



The Best of SQLServerCentral.com Vol 5.



The Best of SQLServerCentral.com – Vol. 5

Aaron Ingold, Alex Grinberg, Amin Sobati, Amit Lohia, Andy Warren, Anthony Bressi,

Arvinder Khosla, Asif Sayed, Bilal Khawaja, Bimal Fernando

Boris Baliner, Brandi Tarvin, Chad Miller, Claudi Rego,

Craig Hatley, David McKinney, David Poole

Dinesh Asanka, Frédéric Brouard, Grant Fritchey, Jack Hummer,

Jacob Sebastian, Jambu Krishnamurthy, Janet Wong, Jason Selburg, Jay Dave, Jeffrey Yang,

Joe Doherty, Johan Bijmens, Kathi Kellenberger, Ken Johnson, Keren Ramot

Kumar Part, Lee Everest, Leo Peysakhovich, Mark Balasundram, Michael Ahmadi, Michael Coles

Michael Lato, Mike Dillon, Paul Mu, Peter He, Peter Larsson

Raj Vasant, Richard Gardner, Rob Farley, Robert Cary, Robert Pearl, Roy Carlson

S. Srivathsani, Sachin Samuel, Santhi Indukuri, Simon Munro, Simon Sabin, Steven Hirsch,

Stephen Laplante, Stephen Lasham, Sushila Iyer, Thomas LaRock, Tim Mitchell

Tim Opry, U.K. Padmaja, Vincent Rainardi, Wayne Fillis, Yaniv Mor

Yaroslav Pentsarskyy, Yoel Martinez, Yousef Ekhtiari

The Best of SQLServerCentral.com – Vol. 5

Simple Talk Publishing

Newnham House
Cambridge Business Park
Cambridge, CB4 0WZ

United Kingdom

Copyright Notice

Copyright 2007 by Simple Talk Publishing. All rights reserved. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced under the Copyright Act of 1976. No part of this publication may be reproduced in any form or by any means or by a database retrieval system without the prior written consent of Simple Talk Publishing. The publication is intended for the audience of the purchaser of the book. This publication cannot be reproduced for the use of any other person other than the purchaser. Authors of the material contained in this book retain copyright to their respective works.

Disclaimer

Simple-Talk Publishing, SQLServerCentral.com, and the authors of the articles contained in this book are not liable for any problems resulting from the use of techniques, source code, or compiled executables referenced in this book. Users should review all procedures carefully, test first on a non-production server, and always have good backup before using on a production server.

Trademarks

Microsoft, SQL Server, Windows, and Visual Basic are registered trademarks of Microsoft Corporation, Inc. Oracle is a trademark of Oracle Corporation.

Editors

Steve Jones, Tony Davis, Anna Larjomaa

Cover Art

Ross King

Table of Contents

Table of Contents	3
Introduction.....	5
Administration.....	6
SQL 2000 to SQL 2005: Where have all the old features gone?	7
Stress Testing Your SQL Server Databases - A Systematic Approach	9
Intro to Database Mail in SQL 2005	12
The OUPUT Command	16
Running a Query Using a Text File for Input	18
Starting SQL Server in Minimal Configuration.....	20
How SQL Server Chooses the Type of Join.....	22
Indexes and Fragmentation in SQL Server 2000 Part 1.....	24
Indexing in SQL Server 2005	30
Database Snapshots	38
Database Snapshots in SQL Server 2005.....	41
Customizable Error Log Scanning.....	46
A Guide to Application Memory Tuning	47
Eliminating Tape.....	49
Full Text Search Follies	52
SQL Stored Procedure to Log Updates, Independent of Database Structure.....	55
Identifying Unused Objects in a Database	62
Upgrading a Database SQL 2000 to SQL 2005	68
Maximum Row Size in SQL Server 2005	75
Dynamic Management Views and Functions in SQL Server 2005.....	76
Backing Up a Database with SMO	81
Document Your Database	89
A Transactional Replication Primer	90
T-SQL	105
Keyword Searching in SQL Server.....	106
Recursive Queries in SQL: 1999 and SQL Server 2005	113
Serializing Procedure Calls Without Unnecessary Blocking.....	134
The Truth Table	137
Introduction to Bitmasking in SQL Server 2005.....	142
Self Eliminated Parameters	151
Finding Primes.....	152
CTE Performance.....	155
The Effect of NOLOCK on Performance	160
Converting Hexadecimal String Values to Alpha (ASCII) Strings.....	163
A Refresher on Joins	166
Using CLR integration to compress BLOBs/CLOBs in SQL Server 2005	172
When To Use Cursors	175
Everybody Reports to Somebody	177
Not In v Not Equal	179
Full Control Over a Randomly Generated Password	183
Performance Effects of NOCOUNT	187
Passing a Table to A Stored Procedure	191
New Column Updates.....	195
The T-SQL Quiz	200

Practical Methods: Naming Conventions.....	205
Large Object Data	208
Row-By-Row Processing Without Cursor	210
Security.....	214
SQL 2005 Symmetric Encryption	215
Ownership Chaining	222
Preventing Identity Theft Using SQL Server.....	223
ETL and BI.....	226
A Common Architecture for Loading Data	227
Overview of SSIS	232
SSIS - Transfer SQL Server Objects Debugged	240
SSIS Is Not Just for SQL Server	244
SSIS Programming.....	247
Loading a 24x7 Data Warehouse	250
More Problems with Data Warehousing	253
Storage Modes in SSAS 2005.....	256
Dynamic Connection Strings in Reporting Services 2005	258
Populating Fact Tables.....	260
Reporting Services 2005 101 with a Smart Client.....	272
Data Driven Subscriptions for Reporting Services (2000 and 2005)	283
Service Broker	285
Adventures With Service Broker.....	286
Building a Distributed Service Broker Application	292
XML	305
The SQL Server 2005 XML Temptress	306
I've Got the XML - Now What?	312
XML Argument Protocols for SQL 2005 Stored Procedures	317
Misc.	323
Windows Utilities for the SQL Server DBA	324
From DBA to DBAA	326
What's a Good Manager.....	328
Mind Your Manners	332

Introduction

Welcome to The Best of SQLServerCentral.com – Vol. 5!

This is our fifth year of producing a book that covers the best articles of the past year. With SQL Server 2005 now on it's second full year of deployment, the amount of knowledge, tips, and techniques for working with this SQL Server paradigm has greatly increased. This year we've substantially enhanced the XML section and added a Service Broker one as well.

We wanted to give you an off-line resource that you can take with you wherever you may need it - most likely at your bedside. To our authors, this book is our way of saying thank you for the effort you put into producing great content for the community and also a nice chance to see your names in print!

We would like to thank everyone for their support on the website and especially in the community. Your visits to the site, clicking through to advertisers, purchasing products, registering for PASS, all help us continue this community and provide you with a valuable resource that hopefully helps you learn, perform better at your job, and grow your career.

We'd like to encourage all of you to submit an article in the next year, whether you are working with SQL Server 2005 or SQL Server 2000! This is a community and we aren't only looking for the "gurus" to contribute. We love hearing about the real world you all live in and deal with on a daily basis. In each "Best Of...", we try to include at least one article from each author and we will send you a couple copies of the book. It's a useful addition to your bookshelf and makes a great Mother's Day present!

Once again, thanks so much for your support and we look forward to 2008.

Steve Jones

Tony Davis

Administration

This is what we do: administer servers and databases. Everyone has their own set of tricks, tips and scripts tailored to the quirks of their own systems and this past year, with many people upgrading from SQL Server 2000 to SQL Server 2005, was one where many of these tools and techniques had to change.

Here's a selection of articles to help you manage your instances a bit better and perhaps learn about some parts of SQL Server that you don't get to deal with in your own systems.

We tackle upgrades, stress testing, mail, and more. As we compile this 5th edition, Microsoft SQL Server is a very mature product that is mostly being limited by its architecture as hardware continues to grow in power, loads increase, and many different stresses occur. Nothing earth-shattering here, just some good information that might help you save the day.

SQL 2000 to SQL 2005: Where have all the old features gone?	7
Stress Testing Your SQL Server Databases - A Systematic Approach	9
Intro to Database Mail in SQL 2005	12
The OUPUT Command	16
Running a Query Using a Text File for Input	18
Starting SQL Server in Minimal Configuration	20
How SQL Server Chooses the Type of Join	22
Indexes and Fragmentation in SQL Server 2000 Part 1	24
Indexing in SQL Server 2005	30
Database Snapshots	38
Database Snapshots in SQL Server 2005	41
Customizable Error Log Scanning	46
A Guide to Application Memory Tuning	47
Eliminating Tape	49
Full Text Search Follies	52
SQL Stored Procedure to Log Updates, Independent of Database Structure	55
Identifying Unused Objects in a Database	62
Upgrading a Database SQL 2000 to SQL 2005	68
Maximum Row Size in SQL Server 2005	75
Dynamic Management Views and Functions in SQL Server 2005	76
Backing Up a Database with SMO	81
Document Your Database	89
A Transactional Replication Primer	90

SQL 2000 to SQL 2005: Where have all the old features gone?

By Boris Baliner

Introduction

As more DBAs across the planet begin using SQL 2005 Tools, but still manage SQL 2000 servers with them, I suspect there will be lots of muffled moaning and wondering where have all the good old features gone. Although Management Studio has some very nice long-awaited features, some of the good old stuff just isn't there.

Where are my tried and true tools, such as taskpad? Where's the IF EXISTS DROP option when I script out the stored procedures? Could someone pinch me and tell me this is just a bad dream? The aspirin industry will profit enormously from that sort of thing.

To name a few good old pals that have all but disappeared into obscurity:

- Taskpad
- Ability to quickly script permissions on stored procedures
- Ability to quickly see disk space in database properties
- Time of creation of stored procedures

Sure, if you're connecting to an instance of SQL Server 2005 with Management Studio you get colorful reports and plethora of professional-looking graphs at your disposal, but what about the majority of us that still did not migrate our servers to SQL 2005, but already upgraded the tools?

The good news is this will tend to convert many GUI DBAs into hardened command-line pros, improve they're typing skills, etc. In the next section I will show how to still take advantage of the old tools functionality.

Taskpad functionality

I don't know about you all, but I really like the Taskpad and use it all the time. I am used to it like to an old slipper; it fits my needs. And even if it did throw a vague error now and then I forgive it now that it's done....forever. But how can we get its functionality back?

The General tab in Database section is now in database properties under the same heading.

Maintenance section-like information can be found by querying the backupset table in msdb:

```
select max(backup_start_date) from backupset  
  
where database_name = 'my_db'
```

Note: Database options, Number of Users, Date Created and Owner can still be found in database properties in the SQL 2005 tools.

Space allocated section info can be found by running this T-SQL:

```
select * from sysfiles
```

or if you just need to find the space used by you log, execute:

```
DBCC SQLPERF (LOGSPACE)
```


Table Info tab

I don't use this one very often, but you can get similar functionality by running:

```
Exec sp_spaceused 'your_table_name'
```

To script multiple stored procedures including permissions:

Right-click the database->Tasks->Generate Scripts, pick your database. Set Include object Level Permissions to True. Note: If you set the Include if NOT EXISTS option to true, the script will not create the stored procedure if it already exists on target database.

Click Next, and select Stored Procedures only. Next select which procs you want to script, review you final options and click Finish.

Unfortunately, if you want to drop/recreate the procedures if they exist on the target server, you will need to manually include the following script in the beginning of each procedure:

```
IF EXISTS (SELECT name FROM sysobjects
           WHERE name = 'your_proc_name' AND type = 'P')
DROP PROCEDURE 'your_proc_name'

GO
```

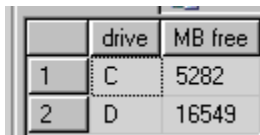
This one is truly beyond me, for reasons known only to Microsoft and the CEO of Bayer (or whoever is the biggest headache drug company these days) this option was excluded from final SQL 2005 RTM.

Check disk space

If you're like I am, you're used to clicking on database properties, and the ellipsis in order to see the free disk space on the server. In SQL Server 2005 you can get this in the report, but until then you can run undocumented extended stored procedure:

```
exec xp_fixeddrives
```

The result would look something like this:



	drive	MB free
1	C	5282
2	D	16549

Time of creation of stored procedures:

For some reason the time part of the Create Date column in the Summary tab of SQL 2005 is depreciated. Why? I guess someone thought DBAs don't need it any longer (rolling eyes). The good news is you can still get this information by querying the sysobjects table, like:

```
Select crdate as DateCreated
From dbo.sysobjects
```

```
where name = 'your_proc_name'
```

Note: make sure NOT to specify the owner, as in `dbo.your_proc_name`, but simply the procedure name. The result is:

	DateCreated
1	2004-12-14 14:51:52.803

Summary

I've shown you here how to get by with available SQL 2005 Tools until you upgrade your servers to SQL 2005 Server edition. People get used to their favorite ways to get the job done, and sometimes get "surprised" when their tools get taken away, and now they have to hammer the nail with the handle of a screwdriver. Hopefully the ways to attain old functionality will increase productivity, and hopefully the tools will continue to improve.

Stress Testing Your SQL Server Databases - A Systematic Approach

By Anthony Bressi

Stress Testing your SQL Server environments is a practice that can yield great benefits. Stress Testing can be used for performance tuning, to uncover bottlenecks, and for capacity planning among other things. In this article we will use the terms 'Load Testing' and 'Stress Testing' interchangeably, even though by definition they are different. So for the sake of this article 'Load Testing' or 'Stress Testing' is simply running a large number of statements/transactions against a SQL Server database from various connections or threads - all to give the database server a good workout. Lastly, this article is not about HOW one physically throws a large number of statements at a server to stress test it, there are 3rd party tools on the market and available code on the web to help you do that. Instead we'll discuss what variables are needed and what questions need to be answered to ensure a successful and productive stress test environment.

WHY You Stress Test Drives HOW You Stress Test

The reasons for Stress Testing vary. You may be trying to determine the apparent cause of poor performance on a production server, or trying to plan for an increase of users, or trying to workout a new application that is not yet "live" to determine if it will be able to handle a production load. The reasons vary, but your specific reason will drive your approach to stress testing. For instance, your approach will vary greatly between a pre-production test and one for a live server.

Production Environments

If it is a performance problem in a production environment that you're dealing with, you can first try to determine your problem using Performance Counters sampled throughout your busy times of day (if possible). I realize that this is a Stress Testing article and I'm starting with a non-Stress Test approach, but in the real-world it is easier to start with performance monitoring before trying to gain approval for stress testing a production server. By comparing the gathered performance counter values with generally accepted thresholds you may be able to figure out your problem without running a Stress Test. We will cover some Microsoft listed thresholds later in the article. You may want to perform your monitoring several times during the week, as the load on Monday might be different from Wednesday's load.

One more note about stand-alone monitoring before we move on to our main focus: at times it can be beneficial to gather samples over a 24 hour period or even just at night if day-time sampling is forbidden. Sometimes your late-night SQL Server Jobs and back-ups can provide some insight. For instance, you may be running back-ups from across the network and notice that your Network Interfaces have less than favorable throughput.

If you can't take samples during peak times in your production environment, you will be best served by running off-hour stress tests using the tool of your choice to create the load. In this case you are still working out the hardware/software configurations on the server in question and capturing performance counter values that can lead you to the source of the problem if one exists. If the captured performance counter values do not lead to insight, then concentrating on the execution times of application stored procedures and scripts may help uncover the problem - perhaps dead-locking, or simply bad indexing. If needed, you can open up SQL Profiler for in-depth tracing information while your stress test is running.

If you're looking to see how your production environment will scale with an increase in users, then stress testing is the way to go. You can dial-up virtual users to hammer your server and monitor its response. In this scenario, as well as all others, you want to gather Performance Monitor samples and execution times. This is also a great way to validate the need for new hardware, walking into your boss' office with a stress test report shows that you have done your homework.

Non-Production Environments

For pre-production tests or any tests that won't be held in a live production environment, Stress Testing your database can be invaluable. You will want to run your tests in a production-like environment. Simulating that can be difficult, but with automated stress testing packages you can attempt to mimic the amount of load that your servers will face and scale that load up to really make your server work. Commercial Stress Testing applications like Agilist's SQL Stress Test or Quest's Benchmark Factory enable you to create a large number of virtual users that will execute the SQL commands of your choice. A good Stress Testing package will create separate threads for each virtual user and will avoid connection pooling. In a non-production environment you typically have a lot of freedom, nevertheless, there are still questions that you will want to think about for testing in both Production and Non-Production environments.

Pre-Stress Test Questions

Here are some questions to ask prior to running your tests:

- How many virtual users will you want to test with? If this will be the back-end of a web or software application, then what is the maximum number of concurrent users that will be using it?
- What are the main SQL statements and procedures that you will execute during the test? If you are testing the procedures called by an application then answering this question should not be difficult. Some third party Stress Testing Tools like Agilist's, will even record the sql statements being executed by users while they use the web or software front-end.
- How long should each statement or procedure take to execute? I.e. what would be an acceptable return time?
- If this is a pre-production database, does it contain a fair amount of data so that your tests are working in a production-like environment?
- What location do you want to run the Stress Test and monitoring from? If monitoring from across the network you will incur some DCOM hit, if monitoring locally on the server you will eat up marginal resources.

A Good Stress Test is Measurable

Throwing a hail storm of statements at your SQL Server is only part of a productive Stress Test. First and foremost, the test must be measurable. So while giving our server a workout we must gather performance related statistics. Usually we'll gather Performance Counters and if possible the actual execution times of the statements themselves. You can gather Performance Counter statistics using Microsoft's Performance Monitor (PerfMon) or the 3rd party tool of your choice.

For these statistics to mean anything we must be able to measure them against something, usually these "somethings" are baselines that we have created some time in the past or accepted thresholds. Baselines are simply previously saved results that we can compare against. We also can compare our tests against thresholds listed by Microsoft and industry professionals as mentioned previously. You can create a baseline the first time you run a test, simply save your results and now you magically have a measurable baseline that can be recalled for future comparisons. As time goes by you will want to periodically create baselines for comparison purposes. Baselines are also great for trending and determining growth rates.

Which Performance Counters To Use?

