Database Management
Views, CTEs, Indexes

By: Meyer Tanuan (2022), Rick Kozak (2019), Glenn Paulley (2015), John Mckay (2011)

Additional Source: Murach's SQL Server 2019 for Developers:

https://www.murach.com/shop/murach-s-sql-server-2019-for-developers-detail

SQL Batches

- When working with tools like SQL Query Analyzer, SQL statements must sometimes be “batched” using “GO”:

```
SQL statement(s)
GO
SQL statement(s)
GO
```

- This is due to SQL Server’s legacy implementation of Transact-SQL that did not require statement delimiters (“;”) at the end of each statement.

Views

- A view is a virtual table based on an SQL SELECT statement
  - SELECT statement with a “view name” attached to it
- Views can:
  - Contain arbitrary joins, GROUP BY, set operations (UNION, INTERSECT, EXCEPT)
  - Specify functions in the SELECT list
  - Be modified as base tables if the server can determine the underlying base table row
    - Updateable View concept
- By convention, VIEW names **start** in **VW_**, for example:
  - VW_VendorShortList, VW_Reservations

Some of the benefits provided by views

- Design independence
- Data security
- Flexibility
- Simplified queries
- Updatability
  - Views are generally used for looking at existing data
  - Although update, insert and delete via views may be allowed with *restrictions*, they are not considered best practice.

Syntax of the CREATE VIEW statement

```
CREATE VIEW view_name [(column_name_1
                    [, column_name_2]...)]
[WITH {ENCRYPTION|SCHEMABINDING}]
AS
select_statement
[WITH CHECK OPTION]
```

Note: If the view is used for insert and update operations, the WITH CHECK OPTION will enforce the SELECT statement's filter condition.

View 1: A view of vendors that have invoices

```
USE AP
GO
CREATE VIEW VW_VendorShortList
AS
SELECT VendorName, VendorContactLName,
    VendorContactFName, VendorPhone
FROM Vendors
WHERE VendorID IN
    (SELECT VendorID
     FROM Invoices)
GO
```

View 2: Naming all columns in the CREATE VIEW clause

```
CREATE VIEW VW_BalanceDue1
    ( InvoiceNumber, InvoiceDate, InvoiceTotal
    , BalanceDue)
AS
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
    , InvoiceTotal - PaymentTotal - CreditTotal
FROM Invoices
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0
GO
```

View 3: Naming only the calculated column in SELECT

```
CREATE VIEW VW_BalanceDue2
AS
SELECT InvoiceNumber, InvoiceDate, InvoiceTotal
    , InvoiceTotal - PaymentTotal - CreditTotal AS
    BalanceDue
FROM Invoices
WHERE InvoiceTotal - PaymentTotal - CreditTotal > 0
GO
```

4. A view that summarizes invoices by vendor

```
CREATE VIEW VW_InvoiceSummary
AS
SELECT VendorName, COUNT(*) AS InvoiceQty,
    SUM(InvoiceTotal) AS InvoiceSum
FROM Vendors JOIN Invoices
  ON Vendors.VendorID = Invoices.VendorID
GROUP BY VendorName
GO
```

View 5: A view that uses a join with three tables

```
USE Hotel
GO
CREATE VIEW VW_Reservations
AS
SELECT FirstName, LastName, City, State
    , CheckInDate, CheckOutDate
    , Rooms.RoomNumber, BedType, Rate
FROM Guests INNER JOIN RoomReservations
        ON Guests.GuestID = RoomReservations.GuestID
    INNER JOIN Rooms
        ON RoomReservations.RoomNumber = Rooms.RoomNumber
GO
```

A SELECT statement that uses the view

```
SELECT *
FROM VW_Reservations;
```

The result set returned by the SELECT statement (1 row)

	FirstName	LastName	City	State	CheckInDate	CheckOutDate	RoomNumber	BedType	Rate
1	Kathy	Reid	Redding	CA	2019-05-15	2019-05-16	201	King	120.00

View 6: Using WITH SCHEMABINDING option

```
IF EXISTS (SELECT TABLE_NAME FROM INFORMATION_SCHEMA.VIEWS
    WHERE TABLE_NAME = 'VW_RoomList')
    DROP VIEW VW_RoomList
GO

CREATE VIEW VW_RoomList
WITH SCHEMABINDING
AS
SELECT RoomNumber, BedType, Rate
FROM dbo.Rooms
WHERE BedType = 'King'
GO
```

Effect of WITH SCHEMABINDING option

```
-- Error unless VW_RoomList is altered or dropped
DROP TABLE Rooms;
```

Result:

```
Msg 3729, Level 16, State 1, Line 15
Cannot DROP TABLE 'Rooms' because it is being referenced by object 'VW_RoomList'.
```

Views and ORDER BY

- You cannot use ORDER BY in CREATE VIEW
- You can include ORDER BY in a SELECT statement that accesses a view

```
SELECT *
FROM AP.dbo.VW_InvoiceSummary
ORDER BY [InvoiceSum] DESC
```

Inlined Views – DT vs CTE

SQL includes the ability to specify a view from within a query

There are two ways to do this:

- Using a **derived table** (**DT**) in the query's **FROM** clause
- Using the **WITH** clause (**Common Table Expression** or **CTE**)

Example:

```
SELECT *
FROM (SELECT lastName, firstName
      FROM Person p
      WHERE p.number
      BETWEEN 1116400 and 1206464) AS DT
WHERE CHARINDEX('S', DT.lastName) = 1
```

