Global Knowledge®

Expert Reference Series of White Papers

SQL Server 2008 Dynamic Duo: Management Views and Functions

SQL Server 2008 DYNAMIC Duo: Management Views and Functions

Jeff Peters, MCSE, MCDBA, MCITP, MCT, A+, Net+

Introduction

Dynamic management views and functions were extraordinary additions to SQL 2005, and they have only improved in function and increased in number in SQL 2008. A minimal list of their uses would include getting performance data, index usage statistics, and object dependency information. Even if you have experience with SQL Server, you may be unsure as to how to use these tools or, for that matter, what they can do. This white paper will explore a few of their capabilities and give examples as to how they can ease the workload of a database administrator.

Where Are They?

Take a look at the full list of these objects. They all reside in the **sys** schema, and the following query will identify them for us. Open a new query window in Management Studio, and type this in (using the AdventureWorks sample database):

```
USE AdventureWorks
SELECT * FROM sys.all_objects
        WHERE [Type] IN (‘V’, ‘TF’, ‘IF’)
AND [Name] LIKE ‘%dm!_%’ ESCAPE ‘!’
ORDER BY [Name]
```

This list of dynamic management objects (DMOs) (Figure 1) gives you some ideas as to what they can show just by looking at their names. They provide plenty of invaluable information that, while mostly available in SQL versions prior to 2005, would have taken comparably enormous amounts of effort and script writing to get.

DMOs can be organized most easily by their scope, which determines whether they are meant to provide data at the level of the server or a particular database. Most DMOs can also be categorized by their area of usefulness with a few examples being: Database Mirroring, Indexes, and the SQL Operating System. DMOs made their debut in SQL 2005, but many new ones are found only in SQL 2008.

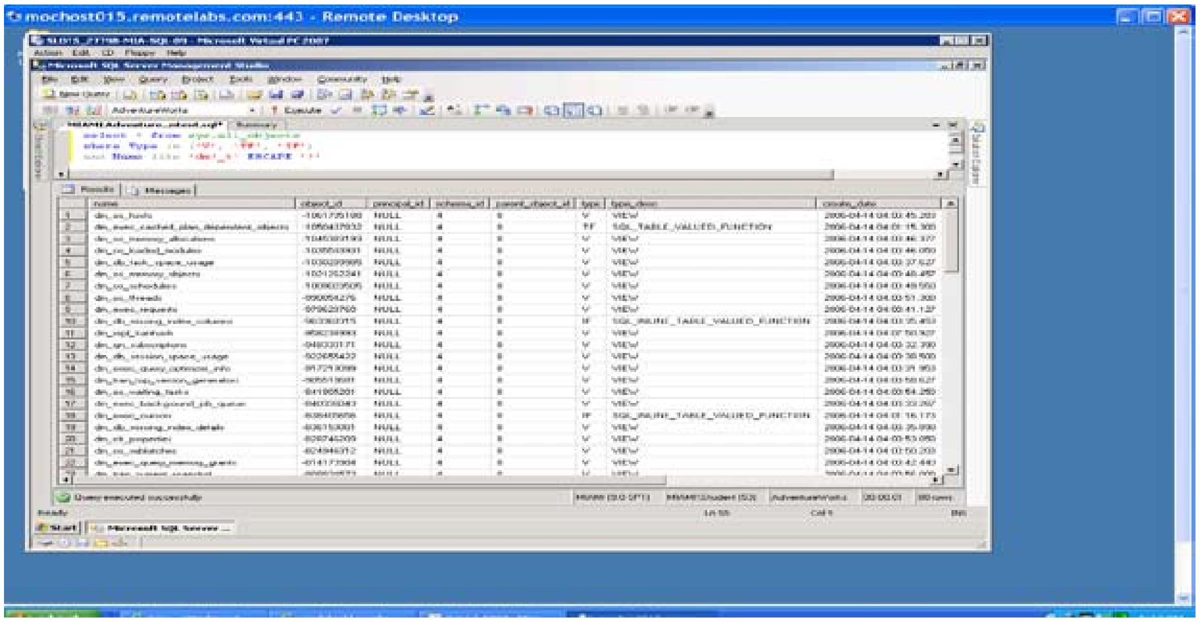


Figure 1

Troubleshooting with DMOs

One of the primary uses of DMOs is to gain immediate access to performance data. This priceless information then can be used to troubleshoot everything from problematic processes to limited free space in the **tempdb** database. Take a look at those two specific examples to see how **sys.dm_os_workers** and **sys.dm_db_file_space_usage** can help.

