Say goodbye to system tables in SQL Server 2005

Microsoft SQL Server DBAs, think fast! Without using any documentation, write a query that pulls a list of indexes from the SQL Server 2000 system tables, enumerating each index's columns and whether the columns are sorted ascending or descending. You have two minutes. Go!

If you actually stopped reading just now and spent the requisite two minutes on this thankless task, you're now entangled in a big mess involving sysindexes, sysindexkeys, syscolumns and a few metadata functions, including such standbys as OBJECT_NAME and INDEXKEY_PROPERTY. And by now it should be obvious that writing this query will take longer than two minutes.

Unfortunately for SQL Server 2000 database administrators, navigating the esoteric system tables is one of the worst parts of working with the database management system (DBMS). The system tables, while generally effective, were not exactly designed with user-friendliness in mind.

Luckily, help is on the way. In SQL Server 2005, the system tables are gone. That's right. Gone. No more working with strange bitmasks or trying to figure out arcane encoding schemes -- that's all in the past. For those of you working with legacy code that references these tables, I know what you're envisioning: hours and hours of mindless updates to ensure compatibility with SQL Server 2005. But don't revolt quite yet. There are still objects around that look like the system tables, for backward-compatibility purposes. But the tables themselves can be -- and really should be -- forgotten, having been cast into the historical dustbin to join such technological greats as 8-tracks and Tab.

So where did these tables go? System data in SQL Server 2005 is now stored in hidden "resource" tables, which can only be accessed directly by the server itself. Lowly users (and DBAs) must use the new series of catalog views, which show data obtained from both the hidden tables and various hidden functions that we also can't see or play with. The system tables from previous versions of SQL Server are now implemented as a series of views called (appropriately enough) *compatibility views*.

Catalog views and their partners, dynamic management views (which are explained below), represent a way of handling meta data that's completely re-designed and re-thought out. Instead of cryptic tables that give the DBA only small tastes of the underlying data, SQL Server now offers lush resources: There are more than 200 catalog and management views in SQL Server 2005, instead of the 50 or so system tables in previous versions.

All of these views can be found in the sys schema. (Schemas are a security feature that's been greatly extended in SQL Server 2005. But that's a subject for another tip.) To see a complete list of the available views, expand the System Views tree for any database in SQL Server Management Studio. Or select a list from the views themselves using T-SQL and check out the friendly, easy-to-understand names:

```
SELECT name
FROM sys.all_views
WHERE is_ms_shipped = 1
```

You'll find that browsing the documentation for hints on how to do something with system data is hardly necessary anymore. These views are quite self-explanatory.

A few hints about the view names: Those prefixed with *dm_* are dynamic management views, which represent the changing state of the server with information like current sessions, locks, and system resources. The other views are all considered catalog views. Those prefixed with *all_* contain information about both system objects (i.e., the views) and user-defined objects. Those without the prefix *all_* contain information about user-defined objects only. In views that include system objects, the *is_ms_shipped* column can be used to differentiate between user-defined and system objects. If the *is_ms_shipped* column has a value of 1, then the row represents a system object; otherwise, it's user-defined.

Finally, let's examine the kind of data you can get from the catalog views. For starters, all of the usual contenders are available. For example: To look at data on indexes, use sys.indexes instead of the older sysindexes -- which, strangely, is now called sys.sysindexes. For constraints, try sys.check_constraints, sys.default_constraints, or sys.key_constraints. See a trend yet?

This tip could not be complete without at least one quick note on the new dynamic management views. These views are the power tools of SQL Server's new meta-data store and are intended to help DBAs quickly troubleshoot and analyze server performance. Some of the star players include sys.dm_exec_query_stats, which reports on how much processor time queries are demanding; and sys.dm_db_index_usage_stats, which can help DBAs determine which indexes are the most useful -- and those that are not.

A lot more can be said about this huge collection of meta-data views. But for now, check out the beta SQL Server 2005 Books Online, recently published on the Web by Microsoft. The System Views topic provides a full overview of the capabilities of these powerful new data repositories.

By the way, here are the solutions to the two-minute challenge. The first uses SQL Server 2000 system tables. The second, easier-to-read version is courtesy of SQL Server 2005 catalog views.

```
SELECT OBJECT_NAME(i.id) AS TableName,
          i.name AS IndexName,
          c.name AS ColumnName,
          CASE INDEXKEY_PROPERTY(i.id, i.indid, ik.keyno, 'IsDescending')
                    WHEN 1 THEN 'DESC'
                    ELSE 'ASC'
          END AS ColumnSort
FROM sysindexes i
JOIN sysindexkeys ik ON ik.id = i.id
          AND ik.indid = i.indid
JOIN syscolumns c ON c.id = ik.id
          AND c.colid = ik.colid
ORDER BY TableName,
          IndexName,
          ik.keyno
```

```
SELECT
          OBJECT_NAME(i.object_id) AS TableName,
          i.name AS IndexName,
          c.name AS ColumnName,
          CASE ic.is_descending_key
                    WHEN 1 THEN 'DESC'
                    ELSE 'ASC'
          END AS ColumnSort
FROM sys.indexes i
JOIN sys.index_columns ic ON i.object_id = ic.object_id
          AND i.index_id = ic.index_id
JOIN sys.columns c ON ic.object_id = c.object_id
          AND ic.column_id = c.column_id
ORDER BY TableName,
          IndexName,
          ic.key_ordinal
```