Marcos Juki

May 14th, 2021

Foundations of Databases & SQL Programming

Assignment06

<<https://github.com/MarcosJuki/MyClassFile>>

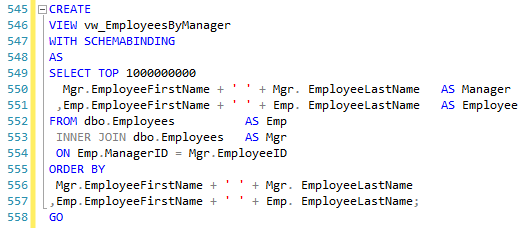
**Views, Functions, and Stored Procedures**

**Introduction |**

This document will provide some clarification and understanding of differences and similarities between SQL Views, Functions, and Stored Procedures. We continue to gather knowledge to be shared with others in my workplace.

**Explain when you would use a SQL View. |**

On previous Modules we discussed a few other protection layers to be added to a scrip that assists with maintaining the integrity of our database, tables, and data itself.

Views are additional security steps we should adopt, to restrict access, and allow DBA to continue to provide the user the data they required, while protecting other data in the same table. They are often used to store or reuse complex queries, manipulate, and rearrange column names with aliases to make them more user friendly, as well as serving as a staging area for data required on complex layered calculations.

Another use for a view is easy deployment of table structure changes, without disrupting queries or programs utilizing the current table.

“Say, you wish to retire a table (T\_OLD) containing data for active users, and instead use a new table with similar data (named T\_NEW) but one that has data for both active and inactive users, with one extra column  active. If your

***Fig. 1: Example of Base View.*** system(s) have gazillion queries that do SELECT

SELECT whatever FROM T\_OLD WHERE

whatever, you have two choices for the roll-out:

1. Cold Turkey - change the DB, and at the same time, change, test and release numerous pieces of code which contained said query. VERY hard to do (or even coordinate), very risky. Bad.
2. Gradual - change the DB by creating the T\_NEW table, dropping the T\_OLD table and instead creating a VIEW called T\_OLD that mimics the T\_OLD table 100% (e.g the view query is SELECT all\_fields\_except\_active FROM T\_NEW WHERE active=1).

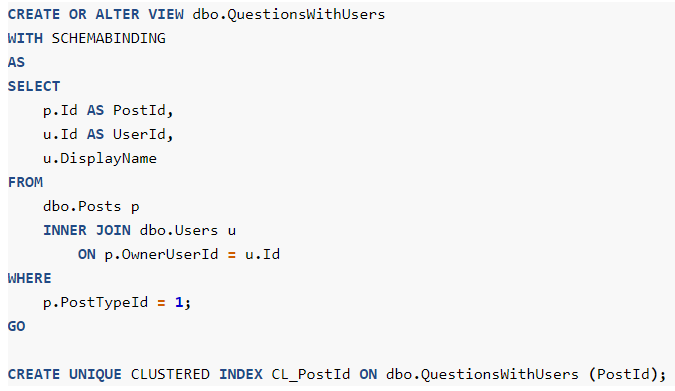
That would allow you to avoid releasing ANY code that currently selects from T\_OLD, and do the changes to migrate code from T\_OLD to T\_NEW at leisure.”

(**Stack Overflow**, <https://stackoverflow.com/questions/2680207/what-is-a-good-reason-to-use-sql-views>, 2021, External Site)

**Explain the differences and similarities between a View, |**

“SQL Server has several ways to store queries for later executions. This makes developers happy because it allows them to follow DRY principles: **D**on't **R**epeat **Y**ourself. The more code you have, the more difficult it is to maintain. Centralizing frequently used code into stored procedures, functions, etc... is attractive. While following the DRY pattern is beneficial in many programming languages, it can often cause poor performance in SQL Server.

Views are similar to inline table valued function - they allow you centralize a query in an object that can be easily called from other queries. The results of the view can be used as part of that calling query; however parameters can't be passed into the view.

Views also have some of the security benefits of a stored procedure; they can be granted access to a view with a limited subset of data from an underlying table that those same users don't have access to.

