Using Table Expressions

Table expressions are named queries.
You write an inner query that returns a relational result set, name it, and query it from an outer query

- Using Views

You preserve the definition of the table expression in the database as an object; it's reusable, and you can also control access to the object with permissions

```
CREATE VIEW HR.EmpPhoneList
AS
SELECT empid, lastname, firstname, phone
FROM HR.Employees;
```

name it

inner query

```
SELECT empid, lastname
FROM HR.EmpPhoneList;
GO
```

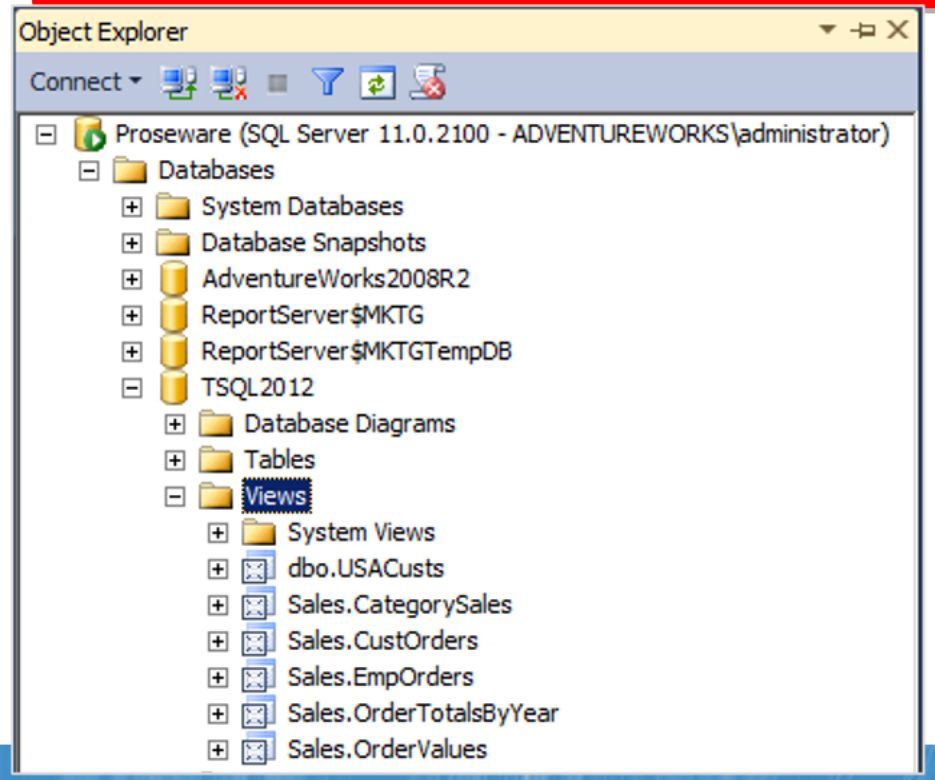
query it from an outer query

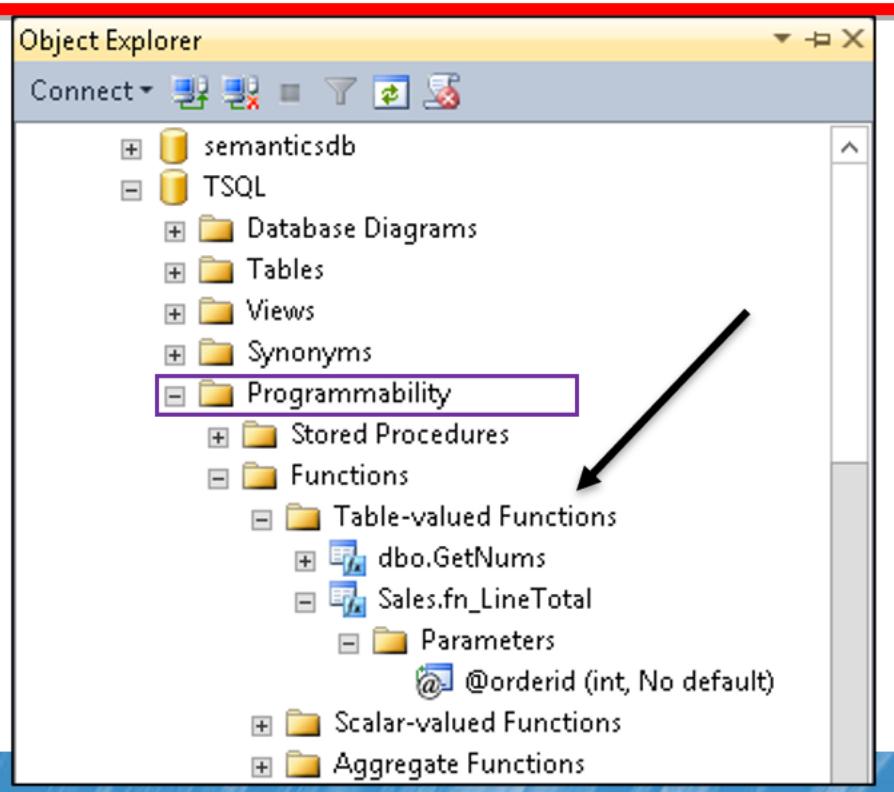
Using Table Expressions

Table expressions are named queries.
You write an inner query that returns a relational result set, name it, and query it from an outer query

- Using Views
- Using Inline Table-Valued Functions **(TVF)**

You preserve the definition of the table expression in the database as an object; it's reusable, and you can also control access to the object with permissions





Using Table Expressions

Table expressions are named queries.
You write an inner query that returns a relational result set, name it, and query it from an outer query

- Using Views
- Using Inline Table-Valued Functions

You preserve the definition of the table expression in the database as an object; it's reusable, and you can also control access to the object with permissions

- Using Derived Tables
- Using Common Table Expressions **(CTE)**

Are visible only to the statements that defines them.
-> are not reusable

Requirements of the inner query (a table expression represents a relation):
- All columns returned by the inner query must have names
- The inner query is not allowed to have an ORDER BY clause (a set has no order)

Using Views

```
SELECT custid, ordermonth, qty
FROM Sales.CustOrders;
```

The partial results are

```
custid      ordermonth                qty
----------- ------------------------- -----
