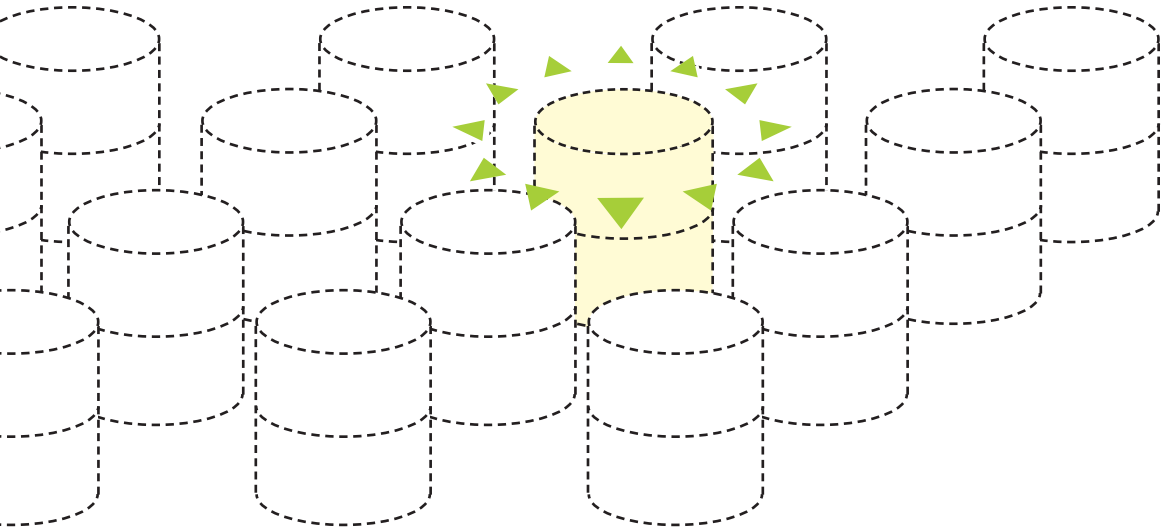
SQL Server INSIDER

Tips for SQL Server pros July 2007

Designing databases for performance



Take time to plan when developing your SQL Server databases for high performing applications from the start.

INSIDE

Designing databases for performance

BY BAYA PAVLIASHVILI

Developing databases might seem relatively simple. However, Microsoft SQL Server database architects and administrators have many choices to make if they want their applications to perform well right from the start.

Many attempt to use the "rapid development" approach by jumping straight to coding. Unfortunately, taking shortcuts often leads to performance problems that are considerably more difficult and costly to overcome once the application is deployed in a production environment.

Thinking through all the options during database development might increase the time it takes to complete your project, but it will pay off in performance dividends. Taking the time to choose appropriate data models, data types, indexing strategies and code modules can save a lot of time and effort tuning the database down the road.

Baya Pavliashvili

is a database consultant helping his customers develop highly available and scalable applications with SQL Server and Analysis Services. Throughout his career he has managed database administrator teams and databases of terabyte caliber. Baya's primary areas of expertise are performance tuning, replication and data warehousing. He can be reached at baya@bayasqlconsulting.com.

WHAT'S THE PURPOSE OF YOUR DATABASE?

The first step in database development is to consider the type of application you're about to build. There is a right tool for each job, and using the wrong tool can have unpleasant side effects. Keeping that in mind, **databases generally have one of the following three purposes:**

1 **ONLINE TRANSACTION PROCESSING (OLTP)** systems are used to collect the data. For example, you could build an OLTP system so your customers can order items from an online bookstore.

Data models for OLTP systems are highly normalized. Each data element is recorded only in one place. By eliminating or at least minimizing data duplication, you can make transactions faster and reduce the amount of storage space needed.

2 **DECISION SUPPORT SYSTEM (DSS)** applications are used to generate detailed, transaction-level reports about data collected by OLTP systems. Because OLTP systems attempt to minimize data duplication, creating reports against such systems requires joining numerous tables, and it generally does not perform well.

DSS applications mold the OLTP model into one that is much easier to report against by de-normalizing tables and introducing data duplication. Each DSS application is usually built to report on data collected by a single OLTP application.

Creating reports against OLTP systems usually makes for poor performance because it requires joining numerous tables. Instead, incorporate DSS applications to produce reports with data collected by OLTP systems.