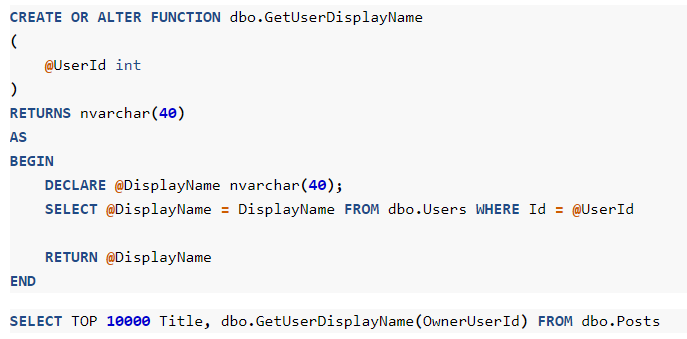
Views also have some performance advantages since they can have indexes added to them, essentially materializing the result set in advance of the view being called (creating faster performance). If considering between an inlined table function and a view, if you don't need to parameterize the input, a view is usually the better option.

***Fig. 2: Example of query to create/alter view.***

**Function, |**

Scalar functions run statements that return a single value.

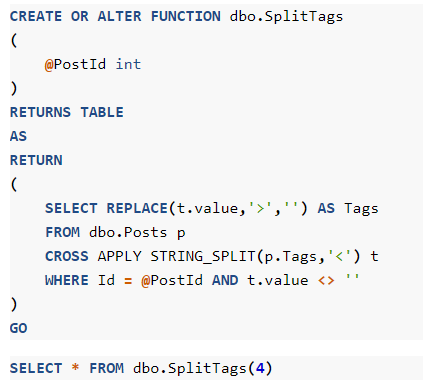
You'll often read about SQL functions being evil, and scalar functions are a big reason for this reputation. If your scalar function executes a query within it to return a single value, that means every row that calls that function runs this query . That's not good if you have to run a query once for every row in a million row table.

[SQL Server 2019 can inline a lot of these](https://docs.microsoft.com/en-us/sql/relational-databases/user-defined-functions/scalar-udf-inlining?view=sql-server-ver15), providing better performance in most cases. However, you can already do this yourself today by taking your scalar function and including it in your calling query as a subquery. The only downside is that you'll be repeating that same logic in every calling query that needs it.

Additionally, using a scalar function on the column side of a predicate will prevent SQL Server from being able to seek to data in any of its indexes; [talk about performance killing](https://bertwagner.com/2017/08/22/how-to-search-and-destroy-non-sargable-queries-on-your-server/).

For scalar functions that don't execute a query, you can always use [WITH SCHEMABINDING](https://bertwagner.com/2018/12/04/two-words-for-faster-scalar-functions/)to gain a performance boost.

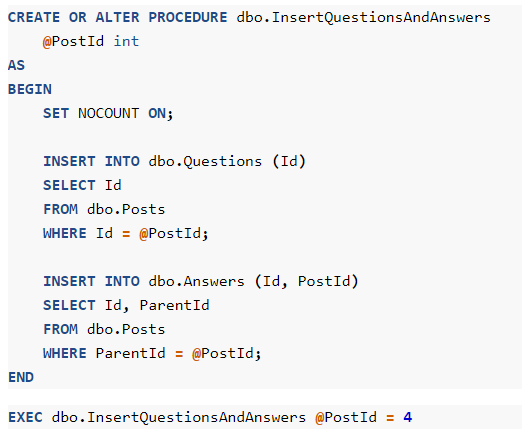
***Fig. 3: Example of query to create/alter function.***

Inline table-valued functions allow a function to return a table result set instead of just a single value. They essentially are a way for you to reuse a derived table query (you know, when you nest a child query in your main query's FROM or WHERE clause).

These are usually considered "good" SQL Server functions - their performance is decent because SQL Server can get relatively accurate estimates on the data that they will return, as long as the statistics on that underlying data are accurate. Generally this allows for efficient execution plans to be created. As a bonus, they allow parameters so if you find yourself reusing a subquery over and over again, an inline table-valued function (with or without a parameter) is actually a nice feature.