When choosing Performance Counters we don't just want to know how our SQL Server software is performing, we also want to know how our hardware and network is performing. The best list of core counters that I have come across, and have been using for years, come from an article by Brad McGehee entitled "How to Perform a SQL Server Performance Audit ". I experiment with some other counters but always use the ones mentioned in the article as the foundation. The counters are:

- Memory: Pages/sec
- Memory: Available Bytes
- Network Interface: Bytes Total/Sec
- Physical Disk: % Disk time
- Physical Disk: Avg. Disk Queue Length
- Processor: % Processor Time
- System: Processor Queue Length
- SQL Server Buffer: Buffer Cache Hit Ratio
- SQL Server General: User Connections

Which Thresholds To Use?

After monitoring your server, you will need to measure your captured counter values against trusted thresholds so that you know whether or not a problem may be at hand. I compiled a list of thresholds from Microsoft resources such as TechNet and others that match the list of Counters above. Below are the MS values along with some comments. When comparing your values to these you should always ask yourself if the value that you have collected was sustained over a period of time or if was just a spike - sustained values are obviously much more appropriate for comparison.

- Memory: Pages/sec: If counter value is consistently > 5 you may likely have a memory issue.
- Available Bytes: Values < 10 MB should raise a serious red flag for you.
- Network Interface: Bytes Total/sec: Microsoft simply advises that if you notice this value dropping it may indicate network problems. By rule of thumb you can use a value of half of the available network interface bandwidth as being acceptable. So for a 100 MBS network adaptor, the value of the Bytes Total/sec performance counter can be 50 MBS or greater.
- Physical Disk: Avg Disk Queue Length: You will need to calculate this value based on the number of physical drives. It is simply the monitor value / # of disks. A value greater than 2 might indicate an I/O bottleneck. The number of waiting I/O requests should be sustained at no more than 1.5 to 2 times the number of spindles making up the physical disk.
- Physical Disk: % Disk time: The recommended numbers for this seem to vary, if the value is greater than 50% you should be concerned and investigate more. If the value is sustained above 80% it is a serious problem and you may have a memory leak or I/O issue at hand.
- Processor: % Processor Time: Values in excess of 80% processor time per CPU are generally deemed to be a bottleneck.
- System: Processor Queue Length: A sustained queue length > 2 (per processor), generally indicates a processor bottleneck. For instance, if you have 2 processors then a value of 4 or less is usually acceptable.
- SQL Server Buffer: Buffer Cache Hit Ratio: A rate of 90 percent or higher is OK. The closer to 100% the better. Less than 85% indicates a problem.

- SQL Server General: User Connections: This one varies of course. It should be tracked though so that you can determine what is "normal" for your environment. With this you can spot trends that may signal growing demand, or you may use it to explain spikes in other counters.

Additional Threshold and Monitoring Resources

There are many good articles that deal with monitoring activity and thresholds. Listed below are some worthwhile resources:

- "Performance Monitoring - Basic Counters" - Steve Jones
<http://www.sqlservercentral.com/columnists/sjones/performancemonitoringbasiccounters.asp>
- Monitoring Disk Activity - Microsoft
http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/sag_mpmperf_19.mspx?mfr=true
- Monitoring Memory - Microsoft
http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/prork/prec_evl_bzcl.mspx?mfr=true
- Troubleshooting Performance in SQL Server 2005 - Microsoft
<http://www.microsoft.com/technet/prodtechnol/sql/2005/tsprfprb.mspx>
- Performance Tuning Checklist 4.5 (NT Based Systems) - Microsoft
<http://www.microsoft.com/technet/prodtechnol/bosi/maintain/optimize/bostune.mspx>

Conclusion

There are many things to think about before stress testing your SQL Server databases. I hope that this article outlined many of those items in a manner that enables you to hit the ground running for your stress testing endeavors. A well-planned stress test can be extremely beneficial, among other things it enables you to test application/procedure/database performance before going into production, troubleshoot performance, and plan for the future.

Anthony Bressi is owner of [Agilist Technologies Inc.](#) which specializes in software for DBA's and SQL Server developers. Mr. Bressi has over 9 years of hands-on experience in the Microsoft SQL Server development environment.

Intro to Database Mail in SQL 2005

By Corey Bunch

Introduction

We have all seen already that there are a ton of new features in SQL 2005. More realistically put, however, there are a ton of differences & things to change when migrating from 2000 to 2005, which a lot of the time invokes groans and moans, because naturally this means more work (and who cares about working?). But Database Mail, my friend, is a different story. No more Outlook installations....no more MAPI profiles....no more 3rd party smtp connector extended stored procedures....no more crossing your fingers and clicking your heels three times in order to get an email sent from your database system. Database Mail has come to the rescue.

Overview

The main difference between SQL Mail in SQL 2000 and Database Mail in 2005 is this: SQL Mail is a headache and Database Mail is not. After experimenting briefly with Database Mail, I see no reason why one would choose the legacy SQL Mail over the new Database Mail, unless of course for backward compatibility, legacy applications, etc.

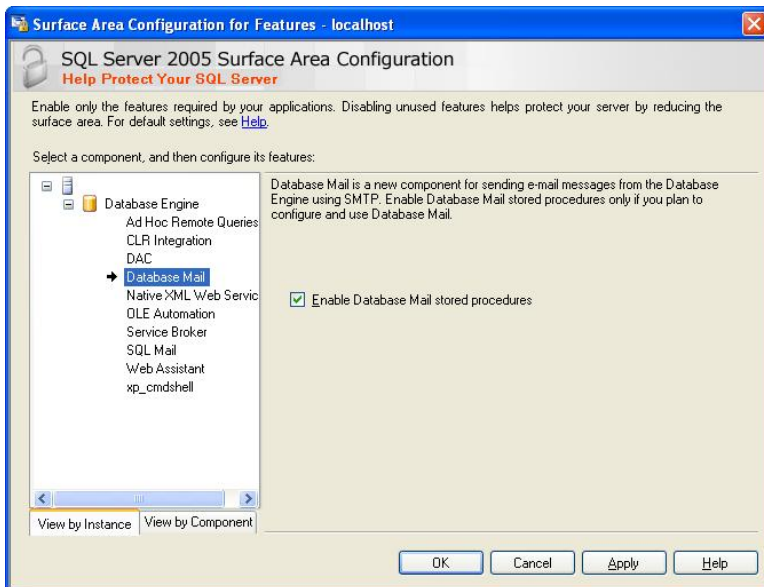
Not only does Database Mail handle the somewhat simple task of sending emails rather well. It has some other robust features that should not go unnoticed, such as...

- Multiple profiles and accounts to specify multiple SMTP servers or different email infrastructure situations
- SQL Server queues messages even when the external mailing process fails
- High security - users and roles have to be granted permission to send mail
- Logging and auditing
- HTML messages, attachment size regulations, file extension requirements, etc.

With all these considerations (plus a good number that I'm not including for purposes of brevity), provided you have (or are) a good developer, you can make some use of 2005's new Database Mail functionality.

Initial Setup

After installing SQL 2005, like a lot of features, Database Mail is not automatically enabled. To enable Database Mail, you must use the Surface Area Configuration Tool. Open the SAC Tool and choose the "Surface Area Configuration for Features". Choose "Database Mail" and click the checkbox.



An alternate way of enabling Database Mail is by using SSMS's (SQL Server Management Studio) object browser. Connect to the server you are interested in, browse to Management and then Database Mail. Right click Database Mail and choose "Configure Database Mail". According to Books Online, if Database Mail has not been enabled, you will receive the message: "The Database Mail feature is not available. Would you like to enable this feature?" If you respond with "Yes", this is equivalent to enabling Database Mail using the SQL Server Surface Area Configuration tool.

Database Mail Accounts and Profiles

Profiles

Database Mail profiles are simply an "ordered collection of related Database Mail accounts" ([Microsoft](#)). 2005 allow you to compile a collection of outgoing (SMTP) servers for your messages, to provide some fault tolerance, as well as

load balancing. SQL Server attempts to send your message through the last successful SMTP server that sent a Database Mail message, or the server with the lowest sequence number if a message has never gone out. If that server fails to transfer the message, then it goes onto the next one. Profiles can be public or private. Private profiles are only available to specified users. Public profiles are available to all users in the mail host (msdb) database. [You can find out more information about public and private profiles here.](#)

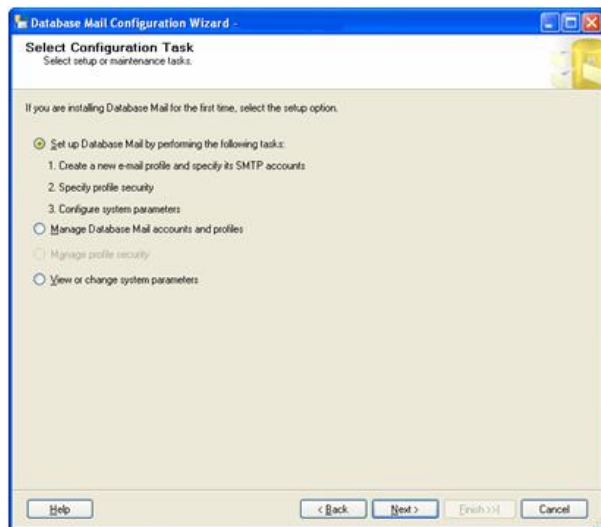
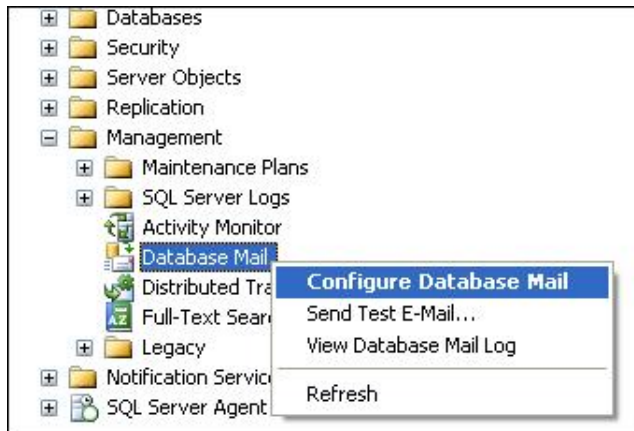
Accounts

Simply put, Database mail accounts contain information relating to email (SMTP) servers. This will remind you of the last time you set up Eudora, Thunderbird, Outlook, or any other simple email client..

Without further procrastination, let's get into setting up the Database Mail....Of course Microsoft takes care of all this by providing you with a wizard.

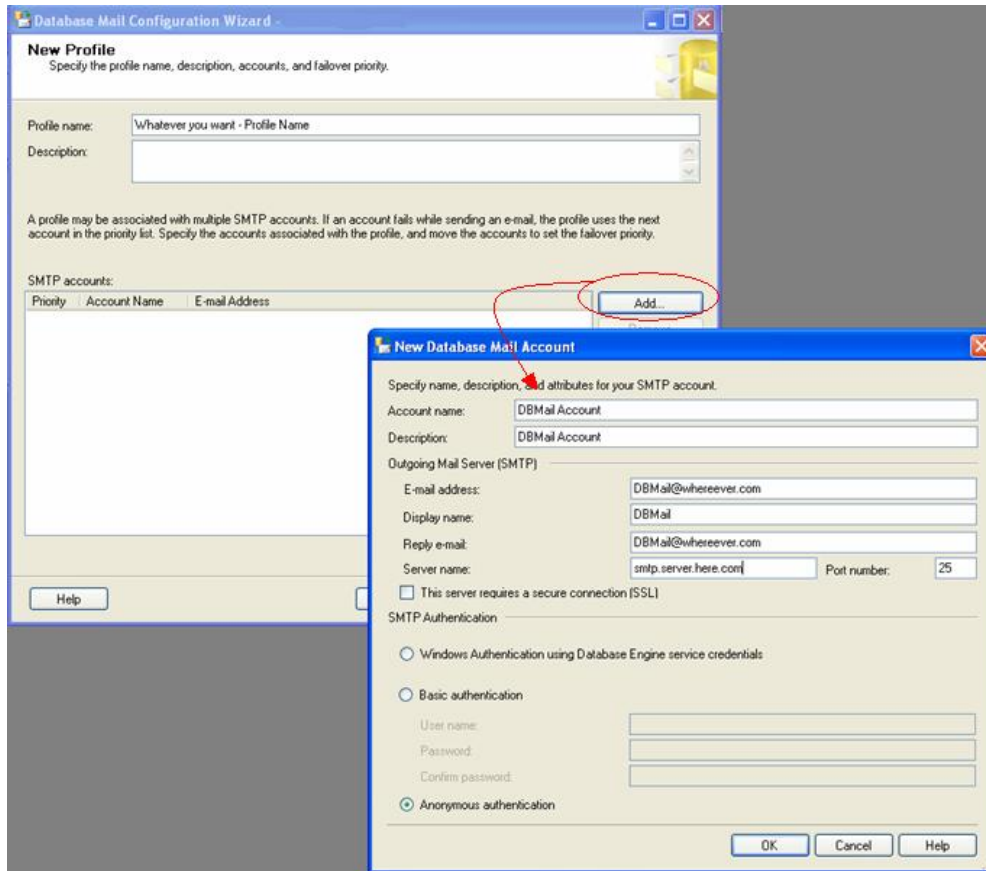
Configuring Database Mail

Using SSMS, browse to Database Mail and right click. Choose "Configure Database Mail". You'll get an initial screen with some different options on setting up or managing Database Mail.

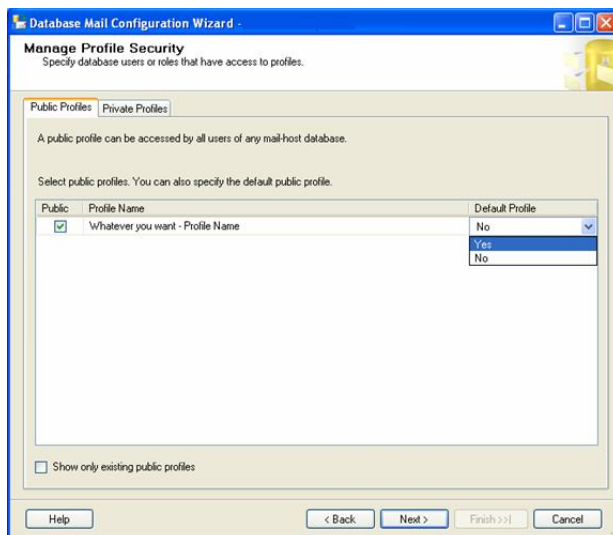


Leave the default for now, and choose next. Now fill in your profile name & description and click "Add". You'll get another window to fill in your SMTP server (or Database Mail account) information. Add multiple Mail accounts and attach them to this profile if you need to. Your email or server administrator should have an SMTP server or gateway

setup, in order for you to complete this form.



After clicking Next, you'll be able to set your security on this profile....whether or not you'd like it to be public or private, if you'd like it to be the default, etc. In order for users to show up on the Private tab, they must exist in the DatabaseMailUserRole on MSDB. See Books online for more details.



Finally, you'll be able to set some default system parameters concerning attachment size limits, file extension trapping and other system level details. From here you can go on to modifying your jobs to use Database Mail to alert you via email, or using `sp_send_dbmail` in your stored procedures or applications.

Summary

I always thought SQL Mail was a pain, and from the sounds of newsgroups and other SQL communication areas out there, others thought the same. The good news is that Database Mail is here to help, and here to stay with SQL 2005.

The OUPUT Command

By Dinesh Asanka

OUTPUT Command in SQL Server 2005

This not about the output parameter that can be used in stored procedures. Instead, this is about returning data affected in a table with a new feature in SQL Server 2005.

SQL Server 2000

A simple question for you. If you want to retrieve the last inserted identity value, what do you do? Obviously `SCOPE_IDENTITY()` or `@@IDENTITY` will be your answer. There is a small different between these too, which I am not going to discuss right now. Even though both will satisfy the current requirement, I will use `SCOPE_IDENTITY()`, which is the correct one.

```
CREATE TABLE TempTable
(
    ID INT IDENTITY(1 , 1)
    , Code VARCHAR(25)
    , Name VARCHAR(50)
    , Salary Numeric(10 , 2)
)

INSERT INTO TempTable ( Code , Name , Salary )
VALUES ( 'A001' , 'John' , 100 )

INSERT INTO TempTable ( Code , Name , Salary )
VALUES ( 'A002' , 'Ricky' , 200 )

SELECT SCOPE_IDENTITY() AS LastInsertID
```

However, this will only be valid when you need the last inserted ID. A Problem arises when you need the last updated or deleted data. In SQL Server 2000, you don't have any other option other than writing a trigger or triggers to capture them via inserted and/or deleted tables.

SQL Server 2005

To satisfy the database developers' cry, SQL Server 2005 has an added feature called OUTPUT. Here is a short example:

```
INSERT INTO TempTable ( Code , Name , Salary )  
  
OUTPUT Inserted.ID  
  
VALUES( 'A003' , 'Martin' , 300 )
```

The INSERT Statement not only inserts data to TempTable but also return the inserted ID value. Unlike, SCOPE_IDENTITY(), you have the ability of getting other affected values as well.

```
INSERT INTO TempTable ( Code , Name, Salary)  
  
OUTPUT Inserted.ID  
  
    , Inserted.Code  
  
    , Inserted.Name  
  
    , Inserted.Salary  
  
VALUES( 'A001' , 'Paul',400 )
```

The code above will return the ID, Code, Name and Salary fields as well.

The major improvement from the output command is that you have the ability of getting the affected values from an Update statement. In most cases you may need the information about the values that were there before they were changed, which you would get normally in an update trigger.

```
UPDATE TempTable  
  
SET Salary = Salary * 1.10  
  
OUTPUT Inserted.ID  
  
    , Inserted.Code  
  
    , Inserted.Name  
  
    , Deleted.Salary PreviousSalary  
  
    , Inserted.Salary NewSalary
```

The code above will return the ID, Code, Name along with previous salary and new salary and there is no need for an additional update trigger, which could have other performance issues.

Finally, let's look at deleting data. Even though it is the same as the above example, just for sake of completeness I will show this example.

```
DELETE FROM TempTable  
  
OUTPUT Deleted.ID  
  
    , Deleted.Code
```

```
, Deleted.Name  
  
, Deleted.Salary  
  
WHERE ID = 1
```

Conclusion

In SQL Server 2005, SCOPE_IDENTITY() and @@IDENTITY still work and there is no need to perform any modifications if you need the value of the Identity from an insert. OUTPUT is another new feature in SQL Server 2005 to improve the productivity of Database developers. So use it whenever you think it is appropriate.