7           2006-07-01 00:00:00.000 50
13          2006-07-01 00:00:00.000 11
14          2006-07-01 00:00:00.000 57
```

There is no way to determine that the FROM clause references a view and not a table

Views:
can be used as a source for queries in much the same way as tables themselves

Writing Queries That Return Results from Views

```
SELECT      <select_list>
FROM        <view_name>
ORDER BY    <sort_list>;
```

- Views may be referenced in a SELECT statement just like a table
- Views are named table expressions with definitions stored in a database, NOT the result set of the view!!
- Like derived tables and CTEs, queries that use views can provide encapsulation and simplification
- From an administrative perspective, views can provide a security layer to a database

CTE = Common Table Expressions

Creating Simple Views

- Views are saved queries created in a database by administrators and developers
- Views are defined with a single SELECT statement
- ORDER BY is not permitted in a view definition without the use of TOP, OFFSET/FETCH, or FOR XML
  - To sort the output, use ORDER BY in the outer query

```
SELECT    <select_list>
FROM      <view_name>
ORDER BY  <sort_list>;
```

```
CREATE VIEW HR.EmpPhoneList
AS
SELECT empid, lastname, firstname, phone
FROM HR.Employees;
```

inner query

```
SELECT empid, lastname
FROM HR.EmpPhoneList;
GO
```

query it from an outer query

Creating Simple Views

empid	lastname	firstname	phone
1	Davis	Sara	(206) 555-0101
2	Funk	Don	(206) 555-0100
3	Lew	Judy	(206) 555-0103
4	Peled	Yael	(206) 555-0104
5	Buck	Sven	(71) 234-5678
6	Suurs	Paul	(71) 345-6789
7	King	Russell	(71) 123-4567
8	Cameron	Maria	(206) 555-0102
9	Dolgop...	Zoya	(71) 456-7890

Virtual table

```
SELECT empid, lastname
FROM HR.EmpPhoneList;
GO
```

The FROM clause:
The view is executed and returns a virtual table

This virtual table is input for the calling SELECT

empid	lastname
5	Buck
8	Cameron
1	Davis
9	Dolgopyatova
2	Funk
7	King
3	Lew
4	Peled
6	Suurs