3 **DATA WAREHOUSE SYSTEMS** are used to consolidate all enterprise data into a single data store and make it possible to correlate data from multiple OLTP systems. Data warehouses allow senior executives to examine a global picture of business performance and to detect areas of weakness for making strategic decisions.

Data warehouses are built using star or snowflake schema data models, which consist of fact and dimension tables. These models are best suited for pre-aggregating data and for examining the business from various perspectives. For example, an executive could review car rental history based on date, make or model of a vehicle, customer demographics or store location.

Although you can use an OLTP system for reporting or a data warehouse for collecting data, you're bound to encounter performance problems if you do so.

CHOOSE APPROPRIATE DATA TYPES

Here is a simple yet often neglected principle: Use the smallest and most appropriate data type for each column. For example, you could use about a dozen different data types for storing a person's age, such as BIGINT, INT, SMALLINT, TINYINT, DECIMAL, FLOAT, CHAR(3), VARCHAR(3) and others.

Because most of us won't live more than 255 years, it is safe to use the data type that requires the least overhead and still supports the necessary range of values — TINYINT, which supports whole numbers between 0 and 255. Similarly, in a data warehousing scenario, you could use a character identifier for a vehicle rental instance to join this dimension to the rental fact. However, for best performance, you should create a surrogate key with an integer data type to make joins between fact and dimension tables as fast as possible.

Use the appropriate data types to stay away from unnecessarily wide indexes when possible. SQL Server scans lean indexes faster than fat indexes.

Using inappropriate data types can also affect your indexing strategies. SQL Server indexes are limited to 900 characters. If you create a VARCHAR column with more than 900 characters, you cannot index such columns. Furthermore, if your columns are wider than necessary, the indexes created on such columns will also be wider than necessary. SQL Server can scan — or seek through — lean indexes faster than fat indexes.

DECIDE ON THE BEST INDEXING STRATEGIES

Appropriate indexes can make a world of difference in performance. SQL Server supports only two index types for most data types — clustered and non-clustered. SQL Server also supports full-text indexes and XML indexes, but those are relevant only for specific data types.

It is crucial to choose the appropriate column or set of columns for your clustered index. The reason is that the table's data is physically sorted by the values in the clustered index column or columns. You can create only a single clustered index on each table. Non-clustered indexes reference the clustered index keys (data values) to determine the physical location of each record.

It is recommended that you

create the clustered index on the columns that do not change often, are highly selective and have lean data types. In many cases, the clustered index on an identity column is the best choice because identity values are highly selective — each record has a unique identity value — and they are never updated and are built using SMALLINT, INT or BIGINT data types.

However, it is not uncommon to find a table that is never queried based on its identity column. If so, carefully consider how the data is commonly retrieved, perhaps by a foreign key to another table or by a character column. Often, you can improve performance by creating the clustered index on the column or set of columns that is most frequently used for retrieving the data.

Some developers like to create composite clustered indexes. These span several columns, a combination of which uniquely identifies each record. This might sound like a good practice because the identity column has no business meaning, whereas other columns — such as hire date, department name and vehicle identification number — definitely translate into something immediately known by application users. However, from a performance perspective, you should avoid composite clustered indexes.

Once again, the leaner the index, the faster SQL Server can scan or seek through it. You might find that for a small data set, composite indexes perform relatively well. But as the number of users grows, you're bound to encounter problems.

After you see performance benefits from building appropriate indexes, you might think your work is finished. But as data is added, modified and removed from tables, the respective indexes become fragmented. The higher the fragmentation, the less effective your indexes become. Now you'll need to implement a plan for removing fragmentation from your indexes to ensure they remain effective.

From a performance perspective, you should avoid composite clustered indexes.

With prior versions of SQL Server, removing fragmentation from large indexes (tables with many millions of rows) often required downtime. Fortunately, SQL Server 2005 supports online index rebuilds that make your life considerably easier. Keep in mind, however, that rebuilding indexes still requires system resources and space in a tempdb database. If possible, schedule index maintenance during periods of minimum user activity.

USE THE MOST APPROPRIATE CODE MODULES

SQL Server supports a number of ways to accomplish the same task, but not all of the methods are appropriate for each task. For example, just because SQL Server supports Common Language Runtime (CLR) stored procedures, doesn't mean you should implement simple data retrieval and update operations using CLR.