Running a Query Using a Text File for Input

By Roy Carlson

How high is up? Well everyone knows it is twice as high as it is half as high. Weird question and even weirder answer but that seems to be the way of SQL. The users relish the idea of stumping the SQL person. Take the case of a user who requested the order amount for all the orders placed by specific users over the last five years and gave out the list of customer ids - 17,000 of them - out of a total field of 300,000+. Yikes!

No problem we add a new table and DTS the text data. Oh! Oh! This is a large third party application of which the MS SQL db is integral to. We can't create a table without causing an issue with the maintenance agreement. We have rights to query the database using Query Analyzer or Crystal Reports, but that is about it.

We could sit there and run 17,000 queries adding the where clause for each customer id. Not a good plan. We could array the where clause using an IN array. The user wanted the customer id, their name, order number, date of order and order amount. The data in real life spanned three tables requiring joins.

To demonstrate another solution we will use the Northwind database and this starting query:

```
SELECT  
  
    Customers.CustomerID AS ID,  
  
    Customers.CompanyName AS COMPANY,  
  
    Customers.City AS CITY,  
  
    Orders.OrderDate AS [DATE],  
  
    Orders.Freight AS FREIGHT  
  
FROM Customers  
  
    INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
```

or as I prefer using aliases:

```
SELECT  
  
    C.CustomerID AS ID,  
  
    C.CompanyName AS COMPANY,
```

```
C.City AS CITY,  
  
O.OrderDate AS [DATE],  
  
O.Freight AS FREIGHT  
  
FROM Customers C  
  
INNER JOIN Orders O ON C.CustomerID = O.CustomerID
```

Next we have the list of CustomerIDs - definitely not 17,000 but enough to get the point of all this.

custID.txt

```
ALFKI  
ANTON  
AROUT  
BERGS  
BOLID  
BONAP  
BSBEV  
CACTU  
CONSH  
DOGGY  
FOLIG  
FOLKO  
FRANK  
FRANR  
FRANS  
FURIB
```

Lets cut-and-paste these names into a text file called custID.txt. Lets also make to other files but keep them empty - custID.qry and custIDResult.txt. Copy and save as custID.vbs the script below in the same folder. (The files are also attached at the bottom in a zip file.) The next part is VBScript code. No dont run away. This is not a big deal - trust me - I am not in politics.

The custid.vbs code is available on the www.sqlservercentral.com site.

All you have to do for the above is make or pick a folder for the files to reside in and change to name of YOURFOLDER - in both locations - to the name of your folder.

Ok you can skip this next part if you dont want an explanation of the VBScript. The two constants - const - just make it easy to see that the code is for either reading or appending to an "existing"file. All things starting with "str" are string variables. We make objects with the Set command using the File Scripting Object. The first bit loops through the CustID.txt file reading each line - strLine - which is the CustomerID. For each CustomerID the value is added to the query. Then the new line containing the full query is appended to the file - custID.qry. Then all files are closed.

NOTE: You cannot run this script from Query Analyzer. It is run from a command line using cscript. Example:

```
C:\YOURFOLDER\>cscript custID.vbs
```

DO NOT RUN THIS WITHOUT THE **cscript** prefix Windows default is wscript which will require you to ok a lot of message boxes for each custID. 17,000 would be a lot of clicks.

Comments about the query itself: I have added SET NOCOUNT code to the front and rear of the query, because I really dont want to know how many rows are returned. If your search requires you to know if no

rows were returned then you might want to delete the SET NOCOUNT ON and OFF. We have another query we run using sql NOT EXISTS which reports on the missing.

Now comes the good old batch file which we run during minimum activity time. custID.bat which consists of:

custID.bat

```
osql /U YOURUSERNAME /P YOURPASSWORD /d NORTHWIND /S YOURSERVER /h-1  
  
/i "c:\YOURFOLDER\custID.qry" /o "c:\YOURFOLDER\custIDResult.txt"
```

This uses SQLs osql command to sign on using YOURUSERNAME AND YOURPASSWORD to YOURSERVER and the NORTHWIND database for this example. If you have not used osql before open "Books On Line." Type in "osql utility." You will get info on this tool. (Please change the YOURUSERNAME, YOURPASSWORD, and YOURSERVER as required.)

For every line in the c:\YOURFOLDER\custID.qry, a query is run, appending the data to the file c:\YOURFOLDER\custIDResult.txt. The custIDResult may not be pretty but we run the file into MS WORD and using a macro to remove extra spaces, replacing hard returns followed by a comma with a comma and so on the file cleans up nicely to a comma delimited file. This is saved as a csv file which we import into Excel, Access or Crystal Reports to get the needed reports.

The sequence of events is:

1. change the path to "YOURFOLDER" in custID.vbs
2. change the server name, username and password in the custID.bat
3. run the custID.vbs from the Command Prompt with "cscript custID.vbs"
4. check the custID.qry with "notepad custID.qry". It should have a query for each custID.
5. run the batch file "custID.bat"
6. check the custIDResult.txt and clean up with search and replace with Word to the format of your choice.

We have learned that with a text file of pre-selected data we can read the file into a query command file that can be processed by an osql command into a delimited text file for reporting. It certainly beats typing in 17,000 queries, but remember it is not the only way to accomplish this feat.

P.S. There is no Doggy in either the window or Northwind.

Starting SQL Server in Minimal Configuration

By Jay Dave

Summary

The SQL Server tools are top notch & one of the tools is "sqlservr.exe" which starts, stops, pauses, and continues Database Engine from the command prompt. This article describes how to start an instance of the Database Engine.

Over-Committing Memory

I had an issue recently where SQL server services didn't start after min & max server memory configuration of SQL server was changed. The Server got configured for AWE along with min & max server memory configuration. The SQL Server didn't start after stopping the services. What I intend to show you here is how problem came to happen & how it got resolved. In this article I'd like to talk about how this over committed memory issue is been addressed when value defined for "max server memory" (RAM) is not physically present on the server. Below topic will take you from Server configuration to Boot.INI file & to enable AWE & configuring server memory using system store procedure sp_configure & finally sqlservr.exe coming in rescue to start SQL server.

My Server Configuration:

- Windows 2000 Datacenter
- Physical memory 36GB
- SQL Server 2000 Enterprise Edition

Configuring AWE for SQL Server 2000

Use of the /PAE switch in the Windows Boot.INI configuration file provides access to physical memory above the 4 GB limit. This is a requirement for AWE memory access above 4 GB. Any change to Boot.INI file requires server restart/reboot. Now is time to enable AWE on SQL Server 2000 by running "sp_configure" from query analyzer. The LOCK PAGE IN MEMORY permission must be granted to the SQL Server account before enabling AWE (SQL SERVER 2005); this may be enabled if Windows 2003 is on SP1

```
(USE MASTER)

sp_configure 'show advanced options', 1

RECONFIGURE

GO

sp_configure 'awe enabled', 1

RECONFIGURE

GO

-- Note: - max server memory is in MB

sp_configure 'min server memory', 1024

RECONFIGURE

GO

-- 30GB is 30720MB but accidentally I typed 307200, which is 300GB

sp_configure 'max server memory', 307200

RECONFIGURE

GO
```

I stopped SQL Server & when I started SQL Server 2000, it didn't start, I tried other possibilities but that didn't help.

Solution

To work around the problem, start SQL Server 2000 in minimal configuration mode by using Sqlservr.exe with the -c -f option and reconfigure "max server memory." For a SQL Server 2000 Default Instance: Navigate to the Binn folder where the SQL Server 2000 default instance is installed and run the following command:

```
sqlservr.exe -c -f
```

For a SQL Server 2000 Named Instance: Navigate to the Binn folder where the SQL Server named instance is installed and run the following command:

```
sqlservr.exe -c -f -s Instance_Name
```

Connect to SQL Server through Query Analyzer, and then run this code:

```
(USE MASTER)

sp_configure 'max server memory', 30720 --- (Which is now 30GB)

RECONFIGURE

GO
```

Navigate to the command prompt and then press CTRL+C. To shut down the SQL Server 2000 instance, type "Y". After that when you start again SQL Server it will come up fine & your "max server memory" issue is been resolved.

How SQL Server Chooses the Type of Join

By Mike Dillon

The problem

A stored procedure that is called via .NET application runs in as short a time as 1-3 seconds some times, but runs for as long as 1.5 minutes at other times. The same stored procedure ran in Query Analyzer always returns in the 1-3 second time frame. The stored procedure is called multiple times in a small time frame and collects data based on start dates and end dates. We can't reproduce it to the degree that we see on the client side, but we are able to see the above mentioned pattern. On the client site, they reach the .NET timeout of 3 Minutes on two of the eight times it is called. When the timeouts occur, these are also the calls made with the largest date ranges.

Finding the Real Problem

So the first thing that jumps out is the fact that this stored procedure fails to complete during the .NET timeout of 3 minutes on the two largest date ranges. However, upon running the stored procedure in Query Analyzer, it finishes with in 3 seconds.

Every time.

No hang ups, no slow downs, nothing. Even with bigger date ranges than that specified by our application and returning as much data as possible, we can't come close to a three minute time out. The procedure still runs in a matter of seconds.

Our thought process jumped to the client environment and possible differences at their site that might explain what it was they were seeing. Were there other things running at the same time? Was there something else blocking tables that we needed to get to? This was tested by again running the procedure in Query Analyzer, which did take longer than at our site. The indexes for the affected tables were rebuilt and now the Query Analyzer test finished in a few seconds. Ahh! Problem solved. Rebuild the indexes and we should be fine. The client ran the procedure via the application again that night, and again it failed. Back to the drawing board

At this point we needed to start over and gather facts. We decided to verify that the procedure ran slow from the application and fast from Query Analyzer. It did run faster. In fact, while running the stored procedure from the application and watching a trace of the server activity we were able to run the stored procedure from Query Analyzer while waiting for it to return from the application. At this point we began to suspect something in the .NET communications layer as that seemed to be the only difference. In order to verify this, we used [ApeXSQL's SQL Edit](#) to run the procedure from as it connects to the DB the same way our application does. That proved to be the jumping off point that would eventually lead to the detection of the problem.

When we tested the procedure from APEX, it ran slow, much like the application. We were now convinced it was something with the way .NET connected to the database server. We made a change to the stored procedure and recompiled it. Sure enough, it was much faster, 1-3 seconds, just like Query Analyzer. We then ran the stored procedure, which didn't seem to improve much over the previous time, so we ran the stored procedure from APEX again and it was slow again. Can you imagine our level of confusion? After running several other tests, we became convinced something was taking place within the application that was causing the query to run slowly during and after the application was run.

So at this point, the facts were as follows;

- The procedure when run via the application was slow, over a minute in some cases.
- The same stored procedure run from APEX was also slow once the application had been run.
- The stored procedure would run fine from SQL Server Query Analyzer.
- The stored procedure would run fine from APEX as long as it had been recompiled.

So what was different about the procedure when it was run from APEX fast versus APEX when it was slow? We decided at this point to start digging into the query plan. After analyzing the query plan, we could see the faster one was much shorter than the slow one, we just didn't know how or why. We discovered that the query plan for the procedure when it took a long time, used nested loop joins as opposed to when it was faster, in which case it used hash joins.

Why would it use nested loops some times and hash joins other times? And if the hash joins were faster, why didn't it use them all of the time. Why would it ever use the nested loops? According to the Microsoft documentation, the engine will chose to use nested loops for the joins in situations where there is not a lot of data being joined. And it will chose hash joins if there is a large amount of data being joined. In our case we had both, so why didn't it choose the right one for the right situation.

The answer lies in one of the other things we mentioned earlier. The procedure would run just fine in APEX if it was recompiled before it ran. But if it wasn't and the application ran first, then APEX would be slow. What was happening is that the procedure when run from APEX would chose the query plan from the procedure cache if there was one there. After a recompile there isn't one there, so it would choose the one for the larger data set as we were testing the procedure with the widest date sets and the largest data sets. When the application ran however, the first set of dates were small in range and the data set returned is relatively small. Based on this SQL Server would choose a query plan using nested loops, which in and of itself is fine. The problem came with the fact that the query plan was now in cache and would continue to tell the engine to use the nested loop join. It didn't know that the data set would change and grow. All the engine knows is that the application called a stored procedure for which it had a query plan stored in cache. It used that plan rather than compiling a new one and the problems described above were revealed.

The Solution

The fix for this was relatively simple. We simply inserted a `sp_recompile {procedure_name}` into the stored procedure so that it would recompile before it ran each time. This would allow a new query plan to be chosen based on the size of the current data set rather than simply using what was in cache. You might be thinking that compiling that stored procedure over and over again would add overhead and you would be right, which is why it is not a best practice to do this. However, in cases where you call the same stored procedure multiple times with different parameters that affect the size of the data set returned you might see a huge gain in performance by doing the recompile each time prior to the stored procedure being run.

Keep in mind this is true in any scenario in which SQL Server is going to choose a query plan based on a set of parameters that can have drastically different size data sets being called by the same stored procedure.

Indexes and Fragmentation in SQL Server 2000 Part 1

By Joe Doherty

I believe it's my academic background that has seen me develop the need to see the facts for myself before I feel confident about a particular subject. It has always been both a strength and a weakness of mine. On one hand it provides the motivation for me to tackle a subject I want to understand, but on the other I have been guilty of reinventing the wheel just so I can see it work for myself. You might say this is natural for guys like us and essential for personal development however I'm sure my past superiors have seen it unnecessary for me to totally rebuild a system from scratch when all that was required was a simple configuration change. But then where does all the fun come from?!

With this in mind I have finally decided to tackle the subject on fragmentation levels within a database. I've always been told that proper index maintenance is essential for optimal performance, and over the years have employed different methods of achieving this. However I have never really seen proof that my maintenance jobs actually have any benefit. Sure I've seen the odd demonstration and even made improvements to systems I've managed myself but these have been isolated incidents where the customer has made a complaint or a developer didn't take into account the amount of data growth within a particular table. These problems have usually been resolved by reindexing a particular index more frequently or applying a more appropriate index.

What I'm talking about here is a system that has been in production for a number of years, where data volumes have continually grown, application patches applied, OS and SQL service packs and patches applied, etc. The system is considerably different now to what it was when it was first released, and chances are you've taken over the responsibility of these systems somewhere in between.

Ideally a performance audit would need to be carried out periodically to ensure your systems are running optimally, and I imagine this would be quite some task to undertake on some of the systems that are out there.

What I am interested in here is looking at what fragmentation is, how it happens and how we go about resolving it. I am also interested in running a few experiments to see how a `SELECT` query performs on tables that have indexes with differing fragmentation levels. This article is more of a discovery voyage rather than a 'how to implement your index maintenance' document and will hopefully give us all a basic foundation to go away with and think about. You might read the entire article and think I could have got this point over in much less text but I want to take you through my entire thought process to aid a better understanding.

After some thought I decided that it was best to revisit the topic on how SQL stores its data and look at the behavior as data is added. And by running a few experiments we can look further into how the system functions under the surface. Then by adding an index or two we can look at the differences in the behavior that SQL exhibits.

As I said earlier I need to see things for myself and so I am going to start at the very beginning and create a blank database with one table.

```
-- This script creates our test database and creates a table within it

-- You will have to modify the 2 filename parameters to values of where you want the DB created

CREATE DATABASE [myIndexes] ON (NAME = N'myIndexes_Data',

FILENAME = N'C:\myIndexes_Data.MDF' , SIZE = 1000, FILEGROWTH = 10%)

LOG ON (NAME = N'myIndexes_Log', FILENAME = N'C:\myIndexes_Log.LDF' , SIZE = 30, FILEGROWTH =

10%)

COLLATE Latin1_General_CI_AS

GO

USE [myIndexes]

GO

CREATE TABLE [dbo].[myTable] (

    [myPK] [uniqueidentifier] NOT NULL ,

    [myID] [bigint] IDENTITY (1, 1) NOT NULL ,

    [Char1] [varchar] (20) COLLATE Latin1_General_CI_AS NOT NULL ,

    [Char2] [char] (200) COLLATE Latin1_General_CI_AS NOT NULL ,

    [Char3] [varchar] (2000) COLLATE Latin1_General_CI_AS NOT NULL ,

    [Num1] [int] NOT NULL ,

    [Num2] [money] NOT NULL ,

    [Date1] [datetime] NOT NULL

) ON [PRIMARY]

GO

ALTER DATABASE myIndexes SET RECOVERY SIMPLE

GO
```

Have a look at our database (myIndexes) and you'll see that it is initially set to a 1GB data file, 30MB log file and simple recovery. You might not have enough disk space so feel free to reduce this but it is useful for what we'll do later on.

Looking in the table (myTable) you'll see I've created a number of fields with differing data types. Hopefully we'll be able to see if these have any effect on performance later on in the article. You'll also notice that there aren't any indexes configured on the table - also known as a heap. Initially we are just going to keep things simple and take a look at the basics of the file structure and how SQL maintains it.

OK, so how and where is the database and table (and system objects) actually stored on the disk? And is there any fragmentation at this point?

Well the database and all its objects are obviously stored in the myIndexes.mdf in the location you specified when creating the database. But where is [myTable] stored within this .mdf file? The objects within a database are stored within data pages which are a little over 8k each. When you create a table you define a number of fields that will store your data. The total number of characters that the combined fields make up cannot exceed this 8k limit. If each character takes up 1 byte then the total number of characters you assign to the table (i.e. each row) cannot be more than 8,000 (8,060 to be exact).

For example if you look at the table we have created in our database you will see it is made up of the following fields:

myPK	uniqueidentifier	16
myID	bigint	8
Char1	varchar	20
Char2	char	200
Char3	varchar	2000
Num1	int	4
Num2	money	8
Date1	datetime	8

total =		2264

You can see that we have used a total 2,264 bytes. Also I have used the term bytes instead of characters - that's the official way field length is measured. Generally a text field will use 1 byte per character but this does not easily convert for a date (8 bytes) or a money field (8 bytes) so it is best that we refer to the length from now on as bytes. Other data types take up different amounts of storage (e.g.. unicode characters use 2 bytes and would therefore limit you to 4,000 characters per row). I want to keep things relatively simple here so you can look them up in BOL if you want to know more.

Note: Certain types can span multiple pages but we won't be covering this here.

OK, so now you see that in general we're limited to a maximum row size of one data page (8k). More often than not however your total row size will not be anywhere near this and so SQL will be able store more than one row within a single data page.