sys.dm_os_workers

This DMO monitors worker processes and tells us if those processes are experiencing any problems. Worker processes handle requests to execute some action on the data. If they are delayed, stuck, or fatally disrupted, you can easily have problems such as CPU red-lining occur. So how do we check on this? Type this in:

```
SELECT        is_sick,
        is_fatal_exception,
        is_in_cc_exception
FROM    sys.dm_os_workers
```

A value of 1 in the **is_sick** or **is_fatal_exception** columns will give an obvious clue as to that particular worker's status. The **is_in_cc_exception** column is helpful in that a value of 1 will identify non-SQL exceptions; this often points to a CLR process gone awry. Other available columns contain information about the amount of time the process has been running and/or waiting, the severity of the last exception it encountered, and I/Os used by or pending for the process.

sys.dm_db_file_space_usage

Tempdb is one of the usual suspects when looking for bottlenecks. Keeping track of the free space within it helps to avoid obvious performance problems. **Tempdb** can fill quickly with sorting operations, cursors, hash joins, temporary tables both local and global, and more. The **sys.dm_db_space_usage** view will reveal exactly how much space is being used and in what way.

Type this in:

```
SELECT SUM(unallocated_extent_page_count),
        (SUM(unallocated_extent_page_count)/128)
        AS [MB of Free Space]
        FROM sys.dm_db_file_space_usage
```

And you get this upon execution.

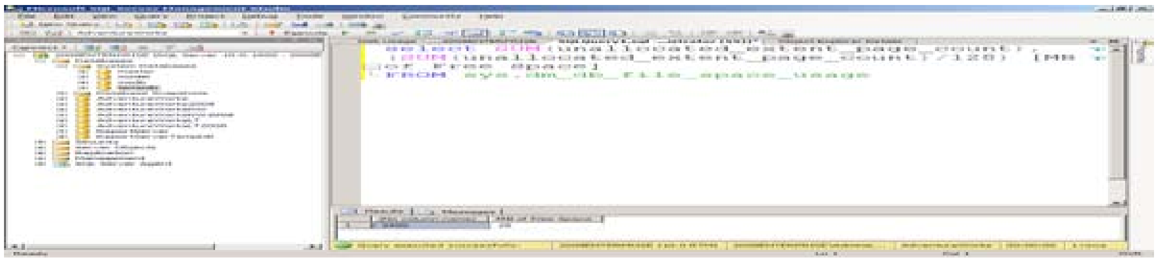
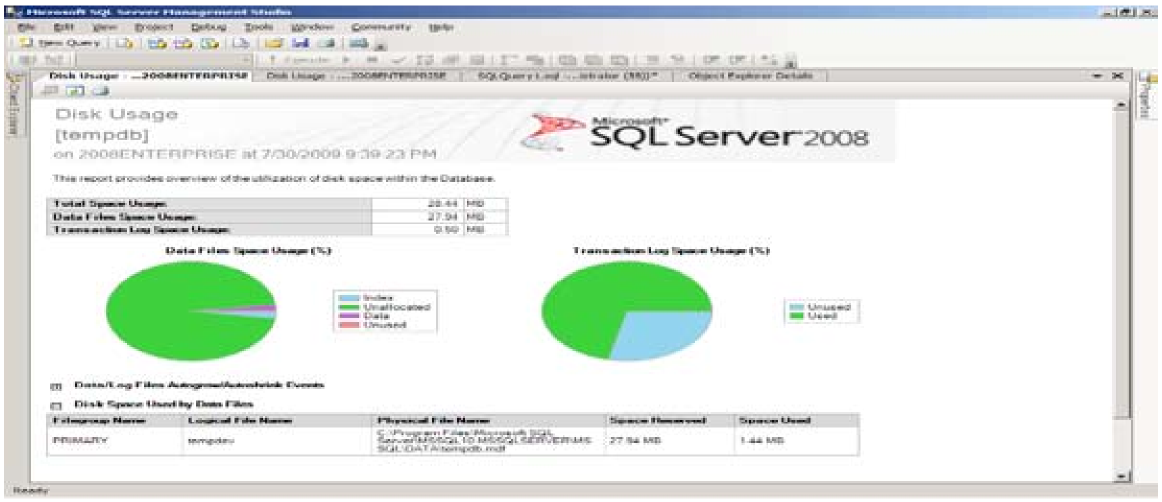


Figure 2

In Figure 2, the first column displays the number of free pages in the **tempdb**. The second column shows the amount of free space measured in MB. (Note: the value is divided by 128 because pages are 8KB blocks of data and 8*128=1024; with 1024 being the number of KB in a MB) To verify, we can compare it against the standard disk usage report on tempdb (Figure 3).