***Fig. 4: Example of query to create/alter function with parameters.***

**and Stored Procedure |**

Stored procedures encapsulate SQL query statements for easy execution. They return result sets, but those result sets can't be easily used within another query. This works great when you want to define single or multi-step processes in a single object for easier calling later.

Stored procedures also have the added benefit of being able to have more flexible security rules placed on them, allowing users to access data in specific ways where they don't necessarily have access to the underlying sources.”

***Fig. 5: Example of query to create/alter Stored Procedure.***

(**Data with Bert**, <https://bertwagner.com/posts/sql-server-stored-procedures-vs-functions-vs-views/>, 2021, External Site)

**Summary |**

Layered views and stored procedure are extremely useful when working with a large amount of data that can potentially slow the communication between server and the end user.

Below is an example of a view created to have visibility on everchanging inventory available and interchangeable parts that can potentially substitute those that are currently out of stock. A Stored Procedure has also been created to update the information and provide changes every 5 minutes.

**Example of View Table |**

USE [HPL];

GO

SET ANSI\_NULLS ON;

GO

SET QUOTED\_IDENTIFIER ON;

GO

ALTER **VIEW** [dbo].[vw\_MJ\_HMV\_PN\_INTERCHANGEABLE\_INVENTORY\_BLUE]

AS

/\*\*\*\*

Created by: MARCOS JUKI & NOELANI MARTIN

Date: 04/27/2021

Function: Stage inventory data of all I/C parts in stock, for

vw\_MJ\_HMV\_OPEN\_MATERIAL\_DETAILS. This view takes the I/C (BLUE\_IC\_PN), finds the parent (BLUE\_PN\_MASTER), and displays the ALTs (BLUE\_ALT\_PARTS) available, and QTY in stock (by outstation, outstation total and total SEA), based on inventory available (BLUE\_PN).

Data Sources:

Object: VIEW [dbo].[vw\_MJ\_HMV\_PN\_INTERCHANGEABLE\_INVENTORY\_BLUE]

Dictionary:

\*\*\*\*/

--Boeing

SELECT \* FROM OPENQUERY

(TXASRPT,

'SELECT \* FROM

(SELECT

PNM.CATEGORY BLUE\_CATEGORY

,PNIC1.PN\_INTERCHANGEABLE BLUE\_IC\_PN

,PNM.PN\_DESCRIPTION BLUE\_PN\_MASTER\_DESCRIPTION

,PNIC1.INTERCHANGEABLE\_TYPE BLUE\_INTERCHANGEABLE\_TYPE

,PNM.PN BLUE\_PN\_MASTER

,PNIC2.PN\_INTERCHANGEABLE BLUE\_ALT\_PARTS

FROM

odb.PN\_MASTER PNM

LEFT JOIN

odb.PN\_INTERCHANGEABLE PNIC1

ON PNM.PN = PNIC1.PN

LEFT JOIN

odb.PN\_INTERCHANGEABLE PNIC2

ON PNM.PN = PNIC2.PN

WHERE PNIC1.STATUS = ''ACTIVE''

) PN

LEFT JOIN

(SELECT DISTINCT

PN S\_BLUE\_PN

,SUM (QTY\_AVAILABLE) OVER (PARTITION BY PN) SEA\_BLUE\_Availability

FROM ODB.PN\_INVENTORY\_DETAIL SP

WHERE LOCATION IN (''SEA'',''SEASW'')

) S

ON S.S\_BLUE\_PN = PN.BLUE\_ALT\_PARTS

LEFT JOIN

(SELECT P2.\* FROM

(SELECT

LOCATION

,PN O\_BLUE\_PN

,QTY\_AVAILABLE

,SUM (QTY\_AVAILABLE) OVER (PARTITION BY PN) OS\_BLUE\_Availability

FROM ODB.PN\_INVENTORY\_DETAIL

WHERE LOCATION IN (''ANC'',''HNL'',''JFK'',''LAS'',''LAX'',''OAK'',''PDX'',''PHX'',''SAN'',''SFO'',''SJC'')

) P1

PIVOT

(SUM (QTY\_AVAILABLE)