Result table

```
CREATE VIEW HR.EmpPhoneList
AS
SELECT empid, lastname, firstname, phone
FROM HR.Employees;
```

```
SELECT empid, lastname
FROM HR.EmpPhoneList;
GO
```

View

orderyear	cust_count
2006	67
2007	86
2008	81

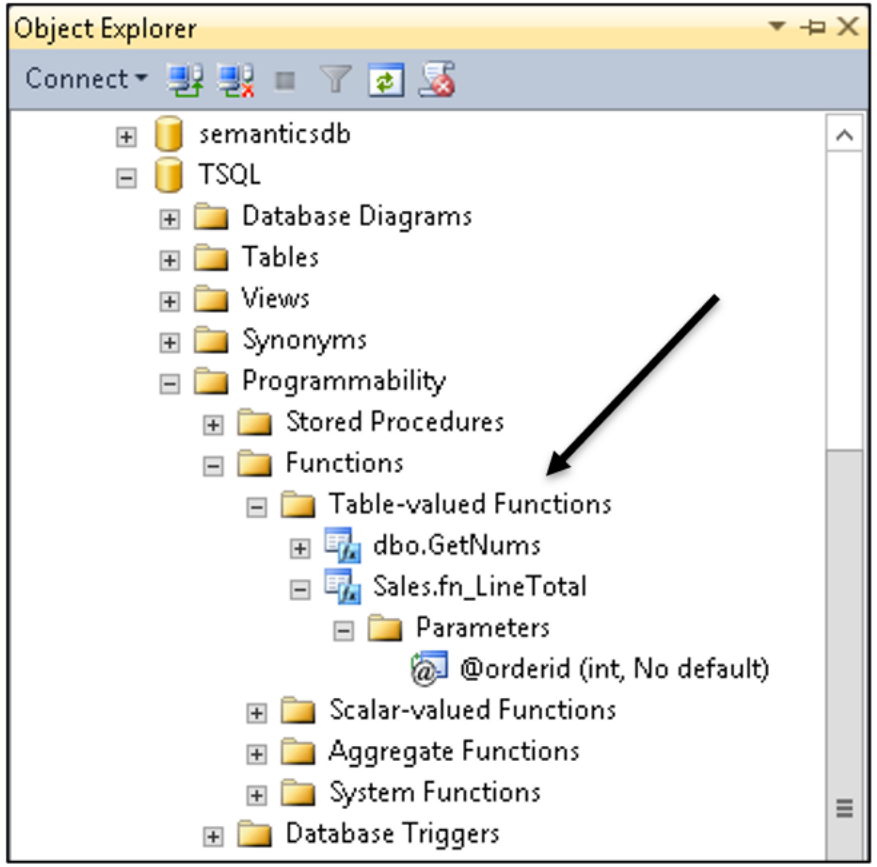
```
CREATE VIEW Sales.vw_year
AS
SELECT YEAR(orderdate) AS orderyear, custid
FROM Sales.Orders ;
```

