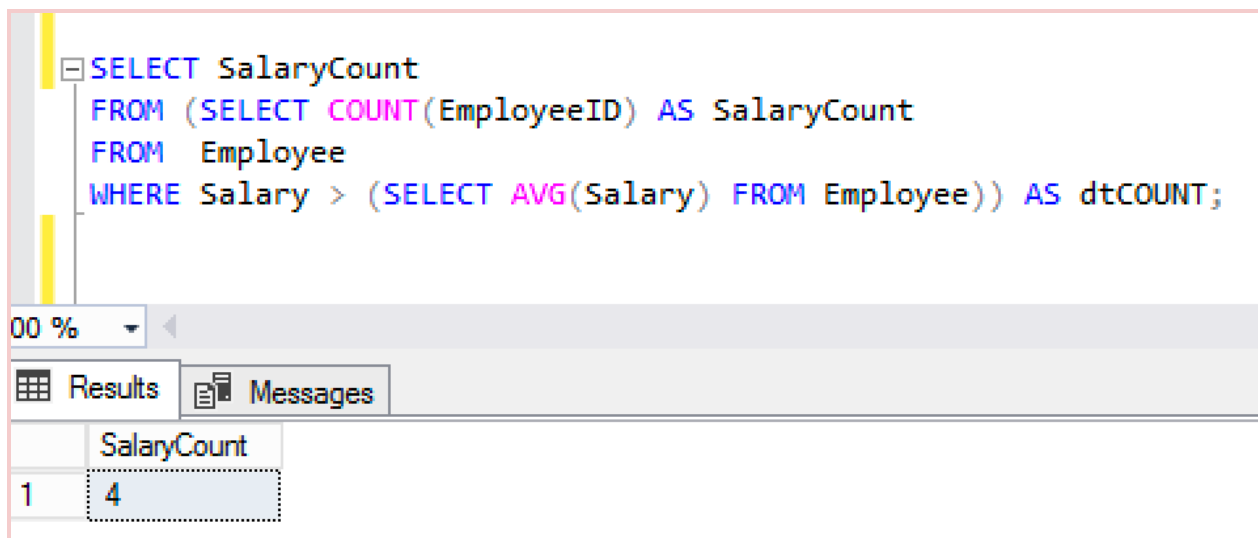


**Lab 3 -Table Expressions (Derived Tables, CTE, & Views)****(Chapter 5)**

- 1) Write a Derived Table called dtCOUNT that counts the employees making over the average salary. Name your COUNT Alias SalaryCount. In addition to the Derived Table, you need to include a Type 1 Subquery. Your SELECT statement should read: SELECT SalaryCount



The screenshot shows a SQL query window with the following text:

```
SELECT SalaryCount
FROM (SELECT COUNT(EmployeeID) AS SalaryCount
FROM Employee
WHERE Salary > (SELECT AVG(Salary) FROM Employee)) AS dtCOUNT;
```

Below the query window, the 'Results' tab is active, displaying a single row of data:

	SalaryCount
1	4

**2) Why were you able to refer to an Alias in the SELECT statement?**

Structured Query Language is a *declarative* language. Declarative languages don't tell the computer *how* to run, they simply describe the desired *result*. The implication here is the alias is only important for the *results*, not the *execution* of the query.

Since SQL only applies aliases *after* the query has completed, SQL will complain if you attempt to use an alias in another column since it doesn't actually understand what the alias represents until the query completes.

Here, the derived table means 'subquery used in the FROM clause'.

There are the sub-queries inside the parentheses. If you do not indicate the alias by using keyword 'as' for those queries, the query engine cannot determine which query is what without their names (or aliases) so, you must give unique names (aliases) for all of your subqueries to make query engine make it's work properly.