Disk Usage
[tempdb]
on 2008ENTERPRISE at 7/30/2009 9:39:23 PM

This report provides overview of the utilization of disk space within the Database.

Total Space Usage	28.44	MB
Data Files Space Usage	27.94	MB
Transaction Log Space Usage	0.50	MB

Data/Log Files Autogrow/Autoshrink Events

Disk Space Used by Data Files

Filegroup Name	Logical File Name	Physical File Name	Space Reserved	Space Used
PRIMARY	tempdev	C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\tempdb.mdf	27.94 MB	1.44 MB

Figure 3

Index Maintenance

Creating indexes is one of the easiest ways to increase performance in joins, sorts, and just about every other operation performed on a table. Unnecessary indexes, however, are often overlooked, taking up hard drive space and requiring processing power for maintenance. Poorly maintained indexes will have reduced performance due to fragmentation requiring reorganization or rebuilding to return them to their original efficiency. DMOs can help with both of these problems.

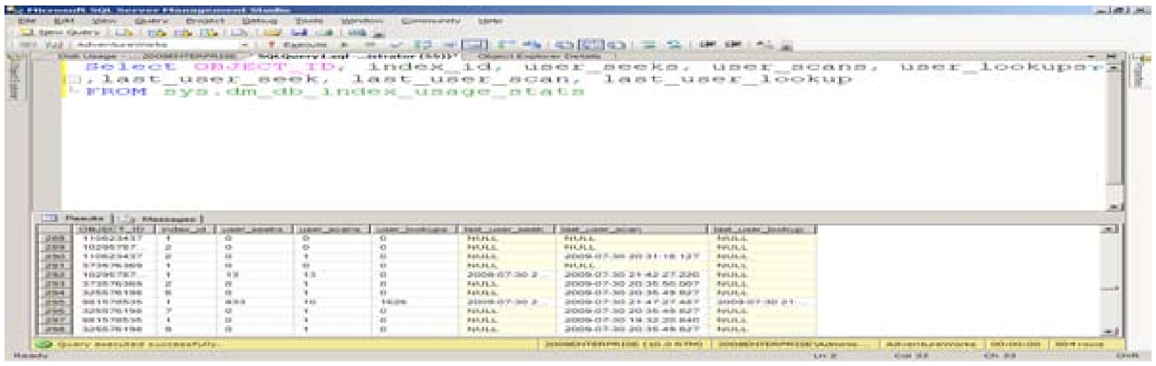
sys.dm_db_index_usage_stats

This view can return information about how many times a particular index has been used, when it was last accessed, and even the specific way in which the index has been used, such as lookups, seeks, or scans. It does not generate the information, but rather accesses data that SQL caches. Be aware that this cache resets during any reboots or database closures, so this is not a fully cumulative total that will always survive from viewing to viewing.

Here is the SQL statement:

```
SELECT          object_id, index_id, user_seeks,
          User_scans, user_lookups,
last_user_seek, last_user_scan, last_user_lookup
                        FROM sys.dm_db_index_usage_stats
```

The only confusion that may arise from using this view is that the **object_id** and **index_id** columns give only identifiers instead of naming the actual tables (Figure 4).



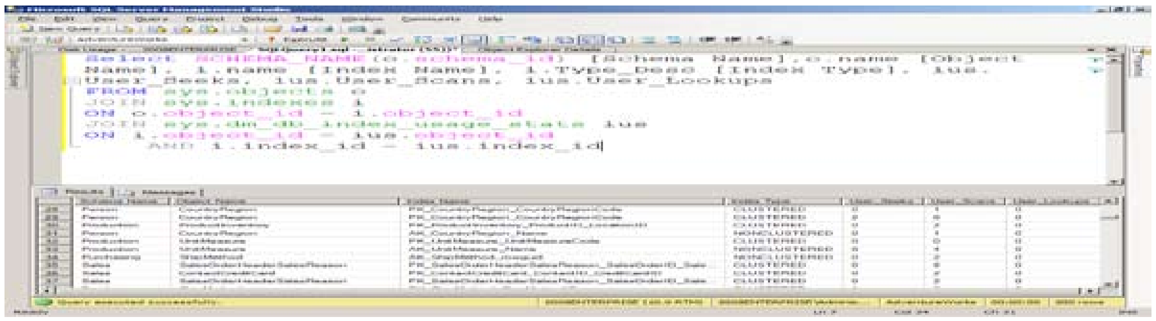
```
Select OBJECT_ID, index_id, user_seeks, user_scans, user_lookups
,last_user_seek, last_user_scan, last_user_lookup
FROM sys.dm_db_index_usage_stats
```