Derived Table (DT) and Outer Join

- Derived tables require less administrative overhead
  - Query specific
  - Are not stored in the system catalog
- Derived tables are commonly used with outer joins
  - The derived table computes a result on-the-fly which is then used as a table for the (outer) join
  - Useful when the null-supplying side of the outer join needs additional SQL constructions such as SELECT DISTINCT

Derived Table (DT) and Outer Join Example

- A correlation name for the inlined view ("DT") is required
  - the column list specification is optional
- Once specified, can be used just like a catalog view
- Example:

```
SELECT *
FROM Course c LEFT OUTER JOIN
(SELECT employeeNumber, sessionCode, courseNumber, sectionNumber
 FROM CourseOffering co
 WHERE co.employeeNumber IN (3514239, 2117745)) DT
 ON (c.number = DT.courseNumber)
WHERE c.name LIKE '%Programming%';
```

Syntax of a Common Table Expression (CTE)

```
WITH cte_name1 AS (query_definition1)
[, cte_name2 AS (query_definition2)]
[...]
sql_statement
```

CTE 1. Employee_CTE Example (acts like a view)

```
USE AdventureWorks2017;
GO

-- This is similar to (re)naming all columns in the CREATE VIEW clause
WITH Employee_CTE (EmployeeNumber, Title)
AS
(SELECT NationalIDNumber, JobTitle
 FROM   HumanResources.Employee)

SELECT EmployeeNumber, Title
FROM   Employee_CTE
GO
```

CTE 2. Sales_CTE Example (inner query)

```
USE AdventureWorks2017;
GO

-- Define the CTE expression name and column list.
WITH Sales_CTE (SalesPersonID, SalesOrderID, SalesYear)
AS
(
    SELECT SalesPersonID, SalesOrderID, YEAR(OrderDate) AS SalesYear
    FROM Sales.SalesOrderHeader
    WHERE SalesPersonID IS NOT NULL
)
```

Sales_CTE Example (outer query)

```
-- Define the CTE outer query that will use the result set from the previous query.
-- The Sales_CTE only exists in the scope of the entire query operation

SELECT SalesPersonID, SalesYear, COUNT(SalesOrderID) AS TotalSales
FROM Sales_CTE
GROUP BY SalesPersonID, SalesYear WITH ROLLUP
GO
```

Basic syntax of the CREATE INDEX statement

```
CREATE [CLUSTERED|NONCLUSTERED] INDEX index_name
    ON table_name (col_name_1 [ASC|DESC]
                [, col_name_2 [ASC|DESC]]...)
    [WHERE filter-condition]
```

- Defaults to nonclustered index and ascending order
- SQL Server automatically creates a clustered index for a table's primary key

Index 1 and Index 2: Single- and Two-column indexes

```
USE AP;

-- Index 1: single column (defaults to nonclustered index)
CREATE INDEX IX_Invoices_VendorID
    ON Invoices (VendorID);

-- Index 2: multiple columns
CREATE INDEX IX_Invoices_InvoiceDate_InvoiceTotal
    ON Invoices (InvoiceDate DESC, InvoiceTotal);
```

Index 3: Explicit nonclustered index

```
USE Hotel;

-- Index 3: explicit nonclustered index
-- LastName column of the Hotel's Guests table
CREATE NONCLUSTERED INDEX IX_Guests_LastName
ON dbo.Guests (LastName ASC);
```

Index 4: Unique index

```
USE Hotel;

-- Index 4: Unique index
-- email column of the Hotel's Employees table
CREATE UNIQUE INDEX UX_Employees_Email
ON dbo.Employees
(
    Email ASC
)
```

DROP INDEX statement

- DROP INDEX index_name_1 ON table_name_1
  [, index_name_2 ON table_name_2]...

- Example:

```
USE Hotel;
DROP INDEX UX_Employees_Email ON Employees;
```

Note: You cannot delete an index that's based on a primary key or unique key constraint. To do that, you have to use the ALTER TABLE statement.

System Views: list view names

- System views in SQL Server:
  - INFORMATION_SCHEMA.*

- To show the list of view names:

```
SELECT TABLE_NAME AS [View Name]
FROM INFORMATION_SCHEMA.VIEWS
```

```
View Name
------------------------------------------------------------------
VW_Reservations
VW_RoomList
```

System Views: Table Constraints

- To show the constraint names for each table

```
SELECT LEFT( TABLE_NAME, 20 ) AS [Table Name]
     ,LEFT( CONSTRAINT_NAME, 50 ) AS [Constraint Name]
FROM INFORMATION_SCHEMA.CONSTRAINT_TABLE_USAGE
```

```
Table Name          Constraint Name
------------------- --------------------------------------------------
Employees           PK_Employees
Guests              PK_Guests
RoomReservations    PK_RoomReservations
RoomReservations    FK_RoomReservations_Guests
Guests              CK_Guests_LastNameLength
```

System Catalog: sys.default_constraints

- To show the default constraint names and definition
  - Note the importance of naming convention to make it easier to read
  - E.g., table name Guests is included in the DF_Guests_Country default constraint name

```
SELECT LEFT( name, 20 ) AS [Default Name], definition
FROM sys.default_constraints
```

```
Default Name         definition
-------------------- ----------------------------------------------
DF_Guests_Country    (N'United States')
DF_Guests_TimeStamp  (getdate())
```

System Catalog: more examples

- More query examples from the system catalog

```
-- list database objects
SELECT * FROM sys.objects
WHERE type IN ('U', 'PK', 'UQ')
ORDER BY type;

-- list user tables
SELECT * FROM sys.tables;

-- list key constraints
SELECT * FROM sys.key_constraints;

-- list FKs
SELECT * FROM sys.foreign_keys;
```