As you keep adding rows to your table SQL will start to add more data pages and populate them accordingly. Now there's one more thing you should know about your data pages and that is SQL groups them into bundles of 8 - and we refer to these as Extents. An extent is a group of 8 data pages, holding up to 64k of data (i.e. 8k x 8 data pages = 64). Why SQL does this is beyond the scope of this article but let's just say it is for optimal disk performance.

Right let's work out how to find which data page is storing our table data. Type the following command... Remember that all the commands that we use will be against the [myIndexes] database unless otherwise stated.

```
SELECT * FROM SYSINDEXES WHERE NAME = 'myTable'
```

This command searches for the system information about our table object. You will see that one row is returned containing a number of fields. The fields we're interested in are:

first	= a pointer to the first data page
dpages	= the number of data pages used for our table data

Currently these values are set to zero because no data has been inserted yet - so lets add our first record by running the following command...

```
INSERT INTO [myTable] VALUES (NEWID(), 'test1', 'test1', 'test1', 123, 321, GETDATE() )
```

and confirm it is there...

```
SELECT * FROM [myTable]
```

Now we have a record in our database lets re-run the command on the system table to find out about where our data is stored...

```
SELECT * FROM SYSINDEXES WHERE NAME = 'myTable'
```

and we now see that these vales have been updated. The values you get may differ from mine so bear this in mind when we use them as parameters for the upcoming exercise. My values are...

```
first      = 0x220000000100
```

```
dpages     = 1
```

So this is saying that the data for [myTable] starts on data page 0x220000000100 (a hex value) and that a total of one data page has been used so far.

Now we'll use one of SQL's undocumented commands to view the contents of the data page, but before we do we need to decipher the hex number by doing the following (more information can be found here - <http://www.sqlservercentral.com/forums/shwmessage.aspx?forumid=5&messageid=291762>).

Split the 0x220000000100 number into two halves which will give us the page id part and the file id part. This gives us page id: 0x220000 and file id: 0x000100. Because we're only using one database file (myIndexes.mdf) then we can be sure that the file id is 1. The page id needs to be reversed which will give us 0x000022 (always leave the 0x bit at the front) and now convert this to decimal. I use Google to do this - just type 'convert 0x000022 to decimal' and I get 34. This is page where our data is stored.

SQL's undocumented command DBCC PAGE allows us to view the contents of a page but we must switch on the trace element first...

```
DBCC TRACEON (3604)
```

```
DBCC PAGE (myIndexes, 1, 34, 3)
```

Remember to replace the value 34 with the one you got from your Google result. Try changing this value to see what else is lurking in other data pages but at this point you'll only see system related data. I'm not going to discuss this command because it is already well documented on the Internet - see <http://support.microsoft.com/kb/q83065/> - you won't find it in BOL and Microsoft say they do not support it seen as it hasn't gone through the usual rigorous testing that supported commands have. Or perhaps it has but they wanted to keep it back as part of their own toolkit for support purposes - I know I've been instructed to use it by Microsoft Product Support services (PSS) in the past while investigating database corruption.

Anyway back to the contents of our data page - you should now see your record within the text and in the bottom section. That's how you view the contents of your data pages. And no, scanning these pages will not reveal system passwords!

So far we have created a blank database, a table and inserted one row of data and have seen that SQL stores this in a data page. Now we need to get a decent amount of data in there so we can start to look at how the system grows and also see if there's anything interesting going on. I have written a script that populates the table with random data - unfortunately this script does take some time to run but can be stopped and started as you see fit. Lets get this running now and then continue talking about the next step.

Before you do run the script please run the following command and store the results - we'll be talking about this next and we need to capture the values before you start adding the records.

```
DBCC SHOWCONTIG ('myTable')
```

Store these results and proceed to run this script to start generating our records...

Editor's Note: This script is available at www.sqlservercentral.com.

You will see at the start of the script that it loops 1000 times (`WHILE @Loop < 1000`) - this also refers to the number of records that will be added so ideally we want to change this to nearer 500000. Perhaps you can run it overnight. I've challenged a colleague at work to see which of us can come up with a script that randomly inserts values into the database the fastest - I have some good ideas but not had the time to try them out. If anyone is interested in joining in the fun (or has any suggestions) let me know and I'll send you the contest rules!

OK while the script is running lets run a check to make sure we can see things happening as we would expect them to. Run the following to check that the number of data pages has been increasing - remember it was at one before...

```
SELECT * FROM SYSINDEXES WHERE NAME = 'myTable'
```

If you repeatedly run this the number of rowcnt and dpages should keep going up. However depending on how busy your system is you may need to issue the following command to force a system update. Remember this command because it is very useful when you need up to date information.

```
DBCC UPDATEUSAGE(0)
```

If you've ever resized your database or inserted a number of records and SQL does not report values to represent this then running this command will update SQL's internal system values for you. We might see more of this in action later. BOL definition: Reports and corrects inaccuracies in the sysindexes table, which may result in incorrect space usage.

If you remember earlier on one of the questions we asked is whether we had any fragmentation after the point we had inserted a record. Well I'm sure you'd agree that this is highly unlikely since we are dealing with a brand new database. And before you ran the script that inserts lots of records I asked that you quickly run the DBCC SHOWCONTIG command and store the results. Well now we'll look at what we actually did here...

DBCC SHOWCONTIG, as BOL puts it, displays fragmentation information for the data and indexes of the specified table. We actually ran this command against our table and would have received results like the following:

```
- Pages Scanned.....: 1
- Extents Scanned.....: 1
- Extent Switches.....: 0
- Avg. Pages per Extent.....: 1.0
- Scan Density [Best Count:Actual Count].....: 100.00% [1:1]
```

```
- Extent Scan Fragmentation .....: 0.00%  
- Avg. Bytes Free per Page.....: 7827.0  
- Avg. Page Density (full).....: 3.30%
```

Lets look at each of the results in turn to gain an understanding of what it is they are telling us.

- **Pages Scanned**

This tells us that 1 page was scanned in total to produce our results. And we are happy with the fact that we only have 1 page.

- **Extents Scanned**

As we know there are 8 pages per extent but seen as we only have 1 page then that would infer we only have 1 extent.

- **Extent Switches**

If there were 100 extents SQL might have to flick between each of them multiple times to read all the records in the data pages. When performing this command think of it that SQL must access every record therefore having to visit every page and extent that contain current records. If the records are not in order then SQL will need to keep flicking between Extents to access the required data. You see when a record is added it isn't necessarily added to the end of the table - it might be inserted somewhere in the middle. Or if a record is updated and it becomes too big to fit in to it's current space allocation SQL will move it and set a pointer to the new location. Therefore when it comes to reading it again it will be passed to the new location which may be on another page and extent. It is this flicking that causes additional disk reads which in turn affects performance.

Ideally the number of extents scanned should be one more than the number of extent switches that take place. This would mean that SQL has started at extent one and read through each of them in turn without being redirected elsewhere.

- **Avg. Pages per Extent**

As we know we only have one data page so SQL has only create one at this point. As data starts to grow SQL will think ahead and create more.

- **Scan Density**

Best count is the ideal number of extent changes if everything is contiguously linked. Actual count is the actual number of extent changes. The number in scan density is 100 if everything is contiguous; if it is less than 100, some fragmentation exists. Scan density is a percentage. As you can see from our results we have a perfect 100%!

- **Extent Scan Fragmentation**

Because we're not using an index yet this value is irrelevant.

- **Avg. Bytes Free per Page**

Average number of free bytes on the pages scanned. The higher the number, the less full the pages are. Lower numbers are better.

- **Avg. Page Density (full)**

Average page density (as a percentage). This value takes into account row size, so it is a more accurate indication of how full your pages are. The higher the percentage, the better.

If you run the DBCC SHOWCONTIG command again on your table you will start to see different results on the extents. Because we are working with a heap (a table without a clustered index) the data is in no particular order. It is generally in the order that it was inserted into the table (providing that we are mainly inserting new rows) however SQL will also insert data in to any available space it comes across. If you do a SELECT * FROM [myTable] you will probably find that the [myID] column doesn't increment serially in the early part of the table and this is because as SQL grows the pages and extents it will copy approximately half of the contents of one page into the next so that there's always room for quick insertion of additional records later on (known as page splits). In the early stages of our data generation script SQL will find free space in some of the initially created data pages and insert some records into these pages rather than at then end of the table. I've never actually had confirmation of this but this is how I understand it - it's as if SQL initially expands quite quickly leaving a lot of free space in the first extent(s) and then returns shortly after to fill them in.

I hope you have picked up on the point that because we are using a heap the data is in no particular order. And if you perform any kind of SELECT statement on a heap the entire table must be scanned. And as we have read if we were to select all the records from the table we wouldn't necessarily get them back in the order they were inserted.

Saying that, and I hope you are with me at this point, can a heap actually become fragmented? And if so is there a command to defrag it? Well I suppose the answer to whether a heap can become fragmented is yes but would that actually cause us any problems? It's not as if we would be scanning the table without any indexes applied and, as we've not specified any order for the table, then we're not loosing out. Fragmentation, of a kind, can occur on a heap when forwarding records are created that point to an updated record that was moved due to the update making it bigger than its currently allocated space (i.e. there was no room for it in the current slot). There is no command that will defrag a heap (unless you apply a clustered index and remove it again as this will force an order which will remain after the index is dropped, but will not be maintained).

Conclusion

So we've not really learnt anything new about how to improve our table fragmentation but we have been through the motions of how data builds up within the database file. And we've seen that having a table without an index, in particular a clustered index, deserves the name 'heap' because basically that is all it is, and of little use to us. Running any form of query on a heap will force a full scan of the table and could take a considerable amount of time. And the results from the DBCC SHOWCONTIG command, which is a tool for measuring fragmentation, is also of little use on a heap.

I am going to finish at this point because it is a big subject to tackle and I want to get some feedback before releasing the next part of this article. In my next installment we will look further into data pages and start analyzing the different types and how these manage the allocation of extents and free space. Then we'll start using indexes. We're a long way off but we'll get there slowly and in an academic fashion.

Do have a play and a think about what we have covered so far. Any suggestions would also be welcome. And please remember I am by no means a SQL expert - this work is my findings that I am willing to share with you. If you feel I have mislead you, have incorrectly stated something or simply need more info then please submit feedback.

Until next time...

Indexing in SQL Server 2005

By Aaron Ingold

Introduction:

Database indexing is not an exact science. It's difficult to weigh the performance benefits adding indexes to a table will have. While adding indexes can improve performance on SELECT queries, it can slow down performance or introduce locking scenarios on highly transactional tables with a high number of INSERT/UPDATE/DELETE statements. With the release of SQL Server 2005, Microsoft has snuck in a few new ways to help you with this strategy. Most people are aware of the Database Engine Tuning Advisor which can provide good information by analyzing a workload from SQL Profiler, however there are other approaches which I'd like to discuss.

For this article, I'm assuming you have a good grasp of indexing basics: the difference between clustered and nonclustered indexes, how they are stored, how the query processor makes use of them, using them to "cover" a query, and the various index options. This isn't to teach someone how to index, per se, but hopefully add some new tools to your approach to indexing in SQL Server 2005. For demonstrations in this article I will be using the AdventureWorks database which comes with SQL 2005 or can be downloaded [here](#). As with all performance "enhancements", things don't always work the way you'd think they would. For each change I would recommend you take a baseline, apply in a test environment first, measure against the baseline to be sure your changes have the desired affect, and always have an undo script.

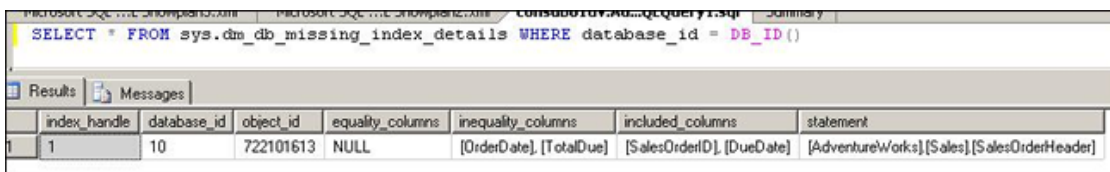
Identifying Potential Indexes: sys.dm_db_missing_index_details

The new dynamic management views in SQL Server 2005 have done a lot to enable quick scripting of management functionality. One which has gone largely undocumented is sys.dm_db_missing_index_details. This stores a list of columns the query processor has gathered since the last server restart which would assist in covering the queries.

For the first example, I will execute the following code to view a list of orders and their details in the last 3 years with a value of \$20,000 or more:

```
SELECT          h.SalesOrderID, h.OrderDate, h.DueDate, h.TotalDue
               , p.[Name] AS Product_Name, d.OrderQty, d.UnitPrice, d.LineTotal
FROM Sales.SalesOrderDetail d
JOIN Sales.SalesOrderHeader h
ON  h.SalesOrderID = d.SalesOrderID
JOIN Production.Product p
ON  d.ProductID = p.ProductID
WHERE          h.TotalDue >= 20000
AND  h.OrderDate >= DATEADD(yy, -3, GETDATE())
```

Now it's time to go check the dynamic management view and see what the query processor picked up:



index_handle	database_id	object_id	equality_columns	inequality_columns	included_columns	statement
1	10	722101613	NULL	[OrderDate], [TotalDue]	[SalesOrderID], [DueDate]	[AdventureWorks].[Sales].[SalesOrderHeader]

This view collects data for the entire server, yet I only want information about the current database (AdventureWorks), hence the WHERE database_id = DB_ID(). The last four columns are the ones you really want to look at here. Here's how they break down:

- **equality_columns** - This lists any columns where the query processor looked for an exact match (i.e. column = value). In this case it's null as we had no equality statements needing to be scanned upon in our WHERE clause. The only equal statements in there are on our JOIN conditions, however since all our key columns in the join have indexes on them they're not listed.
- **inequality_columns** - This lists any columns where the query processor looked for an inequality match. Greater-than, less-than, not-equal or BETWEEN predicates would fall in here. As we were looking for records which fell both above a certain value and a certain date, the OrderDate and TotalDue columns are listed here.
- **included_columns** - SQL Server 2005 allows you to specify non-key columns which will be stored with the index. This allows you to include more columns within the index so that you can cover queries more effectively while not requiring the data in those columns to be sorted during transactional processing.
- **statement** - This is the underlying object on which the index would be created.

Your initial reaction here might be to think that Microsoft has done all the work for you, however this would be premature. This should be used to point you in the right direction for what to index, not build indexes directly off of it. In *general* you should build your indexes by listing the equality (most restrictive) columns first, the inequality columns

second and specifying the included columns in your INCLUDE clause. If you stop here and build your indexes you may be missing a large part of the picture. Let me identify some other considerations you should think about:

- In practice I've seen this technique list columns in SELECT order, rather than the order in which they're scanned by the query processor. You would really gain the most performance by building the index in the order in which they query processor scans them.
- How large is the table? A narrow table and/or one that will not grow above 100 rows might not be worth indexing.
- You don't really know how often the queries which reference these columns are run. Are they one-time ad-hoc queries or part of a larger stored procedure which is run frequently? In the case in the example it's easy, however if you select from the same management view on a production machine you may have many rows returned. It is unlikely that building and maintaining an index which would be referenced only occasionally would do anything to increase performance.
- There may be multiple listings in this view which could all be covered by a single query. For instance if you add another line to the WHERE clause (say "AND DATEDIFF(dd, h.OrderDate, h.ShipDate) > 7" to find any orders that didn't ship within a week) and re-run the query and then check the management view again. You should see an additional column in the list:

	equality_columns	inequality_columns	included_columns	statement
13	NULL	[OrderDate], [TotalDue]	[SalesOrderID], [DueDate], [ShipDate]	[AdventureWorks].[Sales].[SalesOrderHeader]
13	NULL	[OrderDate], [TotalDue]	[SalesOrderID], [DueDate]	[AdventureWorks].[Sales].[SalesOrderHeader]

Instead of making two indexes, it might be better off to only create a single index including the extra column and allow the query processor to use that index to cover both queries. As always, set a baseline, implement the changes in a test environment, and measure the performance impact before implementing in production.

Identifying Potential Indexes: SET STATISTICS XML ON

While sys.dm_db_missing_index_details is good, you still need to tread carefully and think about how you use the columns it lists. A more focused approach is to use SET STATISTICS XML ON to receive an XML showplan result detailing how the query processor fulfilled the request. To return to our original example (with appropriate additions/commenting for this section):

```
SET STATISTICS XML ON;

GO

SELECT          h.SalesOrderID, h.OrderDate, h.DueDate, h.TotalDue
               , p.[Name] AS Product_Name, d.OrderQty, d.UnitPrice, d.LineTotal
FROM Sales.SalesOrderDetail d
JOIN Sales.SalesOrderHeader h
ON h.SalesOrderID = d.SalesOrderID
JOIN Production.Product p
ON d.ProductID = p.ProductID
WHERE          h.TotalDue >= 20000
AND h.OrderDate >= DATEADD(yy, -3, GETDATE())
-- AND        DATEDIFF(dd, h.OrderDate, h.ShipDate) > 7
```

GO

```
SET STATISTICS XML OFF;
```

GO

Executing this block of code returns both the standard result set and a Microsoft SQL Server 2005 XML Showplan result. Clicking on the XML in Management Studio will bring it up in its own document for easier browsing. The element we want to focus in on is <MissingIndexes>:

```
<MissingIndexes>
  <MissingIndexGroup Impact="27.3238">
    <MissingIndex Database="[AdventureWorks]" Schema="[Sales]" Table="[SalesOrderHeader]">
      <ColumnGroup Usage="INEQUALITY">
        <Column Name="[OrderDate]" ColumnId="3" />
        <Column Name="[TotalDue]" ColumnId="24" />
      </ColumnGroup>
      <ColumnGroup Usage="INCLUDE">
        <Column Name="[SalesOrderID]" ColumnId="1" />
        <Column Name="[DueDate]" ColumnId="4" />
      </ColumnGroup>
    </MissingIndex>
  </MissingIndexGroup>
</MissingIndexes>
```

As appropriate you will see various ColumnGroup elements for EQUALITY, INEQUALITY and INCLUDE which work in the same way as the previous example. The Impact value indicates an estimated improvement by the query processor assuming an index was created on the listed columns.