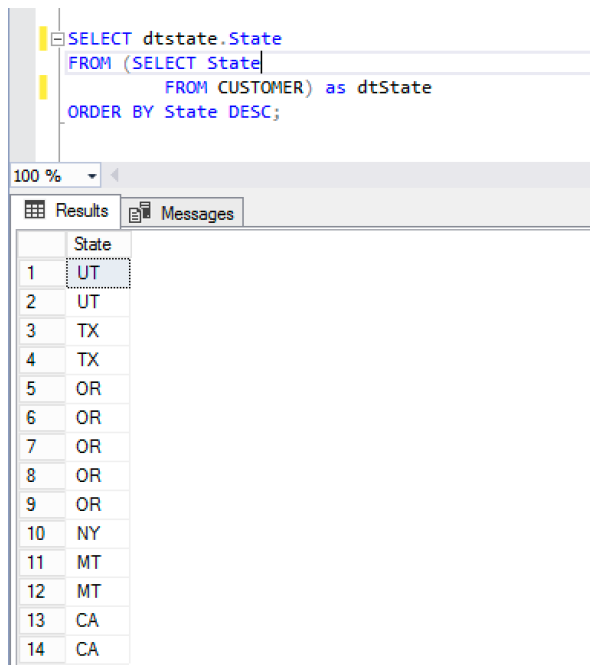
**3) Run the following code and describe the error message you received:**

```
SELECT dtstate.State
FROM (SELECT State
      FROM CUSTOMER
      ORDER BY State DESC) as dtState;
```

The error message we receive is:

The **ORDER BY** clause is invalid in views, inline functions, derived tables, subqueries, and common table expressions, unless **TOP**, **OFFSET** or **FOR XML** is also specified.

To fix this error, the **ORDER BY** clause should be placed in the outer query and be removed from the inner subquery.

**4) Rewrite the code above and place the **ORDER BY** clause in a location that will allow the code to run without an error message.**

The screenshot shows a SQL query window with the following code:

```
SELECT dtstate.State
FROM (SELECT State
      FROM CUSTOMER) as dtState
ORDER BY State DESC;
```

Below the query window, the 'Results' tab is active, displaying a table with 14 rows and one column named 'State'. The results are ordered in descending order of state abbreviations.

	State
1	UT
2	UT
3	TX
4	TX
5	OR
6	OR
7	OR
8	OR
9	OR
10	NY
11	MT
12	MT
13	CA
14	CA

- 5) Examine the following code example and indicate if it will run or not. If not, what is causing the code to fail?

```
SELECT dtstate.State  
  
FROM (SELECT State  
  
      FROM CUSTOMER);
```

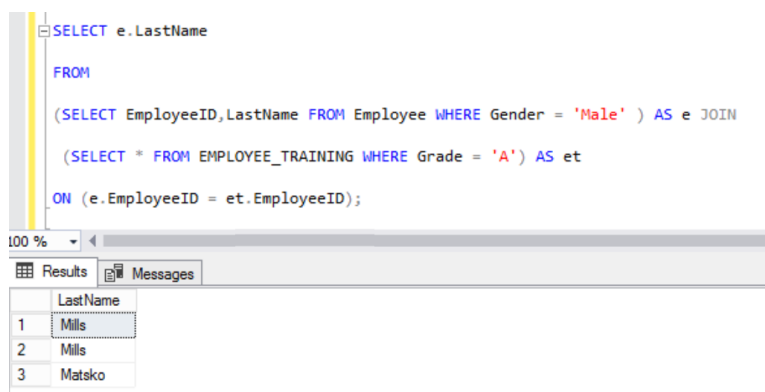
The code won't run because the derived table doesn't have an alias and hence the outer query can't reference the derived table.

When I ran the code, I got an error as expected which said: 'Incorrect syntax near ';''. This meant that our code was still incomplete and an alias was left to be assigned.

- 6) Why doesn't the following Derived Table run? Based on your explanation, fix the code so it will run.

```
SELECT e.LastName  
  
FROM  
  
(SELECT LastName FROM Employee WHERE Gender = 'Male' ) AS e JOIN  
  
(SELECT * FROM EMPLOYEE_TRAINING WHERE Grade = 'A') AS et  
  
ON (e.EmployeeID = et.EmployeeID);
```

The EmployeeID field wasn't included in the first derived table and hence the join couldn't find the field on which to join the two tables. I added EmployeeID in the first table and the code worked.