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM Sales.vw_year
GROUP BY orderyear ;
```

3	2006	34
4	2006	84
5	2006	76
6	2006	34
7	2006	14
151	2006	71
152	2006	83
153	2007	19
154	2007	65
155	2007	20
156	2007	20
559	2007	84
560	2007	27
561	2008	55
562	2008	88
563	2008	42
564	2008	47
828	2008	68
829	2008	9
830	2008	65

Writing Queries That Use Inline Table-Valued Functions

- Table-valued functions (TVFs) are named table expressions with definitions stored in a database that can be queried in much the same way as a view



Writing Queries That Use Inline Table-Valued Functions

- Table-valued functions (TVFs) are named table expressions with definitions stored in a database that can be queried in much the same way as a view

> This enables reuse and centralized management of code in a way that is not possible for derived tables and CTEs as query-scoped table expressions

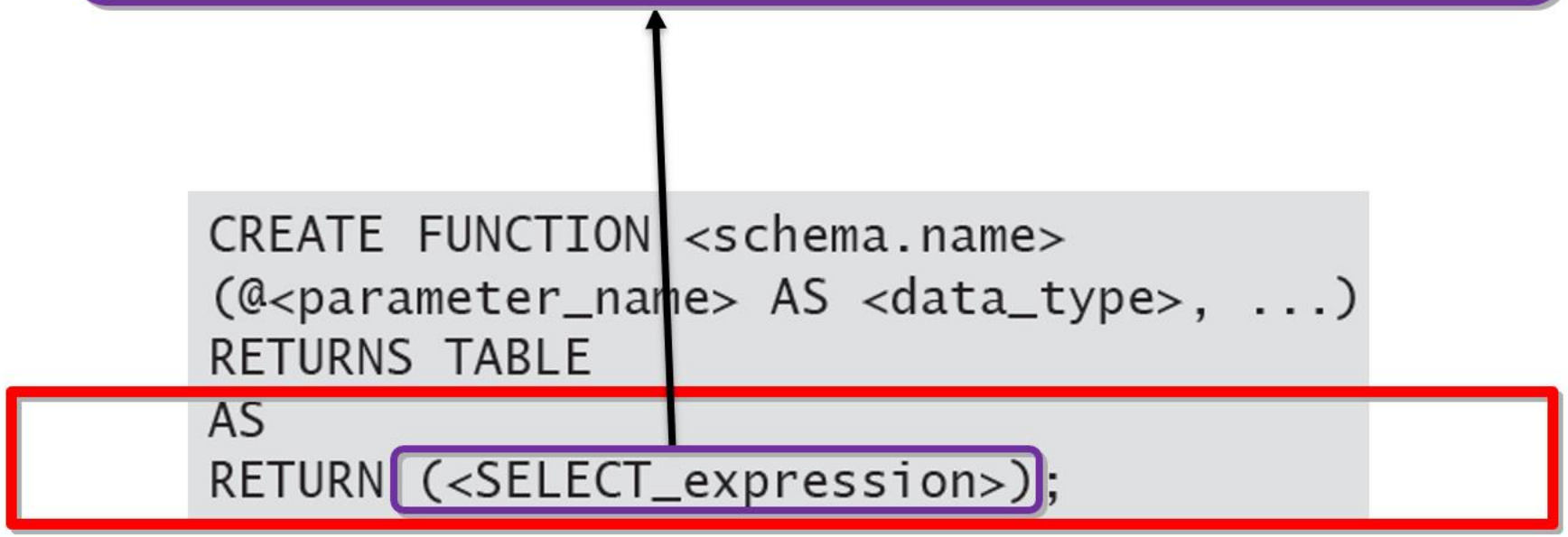
- TVFs return a virtual table to the calling query
- SQL Server provides two types of TVFs
  - Inline, based on a single SELECT statement
  - Multi-statement, which creates and loads a table variable
- Unlike views, TVFs support input parameters
- Inline TVFs may be thought of as parameterized views

Creating Simple Inline Table-Valued Functions

- Create and name function and optional parameters with CREATE FUNCTION
- Declare return type as TABLE to identify this function as a TVF.
- Define inline SELECT statement following RETURN

```
CREATE FUNCTION <schema.name>
(@<parameter_name> AS <data_type>, ...)
RETURNS TABLE
AS
RETURN (<SELECT_expression>);
```

```
CREATE FUNCTION Production.TopNProducts
(@t AS INT)
RETURNS TABLE
AS
RETURN
    (SELECT TOP (@t) productid, productname, unitprice
            FROM Production.Products
            ORDER BY unitprice DESC);
```



```
CREATE FUNCTION <schema.name>
(@<parameter_name> AS <data_type>, ...)
RETURNS TABLE
AS
RETURN (<SELECT_expression>);
```

```
CREATE FUNCTION Production.TopNProducts
(@t AS INT)
RETURNS TABLE
AS
RETURN
   (SELECT TOP (@t) productid, productname, unitprice
           FROM Production.Products
           ORDER BY unitprice DESC);
```

how to query an inline TVF: `SELECT * FROM Production.TopNProducts(3)`

input parameter value

The results:

```
productid productname   unitprice
--------- ------------- ---------
38        Product QDOMO 263.50
29        Product VJXYN 123.79
9         Product AOZBW 97.00

(3 row(s) affected)
```

Retrieving from Inline Table-Valued Functions

- SELECT from function, Use two-part name, Pass in parameters

```
CREATE FUNCTION Sales.fn_LineTotal (@orderid INT)
RETURNS TABLE
AS
RETURN
    SELECT orderid, productid, unitprice, qty, discount,
       CAST((qty * unitprice * (1 - discount)) AS
             DECIMAL(8, 2)) AS line_total
    FROM    Sales.OrderDetails
    WHERE   orderid = @orderid ;
```