SQL Server supports the following classes of code modules:

- **VIEWS** are simply queries that are stored on the server. Views typically have no performance advantage over an ad-hoc set of SQL statements. A view with a nine-table join will not perform any better than the same nine-table join submitted through an ad-hoc SQL statement. Exceptions to this rule include indexed views and distributed partitioned views that, in some cases, can offer significant performance benefits.

- **SCALAR USER-DEFINED FUNCTIONS** allow you to extend the existing arsenal of built-in functions by performing some operation specific to your business. Be careful when using scalar UDFs against large tables. Scalar UDFs get compiled once per execution. Furthermore, if you call a scalar UDF against a column in a table with a million rows, the UDF code will be executed a million times. Scalar UDFs should be short and used only when the same functionality cannot be accomplished through built-in functions.

- **TABLE-VALUED USER-DEFINED FUNCTIONS** come in single statement and multi-statement. They both accept parameters and are sometimes used for implementing business logic. Single-statement UDFs are similar to views, but because they accept parameters, they're more powerful and often faster. If you have a join that involves a number of tables, you can often break it up into multiple statements, each performing a much smaller number of joins to speed up the query through a multi-statement UDF. Table-valued UDFs have many limitations. For instance, they disallow the use of nondeterministic built-in functions. Most business logic implemented in database code should be reserved for stored procedures.

- **TRANSACT-SQL STORED PROCEDURES** should be used for all data retrieval, addition, modification and removal of records from tables. Although you could implement complex business rules with stored procedures, those are best suited for middle-tier code. For two-tiered applications,

though, try implementing business rules with stored procedures rather than any other Transact-SQL code module.

- **EXTENDED STORED PROCEDURES (XPs)** are written using C++ language. XPs are necessary only for implementing functionality that is not directly available with SQL Server, such as querying registry or accessing file systems. SQL Server must use a relatively small area of memory called "MemToLeave" for XPs. If you use XPs heavily, your server is likely to experience memory pressures. You could adjust the amount memory left for "MemToLeave," but then you'll effectively reduce the amount of memory in the other area — buffer pool, or BPool — and can, therefore, hurt performance of Transact-SQL code modules. Modifying registry values and compressing files isn't part of SQL Server functionality and should normally be implemented elsewhere. SQL Server includes built-in extended stored procedures that are thoroughly tested and appropriate for use by database administrators. Developers should be strongly discouraged from using XPs.

- **CLR PROCEDURES** are new with SQL Server 2005 and allow implementing heavy computational logic that normally performs poorly in T-SQL routines. CLR procedures supplement T-SQL code — they don't replace it. As such, CLR procedures should be used sparingly. Don't forget that T-SQL has been around for more than 15 years, and most people have much more experience with tuning T-SQL code rather than CLR code.

PICKING HIGH-AVAILABILITY OPTIONS

With SQL Server, several options exist for ensuring the continuous availability of your applications in the event of hardware failure. Some options can provide better performance by using resources on multiple servers to separate transaction processing and reporting activities. Just as you did when designing data models, think of high-availability architecture at project inception rather than after deployment.

Replication moves transactions and data from the primary server to other servers. Separate transaction processing and reporting functionality by replicating transactions from an OLTP to DSS server. You could physically partition your data across multiple servers and direct a subset of users to each OLTP server. Then transactions from OLTP servers could be combined on the reporting server or servers.

Be sure to replicate only those data and trans-

actions that are necessary for each application. Transactional databases can have many tables and columns that are never used for reporting.

Many DBAs fall into the trap of thinking it's easier to replicate all data as often as possible. Although that might be easier for developers, it's certainly not easier on your servers or your network. Know your limits. Consider network bandwidth, your available disk space, the number and type of scheduled jobs and their effect on the replicated transactions.

Log shipping can back up transaction logs on the primary server and restore these backups to secondary servers. Log shipping can help performance by offloading reporting or ad-hoc querying functionality to secondary servers, but databases on a secondary server will not be available while you restore transaction log backups. Log shipping is easy to implement as long as you have the right hardware. Don't try using your old desktop for log shipping. The standby server should be as powerful as your primary server — after all, if the primary server fails, your stand-by server must become the primary server.

Contrary to popular belief, clustering offers no performance benefit. It is intended solely for automatically bringing the secondary server online when your primary server fails.