The screenshot shows a SQL query window with the following code:

```
SELECT e.LastName  
  
FROM  
  
(SELECT EmployeeID, LastName FROM Employee WHERE Gender = 'Male' ) AS e JOIN  
  
(SELECT * FROM EMPLOYEE_TRAINING WHERE Grade = 'A') AS et  
  
ON (e.EmployeeID = et.EmployeeID);
```

Below the query window, the 'Results' tab is active, displaying the following data:

	LastName
1	Mills
2	Mills
3	Matsko

Alternatively, I made a joined derived table the traditional way and the code worked returning the same answer:

```
SELECT joined_tables.LastName
FROM (SELECT LastName
      FROM Employee AS e
      JOIN EMPLOYEE_TRAINING AS et
      ON e.EmployeeID = et.EmployeeID
      WHERE Gender = 'Male' AND Grade = 'A') AS joined_tables;
```

100 %

Results Messages

	LastName
1	Mills
2	Mills
3	Matsko

- 7) Write a Common Table Expression called **cteCOUNT** that counts the employees making over the average salary. Name your COUNT Alias **SalaryCount**. As part of the CTE, you need to include a Type 1 Subquery. \*Run **SELECT \* FROM cteCOUNT** to see if your CTE runs.

```
WITH cteCOUNT
AS
(SELECT *
 FROM (SELECT COUNT(EmployeeID) AS SalaryCount
       FROM Employee
       WHERE Salary > (SELECT AVG(Salary) FROM Employee)) AS dtCOUNT
)

-- Outer query against CTE
SELECT * FROM cteCOUNT;
```

100 %

Results Messages

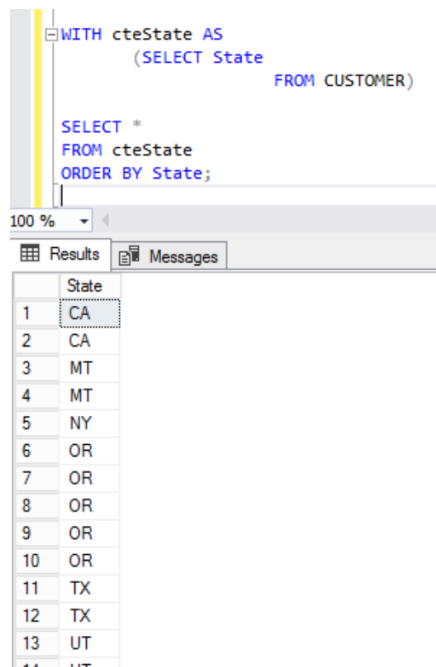
	SalaryCount
1	4

**8) If you change `SELECT * FROM cteCount` to `SELECT SalaryCount FROM cteCount`, does the code still run?**

The code will run as long as the fields stated in the 'SELECT [ ] from CTE' are present in the temporary result set or CTE created.

**9) A Derived Table does not allow the 'ORDER BY' clause. Try the CTE below and see if it allows the ORDER BY clause? If the code won't run, rewrite (move the ORDER BY clause to a new location) so it works.**

The code won't run because the ORDER BY in the derived table doesn't work without using TOP. We can place ORDER BY in the SELECT statement that calls the CTE.



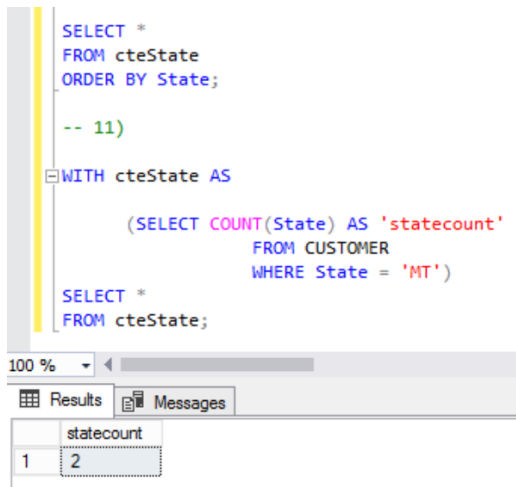
```
WITH cteState AS
    (SELECT State
     FROM CUSTOMER)

SELECT *
FROM cteState
ORDER BY State;
```

	State
1	CA
2	CA
3	MT
4	MT
5	NY
6	OR
7	OR
8	OR
9	OR
10	OR
11	TX
12	TX
13	UT
14	UT

**10) Does the following code use Inline or External Aliasing?**

The code uses External Aliasing since the CTE has 'statecount' defined in its declaration.