FOR LOCATION IN

(''ANC'' ANC\_B,''HNL'' HNL\_B,''JFK'' JFK\_B,''LAS'' LAS\_B,''LAX'' LAX\_B,''OAK'' OAK\_B,''PDX'' PDX\_B,''PHX'' PHX\_B,''SAN'' SAN\_B,''SFO'' SFO\_B,''SJC'' SJC\_B)

) P2

) P

ON P.O\_BLUE\_PN = PN.BLUE\_ALT\_PARTS'

) GT;

GO

**Example of Stored Procedure to feed View above |**

USE [HPL];

GO

SET ANSI\_NULLS ON;

GO

SET QUOTED\_IDENTIFIER ON;

GO

ALTER **PROCEDURE** dbo.spLoad\_tb\_INTERCHANGEABLE\_INVENTORY\_BLUE AS SET NOCOUNT ON

TRUNCATE TABLE HPL.dbo.tb\_Load\_INTERCHANGEABLE\_INVENTORY\_BLUE

INSERT INTO HPL.dbo.tb\_Load\_INTERCHANGEABLE\_INVENTORY\_BLUE

(BLUE\_CATEGORY

,BLUE\_IC\_PN

,BLUE\_PN\_MASTER\_DESCRIPTION

,BLUE\_INTERCHANGEABLE\_TYPE

,BLUE\_PN\_MASTER

,BLUE\_ALT\_PARTS

,S\_BLUE\_PN

,O\_BLUE\_PN

,OS\_BLUE\_AVAILABILITY

,SEA\_BLUE\_AVAILABILITY

,ANC\_B

,HNL\_B

,JFK\_B

,LAS\_B

,LAX\_B

,OAK\_B

,PDX\_B

,PHX\_B

,SAN\_B

,SFO\_B

,SJC\_B)

SELECT

BLUE\_CATEGORY

,BLUE\_IC\_PN

,BLUE\_PN\_MASTER\_DESCRIPTION

,BLUE\_INTERCHANGEABLE\_TYPE

,BLUE\_PN\_MASTER

,BLUE\_ALT\_PARTS

,S\_BLUE\_PN

,O\_BLUE\_PN

,OS\_BLUE\_AVAILABILITY

,SEA\_BLUE\_AVAILABILITY

,ANC\_B

,HNL\_B

,JFK\_B

,LAS\_B

,LAX\_B

,OAK\_B

,PDX\_B

,PHX\_B

,SAN\_B

,SFO\_B

,SJC\_B

FROM vw\_MJ\_HMV\_PN\_INTERCHANGEABLE\_INVENTORY\_BLUE (nolock)

TRUNCATE TABLE HPL.dbo.tb\_INTERCHANGEABLE\_INVENTORY\_BLUE

INSERT INTO HPL.dbo.tb\_INTERCHANGEABLE\_INVENTORY\_BLUE

(BLUE\_CATEGORY

,BLUE\_IC\_PN

,BLUE\_PN\_MASTER\_DESCRIPTION

,BLUE\_INTERCHANGEABLE\_TYPE

,BLUE\_PN\_MASTER

,BLUE\_ALT\_PARTS

,S\_BLUE\_PN

,O\_BLUE\_PN

,OS\_BLUE\_AVAILABILITY

,SEA\_BLUE\_AVAILABILITY

,ANC\_B

,HNL\_B

,JFK\_B

,LAS\_B

,LAX\_B

,OAK\_B

,PDX\_B

,PHX\_B

,SAN\_B

,SFO\_B

,SJC\_B)

SELECT

BLUE\_CATEGORY

,BLUE\_IC\_PN

,BLUE\_PN\_MASTER\_DESCRIPTION

,BLUE\_INTERCHANGEABLE\_TYPE

,BLUE\_PN\_MASTER

,BLUE\_ALT\_PARTS

,S\_BLUE\_PN

,O\_BLUE\_PN

,OS\_BLUE\_AVAILABILITY

,SEA\_BLUE\_AVAILABILITY

,ANC\_B

,HNL\_B

,JFK\_B

,LAS\_B

,LAX\_B

,OAK\_B

,PDX\_B

,PHX\_B

,SAN\_B

,SFO\_B

,SJC\_B

FROM HPL.dbo.tb\_Load\_INTERCHANGEABLE\_INVENTORY\_BLUE (nolock);

GO