```
SELECT orderid, productid, unitprice,
       qty, discount, line_total
FROM Sales.fn_LineTotal(10252) AS LT;
```

orderid	productid	unitprice	qty	discount	line_total
10252	20	64.80	40	0.05	2462.40
10252	33	2.00	25	0.05	47.50
10252	60	27.20	40	0.00	1088.00

View

orderyear	cust_count
2006	67
2007	86
2008	81

```
CREATE VIEW Sales.vw_year
AS
SELECT YEAR(orderdate) AS orderyear, custid
FROM Sales.Orders ;
```

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM Sales.vw_year
GROUP BY orderyear ;
```

Table-Valued Function

orderyear	cust_count
2006	67
2007	86
2008	81

```
CREATE FUNCTION Sales.TVF_year()
RETURNS TABLE
AS
RETURN
    (SELECT YEAR(orderdate) AS orderyear, custid
    FROM Sales.Orders) ;
```

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM Sales.TVF_year()
GROUP BY orderyear ;
```

Writing Queries with Derived Tables

- Derived tables are named query expressions created within an outer SELECT statement
- Not stored in database – represents a virtual relational table
- When processed, unpacked into query against underlying referenced objects
- Allow you to write more modular queries

```
SELECT <column_list>
FROM   (
       <derived_table_definition>
       ) AS <derived_table_alias>;
```

outer query

inner query

- Scope of a derived table is the query in which it is defined

Writing Queries with Derived Tables

```
SELECT <outer query column list>
FROM (SELECT <inner query column list>
      FROM <table source>) AS <derived table alias>
```

To create a derived table,
write the inner query between parentheses,
followed by an AS clause and
name for the derived table

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM (SELECT YEAR(orderdate) AS orderyear, custid
      FROM Sales.Orders) AS derived_year
GROUP BY orderyear;
```

Writing Queries with Derived Tables

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM (SELECT YEAR(orderdate) AS orderyear, custid
      FROM Sales.Orders) AS derived_year
GROUP BY orderyear;
```

Inner query result:

orderyear	custid
2006	85
2006	79
2006	34

A partial result for the inner query

Outer query result:

orderyear	cust_count
2006	67
2007	86
2008	81

The inner results are passed to the outer query, which operates on the derived table's orderyear and custid columns:

Using Aliases for Column Names in Derived Tables

- Column aliases may be defined inline:

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM (
       SELECT YEAR(orderdate) AS orderyear, custid
       FROM Sales.Orders) AS derived_year
GROUP BY orderyear;
```

- Column aliases may be defined externally:

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM (
       SELECT YEAR(orderdate), custid
       FROM Sales.Orders) AS
       derived_year(orderyear, custid)
GROUP BY orderyear;
```

```
SELECT <outer query column list>
FROM (SELECT <inner query column list>
      FROM <table source>) AS <derived table alias>
```

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM (SELECT YEAR(orderdate) AS orderyear, custid
      FROM Sales.Orders) AS derived_year
GROUP BY orderyear;
```

Derived Tables Must

- Have an alias
- Have names for all columns
- Have unique names for all columns
- Not use an ORDER BY clause (without TOP or OFFSET/FETCH)
- Not be referred to multiple times in the same query

Derived Tables May

- Use internal or external aliases for columns
- Refer to parameters and/or variables
- Be nested within other derived tables

Passing Arguments to Derived Tables

- Derived tables may refer to arguments
- Arguments may be:
  - Variables declared in the same batch as the SELECT statement
  - Parameters passed into a table-valued function or stored procedure

```
DECLARE @emp_id INT = 9;
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM (
        SELECT YEAR(orderdate) AS orderyear, custid
        FROM Sales.Orders
        WHERE empid=@emp_id
) AS derived_year
GROUP BY orderyear;
```

The results:

```
orderyear   cust_count
----------- -----------
2006        5
2007        16
2008        16

(3 row(s) affected)
```

Nesting and Reusing of Derived Tables

- Derived tables may be nested, though not recommended:

```
SELECT orderyear, cust_count
FROM ( SELECT  orderyear, COUNT(DISTINCT custid) AS cust_count
       FROM ( SELECT YEAR(orderdate) AS orderyear,custid
              FROM Sales.Orders) AS derived_table_1
       GROUP BY orderyear) AS derived_table_2
WHERE cust_count > 80;
```