Specifying another predicate in the where clause which references another table (for instance "AND d.SpecialOfferID = 1") changes the XML plan when run to include multiple suggested indexes to improve overall query performance. In this case, adding the example above gives me this for my <MissingIndexes> element:

```
<MissingIndexes>
  <MissingIndexGroup Impact="52.9893">
    <MissingIndex Database="[AdventureWorks]" Schema="[Sales]" Table="[SalesOrderDetail]">
      <ColumnGroup Usage="EQUALITY">
        <Column Name="[SpecialOfferID]" ColumnId="6" />
      </ColumnGroup>
      <ColumnGroup Usage="INCLUDE">
        <Column Name="[SalesOrderID]" ColumnId="1" />
        <Column Name="[OrderQty]" ColumnId="4" />
        <Column Name="[ProductID]" ColumnId="5" />
        <Column Name="[UnitPrice]" ColumnId="7" />
        <Column Name="[LineTotal]" ColumnId="9" />
      </ColumnGroup>
    </MissingIndex>
  </MissingIndexGroup>
  <MissingIndexGroup Impact="26.7019">
    <MissingIndex Database="[AdventureWorks]" Schema="[Sales]" Table="[SalesOrderHeader]">
      <ColumnGroup Usage="INEQUALITY">
        <Column Name="[OrderDate]" ColumnId="3" />
        <Column Name="[TotalDue]" ColumnId="24" />
      </ColumnGroup>
      <ColumnGroup Usage="INCLUDE">
        <Column Name="[SalesOrderID]" ColumnId="1" />
        <Column Name="[DueDate]" ColumnId="4" />
      </ColumnGroup>
    </MissingIndex>
  </MissingIndexGroup>
</MissingIndexes>
```

This shows two separate indexes on two separate tables which could assist in query processing.

This is very helpful in the early stages of improving problem queries. If you have an idea that a certain SQL statement is going to be repeatedly executed, using this technique can assist you in building better indexes on your tables.

Removing Unused or Problem Indexes: sys.dm_db_index_usage_stats

I once had a developer look me straight in the eye and say that there would never be a problem with adding as many indexes as possible to a table; the query processor would simply select the ones it needed and ignore the rest. While this is *somewhat* correct, it really falls far from the whole truth. Unnecessary indexes can force the query processor to evaluate more before deciding on an optimum execution plan, can slow down transactional processing and can take up a great deal of space on disk.

Index usage information is stored in another dynamic management view called `sys.dm_db_index_usage_stats`. Like the previous views, information available inside only persists since the last SQL Service startup. I would highly recommend you not evaluate information in this view until after the system has been running for an adequate period of time under normal load and usage conditions. This will allow you to obtain a decent set of information before determining which indexes to potentially remove.

With my warning out of the way, let's get to the code I like to use:

```
SELECT          o.name AS object_name, i.name AS index_name
               , i.type_desc, u.user_seeks, u.user_scans, u.user_lookups, u.user_updates
FROM sys.indexes i
JOIN sys.objects o
ON   i.object_id = o.object_id
LEFT JOIN      sys.dm_db_index_usage_stats u
ON   i.object_id = u.object_id
AND   i.index_id = u.index_id
AND   u.database_id = DB_ID()

WHERE          o.type <> 'S'                -- No system tables!

ORDER BY       ( ISNULL(u.user_seeks, 0) + ISNULL(u.user_scans, 0) + ISNULL(u.user_lookups, 0)
               + ISNULL(u.user_updates, 0)), o.name, i.name
```

The reason I use a LEFT JOIN to `sys.dm_db_index_usage_stats` is to ensure that any indexes which have not been used since the last server restart are still listed accordingly. If my server has been up and running for six months without using that index, perhaps its time to consider dropping it! By sorting it in the order which I do, it moves the least used indexes to the top of the list. It's also important to note that indexed views will be listed in the `object_name` column.

I would highly recommend again that you factor in other considerations before removing or disabling indexes. It's important to make sure the database will not be using the index. It's also good to read this information even for your more commonly used indexes and examine the scans vs. seeks. This can provide good hints for optimizing your existing indexes and queries (for a good explanation of a scan vs. a seek, check out Craig Freedman's WebLog – <http://blogs.msdn.com/craigfr/archive/2006/06/26/647852.aspx>).

Summary:

As I said at the start, the intent of this article is not to teach you how to index. There are numerous resources devoted to this topic and it is beyond the scope of this article (see below for recommended reading). SQL Server 2005 gives you some great new tools to better understand index usage and gain insight into where new indexes might be helpful. While there is still no tool which can really replace a good understanding of how SQL Server makes use of indexes and where they can be beneficial, the information you can glean from the new dynamic management views and enhancements to viewing query statistics can assist you in optimizing your indexing to see real performance gains.

Recommended Reading:

- <http://www.sqlservercentral.com/columnists/Peysakhovich/indexcreationguidelines.asp> – Index Creation Guidelines by Leo Peysakhovich
- <http://www.sqlservercentral.com/columnists/gjackson/whocaresaboutfillfactor.asp> – Who Cares about FillFactor? by Gregory Jackson
- Index Basics - SQL Server 2005 Books Online (link doesn't work with FireFox)

Moving Your Database to a New Server

By Paul Mu

Time to Give Your Databases a New Home?

Do you have a server that is underperforming? Do you have end-users constantly reminding you that a server is slow at certain times of the day? Maybe the server has passed its used-by date and requires an overhaul, or maybe that you have a need to implement load balancing by having a second server (or a third). If this (or any other scenario that requires a replacement server) fits you then read on to find out what you need to consider and the approaches that are available to you to have your new server up and running in as little time as possible.

We will assume here that your aim is to have a replacement server, and that this server already has the OS, services and applications installed, which may or may not be an exact replica of the existing server. Therefore, what is left is the task of moving all relevant databases (and other SQL Server related objects) from one server to another.

Understanding What is Involved

Before delving into specifics, there are questions that need consideration upfront. The following is not meant to be an exhaustive list, but gives some idea as to what might be involved:

1. What is your operation's system downtime policy?
2. Is your database server also involved in replication (either as publisher or subscriber)?
3. Is the move between same releases of SQL Server or not?
4. Will the existing server continue to operate or be retired?
5. Will the system databases be moved, in particular the master database?
6. How should all this be implemented? Using scripts and/or SSIS packages?

Means of Approach

The fact that some operations are 12x7 has meant that system downtime is a premium that needs to be considered carefully. The following discussion will principally be concerned with restoring a database to another machine, and how quickly this can be achieved.

Most companies already have a database maintenance plan setup that performs regular full and differential (optional) database backups followed by frequent transaction log backups. Figure 1 shows a remote database restore process that makes use of the full and differential backups only, making it applicable for all database recovery models. Note the serial nature of the process and the sequence to be followed by each individual task. Steps 1 and 4 are tasks to perform against the current system, whereas steps 2, 5 and 6 are tasks to perform against the replacement server. Steps 3 and 7 are dependent on your system downtime policy, which may (in its simplest form) require all affected databases to be set in read-only mode.

The overall process execution time will largely depend on the time taken to perform the differential backup on the existing databases and the corresponding restores on the replacement server. Therefore, the smaller the differential backup the shorter the execution time. This can be achieved by performing the maintenance process as close as possible to the completion of the full backup (shaded regions between steps 1 and 4), thus reducing the size of the differential backup.

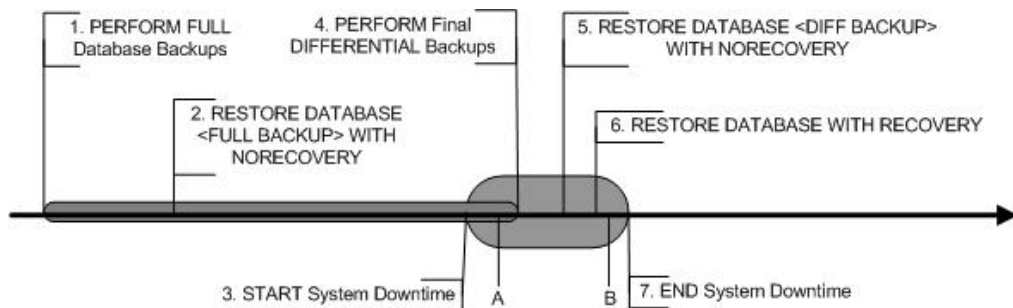


Figure 1. Restoring a database to a remote server using the full and differential backups.

In the event that transaction log backups exist, then the approach would be to perform a NORECOVERY restore of the full, differential (if available) and all transaction logs (in the appropriate order) on the replacement server before the start of system downtime, then after system downtime is initiated, perform a final transaction log backup on the existing system and then to restore it on the replacement server. As with the previous approach, the closer the transaction log backup is to the completion of the differential backup, the quicker will be the backup and restore process.

What happens if you have a replicated database? Well, it depends on whether you are dealing with a publisher or a subscriber. With the former, there is the issue of whether or not the replacement server takes over completely and whether it will have the same name as the existing server. The consequence being that replication may need to be rebuilt on the publisher as well as the need to move the existing server off the network (or offline it). This is a subject for another time, suffice to say that your network admin staff would be help you with such matters.

In the case where you are only interested in moving a subscription database, then the approach is much simpler. The trick is to use a no-sync subscription (to be created at point B' in Figure 1), particularly where very large tables are involved (There are caveats to using a no-sync subscription, but nothing that a DBA cannot handle). A prerequisite for using a no-sync subscription is to ensure that data in both publisher and subscriber databases are in sync. That is, to ensure that all undelivered commands in the distribution database have been delivered to the subscriber before performing the final backup.

The following script (to be executed at point A' in Figure 1) will help with this check, which displays the number of undelivered commands in the distribution database for all articles in all publications on that server.

```
use <publication database>

go

if object_id('tempdb..#dist_status') is not null
```

```
drop table #dist_status

go

select *

into #dist_status

from distribution..MSdistribution_status

go

select p.name as publication, a.name as article, t.*

from #dist_status t

    left outer join sysarticles a

on t.article_id = a.artid

    inner join syspublications p

on a.pubid = p.pubid

where UndelivCmdsInDistDB > 0

order by p.name,a.name,t.agent_id
```

Other Considerations

If you are migrating, say from SQL Server 2000 to SQL Server 2005, then you will need to run the SQL Server 2005 Upgrade Advisor against your server to identify an incompatibilities that will need taken care of beforehand. This includes DTS packages, whose meta-data are stored in the msdb database. There may be a need to rework some of the DTS packages before it will execute in 2005, otherwise look at migrating these packages using the Package Migration Wizard in SQL Server 2005.

If you do not wish to restore the system databases then there are other considerations for you to take note of. One of the most important being the user logins. There are three approaches to this, including (a) manually create the logins using the system stored procedures; (b) use off-the-shelf scripts to generate the logins (e.g. <http://support.microsoft.com/default.aspx?scid=kb;en-us;246133>) ; or (c) use the SSIS Transfer Logins Task. Method (a) will require some extra work to realign the SIDS for the logins, so the approach I would take is either (b) or (c). Then, there are also linked servers, which are easy enough to script out and apply.

Finally, there is a tool that can help with putting your scripts and process together. SQL Server Integration Services (or SSIS for short) is a powerful tool that can be used for such a task. I used it in my last server migration project, and I can say that it beats running scripts in QA (or the Management Studio equivalent in 2005). Once you are familiar with the SSIS interface and know how to configure the various components that you will be using, it is rather fun to design and to deploy.

Conclusion

In conclusion, the aim of this article is to provide you with some approaches that are available to you as you ponder the need to provide a new home for your databases. The focus of the discussion is more on the need to minimize the downtime to your production system. With some proper planning - and this involves doing as much of the work outside of the system downtime window - it is possible to achieve minimal impact to your production systems. Hopefully, there is sufficient information here for you to plan your next database-move project.

Database Snapshots

By Kumar Part

This article is a first part of the three part series. In this series of articles, I will focus on Microsoft's different offerings for SQL Server data availability. In this first part, the focus is on database snapshots, in the second part, the focus will be on snapshot backups and in the last part I will focus on Microsoft Data Protection Manager (DPM).

Recently IT Managers have started talking about the acronyms RPO and RTO. RPO stands for Recovery Point Objective and RTO stands for Recovery Time Objective. RPO can be defined as the acceptable amount of data loss and RTO can be defined as the acceptable amount of business downtime. Lowering RPO and RTO should be an important objective in your backup and recovery strategy.

Microsoft offers following data protective measures to minimize your RPO (data loss) and RTO (down time).

1. Full database and differential backup
2. Granular backups (file level or log level)
3. Log Shipping or Database Mirroring
4. Shadow copying data
 - o Database Snapshots
 - o Snapshot Backup
5. Data Protection Manager

RPO = 0 (or near-zero) and RTO > 0, means some data loss and some business down time. For example, if you perform a point in time restore up to a minute before the disaster, you have achieved an RPO of 1 minute, but your RTO is equal to your recovery time. Conventional recovery strategy (bullet 1 & 2) involving full base and log backups would help you in lowering RPO but not in lowering RTO.

To minimize RTO, you can add log shipping or database mirroring (supported only in SQL Server 2005) in your data protection strategy. If you have a well established standby server, during disaster (example, disk crash) you can promote your standby as production and perform a recovery of original production during off-peak hours. This way you can achieve a smaller RTO. The mirroring solution does protect your data against the physical disaster but does not protect your data against the logical disaster (example, data corruption) because the mirroring solution will simply replicate your corrupted data to a mirror server.

To achieve an infinitesimally small RPO and RTO from the logical disaster, consider using 3rd party transactional log management products such as Lumigent's Log Explorer. If you do not want to pay and maintain a 3rd party product, you can consider using SQL Server 2005 database snapshots. So, conventional backups plus mirroring plus database snapshot will protect your data against the physical and logical disasters.

Microsoft is also working towards addressing low RPO and RTO through another product called Data Protection Manager (DPM). The current DPM offerings do not support SQL Server or Exchange.

Database Snapshots

SQL Server 2005 Database snapshots use NTFS sparse files with the Alternate Data Stream (ADS) mechanism. In this scheme, before updating the original data block in a file, it will be copied to the respective sparse file; this concept is called copy-on-write. A database snapshot operates on a data file level, so for each data file of a database 'db' you have to associate a sparse file to create a successful database snapshot. The sparse file mechanism is supported in NTFS but not in FAT32. If you delete the original file, the sparse file will also be deleted. This mechanism is also used in certain DBCC commands such as DBCC CHECKDB, DBCC CHECKTABLE and DBCC CHECKALLOC

In the sparse file system, storage is not fully allocated as in the original data file. When you create a snapshot it does not take much storage, but it grows when your disk blocks change in the original file. So, the older the snapshot is, the larger the disk space requirements (assuming your database changes regularly).

SQL Server Management Studio does not support creating and managing the snapshots. So, you need to rely on T-SQL for creating, reverting and deleting the snapshots.

To create a snapshot database *sample_snap1* for the *sample* database use the following query.

```
CREATE DATABASE [sample_snap1]

ON ( NAME=[sample]

,FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\sample_snap1')

AS SNAPSHOT OF [SAMPLE]
```

The above query takes logical name of the *sample* as the database's data file. To get logical name of data and log files use the following query

```
USE SAMPLE

SELECT [name] FROM sys.sysfiles
```

If you try creating a snapshot, when the original database is in the middle of running a transaction, you would get the following error

```
Server: Msg 226, Level 16, State 5, Line 2

CREATE DATABASE statement not allowed within multi-statement transaction.
```

After the successful creation of a snapshot, you will see the snapshot database in the object explorer like any other SQL Server database. Also, *sys.databases* field *source_database_id* will point to original database id.

Transactional log files, offline files or files in loading state are not considered for database snapshots because a database snapshot is read-only and a snapshot must always be created in a consistent state. You can compare the amount of disk space that is used by normal and snapshot database by using the following query.

```
SELECT BytesOnDisk

FROM fn_virtualfilestats(DB_ID('sample_snap1')

,FILE_ID('C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\sample.snap1'))
```

The above query returned 192K for the snapshot database and 1.18MB for the original database. If you make some updates to original database, size of the snapshot database will grow. For example, if you add a file to your *sample* database, as shown by the following query

```
ALTER DATABASE [sample]

ADD FILE (NAME=sample1, FILENAME='C:\Program Files\Microsoft SQL

Server\MSSQL.1\MSSQL\Data\sample1')
```

Note, ALTER DATABASE command is not allowed on database snapshots. You cannot alter the snapshot; you need to newly create one, including all the file specifications as shown below.


```
CREATE DATABASE [sample_snap2] on

(NAME=[SAMPLE1],FILENAME='C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\sample1_snap2'),

(NAME=[SAMPLE],FILENAME='C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\sample_snap2')

AS SNAPSHOT OF [sample]
```

If you do not specify a file specification, you will get the following error

```
Server: Msg 5127, Level 16, State 1, Line 1

All files must be specified for database snapshot creation. Missing the file "sample".
```

One of the key benefits of a snapshot is guarding against data corruption. If you are planning to perform a bulk copy or concerned that your data more prone to corruption, consider taking frequent snapshots. Note having too many snapshots will impact your production environment performance because snapshot access requires access to the original database.

Now, create a table called *mytab* in the sample database and insert some rows.

```
CREATE TABLE mytab(c1 int, c2 varchar(12))

go

INSERT INTO mytab VALUES(1,'hello')
```

Take a snapshot *sample_snap3* from *sample*.

If the original table is truncated, you can recover the original table data using the following query.

```
INSERT sample..mytab SELECT * FROM sample_snap3..mytab
```

If the original table data is updated as shown below,

```
UPDATE sample..mytab SET c2='corrupted' WHERE c1=1
```

You can recover from the corruption using the following query

```
UPDATE sample..mytab SET c2=(SELECT c2 FROM sample_snap3..mytab WHERE c1=1)
```

If your table is huge, the above described two methods of data recovery might take longer than recovering (reverting) the original database using one of the snapshots. Before you recover (or revert) to a snapshot, you have to delete rest of the snapshots.

So, I drop *sample_snap1* & *sample_snap2* snapshots

```
DROP DATABASE sample_snap1

DROP DATABASE sample_snap2
```

Then recover to *sample_snap3* using the following query

```
RESTORE DATABASE sample from DATABASE_SNAPSHOT= 'sample_snap3'
```

Note you cannot drop or detach the original database unless you drop the snapshots associated with it. SQL Server uses a reference counting mechanism and prevents you from accidentally dropping the original database.

Reverting to a snapshot is not supported, if

- The source database contains any read-only or compressed files
- Any files are offline that were online during snapshot

During the revert operation, both the original and snapshot will be marked as "in restore". Before reverting, it is recommended you perform a log backup and after reverting, perform a full database backup because reverting breaks the log chain.

