# Module 11

## Using Table Expressions

# Module Overview

- Using Views
- Using Inline TVFs
- Using Derived Tables
- Using CTEs

# Lesson 1: Using Views

- Writing Queries That Return Results from Views
- Creating Simple Views
- Demonstration: Using Views

# Writing Queries That Return Results from Views

- Views may be referenced in a SELECT statement just like a table
- Views are named table expressions with definitions stored in a database
- 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

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

# 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
- View creation supports additional options beyond the scope of this class

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

# Demonstration: Using Views

In this demonstration, you will see how to:

- Create views

# Lesson 2: Using Inline TVFs

- Writing Queries That Use Inline TVFs
- Creating Simple Inline TVFs
- Retrieving from Inline TVFs
- Demonstration: Inline TVFs

# Writing Queries That Use Inline TVFs

- TVFs are named table expressions with definitions stored in a database
- 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 TVFs

- TVFs are created by administrators and developers
- Create and name function and optional parameters with CREATE FUNCTION
- Declare return type as TABLE
- Define inline SELECT statement following RETURN

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

# Retrieving from Inline TVFs

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

```
SELECT orderid, line_total
FROM Sales.fn_LineTotal(10252) AS LT;
```

| orderid | line_total |
|---|---|
| 10252 | 2462.40 |
| 10252 | 47.50 |
| 10252 | 1088.00 |

# Demonstration: Inline TVFs

In this demonstration, you will see how to:

- Create inline TVFs

# Lesson 3: Using Derived Tables

- Writing Queries with Derived Tables
- Guidelines for Derived Tables
- Using Aliases for Column Names in Derived Tables
- Passing Arguments to Derived Tables
- Nesting and Reusing Derived Tables
- Demonstration: Using Derived Tables

# 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>;
```

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

# Guidelines for Derived Tables

## 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

# 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;
```

# Passing Arguments to Derived Tables

- Derived tables may refer to arguments
- Arguments might 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;
```

# Nesting and Reusing 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

# Demonstration: Using Derived Tables

In this demonstration, you will see how to:

- Write queries that create derived tables

# Lesson 4: Using CTEs

- Writing Queries with CTEs
- Creating Queries with Common Table Expressions
- Demonstration: Using CTEs

# Writing Queries with CTEs

- 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 a WITH clause
  - Assign column aliases (inline or external)
  - Pass arguments if desired
  - Reference the CTE in the outer query

```
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;
```

# Demonstration: Using CTEs

In this demonstration, you will see how to:

- Write queries that create CTEs

# Lab: Using Table Expressions

- Exercise 1: Writing Queries That Use Views
- Exercise 2: Writing Queries That Use Derived Tables
- Exercise 3: Writing Queries That Use CTEs
- Exercise 4: Writing Queries That Use Inline TVFs

**Logon Information**
Virtual machine: **20761C-MIA-SQL**
User name: **ADVENTUREWORKS\Student**
Password: **Pa55w.rd**

**Estimated Time: 90 minutes**

# Lab Scenario

As a business analyst for Adventure Works, you will be writing reports using corporate databases stored in SQL Server. You have been given a set of business requirements for data and you will write T-SQL queries to retrieve the specified data from the databases. Because of advanced business requests, you will have to learn how to create and query different query expressions that represent a valid relational table.

# Module Review and Takeaways

- Review Question(s)