- Derived tables may not be referred to multiple times in the same query
  - Each reference must be separately defined

Nesting and Reusing of Derived Tables

```
SELECT orderyear, cust_count
FROM  (
      SELECT  orderyear, COUNT(DISTINCT custid) AS cust_count
      FROM (
            SELECT YEAR(orderdate) AS orderyear ,custid
        FROM Sales.Orders) AS derived_table_1
      GROUP BY orderyear) AS derived_table_2
WHERE cust_count > 80;
```

derived_table_1:

orderyear	custid
2006	85
2006	79
2006	34

(830 row(s) affected)

derived_table_2:

orderyear	cust_count
2006	67
2007	86
2008	81

(3 row(s) affected)

outer query:

orderyear	cust_count
2007	86
2008	81

(2 row(s) affected)

View

orderyear	cust_count
2006	67
2007	86
2008	81

```
CREATE VIEW Sales.vw_year
AS
SELECT YEAR(orderdate) AS orderyear, custid
FROM Sales.Orders ;
```

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM Sales.vw_year
GROUP BY orderyear ;
```

Table-Valued Function

orderyear	cust_count
2006	67
2007	86
2008	81

```
CREATE FUNCTION Sales.TVF_year()
RETURNS TABLE
AS
RETURN
    (SELECT YEAR(orderdate) AS orderyear, custid
    FROM Sales.Orders) ;
```

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM Sales.TVF_year()
GROUP BY orderyear ;
```

Derived Table

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM    (
        SELECT YEAR(orderdate) AS orderyear, custid
        FROM Sales.Orders) AS derived_year
GROUP BY orderyear ;
```

Writing Queries with Common Table Expressions

- CTEs are named table expressions defined in a query
- CTEs are similar to derived tables in scope and naming requirements
- Unlike derived tables, CTEs support multiple definitions, multiple references, and recursion

```
WITH <CTE_name>
AS (
   <CTE_definition>
      )
<outer query referencing CTE>;
```

Creating Queries with Common Table Expressions

- To create a CTE:
  - Define the table expression in WITH clause
  - Assign column aliases (inline or external)
  - Pass arguments if desired
  - Reference the CTE in the outer query

```
WITH <CTE_name>
AS (
   <CTE_definition>
      )
<outer query referencing CTE>;
```

```
WITH CTE_year AS
       (
       SELECT YEAR(orderdate) AS orderyear, custid
       FROM Sales.Orders
       )
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM CTE_year
GROUP BY orderyear;
```

Writing Queries with Common Table Expressions

Creating Queries with Common Table Expressions

```
WITH CTE_year AS
       (
       SELECT YEAR(orderdate) AS orderyear, custid
       FROM Sales.Orders
       )
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM CTE_year
GROUP BY orderyear;
```

The results:

```
orderyear   cust_count
----------- -----------
2006        67
2007        86
2008        81

(3 row(s) affected)
```

```
WITH <CTE_name>
AS (
   <CTE_definition>
      )
<outer query referencing CTE>;
```

View

orderyear	cust_count
2006	67
2007	86
2008	81

```
CREATE VIEW Sales.vw_year
AS
SELECT YEAR(orderdate) AS orderyear, custid
FROM Sales.Orders ;
```

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM Sales.vw_year
GROUP BY orderyear ;
```

Table-Valued Function

orderyear	cust_count
2006	67
2007	86
2008	81

```
CREATE FUNCTION Sales.TVF_year()
RETURNS TABLE
AS
RETURN
    (SELECT YEAR(orderdate) AS orderyear, custid
    FROM Sales.Orders) ;
```

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM Sales.TVF_year()
GROUP BY orderyear ;
```

Derived Table

```
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM    (
        SELECT YEAR(orderdate) AS orderyear, custid
        FROM Sales.Orders) AS derived_year
GROUP BY orderyear ;
```

CTE Common Table Expression

```
WITH CTE_year AS
    (
    SELECT YEAR(orderdate) AS orderyear, custid
    FROM Sales.Orders
    )
SELECT orderyear, COUNT(DISTINCT custid) AS cust_count
FROM CTE_year
GROUP BY orderyear ;
```