Database mirroring is the newest kid on the high-availability block. It is somewhat similar to log shipping because the same transactions are applied to two identical databases on different servers. Unlike log shipping, mirroring offers automatic failover. Unlike clustering, you must set up separate mirrors for each database.

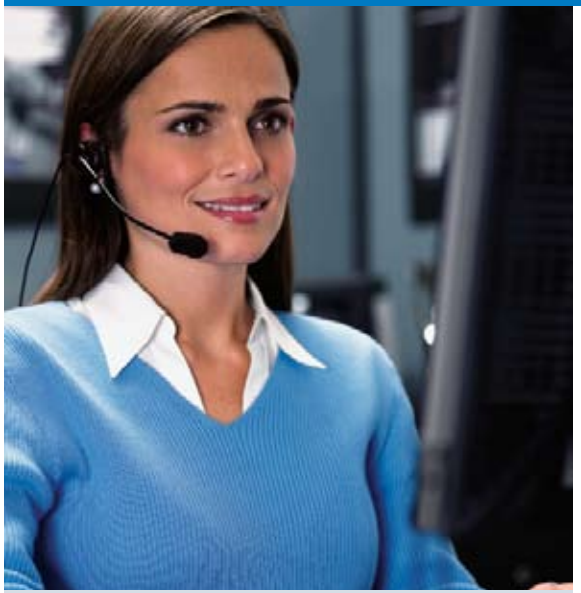
You can use mirroring to separate transactional and read-only activity. Databases on the secondary server must remain offline until the failover occurs. However, you can take snapshots of the mirror databases on a secondary server and make them available for reporting purposes. Once you have a database snapshot, you could also use integration services to transport the data to a data warehouse and correlate it with data from other transactional systems.

Many options must be carefully considered when developing SQL Server applications. To ensure success, take the time to make the best choices during development rather than after deployment.

Choosing the most appropriate data model, data types, indexing strategies and code modules are essential steps for developing an application that satisfies your needs. Consider some of the high-availability options for separating transactional and reporting activities to keep your application performing at its peak.*

Ticket to Growth

DELL
ServiceU



"The Dell and SQL Server 2005 database mirroring solution allows us to eliminate risks from disasters and assure our customers that they will always have the same level of service that they rely on."

— **David P. Smith**,
Chief Technology Officer,
ServiceU

ServiceU gains 99.999% uptime, more than $900,000 in benefits, and 595% return on investment thanks to Dell PowerEdge Servers and Microsoft SQL Server 2005

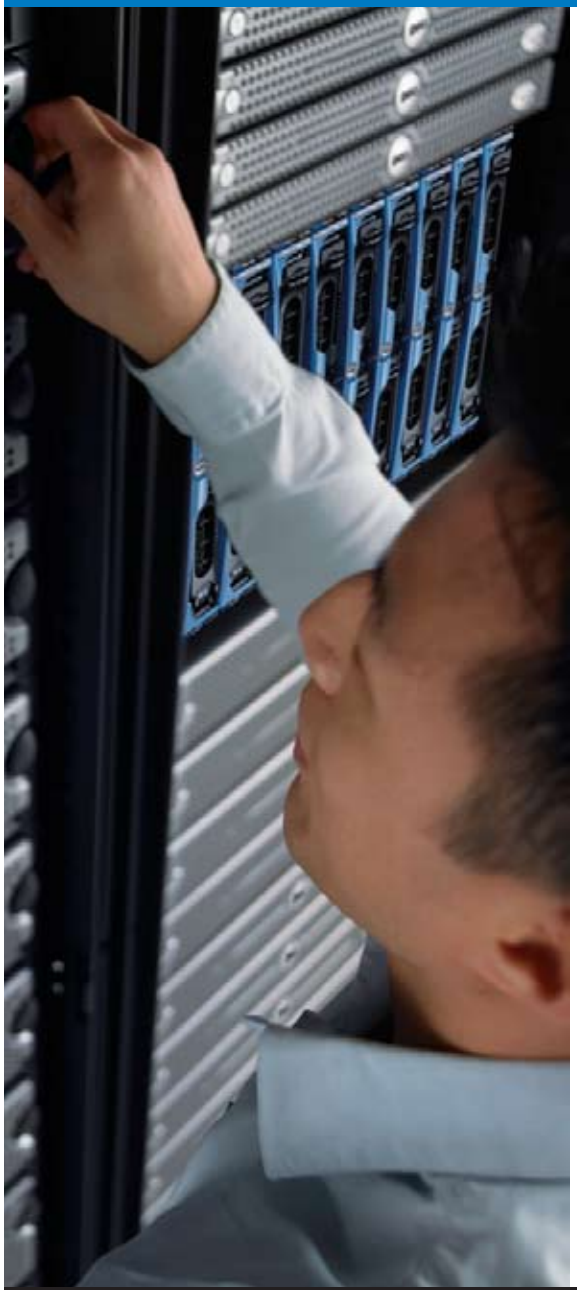
Memphis-based ServiceU, in business since 1997, is an on-demand service provider which delivers Web-based software for event management. It serves more than 1,000 organizations worldwide, ranging from Fortune 500 companies to public universities and small, nonprofit institutions. The company's software has been used to handle scheduling for more than 12 million events.

