SQL View and Common Table Expression

.NET

A View is a way to create a SQL table virtually to only present data to a user. This safeguards the data from malicious intent.

HTTPS://DOCS.MICROSOFT.COM/EN-US/SQL/T-SQL/STATEMENTS/CREATE-VIEW-TRANSACT-SQL?

SQL – Computed Columns

https://docs.microsoft.com/en-us/sql/relational-databases/tables/specify-computed-columns-in-a-table?view=sql-server-ver15

A ***Computed Column*** is a virtual column whose value is based on some computation done on other columns within the table. It is not physically stored in the table unless the column is marked PERSISTED.

A ***Computed Column*** expression can use data from other columns to calculate a value for the column to which it belongs.

When creating a table, use the keyword AS to designate a column as a Computed Column.

```
CREATE TABLE dbo.Products
(
ProductID int IDENTITY(1,1) NOT NULL,
QtyAvailable smallint,
UnitPrice money,
InventoryValue
AS
(QtyAvailable * UnitPrice)
);
```

SQL – Computed Tables (Views)

https://docs.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql?view=sql-server-ver15

A ***Computed Table (AKA, View)*** is a virtual table whose contents are defined by a query. A ***View*** can be used:

- To focus, simplify, and customize the perception each user has of the database.
- As a security mechanism by allowing users to access data through the ***View*** without granting the users permissions to directly access the underlying base tables.
- To provide a backward compatible interface to emulate a table whose schema has changed.

```
GO
CREATE VIEW MyNewView
AS
SELECT * FROM Customers
WHERE City = 'Crowley';
GO
```

View – WITH SCHEMABINDING

https://docs.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql?view=sql-server-ver15#arguments
https://www.tutorialspoint.com/sql/sql-using-views.htm

When ***WITH SCHEMABINDING*** is specified:

- The base table(s) cannot be modified in a way that would affect the ***View*** definition.
- The ***View*** definition itself must first be modified or dropped to remove dependencies on the table that is to be modified.
- The ***SELECT*** statement must include the two-part names (schema.object) of tables, ***Views***, or user-defined ***Functions*** that are referenced.
- All referenced objects must be in the same database.

```
CREATE VIEW view_name
WITH SCHEMABINDING
AS
SELECT column1, column2...
FROM table_name
WHERE [condition];
```

View – WITH SCHEMABINDING

WITH SCHEMABINDING sets up a "hard" reference from the ***View*** to the table. The ***View*** prevents any changes to that table that would “break” the ***View***'s query

```
CREATE VIEW dbo.MyNewView
WITH SCHEMABINDING
AS
SELECT * FROM dbo.Customers
WHERE City = ‘Crowley’;

SELECT * FROM dbo.MyNewView
WHERE City = ‘Crowley’;

UPDATE dbo.MyNewView SET Address =
‘Addy’;
DROP VIEW dbo.MyNewView;
```

CREATE OR ALTER VIEW

https://www.sqlservertutorial.net/sql-server-views/sql-server-create-view/

If a view already exists, you need to alter it.

Create or Alter a View:

```
CREATE OR ALTER VIEW
CustomerView
 AS
 SELECT Name, Address, Age
 FROM Customers
 WHERE Age > 40';
```

Create or Alter a View:

```
ALTER VIEW CustomerView
  RENAME TO C_View;
```

Common Table Expression (CTE)

https://docs.microsoft.com/en-us/sql/t-sql/queries/with-common-table-expression-transact-sql?view=sql-server-ver15

A ***Common Table Expression*** is an expression that first creates a temporary table that is then queried by the following query. A ***Recursive Common Table Expression*** is a CTE which references itself.

1. Define the CTE expression name and column list.
    1. Use the WITH keyword followed by the CTE name and a list of comma-separated columns.
2. Define the CTE query.
    1. Use the AS keyword and write a query (in parenthesis) to query an existing table.
3. Define the outer query referencing the CTE name.
    1. Create a regular query with one exception.
    2. After SELECT and the column list,
    3. use the FROM keyword and state the name of the desired CTE
    4. Complete the query as normal
4. Now your outer query will use the CTE as its source.

```
WITH Sales_CTE (SalesPersonID, SalesOrderID,
SalesYear)
AS
(
    SELECT SalesPersonID, SalesOrderID,
    YEAR(OrderDate) AS SalesYear
    FROM Sales.SalesOrderHeader
    WHERE SalesPersonID IS NOT NULL
)

SELECT SalesPersonID,
                COUNT(SalesOrderID) AS TotalSales,
                SalesYear
FROM Sales_CTE
GROUP BY SalesYear, SalesPersonID
ORDER BY SalesPersonID, SalesYear;
```