	OBJECT_ID	index_id	user_seeks	user_scans	user_lookups	last_user_seek	last_user_scan	last_user_lookup
[illegible]	110623437	1	0	0	0	NULL	NULL	NULL
[illegible]	102295787	2	0	0	0	NULL	NULL	NULL
[illegible]	110623437	2	0	1	0	NULL	2009-07-30 20:31:18.127	NULL
[illegible]	373576369	1	0	0	0	NULL	NULL	NULL
[illegible]	102295787	1	13	13	0	2009-07-30 2...	2009-07-30 21:42:27.220	NULL
[illegible]	373576369	2	0	1	0	NULL	2009-07-30 20:35:50.067	NULL
[illegible]	325576198	6	0	1	0	NULL	2009-07-30 20:35:49.827	NULL
[illegible]	981578535	1	433	10	1626	2009-07-30 2...	2009-07-30 21:47:27.487	2009-07-30 21...
[illegible]	325576198	7	0	1	0	NULL	2009-07-30 20:35:49.827	NULL
[illegible]	981578535	1	0	1	0	NULL	2009-07-30 19:32:20.840	NULL
[illegible]	325576198	8	0	1	0	NULL	2009-07-30 20:35:49.827	NULL

Query executed successfully.

Figure 4

This confusion can be cleared up by joining the view to the **sys.objects** and **sys.indexes** system views to resolve things such as object names and index types. (Figure 5).



```
Select SCHEMA_NAME(o.schema_id) [Schema Name],o.name [Object
Name], i.name [Index Name], i.Type_Desc [Index Type], ius.
User_Seeks, ius.User_Scans, ius.User_Lookups
FROM sys.objects o
JOIN sys.indexes i
ON o.object_id = i.object_id
JOIN sys.dm_db_index_usage_stats ius
ON i.object_id = ius.object_id
    AND i.index_id = ius.index_id
```

```
    Schema Name  Object Name                  Index Name                                           Index Type     User_Seeks  User_Scans  User_Lookups
28  Person       CountryRegion                PK_CountryRegion_CountryRegionCode                   CLUSTERED      0           1           0
29  Person       CountryRegion                PK_CountryRegion_CountryRegionCode                   CLUSTERED      2           6           0
30  Production   ProductInventory             PK_ProductInventory_ProductID_LocationID             CLUSTERED      0           2           0
31  Person       CountryRegion                AK_CountryRegion_Name                                NONCLUSTERED   0           1           0
32  Production   UnitMeasure                  PK_UnitMeasure_UnitMeasureCode                       CLUSTERED      0           0           0
33  Production   UnitMeasure                  AK_UnitMeasure_Name                                  NONCLUSTERED   6           4           0
34  Purchasing   ShipMethod                   AK_ShipMethod_rowguid                                NONCLUSTERED   0           2           0
35  Sales        SalesOrderHeaderSalesReason  PK_SalesOrderHeaderSalesReason_SalesOrderID_Sale...  CLUSTERED      0           8           0
36  Sales        ContactCreditCard            PK_ContactCreditCard_ContactID_CreditCardID          CLUSTERED      0           2           0
37  Sales        SalesOrderHeaderSalesReason  PK_SalesOrderHeaderSalesReason_SalesOrderID_Sale...  CLUSTERED      0           2           0

Query executed successfully.
```

Figure 5

sys.dm_db_index_physical_stats

Now that we know which indexes are actually being used, we can address their maintenance issues. Fragmented indexes perform inefficiently. This is a problem we can correct, and this DMO will help us identify those indexes in need of correction. Enter the following statement on a single line to get index information for the **Person.Contact** table in the AdventureWorks database:

```
SELECT * FROM sys.dm_db_index_physical_stats(
DB_ID(N'AdventureWorks'),(Object_ID(N'AdventureWorks.Person.
Contact'),NULL,NULL,NULL)
```

Look at the **avg_fragmentation_in_percent** column to identify the current level of fragmentation within each index. A general guideline is to accept up to 10% fragmentation. Up to 30% fragmentation is normally dealt with by reorganizing the index. Higher levels of fragmentation might require you to rebuild the index. The closer to zero fragmentation that the index stays, the more efficiently the index performs.