The always-on availability of ServiceU's databases and Web-based software is the key to the company's success. "The entirety of our business is done online," explains ServiceU Chief Technology Officer, David P. Smith. "It accounts for all of our revenue, so uptime is crucial to us."

SERVICEU™

Ticket to Growth

ServiceU



ServiceU faces some unique challenges in maintaining such high levels of availability. Payment Card Industry (PCI) standards mandate that level-one PCI service providers like ServiceU meet a rigorous set of disaster recovery (DR) requirements. In addition, the company's Memphis offices sit atop an active fault. Says Smith, "according to seismologists, we are within 40 years of another major earthquake, so we have to be prepared."

Faced with the requirement of always-on availability, quick disaster recovery (DR), and compliance with PCI standards, ServiceU realized that it needed to improve on a DR plan that called for physical tapes to be sent by helicopter between facilities. After a detailed analysis, ServiceU turned to Dell and Microsoft SQL Server 2005 with Database Mirroring, to mirror the databases from its main facility in Memphis to its disaster recovery facility in Atlanta.

"Database mirroring allows us to have the Atlanta facility functional, compliant, and ready to use at a moment's notice," says Smith. "Additionally, the Dell and SQL Server 2005 database mirroring solution allows us to eliminate risks from natural disasters, such as earthquakes, and assure our customers that they will always have the same level of service that they rely on."

Thanks to the mirroring solution, ServiceU can guarantee high availability that will help it expand into new markets. ServiceU expects to realize a cumulative, projected, three-year net benefit of US$908,985 which will result in an ROI of 595 percent and a payback period of five months.

To view the entire story, click here.

SQLServer Insider BACKUP AND RECOVERY

Speed up SQL Server backup and restore time

BY GREG ROBIDOUX

PERHAPS THE MOST important task that occurs on every SQL Server system is running backups and restores. Backup copies of your database give you the security of having a complete copy to fall back on if any issues surface with your production database. In most cases the restore process is done for non-production critical means such as refreshing development\test environments or refreshing a reporting environment. But in the most critical mode, you are restoring these backup copies to replace or fix a production environment.

Based on the importance of creating backups and the critical need for restoring backups to correct a production problem, time is of the essence. Backups are an online operation, but they do use system resources. Restores, however, require exclusive access to the database, so in a failure state this is an even more critical task.

GREG ROBIDOUX

is the president and founder of Edgewood Solutions LLC, a technology services company delivering professional services and product solutions for Microsoft SQL Server. He has authored numerous articles and has delivered presentations at regional SQL Server users groups and national SQL Server events. Robidoux also serves as the SearchSQLServer.com Backup and Recovery expert.

In light of the time factor to complete these tasks, there are several things that can be done on both the backup side and the restore side to improve the speed of these operations.

Hardware backup and restore time is affected by your hardware as well as the configuration of that hardware. From a hardware perspective, here are a few things you should consider doing to improve performance:

- Spread the disk I/O. By using as many drives as possible, you can make sure the disk I/O is not the bottleneck. Also make sure you are not using the same drives for both reading and writing.
- Employ the newest hardware technology.
- Use the fastest RAID configurations: RAID 0, RAID1, RAID10 and then RAID 5.
- Use the fastest drives.
- Use the fastest controllers, and separate disk activity onto different controllers or different channels.
- Use locally attached disks instead of backing up across the network.
- Backup to disk and then archive to tape.
- Use SAN technologies for snapshot and split mirror backups.
- If you do need to back up to other machines, use the fastest network cards and switches possible. And if you can segment this traffic from normal traffic, you'll reduce the network I/O bottleneck.