You cannot backup database snapshots, therefore you cannot restore them. Though a database snapshot protects your data from logical disaster, you need to note the following limitations.

- The original database performance is reduced due to increased I/O between original database and snapshots whenever a page is updated. You can minimize this by limiting the number of snapshots.
- The source database is not scalable with snapshots
- Reverting will recover your data from corruption, but you have to perform the full database backup following the revert operation.
- Snapshots will take up your disk space. So consider deleting older snapshots, if they are not needed.

Understanding these limitations and exercising caution will definitely minimize your Recovery Time Objective.

Database Snapshots in SQL Server 2005

By Arvinder Khosla and S. Srivathsani Murthy

Introduction

Database snapshots is a new feature added in SQL Server 2005. Database Snapshot can be used to create a read-only copy of the database. Database snapshot consists of the read-only static view of the database without including the uncommitted transactions. The uncommitted transactions are rolled back to make the database snapshot transactionally consistent.

Uses:

Database snapshots can be used for

- Protecting your system from user or administrator error
- Offloading reporting
- Maintaining historical data
- System upgrades
- Materializing data on standby servers
- Recovering data

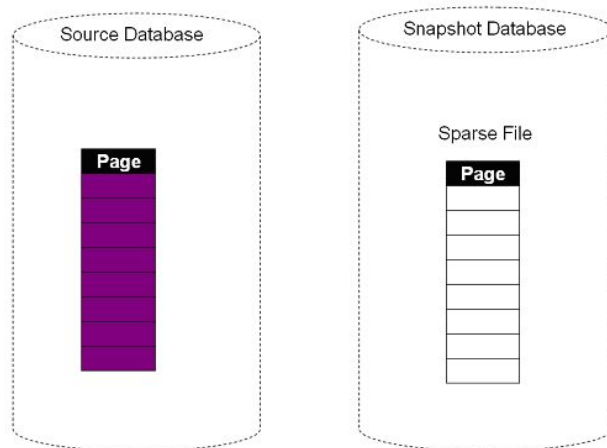
How Database Snapshots Work?

Database snapshots occur at the page level. Before a page of the source database is modified for the first time, the original page is copied from the source database to the snapshot. This process is called a copy-on-write operation.

The snapshot stores the original page, preserving the data records as they existed when the snapshot was created.

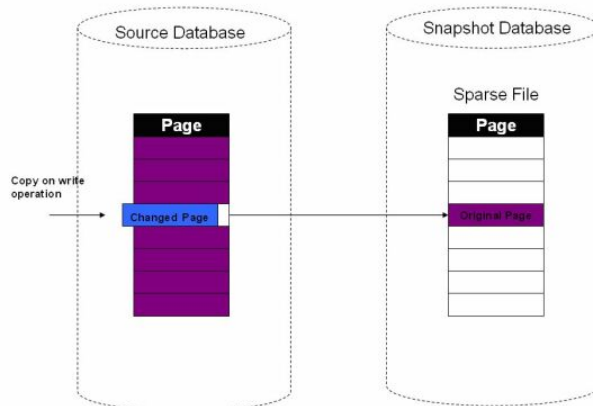
To store the copied original pages, the snapshot uses one or more sparse files. Initially, a sparse file is an essentially empty file that contains no user data and has not yet been allocated disk space for user data. As more and more pages are updated in the source database, the size of the file grows. When a snapshot is taken, the sparse file takes up little disk space. As the database is updated over time, however, a sparse file can grow into a very large file. This is illustrated with the help of this diagram:

1. At first when the snapshot is created, the sparse file is empty.

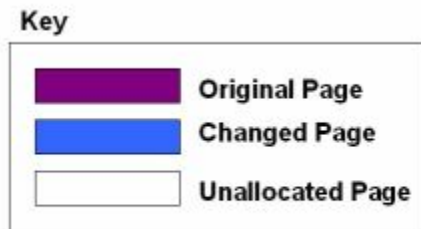


Case 1: When the database has no updations since the snapshot was taken

2. When the first update operation happens, the original data is copied into the sparse file. The database engine then updates the data in the source database.



Case 2: When the database has updates since the snapshot was taken



When the first update operation happens, the original data is copied into the sparse file. The database engine then updates the data in the source database. During the read operation on the snapshot, the original data is read from the source database. For the data which has been changed, the snapshots read from the sparse file.

↑ Advantages

1. The user can create as many snapshots as he/she wants quickly in no amount of time. The user can schedule to take snapshots every hour. This will be useful in auditing scenarios
2. The snapshots can be used in restore operations.
3. The corrupted or deleted data can be recovered from the snapshot to repair the primary database.
4. In case of user error, the administrator can revert back to the snapshot taken just before the error.

↓ Disadvantages

1. Database snapshots are available only in the SQL Server 2005 enterprise edition.
2. Database snapshots are dependent on the primary database. If the primary database goes offline, then the snapshots are not accessible.
3. Performance of the source database is reduced, due to increased I/O on the source database resulting from a copy-on-write operation to the snapshot every time a page is updated.
4. Full-text indexing is not supported in database snapshots.
5. If the data changes rapidly, the snapshot might run out of disk space.

Creating a Database Snapshot-TSQL

In the **PUBS** database we have 3 data files under 2 file groups:

1. **Primary File group** contains Pubs and pubs_logicalname
2. **Pubs_data filegroup (Secondary Filegroup)** contains Pubs_2

The statement below creates a snapshot on the **PUBS** database

```
CREATE DATABASE pubs_Snapshot1800 ON

(Name=pubs,Filename='C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\pubs_data1800.ss'),

(Name=pubs_logicalname,Filename='C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\pubs_logicalname_data1800.ss'),

(NAME = pubs_2,

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\pubs2_data1800.ss')

AS SNAPSHOT OF pubs
```

In pubs database, there are three files. The user should include all the three files while taking the snapshot. The database is a combination of data present in all three files. So we need to mention the all the three logical file names, so that all the data from these files will get stored in the location 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\filename_with_timestamp.ss'. In the example given above,

- pubs_Snapshot1200 --> Snapshot Name
- pubs --> Logical File Name
- C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\Northwind_data1000.ss --> Physical File Path

Suppose if the user wants to create the snapshot at 2400 hrs then he has to just change the name of the database and the physical file name.

```
CREATE DATABASE pubs_Snapshot2400 ON

(Name=pubs,Filename='C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\pubs_data2400.ss'),

(Name=pubs_logicalname,Filename='C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\pubs_logicalname_data2400.ss'),

(NAME = pubs_2,

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\pubs2_data2400.ss')
```

The snapshot can be used just like the database. The user has to remember that the snapshot is a read-only copy of the database. So he/she cannot issue the update, delete or insert commands.

Viewing Snapshots from SQL Server Management Studio

1. Expand Databases.
2. Expand Database Snapshots.
3. Select the snapshot created in the above section

Restoring from the Database Snapshot

Consider the database Pubs. Suppose the user deletes dbo.table_1 from the Pubs database. We have to recover it from the database snapshot by issuing the following command.

```
Restore database pubs from database_snapshot = 'pubs_Snapshot1800'
```

Dropping the Snapshot

```
Drop Database 'Snapshot_Name'
```

Frequently Asked Questions on Snapshots

Question 1. How is Snapshot different from the full backup of the database?

Answer 1. Differences are as follows:

1. In case of backup, we have to restore it somewhere to actually use it. The user cannot query the backup file. On the other hand snapshots are static read only data at that point of time. So it can be used just like any other database.
2. Another advantage or difference that snapshots have over backups is that backups occupy lot of space.
For example: Suppose if there is a database in which only few updates takes place. Then definitely snapshot is a better option because less data has to be stored in the sparse file.

If we remember that in case of snapshots, only the pages which are updated are stored in the sparse file. When we fire the select query, the snapshots will read the pages which are not changed from the source database and the changed ones are read from the sparse file. Thus snapshot proves more advantageous than full backups in these scenarios.

Question 2. Is ss the default extension of the data files?

Answer 2. The default extension is not ss. It can be anything

Question 3. When to use Snapshot and when to use Full backups? And are differential backups influenced / dependent on Snapshot?

Answer 3.

Backups and Snapshots are two different concepts. Snapshots are definitely NOT a replacement for backups. Snapshots should be used for auditing or reporting purposes. Recovery operations can be done with the help of snapshots. But it is not the primary resource. Snapshots provide a read only view of the database at that point of time. It does not include uncommitted transactions. When we take backups, the uncommitted transactions are also included in the backups.

Differential backups are not dependent on snapshots. Differential backup is the differential of the last full backup and the current state of the database.

Got more questions?

Please mail us ([Arvinder Singh Khosla](#) or [Srivathsani Murthy](#))

Customizable Error Log Scanning

By Thomas LaRock

Background

The SQL Server error log has a wealth of information contained inside. Sometimes it is too much, sometimes it is not enough. And then situations occur when you do not know if there is enough in the log or not until well after you have a resolution. In my opinion, things are even more complex with 2005, as there seems to be a lot of messages that end with "This is an informational message only; no user action is required." If that is the case, then I may or may not want to be alerted about this, right? User action may not be "required", but what if it is actually "desired"?

Like many shops, we developed an in-house system for monitoring our database servers. Built over the past two years, the system is called DBA_Perform. One of the very first items we incorporated into DBA_Perform was the error log scan. Code for all of the tables and stored procedures for the solution can be found in the downloadable zip files. There is also a readme file included to assist with your implementation. I have tried to keep things as general as possible, but some additional modifications may be necessary based upon your specific environment.

The Need To Know

I believe that many shops have processes in place to scan the logs for review. Some shops are small enough (less than ten total servers) that no process is really necessary because they can simply review the logs by hand as needed. However, I would argue that using an automated process to scan the log periodically is the smart thing to do, so that you can be alerted should something occur. That is exactly what we set out to build over two years ago.

What we found was that setting up a process to scan the error log is very easy to do. There were lots of examples available from community sites. We sampled some and found that while they did work, there was always something missing. What one solution would have, another solution would not. So, we took the best of everything we could find, added in our own ideas, and ended up with the solution being presented. What set ours apart from others at the time was the ability to include (or exclude) certain keywords that would appear in the error log.

This was very important for our group as the number of servers we managed went from around ten to over seventy in less than fifteen months. The combination of production, test, and development servers also meant that the errors we would be concerned about varied. For example, our production servers are set to log any failed logins, but test and development are not. Any solution we would roll out to a new server had to work regardless of environment. But do we want to be alerted by email each and every time someone fails to log in? No, probably not. But what if that failed attempt was by someone trying to login as "sa", or "admin", or "root"? That could be something that we want to know about quickly, as it could be an attack or a virus.

The Magic

First, get the code from sqlservercentral.com. This has been tested with SQL Server 2000 and SQL Server 2005.

We needed to build a process that would scan and dump the contents of the error log, clean up the contents based upon keywords we wanted to include and/or exclude, and then send us an email if necessary. The procedure we currently use is named `Usp_error_log_Notification`, and is included in the zip files. The procedure makes use of the `xp_readerrorlog` extended stored procedure in order to gather all details from the log into a temporary table as an initial starting point. The process does a check to see

what version of MS SQL Server you are running, because Microsoft changed the output of the xp_readerrorlog extended stored procedure in SQL 2005.

While checking the version, we also verify that the reading of the error log results in a valid date that will ultimately be inserted before our emails are sent. This is essential because the very next step does a calculation of the maximum date, so we need to make certain the date is valid in order for the max() function to return a proper result. If this is the first time a scan has been run, then a default date is assigned. Otherwise, it will retrieve the max date from an earlier run of the process from the error tables and that date will be used to filter out rows from the temporary tables later in the process.

Before that cleanup happens, a cursor is built to go through the entries in the temporary table and filter out the rows that do not have a keyword that we want to scan for. The keywords for this part are found in the ErrorlogScanKeyword table, and include words and phrases such as 'stack', 'is full', and 'error'. If a row exists in the #Errors temporary table that do not have any of the words listed in this table, then they are filtered out.

After filtering the temporary table, leaving behind only rows that have keywords we are scanning for, we then cleanup the table by using the calculated date mentioned in the previous paragraph. As our #Errors table gets smaller, we next need to force the inclusion of specific words. We take the current rows in #Errors and insert them into a separate temporary table named #ErrorsHOLD. At this point we remove rows from #ErrorsHOLD that do not have words and phrases found in the ErrorlogForceInclude table. In our experience, we found that is not sufficient to only exclude certain words, there also needs to be the forced inclusion of keywords as well. By forcing the inclusion of certain keywords (for example, the word 'admin' would be a forced include), we can tell the system to be more flexible. This is the magic. We can tell the system to not bother us every time the phrase "Login failed for user", but to make certain we know whenever "Login failed for user 'sa'" appears in the error log.

After setting those rows off to the side, we do one final scan on the #Errors temporary table to exclude certain keywords outright (for example, certain error numbers above 50000, and this is where we would exclude the generic 'Login failed for user'). From there, it is a quick reinsert into the #Errors from #ErrorsHOLD and we have the desired result set. The last step is to generate an email and notify our team that there is something in the error log that may be worth further investigation.

Create the Job

The last thing we needed to do was to put together a job that would run every ten minutes. We created a job that simply calls the Usp_error log_Notification procedure (also included in the zip file). Feel free to modify the job as you see fit, changing the frequency of the schedule or whatever you desire.

Conclusions

As your environment gets more and more complex, you may find that the number of alerts increase substantially. Some of the processes that you build while you have a small shop may not be as effective as more and more servers come under your control. If you start with a customizable toolbox you can save yourself some headaches later. With the above solution, you can decide for yourself what you want to be notified about during an error log scan, and not have to worry about filtering through the white noise that we all seem to be swimming in these days.

A Guide to Application Memory Tuning

By Paul Mu

This article will provide the DBA with appropriate background information and links to resources in order to have a good understanding of application memory tuning, with specific application to SQL Server 2000 EE under Windows Server 2003.

Lets start by defining some terms that are important to our discussion. A **process** is an instance of a running application. A **physical storage** consists of the physical memory (RAM) and system paging files. A **virtual memory** is defined as a range of **address space** (or pointers) with no associated physical storage.

For a 32-bit Windows operating system, Windows allocates **all processes** a 4GB address space. This 4GB limit is because a 32-bit pointer can have a maximum of 2^{32} values, whereas a 64-bit system will be able to handle 2^{64} values, a very large number indeed! The 4GB address space is divided into two partitions, a **user mode partition** and a **kernel mode partition**. By default, each of these is sized at 2GB. It is however possible for a process to use more than the default 2GB of addressable memory. The /3GB switch offers such an option.

The /3GB Switch

The /3GB switch, when configured in Windows, will expand the user mode address space for a process from 2GB to 3GB. The down-side is that the kernel mode address space must also decrease to 1GB in order to preserve the 4GB limit for a 32-bit OS. To configure this in Windows, add the /3GB switch to the appropriate line in the BOOT.INI file, or follow the steps in this link for Windows Server 2003 (<http://support.microsoft.com/kb/317526>).

The /3GB switch is only applicable to systems having between 2GB and 4GB of configurable RAM. Where more than 4GB of memory is available, Windows can be configured to use the /PAE switch instead.

The /PAE Switch

PAE stands for Physical Address Extension, and according to Microsoft '*PAE is the added ability of the IA32 processor to address more than 4 GB of physical memory*'. To enable PAE you add the /PAE switch to the appropriate line in the BOOT.INI file.

It is also possible to use the /PAE switch in conjunction with the /3GB switch. However, in this instance, the maximum memory that windows can manage is limited to 16 GB due to the **kernel mode partition** being reduced to only 1GB of addressable memory.

Not all Windows OS support these switches. According to the following article (<http://support.microsoft.com/?kbid=291988>), the **/3GB** and **/PAE** switches in the Boot.ini file are to be used with the following products: Windows 2000 Server (Advanced and Datacenter editions) and Windows Server 2003 (Enterprise, Datacenter and Small Business editions). Other scenarios where the switches are supported are for testing purposes only.

Configure SQL Server to Use More Memory

According to Microsoft *AWE is a set of APIs to the memory manager functions that enables applications to address more memory than the 4 GB that is available through standard 32-bit addressing*. The important thing to note from this statement is that AWE is related to a process (or application), and that not all applications or versions of an application are AWE-aware.

Further information on the particular version of SQL Server and the underlying Windows Operating System that support this type of memory configuration, as well as instructions on how to enable AWE on SQL Server, can be found in this website (<http://support.microsoft.com/?id=274750>). For SQL Server 2000 EE with SP4, there is a fix for AWE which you can download from this website (<http://support.microsoft.com/?kbid=899761>).

Once you have enabled AWE on your SQL Server, you can set an appropriate min/max server memory thresholds. Understand that if the operating system does not have enough memory it will request memory from SQL Server, thus setting an inappropriate min/max server memory will impact on the performance of SQL Server. For a dedicated database server, the default memory settings for SQL Server should suffice. Note that a value of 0 for min server memory simply means that SQL Server will dynamically manage its memory.

Conclusion

For certain applications running under various Windows operating system, it is possible to tune the memory used by both the OS and the application. For the OS, the two configurable switches are the /3GB switch and the /PAE switch. Once configured, an AWE-aware application (such as SQL Server 2000 EE) can be enabled to access memory in excess of 4GB.

References

- A highly recommended book for the serious DBA is *The Gurus Guide to SQL Server Architecture and Internals* by Ken Henderson.
- Further information on Memory Configuration Options for SQL Server can be found at this website: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/optimsq/odp_tun_1a_2f3b.asp
- Further information on PAE can be found at this Microsoft support website: <http://support.microsoft.com/default.aspx?scid=kb;en-us;283037>

Eliminating Tape

By Tim Opry

Disk-based Backup in a Small-Business Environment

Overview:

This article is a follow-up to a recent editorial by Steve Jones (<http://www.sqlservercentral.com/forums/shwmessage.aspx?forumid=263&messageid=326270>) on this topic. There have been numerous trade journal articles over the past two plus years about the efficacy of using inexpensive IDE or SATA disks for backup vs tape. The purpose of this article is not to rehash those well documented pros and cons, only to describe our transition to a home-grown disk based solution.