Dependency Checking

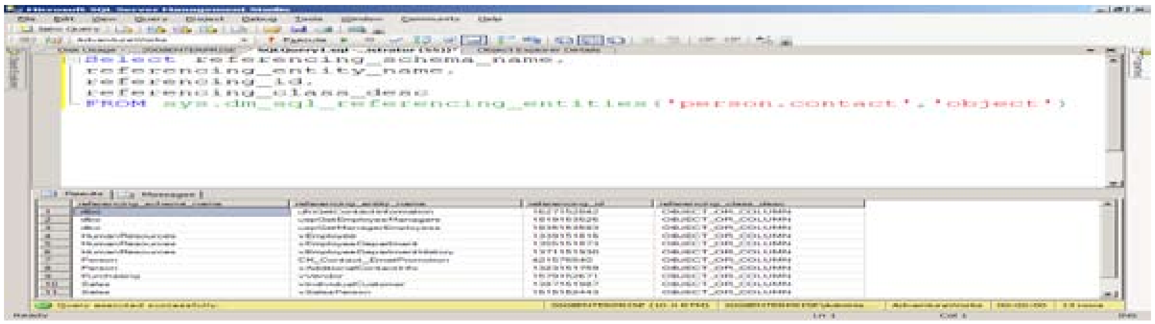
Creating a new table in a database is usually pretty straight forward. The problems always seem to present themselves when you want to modify existing objects. In earlier versions of SQL, dependencies, or references to an object by another object, were tricky and time consuming and were often missed when redesigning objects within a user database. If you have ever spent time tracking down triggers or schema-binding issues, this new functionality can save you hours! (Note that these DMOs are newly introduced in SQL 2008 and are not available in SQL 2005.)

sys.dm_sql_referencing_entities and sys.dm_sql_referenced_entities

A dependency occurs when there are two entities and the name of entity B appears in a persisted SQL expression of entity A. In that example, A would be considered the "referencing entity" and B would be considered the "referenced entity." The following code gives a list of all objects referencing the **Person.Contact** table.

```
USE AdventureWorks
SELECT referencing_schema_name,
        referencing_entity_name,
        referencing_id,
        referencing_class_desc,
        is_caller_dependent
    FROM sys.dm_sql_referencing_entities(
        'Person.Contact', 'Object')
```

The result set is a concise list of all objects that reference **Person.Contact**. (Figure 6.) If you want to see things from the opposite direction, just write a similar query using the **sys.dm_sql_referenced_entities** DMO. The result set would show all objects that the **Person.Contact** table references.



```
select referencing_schema_name,
referencing_entity_name,
referencing_id,
referencing_class_desc
FROM sys.dm_sql_referencing_entities('person.contact','object')
```

	referencing_schema_name	referencing_entity_name	referencing_id	referencing_class_desc
1	dbo	ufnGetContactInformation	[illegible]	OBJECT_OR_COLUMN
2	dbo	uspGetEmployeeManagers	[illegible]	OBJECT_OR_COLUMN
3	dbo	uspGetManagerEmployees	[illegible]	OBJECT_OR_COLUMN
4	HumanResources	vEmployee	[illegible]	OBJECT_OR_COLUMN
5	HumanResources	vEmployeeDepartment	[illegible]	OBJECT_OR_COLUMN
6	HumanResources	vEmployeeDepartmentHistory	[illegible]	OBJECT_OR_COLUMN
7	Person	CK_Contact_EmailPromotion	[illegible]	OBJECT_OR_COLUMN
8	Person	vAdditionalContactInfo	[illegible]	OBJECT_OR_COLUMN
9	Purchasing	vVendor	[illegible]	OBJECT_OR_COLUMN
10	Sales	vIndividualCustomer	[illegible]	OBJECT_OR_COLUMN
11	Sales	vSalesPerson	[illegible]	OBJECT_OR_COLUMN

Figure 6

Summary

Dynamic management views and functions have greatly improved SQL's arsenal of troubleshooting tools. Taking advantage of these new objects give you quick and easy access to in-depth information on nearly every aspect of SQL's internal workings. While they may be slightly intimidating or, more likely, entirely unknown to the average DBA, these are tools well worth the investment of your time and effort.

Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge. Check out the following Global Knowledge courses:

SQL Server T-SQL with Advanced Topics

SQL Server 2005 Tuning, Optimization, and Troubleshooting

Writing Queries Using Microsoft SQL Server 2008 Transact-SQL

SQL Server 2005 Administration

SQL Server 2005 for Developers

For more information or to register, visit **www.globalknowledge.com** or call **1-800-COURSES** to speak with a sales representative.

Our courses and enhanced, hands-on labs offer practical skills and tips that you can immediately put to use. Our expert instructors draw upon their experiences to help you understand key concepts and how to apply them to your specific work situation. Choose from our more than 700 courses, delivered through Classrooms, e-Learning, and On-site sessions, to meet your IT and management training needs.

About the Author

Jeff Peters is a trainer, network administrator, and consultant specializing in Microsoft products. He currently works for Azen Tech, LLC in Metuchen, NJ.