NATIVE BACKUPS

Another area that affects the time it takes to complete backups is when and how the backups are run.

- Execute during low server usage times.
- Don't run all of your backups at the same time.
- Don't run batch processing at the same time as large backups.
- Use backup options to write to multiple files. This will spread your I/O as well as increase the number of threads.
- Use a combination of back-up techniques: full, differential and log.

NATIVE RESTORES

From a restore perspective, most of the items mentioned above apply to restores as well. Here are a few additional tips:

- Use a staging area so backups are partially restored, instead of having to restore all backups at the same time.
- Use a restore process such as log shipping.
- Use other technologies

besides backup and restore for data recovery, such as clustering, replication, CDP, etc.

THIRD-PARTY SOFTWARE

One key time-saving function is using a backup compression tool built specifically for SQL Server. There are several on the market, and these tools provide the biggest gains with the least amount of effort. Use compression software such as Idera's SQLsafe, Quest Software Inc.'s LiteSpeed, and Red-Gate's SQL Backup.

Based on a test by Idera and various hardware vendors, Idera was able to achieve backup rates of 4.5 terabytes per hour and restore rates in excess of 2.3 terabytes per hour by using SQLsafe. Take a look at this link for additional info on setting new performance records. This is probably the extreme for most SQL environments, both from the cost to configure the hardware to the need to back up a 4.5 terabyte database. But the fact is that it is possible by configuring the total solution in both hardware and software. *

You must be prepared to efficiently manage growing SQL Server databases.

SUMMARY There are several things you can do to increase the throughput of your backup and restore processing. Some of them are pretty simple fixes while others require reconfiguring your hardware, purchasing new hardware or purchasing tools that can help increase the speed.

Based on the speed of the Idera tests to achieve 4.5 terabytes in an hour, using a third-party backup compression tool seems like the simplest and easiest way to go. I don't think there are many databases that fall into this realm of database size, so, based on the test, most full backups could be completed in under an hour. In a report of the largest SQL Server databases, you can see there are still not many databases that cross the terabyte threshold. From the report, the number almost tripled in two years, and I am sure the number will triple again — if not by even more in the next two years.

By using a combination of all these options, you will be able to achieve faster backup and restore times; but like most things, there will always be some kind of limitation.

SQLServer Insider DEVELOPMENT

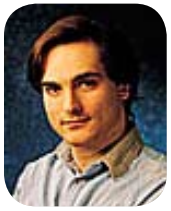
T-SQL in SSIS: The power and the weaknesses

BY SERDAR YEGULALP

SQL SERVER INTEGRATION services, or SSIS, provides a number of different mechanisms to create and pull data from a data source. One is the ExecuteSQL task, which lets you use a T-SQL statement, much as you might pass a bit of T-SQL from a front-end application to SQL Server on the back end. The T-SQL in question can be a full statement or a reference to an existing stored procedure.

T-SQL, however, is not the only way to get data into SSIS. There's also the Data Flow task, which leads people to ask: Why use T-SQL in SSIS? Why not just use data flows? What's the advantage to T-SQL, if any? I went searching for a response to those questions and found a number of very well-explained answers at Jamie Thomson's SSIS Junkie blog at Conchango. I'll rephrase a few of them here, along with some of my own commentary gleaned from personal experience.

SERDAR YEGULALP *has been writing about Windows and related technologies for over 10 years and is a regular contributor to various sections of TechTarget, as well as other publications. He hosts the Web site WindowsInsider.com, where he posts regularly about Windows and has an ongoing feature guide to Vista for emigrants from Windows XP.*



- **USING AN EXISTING PIECE OF T-SQL OR STORED PROCEDURE IS A TIME-SAVER,** rather than recreating the whole thing in a data flow. If you have an existing stored procedure that took a good deal of sweat and concentration to build, there's no point in tearing it down and rebuilding it; you can simply use it as is. There are some slight differences in the syntax for evoking a stored procedure (mostly in the way parameters are passed), but very little that would prevent existing stored procedures from being reused in ExecuteSQL. There are, however, a couple of things that might cause problems.