Environment

Our company is a small financial services firm with approximately 50 employees that utilize a combination of thin clients and traditional PCs all running MS Windows. The servers are a combination of rack-mounted Dell PowerEdge servers which are being replaced as they reach their EOL with SuperMicro 1U servers, all of them running W2k or W2k3. While it is beyond the scope of this article to discuss why we have migrated from Dell to a white-box solution, it has been my experience over the past many years that if you have the technical expertise in-house, a SMB is much better served by this type of solution than with the major vendors (Dell, HP or IBM). Even with the highest level of warranty/maintenance offered, SMBs simply do not have the buying power to get the attention of those firms. Additionally, the price differential allows you to provide a more robust solution for the average SMB back-office environment.

From late 2001 until late 2005, we had been using a traditional backup methodology with two LTO (rev 1) tape devices (DAT prior to that). Each device was capable of 120GB (compressed) and data was backed up using the latest version of Backup Exec with appropriate agents for SQL, Exchange and open files. We used a traditional tape rotation sequence and stored a full set of backup tapes at an offsite facility once per week.

As the business grew and with the advent of SOX (Sarbanes Oxley) as well as our own internal disaster recovery requirements, the backup window and protection provided by the tape devices was reaching the limits of that solution (both in space and the length of time in the backup window).

Objective

To create a backup solution that would exceed the governmental regulations as well as our internal requirements and be flexible enough to grow and improve our disaster recovery / restoration requirements. With tape, our exposure for data loss in the event of total system failure or a disaster (assuming the most recent tape was 100% accessible) was a minimum of 1 day up to a potential of 1 week. This window did not apply to the actual financial data which is maintained separately by the custodians, only the normal day to day, routine business data. Regardless, we wanted to reduce the exposure to a fraction of that, ideally to 1 hour or less (depending on the type of data).

We had been using software imaging for several years. Originally in the form of Ghost to push desktop images to clients to decrease downtime in system failure and later to the original Power Quest V2i solution for servers. (PQ was later acquired by Symantec [as was the company that made Ghost], and this solution was recently re-branded as part of the BackUp Exec family). I had tested the PQ product in the server environment and it worked well, but was limited to restoring to like hardware. While this was acceptable for the majority of system failures we were likely to experience, it would not meet our goals for disaster recovery. The thought of having to reinstall and reconfigure 6 servers with all of the associated patches, was not something I wanted to face. The effort of simply keeping the documentation current for that possibility, was extremely time consuming and tedious.

Solution:

This has been an evolving solution for the past 24+ months and it will continue to evolve and change as hardware and software solutions improve. The most notable improvement to backup systems that I have seen is Continuous Data Protection (CDP) solutions. At the time I was researching this almost 2 years ago, there were very few options in the CDP field and all were outside our budget range. Since that time, there are several vendors in this space (including an MS solution) and the price point is becoming feasible for an SMB. As such, we are looking at including CDP to compliment our current methodology.

Since we have the requisite IT experience in house and the market was evolving rapidly due to both technology and acquisitions (Symantec buying Ghost, PQ, Veritas, etc), I chose to start with a homegrown solution to validate the plan and then add to and improve it over time.

STEP 1: Purchased an inexpensive 4 disk IDE JBOD enclosure from Promise Technologies and connected it to our existing backup device (a 1U Dell NAS) via SCSI.

We installed 4, 400GB (largest available at the time) IDE drives and configured it as JBOD, giving us a 1.6TB device (1.45 usable). This was to be used as nothing more than incremental storage, so the amount of space was more important than RAID 5. Total cost (2 years ago): Under \$1500 (not including the NAS). Of course this same solution would be much less expensive today and have a greater capacity.

Using the imaging software (V2i) we created weekly base (full) images directly to the JBOD device. We then pulled the images to our LTO tape drives for offsite storage. Since our total compressed image size at that time was under 120GB, we could store over 2 months worth of weekly images on the JBOD device and move an entire weeks worth of images onto one LTO tape cartridge.

While using server images improved our restoration time in the event of a normal failure, in the event of a disaster where all hardware was lost, we were still likely to be facing a full OS installation with data restore, which while an improvement, failed to meet all of our goals.

STEP 2: Approximately 1-year later, Symantec added the ability to create incremental images. This would allow us to easily capture incremental changes on an hourly (or less) basis and reduce the likelihood of data loss from 1 day to 1 hour or less. However, we still faced the daunting task of OS reinstallation in the event of total system loss.

STEP 3: This year (2006), Symantec (and others) added the ability to restore an image to totally different hardware. In the event of a total system loss, we can bring up a system on most any hardware that is capable of running the imaged software vs being limited to sourcing hardware that is technically similar to the system lost (or keeping spare parts for emergencies).

To test this, we purchased a SuperMicro 1U server with two SATA-II 250GB drives, configured as RAID-1 with a third drive configured as a hot-spare and redundant power supplies. We had a 2U Dell PowerEdge that was almost 5 years old that had reached EOL. This was a 5-drive SCSI system with the OS on two drives configured as RAID 1 and the data partition on 3-drives configured as RAID 5. The new system had twice the total disk space and was a much higher performance system that cost less than 50% of what a comparable system would cost from Dell or HP IF you could get a similar configuration which you cannot. The Dell system hosted W2k server configured as a Terminal Server.

We first migrated the OS and Data partitions (C and D logical devices) to the new hardware. After a phone call or two to Symantec, (their documentation at the time had no details on how to do this) the system came up and ran flawlessly. We documented the process, wiped the drives clean and repeated the installation in less than 1 hour!

We then tested upgrading this system to W2k3 knowing that if we had any issues, we could take it back to its original state in less than 1 hour. The upgrade ran without problems and the server has been running without any problems or a single event log error for several months.

STEP 4: Get rid of tape! With the increasing size and speed of SATA devices and decreasing prices we purchased a small 4 drive hot-swappable external SATA-II device from Addonics along with a PCI card and several removable drive bays that allowed us to connect it to a standard PC. Now instead of pulling the images off to tape each week, we copy it across our LAN onto a single 250GB SATA II drive, which is then removed and taken offsite. The SATA tower, card and removable drive bays were around \$800 and the cost of the drives only gets cheaper. Granted, you can buy solutions from Dell, HP, Iomega and many others that do these same tasks, but you will pay appreciable more for the same functionality.

All of this process is automated using a batch file (some habits die hard) that first deletes the oldest backup set from the JBOD and then creates a folder for the current set and copies the images and incrementals into a new folder named for the current week. The imaging software creates a new base/full image of ALL drives on all of the servers which currently totals about 175GB. Since this is run over the weekend over a gigabit switch, the total time to create all of the images is less than 8 hours. A copy of the latest images is then moved to the removable SATA drive which is sent offsite first thing Monday morning. I also use some in-house developed software to copy the images to the SATA drive and check for errors and send notifications via SMS if problems are detected. (All backup data is both password protected and encrypted.)

Since the most likely catastrophic event we will experience is a fire and secondarily theft, we needed a way to move data offsite more frequently than once per week or we still faced the possibility of a one week loss. We accomplish this by transferring the incremental images offsite via secure FTP. Fortunately, the amount of data we have to push offsite hourly is relatively small and can be accomplished with our current bandwidth.

Going forward, as we look at integrating CDP into this solution we will also explore moving the incremental CDP data offsite in real-time as well. In theory this would give us a solution that until recently would be impossible for a SMB. In the event of a total loss, we can purchase off the shelf components from local vendors and have basic services restored within 24 hours (assuming hardware availability). I have also tested bringing up base images as virtual machines. VMWare supports the V2i disk format and I have successfully booted several of my images as virtual machines this is a great timesaver for testing.

There are some limitations and concerns with imaging software, specifically as it pertains to SQL, Exchange and AD stores (transactions getting out of sync). These limitations are well documented by the imaging vendors and we do still backup this data separately each and every day using more traditional (agent based) solutions. The data is still stored on disk and moved offsite with the images, however I have restored all of these from full and incremental images without problems in both test and migration scenarios. However, I do understand the potential for failure and this is yet another reason for my wanting to add CDP to compliment this solution.

Final note

This is meant only to give you a brief overview of our current approach. If you have suggestions on how we can improve this (within the constraints of a SMB budget), I welcome any and all feedback.

There are several areas that I did not mention that are part of our disaster recovery plan and current backup methodology. Those were excluded primarily as they were not relevant to this context or due to security concerns. Additionally, I have no affiliation with any of the vendors mentioned. There are many that make these same devices and software. The ones mentioned are simply those I chose at the time and continue to use.

About the author: I have been involved with computers since the dark ages learning how to program originally in Fortran using lined paper and punch cards. I have been the partner/co-founder of 3 different software startups, each having varying degrees of success. Currently in semi-retirement, serving as the ersatz CTO for The Henssler Financial Group in Atlanta, GA.

Full Text Search Follies

By Robert Pearl

The following story you are about to read is true. In no way is the attempted humor meant to mask the seriousness of the issue, nor harm, insult or embarrass anyone, except perhaps myself. I write this article to share my experiences and give you a heads up as to what you might encounter during a database migration from SQL 2000.

One of the most common ways of migrating our databases from SQL Server 2000 to 2005, is the old trusted method of backup and restore. SQL Server 2005 allows us to restore a 2000 database, and gives us the option of bringing the compatibility level up to 9.0 (2005). Therefore, it is perhaps one of the quickest, simple, and reliable ways of migrating the database, ensuring that all data, objects, and users come over.

I know that I have migrated many SQL Server databases in this manner. However, until recently, I ran into some full-text search (FTS) issues with a particular database where a full-text index and catalog was created on the database, but never really utilized in production. Nonetheless, several entries with respect to this Full-Text definition were made into the system tables, and caused some major headaches.

What happened next ranges from the ridiculous to the sublime. Either occurrence I am about to describe is unacceptable, and certainly would be considered a show-stopper to deploying your SQL2K5 instance into production. First, take notice of the fact that in SQL 2005, FTS is installed as a separate service/component. Depending on how you specify the start-up option during the install, usually it will show up in the control panel as 'Automatic' startup.

Quick background on our server resources. We're running SQL Server 2005 64-bit Standard Edition SP1; Windows Server 2003 R2 x64 Enterprise Edition, 4 Dual-Core Processors (with hyper-threading that's 16 logical CPUs), with 12 GB of purely addressable RAM. One powerful box!

In our environment, when we first attempted to restore the 2000 database to our new 2005 server, we watched the restore complete successfully, and then the full-text engine kicked in attempting to rebuild the catalog on the database. Here we watched helplessly, as if witnessing a tsunami approaching but unable to get out of its way. We monitored the CPU usage and saw it climb incrementally from 0 to 100% flat-lining across all processors, maxing out memory and page-faulting all over itself. At that point, crucial migration on the line, we checked our pulses as well!

Fortunately, breathing resumed and we were ok, but no such luck for the server. It was hung, and any attempt to regain control was unsuccessful, leaving us no choice but to reboot the machine. By the way, we did try one of SQL 2005's newest features, the dedicated administrator's connection DAC "BOL: a special diagnostic connection for administrators when standard connections to the server are not possible. It allows an administrator to access SQL Server to execute diagnostic queries and troubleshoot problems even when SQL Server is not responding to standard connection requests." (In our case, Do not Attempt Control :-). However, please **do** remember to enable the DAC via Surface Area Configuration for Features, as it might come in handy since it's not enabled by default oops!

The good thing about SQL Server 2005 is that with the help of its very extensive and verbose error logging, we very quickly realized where the problem was, and identified that Full-Text Search was indeed the *causus-belli*. We even were able to reproduce the behavior at-will. I'd say we're now resident experts on how to crash the mightiest of

servers. One message appearing in the SQL FT log clued us in, "The full-text catalog monitor reported catalog 'MyCatalog' in database 'MyDB' in REINITIALIZE state. This is an informational message only. No user action is required." No kidding! All joking aside, all the above was useful for our support call with Microsoft.

Upon further review of our logs, we noticed this: "Warning: No appropriate filter was found during full-text index population for table or indexed view '[MyDB].[dbo].[MyTable]' (table or indexed view ID 'X', database ID '1784393426'), full-text key value 0x0001E119. Some columns of the row were not indexed" So, it appeared that we had some sort of corruption with the full-text index, and the problem likely to be with the Filter that is not available for FTS.

Once we finally got the situation under control, where we would disable the FTS service temporarily while working to resolve issue, we proceeded to create a simple backup plan to regularly backup our newly migrated 2005 database. To our horror, when the backup kicked off it failed straight away, and we receive the error:

"Error message when you perform a full backup of a database in SQL Server 2005: Backup failed for Server 'ComputerName\SQLInstanceName' or The backup of the file or filegroup 'sysft_FullTextCatalog' is not permitted because it is not online"

So, now this was personal. How can we deploy this database and application in production when we can't even take a backup?! Fortunately, the aforementioned error appears to be a known issue at Redmond, and has a KB-article on same: <http://support.microsoft.com/kb/923355/en-us>.

According to the article, "This behavior occurs because a full-text catalog in the database is not online. To perform a full backup, SQL Server 2005 requires all the database files and full-text catalogs in the database to be online." Great! So, in order to be able to backup the database, our choices were 1) enable FTS, and therefore crash the server, or 2) hey, who needs a backup anyway?

Although the article gives various potential causes and resolutions, it doesn't seem to clearly indicate what commands, and in what order, to execute to get around the issue.

For example, the KB-article suggests, "If you do not need the full-text catalog any longer, you can drop the problematic full-text catalog. Then, perform a full backup of the database in SQL Server 2005." Ok, in our case we no longer needed the catalog. Whew! That sounds simple enough. So I proceed with the system stored procedure: **`sp_fulltext_catalog 'AdiosMyCatalog', 'drop'`**. Result: **"Full-text catalog 'AdiosMycatalog' has been lost. Use `sp_fulltext_catalog` to rebuild and to repopulate this full-text catalog"** Lost!? I don't remember losing it anywhere. I looked around for it, but no luck.

Back to the KB-article for more clues:

The full-text catalog folder is either deleted or corrupted.

You did not enable the database for full-text indexing.

The database is restored from a Microsoft SQL Server 2000 database backup. Therefore, the folder of the full-text catalog in the database does not exist on the server where you restore the database.

Sounds plausible. Full-text is *not* enabled on the database, and it was restored from a MS SQL 2000 backup. Must be getting warmer. I figured I'd do as the error msg says (not as the error msg does), and attempted to rebuild the catalog (thinking it would at least refresh any values it needed, to then be able to successfully drop the catalog.) Upon query execution, I get my confirmation:

"Full-Text Search is not enabled for the current database. Use `sp_fulltext_database` to enable Full-Text search." A no-brainer. Use MyDB, `sp_fulltext_database 'enable'` slam dunk what the !@#%\$? **"Full-text catalog 'AdiosMycatalog' has been lost" Again??** Seems now that I was lost too.

After some research on the 'Net searching for fellow survivors of "Lost" no, not the TV show (<http://www.tv.com/lost/show/24313/summary.html>), I came across a potential solution in the MSDN forums. Suggested was the following:

Look at the sysfulltextcatalogs table in your database: `SELECT * FROM sysfulltextcatalogs`. If there is an entry, get the Ftcatid and look your sysobjects:

```
SELECT * FROM sysobjects

WHERE ftcatid IN (SELECT ftcatid FROM sysfulltextcatalogs)
```

OR

```
SELECT * FROM sysobjects WHERE ftcatid > 0
```

If you encounter the objects, update the sysobjects table:

```
UPDATE sysobjects SET ftcatid = 0

WHERE ftcatid > 0

or IN (SELECT ftcatid FROM sysfulltextcatalogs)
```

Well, I had nothing to lose, since I already lost my catalog ok, enough with the "lost" jokes. I certainly won't forget the wisdom of the poster who warns us in red letter italics (I kid you not): "*But, remember Backup Full before [running] these procedures*" I took my chances. After executing the above statements, I proceeded to run the below system stored procs, in the following order:

```
sp_fulltext_table 'SoLongMyTable', 'drop'

go

sp_fulltext_database 'enable'

go

sp_fulltext_catalog 'HastaLaVistaMyCatalog', 'drop'

go

sp_fulltext_service 'clean_up'

go
```

And, alas, no error messages! All was not lost. :-P Definitely, a positive sign, but now for the ultimate test the database backup. Right-click >> Tasks >> Backup. After several minutes, I was satisfied that indeed the issue has been resolved, and I had my first good backup of the SQL Server 2005 database.

Now, although the above full-text SQL statements are valid in SQL 2005, here is a list of those that are deprecated and its replacements. Keep in mind that they may be removed altogether in future versions of SQL Server.

DEPRECATED	REPLACEMENT
<code>sp_fulltext_catalog</code>	CREATE/ALTER/DROP FULLTEXT CATALOG

sp_fulltext_table	CREATE/ALTER/DROP FULLTEXT INDEX
sp_fulltext_column	ALTER FULLTEXT INDEX
sp_fulltext_database	
sp_help_fulltext_tables	sys.fulltext_indexes
sp_help_fulltext_columns	sys.fulltext_index_columns
sp_help_fulltext_catalogs	sys.fulltext_catalogs

It might be worthy to note that upgrading to SQL Server 2005 from a previous SQL Server version is supported by the SQL Server 2005 Setup program. You can also migrate databases from previous SQL Server versions to an instance of SQL Server 2005. If you have full-text indexes present, the upgrade process marks your databases as full-text disabled, and hence catalogs must be repopulated. Since this operation can be time and resource consuming, it is not run automatically by the setup. Therefore, it is quite conceivable that because we chose to migrate our database via backup/restore, we did not have the full-text properly disabled. So plan carefully and consider what is the best upgrade method based on your current database environment.

There is one more solution that was suggested by another poster, which has yielded positive results. Make sure you're MSSQL 2005 instance is on SP1, then right-click, tasks, and select 'Detach' to detach the database. Uncheck the option "Keep Full Text Catalogs". Then, reattach the database by right-clicking, tasks, 'Attach', and search to the path of the data files. The backup now completes successfully!

Many thanks to those folks out there who posted their similar issues and resolutions it was indeed a great help. Subsequently, I requested a full-refund from Microsoft support, for being a known issue, yet unable to get to a quick resolution. I'm still waiting to hear back from them, I guess their server is hung indefinitely.

Written by: Robert Pearl, President
Pearl Knowledge Solutions, Inc.

rsp05@pearlknows.com
<http://www.pearlknows.com>

Copyright 2007 - All Rights Reserved.

Note: Not to be reprinted or published without express permission of the author.

SQL Stored Procedure to Log Updates, Independent of Database Structure

By Keren Ramot

Have you ever needed a quick, retroactive solution to track changes on your database? Well, if you're a good system designer you wouldn't, but when contracting for clients you may find often that the characterization of the project is ever changing, and there is only so much you are able to foresee.

