clauses to filter rows based on whether certain conditions are met in a related table or query.

- `Exist()` is suitable for one to many relation table



Results Messages

	SalesOrderID	RevisionNumber	OrderDate	DueDate	ShipDate	Status	OnlineOrderFlag	SalesOrderNumber	PurchaseOrderNumber	AccountNumber	CustomerID	SalesP
1	43659	8	2011-05-31 00:00:00.000	2011-06-12 00:00:00.000	2011-06-07 00:00:00.000	5	0	SO43659	PO522145787	10-4020-000676	29825	279

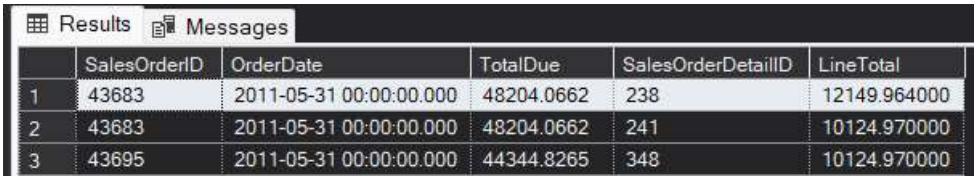
	SalesOrderID	SalesOrderDetailID	CarrierTrackingNumber	OrderQty	ProductID	SpecialOfferID	UnitPrice	UnitPriceDiscount	LineTotal	rowguid	ModifiedDate
1	43659	1	4911-403C-98	1	776	1	2024.994	0.00	2024.994000	B207C96D-D9E6-402B-8470-2CC176C42283	2011-05-31 00:00:00.000
2	43659	2	4911-403C-98	3	777	1	2024.994	0.00	6074.982000	7ABB600D-1E77-41BE-9FE5-B9142CFC08FA	2011-05-31 00:00:00.000
3	43659	3	4911-403C-98	1	778	1	2024.994	0.00	2024.994000	475CF8C6-49F6-486E-B0AD-AFC6A50CDD2F	2011-05-31 00:00:00.000

Lets us see an example to understand the exit

- let us find salesorderId which has alteast one line tota > 10000$

```
select
    ssoh.SalesOrderID,
    ssoh.OrderDate,
    ssoh.TotalDue,
    ssod.SalesOrderDetailID,
    ssod.LineTotal

from
    sales.SalesOrderHeader ssoh
INNER JOIN
    sales.SalesOrderDetail ssod
ON
    ssoh.SalesOrderID = ssod.SalesOrderID
where ssod.LineTotal>10000
```



Results Messages

	SalesOrderID	OrderDate	TotalDue	SalesOrderDetailID	LineTotal
1	43683	2011-05-31 00:00:00.000	48204.0662	238	12149.964000
2	43683	2011-05-31 00:00:00.000	48204.0662	241	10124.970000
3	43695	2011-05-31 00:00:00.000	44344.8265	348	10124.970000

- Here we can see there are duplicate value ..although we wanted only salesorderid with atleast 1 order with linetotal > 10000 [single salesorderId]
- In situation like these we use `EXISTS()`

```
select
    ssoh.SalesOrderID,
    ssoh.OrderDate,
    ssoh.TotalDue

from
    sales.SalesOrderHeader ssoh
where EXISTS(
    Select 1
    FROM sales.SalesOrderDetail ssod
    where ssod.linetotal > 10000
    AND ssoh.SalesOrderID = ssod.SalesOrderID
    )
```

Results | Messages

	SalesOrderID	OrderDate	TotalDue
1	43683	2011-05-31 00:00:00.000	48204.0662
2	43695	2011-05-31 00:00:00.000	44344.8265
3	43843	2011-07-01 00:00:00.000	37106.2915
4	43864	2011-07-01 00:00:00.000	43335.7219

- Here we can see only single salesorderID is returned
- **`SELECT 1`** is used when we only care about the existence of rows and not their contents.