- **YOUR SQL SERVER PROGRAMMERS DON'T HAVE TO LEARN MUCH OF ANYTHING NEW** to make their code work in SSIS. They can write T-SQL code, and it can simply be dropped into SSIS workflows, with relatively little modification. If you're taking over someone else's work and migrating it progressively into SSIS, you may be more inclined to reuse the existing work instead of trying to re-engineer something from scratch.

- **THE MAJORITY OF WORK IS DONE WHEREVER YOUR DATABASE IS,** instead of where SSIS itself is running (which may not be the same machine). This may not be an advantage, but it is a behavior worth noting. It can work for you or against you.

There's no learning curve for SQL Server developers to experience in order to make their T-SQL code work in SSIS. With little modification, T-SQL code can be dropped into SSIS workflows.

- **T-SQL USES TRANSACTIONS WITHIN THE DATABASE ITSELF** instead of across MSDTC (as SSIS does).

Now, here are some reasons why T-SQL in a SQL Server Integration Services data flow may not be a good idea:

- **SOME STORED PROCEDURES DO NOT EXPOSE DATA IN A PREVIEW.** The simple reason is that it's not always possible to predict what columns a given stored procedure will produce when it's run. That said, some stored procedures can't be used in place of a data flow if you need a predictable "output contract" for that data flow. However, you can use a view to expose a data flow preview. The view is bound

tightly to the schema(s) it works with and doesn't change, unless you explicitly recast it (i.e., you edit it). So if you have an existing view that you want to use as a data source in SSIS, you'll have previewing as a bonus feature.

- **SSIS DATA FLOWS ARE BETTER FOR MORE COMPLEX, MULTI-STEP OPERATIONS, SUCH AS JOBS THAT REQUIRE:**

- a lot of programmatic work
- aggregation from different data sources or type aggregation from different data sources or types
- structured exception handling
- additional transformation
- jobs that can't be accessed from SQL itself

These are things that are not always done elegantly or efficiently in the context of T-SQL:

- **SSIS DATA FLOWS ARE SELF-DOCUMENTING,** which is another possible advantage of SSIS data flows over T-SQL. But, it's true only to a point. If you don't have any understanding of the schema being accessed, you're just as likely to be in the dark. On the other hand, stored procedures and T-SQL in general are not self-documenting at all — unless whoever wrote them took the time to document each in detail (and how often does that happen?).

From all of this, we derive two simple rules about using T-SQL versus data flow:

1. **T-SQL IS BEST SUITED** for operations in which you're simply gleaning data from an existing set, where the process for doing so is not likely to change soon and doesn't involve a lot of additional transformation.

2. **THE DATA FLOW TASK** works best when you're devising an entirely new data transformation — something where existing T-SQL (or T-SQL itself) won't comfortably do the job. You can't really gauge this unless you've worked a great deal with T-SQL and know its limitations. You need to know at least as much about T-SQL as you do about SSIS in order to take advantage of both. *

Additional Resources from Dell

- **Dell's SQL Server 2005 Tested & Validated Configurations**
  http://www.dell.com/content/topics/global.aspx/sitelets/solutions/software/db/microsoft_sql_2005_se?c=us&cs=555&l=en&s=biz

- **Dell's SQL Server 2005 Reference Architecture**
  http://www.dell.com/downloads/global/solutions/sql_server_2005_reference_architecture_w2k3_std.pdf?c=us&cs=555&l=en&s=biz

- **Dell SQL Server 2005 Advisor Tool**
  http://www.dell.com/content/topics/global.aspx/tools/advisors/sql_advisor?c=us&cs=555&l=en&s=biz

- **Dell Tech Center Wiki**
  http://www.delltechcenter.com/

- **Case Study: University of Mary Hardin - Baylor**
  http://www.dell.com/downloads/global/casestudies/456_UMHB_9.pdf

- **Optimizing Microsoft SQL Server 2005 Environments with EMC Assessments and Quest Software**
  http://www.dell.com/downloads/global/power/ps4q06-20070103-EMC-Quest.pdf

SQL Server INSIDER

is brought to you by SearchSQLServer.com. The articles "Speed up SQL Server backup and restore time" and "T-SQL in SSIS: The power and the weaknesses" originally appeared on SearchSQLServer.com.

EDITORS
Heidi Sweeney
Christine Casatelli

COPY EDITOR
Martha Moore

DESIGN
Ronn Campisi
www.ronncampisi.com
Heather Luipold