**Internal Aliasing:**

```
SELECT *
FROM cteState
ORDER BY State;

-- 11)

WITH cteState AS
    (SELECT COUNT(State) AS 'statecount'
     FROM CUSTOMER
     WHERE State = 'MT')
SELECT *
FROM cteState;
```

The screenshot shows a SQL query in SQL Server Enterprise Manager. The query uses a Common Table Expression (CTE) named 'cteState' which is defined within the same query block. This is an example of internal aliasing. The CTE calculates the count of customers for the state 'MT'. The main query then selects all columns from 'cteState' and orders them by the 'State' column. The results pane at the bottom shows a single row with the value '2' under the column 'statecount'.

statecount
2

**11) Henry has finished a class on SQL Server. He is now working for a company that uses Oracle. Describe how you would create a VIEW in Oracle.**

Views in Oracle can be created using nearly the same syntax as you create it in SQL Server. The minor difference that can occur is shown as following:

**SQL SERVER:**

CREATE VIEW OR ALTER 'cars\_view' AS

**ORACLE:**

CREATE VIEW OR REPLACE 'cars\_view' AS

ALTER and REPLACE create a view if it doesn't exist or replace the current one.

- 12) Create a VIEW called vCOUNT that uses a CTE called cteCOUNT to count the employees making over the average salary. Name your COUNT Alias SalaryCount. Add encryption to your VIEW. \*Hint: You need to include SELECT \* FROM cteCOUNT in your VIEW.**

```
CREATE VIEW vCOUNT With Encryption
AS
WITH cteCOUNT AS
(SELECT COUNT(EmployeeID) as SalaryCount
FROM EMPLOYEE
WHERE Salary > (SELECT AVG(Salary)
FROM EMPLOYEE))

--outer query against CTE
SELECT *
FROM cteCOUNT;
```

- 13) Conduct a brief Google search and identify 2-3 advantages of using:**

- 1) derived tables,
- 2) common table expressions,
- and 3) views.

#### **Derived Tables**

- Derived tables are one of my most useful ways of figuring out complex data for reporting as well. You can do this in pieces using table variables or temp tables too, but if you don't want to see the code in procedural steps, people often change them to derived tables once they work out what they want using temp tables. Hence, you have absolute control of what it looks like
- Doesn't automatically expand when columns are added
- SQL Statements with complex joins, expressions and conditional prompts can be used to create derived tables when those are not possible in the BO universe through normal tables. Derived tables will reduce the amount of data returned to the document for analysis, and reduce the maintenance effort of database tables.
- Derived tables can be used as a normal table in the universe and do join with other tables.
- Updating and changing the derived table structure in the universe is very easy.

**CTE**

- Better code readability, so it's easier to understand.
- Provides recursive programming.
- Maintaining code is easier.
- Though it provides similar functionality as a view, it will not store the definition in metadata. The query can be divided into separate, simple, logical building blocks which can be then used to build more complex CTEs until the final result set is generated.
- CTE can be defined in functions, stored procedures, triggers or even views.

**View**

- **Security:** Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data
- **Query Simplicity:** A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.
- **Structural simplicity:** Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for that user.
- **Consistency:** A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed.
- **Data Integrity:** If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets the specified integrity constraints.
- **Logical data independence:** View can make the application and database tables to a certain extent independent. If there is no view, the application must be based on a table. With the view, the program can be established in view of above, to view the program with a database table to be separated.
- It can allow for **massive performance improvements**