Example: Find SalesOrderID where none of the order are greater than 1000

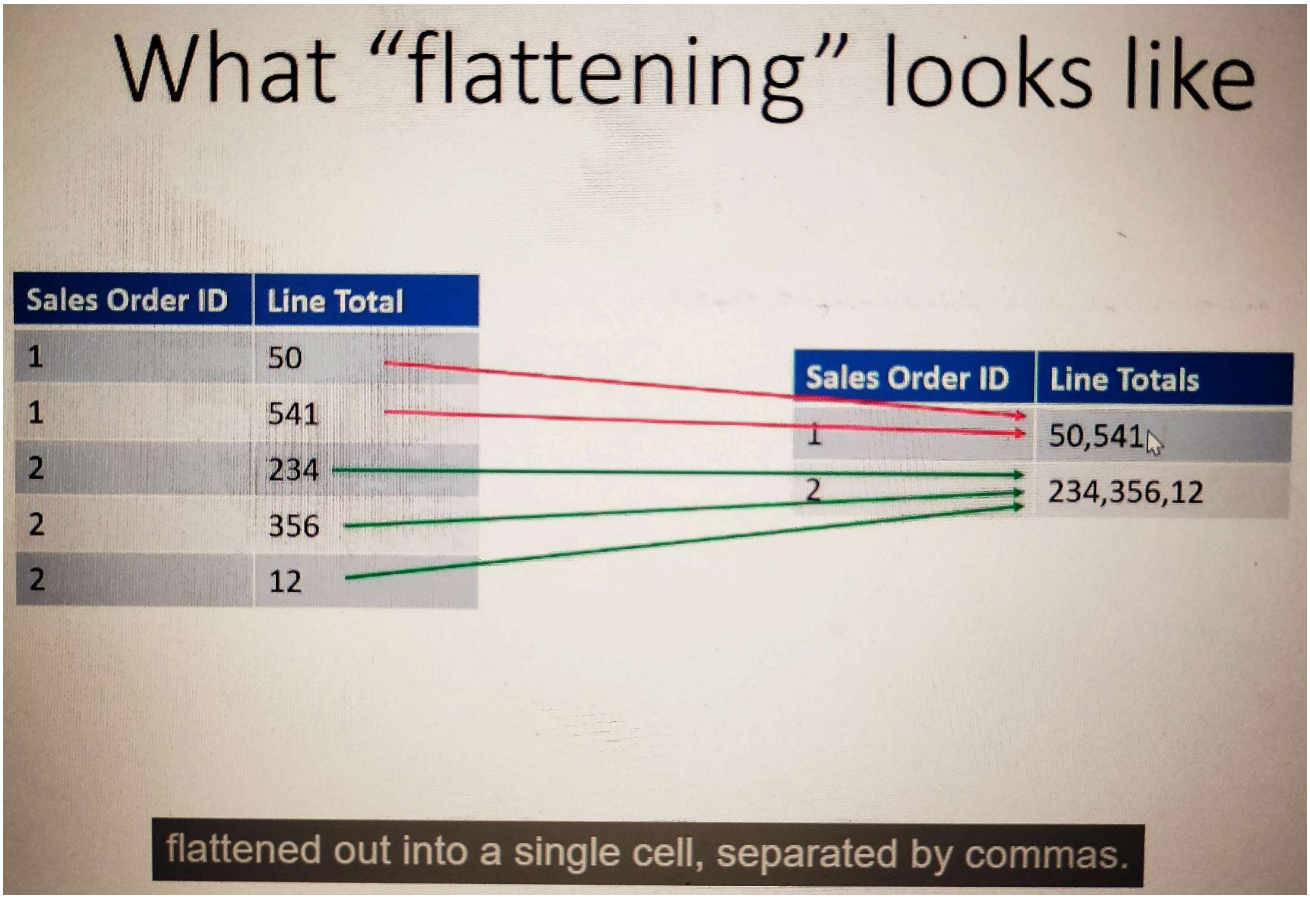
```
select
    ssoh.SalesOrderID,
    ssoh.OrderDate,
    ssoh.TotalDue

from
    sales.SalesOrderHeader ssoh
where NOT EXISTS(
    Select 1
    FROM sales.SalesOrderDetail ssod
    where ssod.linetotal > 10000
    AND ssoh.SalesOrderID = ssod.SalesOrderID
    )
```

Results | Messages

	SalesOrderID	OrderDate	TotalDue
1	43659	2011-05-31 00:00:00.000	23153.2339
2	43660	2011-05-31 00:00:00.000	1457.3288
3	43661	2011-05-31 00:00:00.000	36865.8012

Flatteing row



Results Messages

	linetotal
1	2024.994000
2	6074.982000
3	2024.994000
4	2039.994000
5	2039.994000
6	4079.988000
7	2039.994000
8	86.521200
9	28.840400
10	34.200000
11	10.373000
12	80.746000

```
Select
STUFF(

(
select
    ','+cast(cast(LineTotal as money) as varchar)

from
    sales.SalesOrderDetail ssod
where
    SalesOrderID=43659
for xml path('')
),
1,1,''
)
```

PIVOT

```
select
*
FROM
(
select
    ppc.Name as ProductCategoryName,
    ssod.LineTotal

from sales.SalesOrderDetail ssod
inner join Production.Product pp
    on pp.ProductID=ssod.ProductID
inner join Production.ProductSubcategory pps
    on pps.ProductCategoryID = pp.ProductSubcategoryID
inner join Production.ProductCategory ppc
    on ppc.ProductCategoryID = pps.ProductCategoryID
)temp

Pivot(
    sum(linetotal)
    for ProductCategoryName IN(Bikes,Clothing,Accessories,Components)

) b
```

Example

```
select
*
FROM
(
select
    ppc.Name as ProductCategoryName,
    ssod.LineTotal,
    ssod.OrderQty
-- The row which is neither pivoted nor aggregated will still remain a row

from sales.SalesOrderDetail ssod
inner join Production.Product pp
    on pp.ProductID=ssod.ProductID
inner join Production.ProductSubcategory pps
    on pps.ProductCategoryID = pp.ProductSubcategoryID
inner join Production.ProductCategory ppc
    on ppc.ProductCategoryID = pps.ProductCategoryID
)temp

Pivot(
    sum(linetotal)
    for ProductCategoryName IN(Bikes,Clothing,Accessories,Components)
```

```
) b
order by OrderQty
```

Results | Messages

	OrderQty	Bikes	Clothing	Accessories	Components
1	1	34571756.603175	40601949.642000	274356.164400	243016664.770600
2	2	13788854.493000	15651671.092800	467519.990400	69887353.317600
3	3	11555001.819900	14545694.107800	433713.085200	75776404.107600
4	4	13029439.598400	9388287.984000	323043.451200	63446257.106400
5	5	9871671.085125	6671883.768000	198859.944000	49952968.653000
6	6	8233680.152700	6143700.621600	135899.460000	34140739.290000
7	7	5264803.765200	3176649.957600	89055.246000	22926321.350800
8	8	5158873.021200	4910370.792000	40394.592000	16975884.654400
9	9	2913311.872575	3187596.715200	25291.310400	9202171.910400