When working with multiple clients, with only a certain amount of hours dedicated to each project, you may want your patches to be applicable for multiple projects. The following is such a patch. It functions to track updates done through

the website, storing information such as the user information, the page on which the action took place, the old value and the new value of the changed information and a date time stamp.

THE LOG TABLE

Once you know what information you'd like to store you can easily knockoff stage one of the process by creating the log table. It may look something like this:

LogID	int	4
TableName	NVARCHAR	100
RecordNumber	int	4
ActionBy	NVARCHAR	50
ActionPage	NVARCHAR	100
ChangeClmn	NVARCHAR	100
OldValue	NVARCHAR	75
NewValue	NVARCHAR	75
ActionDate	datetime	8

- The LogID is just a unique identifier
- The TableName, RecordNumber and ChangeClmn would indicate where the change took place in the database
- We need the OldValue and the NewValue of the field.

The field's length would have to be as long as the longest field in the database that is being tracked. If your database has fields too big to be tracked, you may want to consider truncating them. You can maintain smaller OldValue and NewValue, but keep in mind that if you have a field that's say a 1000 characters long, but you would only like to maintain 250 characters, if the field is changed outside of the first 250 characters, it will not be logged as a changed field. This will become clearer as you read on.

Some of you may argue that the old value is not necessary, and indeed it isn't if you store the initial values, but aside from the fact that for the sake of reporting it would be much easier to have the old value and the new value in the same record, one would also have to assume that this table is going to grow rapidly and you may need to archive records at some point, then you would potentially have to go between the log table and the archived tables to get the old values.

- ActionBy would store the user information It would be a char if you would like to store the user name, or an integer if you would like to associate it with a login table or a user table.
- ActionPage would be the name of the page the action was taken on.
- ActionDate would be a getdate()

THE PLAN

If you weren't interested in the client information (the user and page name), and didn't want to target certain pages on your site, rather than tracking all the pages and changes made directly on your database, the easiest way to create such a log would be through triggers. Of course, these triggers would be database specific, and you would have to write one for each and every table you want to track.

If you really really are willing to do just about anything to avoid coding these triggers one by one, keep reading. You would be able to run lines 15 through 91 in query analyzer to automatically produce these triggers. I am sure that you would find this method to be a bit splurgy with the system resources relative to the resources a simple hard coded trigger would require.

In order to track user and page information, as well as being able to track only targeted pages, you would have to make some code changes. A good patch constitutes the fewest possible changes to existing code specially when that code is already tested, or god forbid already running on a live site.

Depending on your work methods, you may find that often the least intrusive way to add such tracking functionality as far as your code is concerned, would be to change the stored procedure calls. A lot of us maintain separate include files for stored procedure calls, and would only have to go over one directory in the website. If you were to reroute your existing stored procedures through one main stored procedure, you wouldn't have to change the stored procedures themselves, just the code that calls them.

Now, the stored procedure would take care of carrying the user and page information on to the database, but as you may have suspected getting the old values is a bit more complicated.

Getting the old value with a trigger is simple, you simply ask for the deleted value, something that can't be done in a stored procedure. In a stored procedure you would select all the fields that are about to be updated before the update occurs, and store them into variables. That would require changing the original stored procedure. If you were to do that, you might as well add 2 more parameters to be passed from the code that calls the stored procedure (ActionBy and ActionPage) and then have the stored procedure insert records into the Log table. This method would force you to change all your update stored procedures, change the code calling these procedures, and would only apply to the one project.

So how do you maintain a log that contains the user information (something a trigger alone can not do), and get the deleted values of the updated fields, without massive changes to your project and the ability to apply it to any project? Well, as you may have already gathered it would have to be a combination of the 2. This brings me to the plan:

A routing stored procedure would create a trigger on the table you are about to update, run your stored procedure, and then drop the trigger.

THE ROUTING STORED PROCEDURE

A routing stored procedure would expect the following parameters:

- @SPName would pass the name of the stored procedure to be run.
- @str1 would pass the parameters for the @SPName stored procedure.
- @TableName would pass the name of the table on which the trigger should be created.
- @RecordId would pass the value of identifier of the table
- @OpName would pass the name of the user making the changes.
- @PageName would pass the name of the page the action was taken on.

Rules of passing the @str1:

1. Make sure Null values don't have quotes.

2. Bit values don't come in as true/false but as 1/0
3. Each string is surrounded by quotes

You may create a class to pass these variables to the stored procedure. That way you will not have repeated code on your site. In ASP the class may look something like this:

Editor's Note: The class code is available at www.sqlservercentral.com

And the call to it would look like this:

```
<%  
  
    set SObj = New SPRouter  
  
    SObj.Connection = objConnection  
  
    'get connection string  
  
    SObj.AddParameter Request.form("FieldName")  
  
    'get the form field from a post form  
  
    SObj.StoredProcedure="StoredProcedureName"  
  
    'stored procedure name  
  
    SObj.ReferID= 1  
  
    'record ID  
  
    SObj.TableName="table_name"  
  
    'updated table name  
  
    SObj.Operator = Session("Operator")  
  
    'User name  
  
    SObj.PageName=request.ServerVariables("SCRIPT_NAME")  
  
    'User location  
  
    SObj.SPRouterForLog  
  
    Set SObj=Nothing  
  
%>
```

The sorted procedure will look like this:

```
CREATE PROCEDURE dbo.SPRouterForLog  
  
(  
  
    @SPName NVARCHAR(50),
```

```
@RecordId NVARCHAR(10),

--The record ID doesn't have to be passed to the SPRouterForLog.

-- You can get that identifier from the syscolumns table.

-- The syscolumns would be discussed below.

@TableName NVARCHAR(100),

@OpName NVARCHAR(50),

@PageName NVARCHAR(50)

)

AS

    --Do something

Go
```

THE TRIGGER

What does the trigger need to do? To simplify, it needs to find the updated values, and for each of those values insert a record into the tbl_log with the old value, the new value, the database location information, and the user and page information.

How would we go about doing that? We would have to iterate through each of the updated columns and compare the old value to the new value. The general idea is as follows:

```
Select * from inserted

Select * from deleted

For each column

    If Deleted <> Inserted

        Insert values into tbl_log

Next
```

Getting the column names

In order to compare the columns, we need to know the column names. The column names can be easily obtained from the syscolumns table or information_schema.columns.

The syntax would be as follows:

```
SELECT name

FROM syscolumns

where id = object_id(''+@TableName+'')
```

Iterating through the columns one by one

The iteration can be done using the colid from the syscolumns

The simplest syntax for looping in SQL would be with a cursor. And the syntax is as follows:

```
declare @colid int declare curFields cursor fast_forward for
select colid, name from syscolumns where id = object_id(''+@TableName+'')
open curFields
fetch curFields into @colid, @tmpName
while @@fetch_status = 0
begin
    --****Compare your values, insert your record****
end
fetch curFields into @colid, @tmpName
end
close curFields
deallocate curFields
```

Of course, a cursor will eat up your system resources. Which is why I suggest iterating with a while loop, like so:

Editor's Note: The code is available at www.sqlservercentral.com

Columns_updated()

We are going to iterate through all the columns in the table, but there is no reason to compare columns that have not been updated. For the sake of shortening the process, we should make sure the column has been updated. Unfortunately the syntax "if updated(@tmpName)" will not work. So we shall use the columns_updated() function.

```
if (substring (columns_updated(), 1+ round ((@CurrentRowId - 1) / 8, 0), 1)
    & power (2, (@CurrentRowId - 1) % 8) <> 0 )
BEGIN
    --Do your thing
END
```

Columns_updated() brings back a varbinary(8 base), so it needs to be formatted like shown above.

The fact that a field has been updated, is no guarantee that it's value has changed. We would still have to compare the old value against the new value.

Comparing the values

At the point of comparison we would want to do something like this:

```
if @new<>@old
begin
    --insert
```

End

Unfortunately, the query "SELECT @old=@tmpName FROM deleted" will bring back the column name and not the field value. So we will be forced to get the value from an executable. Like so:

```
DECLARE    @old varchar(75),@tmpName sysname ,@subsql NVARCHAR(200)

SELECT @subsql = N'SELECT @old = convert(varchar(75), d.''

    + @tmpName+'' )  from deleted as d ''

EXEC sp_executesql @subsql, N'@ old varchar(75) OUTPUT'', @ old OUTPUT
```

But deleted would not work in an executable, because it is outside the trigger, which means that we would have to create a temp table, and then get the value from that table:

```
SELECT * INTO #deleted FROM deleted

SELECT @subsql = N'SELECT @old = convert(varchar(75), d.''

    + @tmpName+'' )  from #deleted as d ''

EXEC sp_executesql @subsql, N'@old varchar(75) OUTPUT'', @old OUTPUT
```

When creating temp tables, we must make sure they don't already exist. A good way to do that, would be giving them dynamic names. The way I chose is creating a random number to be added to the name by using the **Rand()** function:

```
declare @VarRandom NVARCHAR(50)

set @VarRandom=ltrim(str(replace(Rand(), '.', '')))

-- Creates a random number as a string

SELECT * INTO #deleted'+@VarRandom+' FROM deleted

SELECT @subsql = N'SELECT @old = convert(varchar(75), d.''

    + @tmpName+'' )  from #deleted'+@VarRandom+' as d ''

EXEC sp_executesql @subsql, N'@old varchar(75) OUTPUT'', @old OUTPUT
```

Putting everything together:

Editor's Note: The code for this is available at www.sqlservercentral.com

Now, all that's left is to call the stored procedure to update the table, and then drop the trigger from the table.

```
declare @strSqlExec NVARCHAR(4000)

set @strSqlExec=@SPName+' '+@str1

EXEC (@strSqlExec)

drop trigger PreUpdateTrigger
```

GO

LIMITATIONS

This patch will not work if you try to log a text, ntext or image datatype.

It's also highly recommended to maintain small field lengths for a couple of reasons:

- The practical reason would be that the @str1 can only sustain so many characters, depending on the length you set it to. Your updated values including the commas and the apostrophes can not exceed the length of the @str1 string.
- The methodological reason would be that certain field changes are illogical to store. If you have a comments field that's an NVARCHAR 4000, why would you want a log of every time a comma was added on there? Maintaining these changes would reek habit on your system's resources - Your tbl_Log oldValue and newValue would have to be a datatype of at least that length, and querying the log table would require sorting through more unnecessary records and pulling much bigger datatypes then necessary.

If you already have big datatypes in your database, while you do have to make sure that the @str1 is not exceeded, you don't have to go back and make changes to your database to instate this patch. All you need to do is truncate these values when you convert them (@old = convert(varchar(75), d."'+@tmpName+'")), keeping in mind that only a change within the first however many characters you are truncating it to would be recorded in the tbl_log.

IN CONCLUSION

This cool little patch, while not being the most efficient way to deal with such functionality, is certainly one of the fastest ways. It took me a couple of highly frustrating days to come up with it, and I would like to take this opportunity to thank my co-worker and good friend Tejal Shah for all her help and for always supporting my crazy ideas.

Identifying Unused Objects in a Database

By Leo Peysakhovich

Recently, when I chatted about SQL Server issues, I found the following email: "Is anyone aware of any freeware/shareware tools that can assist in identifying unused objects in a database? I have inherited a database from a previous developer, and there seems to be a lot of unused database objects in the database. I know than Apex SQL clean can do the job, but my boss won't agree to the purchase the tool. Any suggestions would be appreciated."

How many times have we had such an answer from the boss! So, I started to check the ways to define the unused objects without using any tools. I understand that tools may do the job better but I was interesting to do it myself at least just for basic analysis and for fun.

You know that in many cases deletion of unused objects will help make easier and quicker development maintenance and may improve the efficiency and the performance of a database. There are many methods that can be used. At the end, the main task is to answer what objects were not used for the month or two. I am saying a month or two because the time depends on the fact that a database may be used not only by the applications but by the backend processes as well. For example, in my company we have daily, weekly and monthly processes that define the time of how often object may be used. The task can be achieved with many methods, one of them by using SQL Profiler, but it will require keeping the tool running constantly for a long time which is not practical and may even degrade the system's performance.

In my company the task becomes a little bit easier by knowing the fact that any database can be accessed only via call to a stored procedure. E.g. I have to split the task to the 2 sequential subtasks:

1. Find unused stored procedures and drop them
2. Find the objects that are not referenced by any stored procedure and are not lookup tables at the same time.

The main idea of my method is to constantly query the system cache to find the procedures that have no execution plan for long periods of time. This can be achieved by using system table syscacheobjects which contains information on how the cache is used. Syscacheobjects belongs to the master database. At first glance, it may be hard to fully appreciate the value of this technique and the information produced by the output report but the method is very easy and provides a very accurate picture for the unused stored procedures. The technique is not as flawless as it may appear, but it offers a good available way for DBAs to find unused stored procedures, functions and the other objects by checking the database-execution plans.

Let's see the step by step and ideas and implementation. Will check an idea by analyzing the cache behavior for one procedure USP_Checkuser

```
select name, id from sysobjects where name = 'USP_Checkuser'
```

name	id

USP_Checkuser	1093578934

```
select bucketid, cacheobjtype, objtype, objid, dbid
from master.dbo.SYSCACHEOBJECTS
where dbid = 6 and objid = 1093578934
```

bucketid	cacheobjtype	objtype	objid	dbid

545	Executable Plan	Proc	1093578934	6
545	Compiled Plan	Proc	1093578934	6

```
sp_recompile 'dbo.USP_Checkuser'
```

Object 'dbo.USP_Checkuser' was successfully marked for recompilation

```
select bucketid, cacheobjtype, objtype, objid, dbid
from master.dbo.SYSCACHEOBJECTS
where dbid = 6 and objid = 1093578934
```

bucketid	cacheobjtype	objtype	objid	dbid

```
Exec dbo.USP_Checkuser
```

```
go
```

```
select bucketid, cacheobjtype, objtype, objid, dbid
from master.dbo.SYSCACHEOBJECTS
where dbid = 6 and objid = 1093578934
```

bucketid	cacheobjtype	objtype	objid	dbid
545	Executable Plan	Proc	1093578934	6
545	Compiled Plan	Proc	1093578934	6

If you would like to make sure that object is marked for recompilation the next statement will show you the changes in object base schema and schema versions:

```
select name, id, base_schema_ver, schema_ver from sysobjects
where name = 'USP_Checkuser'
```

BEFORE RECOMPILATION

name	id	base_schema_ver	schema_ver
USP_Checkuser	1093578934	48	48

AFTER RECOMPILATION

name	id	base_schema_ver	schema_ver
USP_Checkuser	1093578934	64	64

As you can see the fields `base_schema_ver` and `schema_ver` are changing the value from 48 to 64. Every time the procedure will be marked for recompilation the field's value will be changing.

Step one is to cleanup the cache by using the stored procedure sp_recompile that causes stored procedures and triggers to be recompiled the next time they are run. Or DBCC FREEPROCCACHE can be used to clear the procedure cache. Freeing the procedure cache would cause, for example, an ad-hoc SQL statement to be recompiled rather than reused from the cache. While you do this the object name and id can be written into the table to show the time the cache for the object was cleared.

```
create table MONIT_ObjectRecompilation (

    MOR_ID int not null identity(1,1),

    SEQ_ID int,

    DB_NM varchar(50),

    Object_NM varchar(200),

    ObjectID int,

    Object_type varchar(2),

    Status char(1),

    -- R - recompiled;

    -- S - from syscacheobject table


    Create_DT datetime default( getdate() ) )
```

The next step is to add each existing object while marking it for the recompilation. It can be done dynamically by the next batch.

```
Begin

declare @minid int,@maxid int, @cmd Nvarchar(1000)

        , @dbnm varchar(50), @seq_id int, @objectid int

declare @tmp table ( objectid int, rcmd varchar(1000), tid int identity(1,1))

set @dbnm = db_name()

select @seq_id = max(seq_id) from dbo.MONIT_ObjectRecompilation

set @seq_id = ISNULL(@seq_id,0) + 1

insert into @tmp( rcmd, objectid)

select 'EXEC sp_recompile [' + name + ']' , id

        from sysobjects

        where type in ('P', 'FN', 'TR', 'TF')
```

```
select @minid = 1, @maxid = max(tid) from @tmp

while (@minid <= @maxid)

begin

    select @cmd = rcmd, @objectid = objectid from @tmp where tid = @minid

    EXEC sp_executesql @cmd

    insert into dbo.MONIT_ObjectRecompilation

        ( SEQ_ID, DB_NM , Object_NM, objectID, Object_Type, Status)

        select @seq_id, @dbnm, name, id, type, 'R'

            from dbo.sysobjects

            where id = @objectid

    select @minid = @minid +1

end

end
```

Next step will be inserting into the table the objects that are not recompiled yet. I setup a job that inserted a not compiled list of objects to the table. The job is running every 10 minutes.

```
declare @dbnm varchar(50), @seq_id int

set @dbnm = db_name()

select @seq_id = max(seq_id) from dbo.MONIT_ObjectRecompilation

set @seq_id = ISNULL(@seq_id,0) + 1

insert into dbo.MONIT_ObjectRecompilation (SEQ_ID, DB_NM, Object_NM, objectID, Object_Type,
Status)

select @seq_id, @dbnm, so.name, so.id, so.type, 'S'

from dbo.sysobjects so

    left join master.dbo.SYSCACHEOBJECTS sc

        on sc.objid = so.id

    left join master.dbo.sysdatabases sd

        on sd.dbid = sc.dbid and sd.name = @dbnm

where so.type in ('P', 'FN', 'TR', 'TF')
```

```
and sc.objid is null
```

This job will run for a month before I can define a usage of the stored procedures and functions. Then, let's find unused procedures and functions. If object name is in every insert for non recompiled object then the object is never was used.

```
declare @seq_id int
```

```
select @seq_id = max(seq_id) from adm_support.dbo.MONIT_ObjectRecompilation
```

```
select DB_NM, Object_NM, objectID, Object_Type, Status, count(*)  
  
from dbo.MONIT_ObjectRecompilation mor  
  
where mor.status = 'S'  
  
group by DB_NM, Object_NM, objectID, Object_Type, Status  
  
having count(*) = (@seq_id - 1)
```

Seq_id 1 was used to get all recompiled objects.

The research can be extended by usage of additional columns from table syscacheobjects that will allow you to analyze all databases on a server, how often object is used, and get over unknown ad-hoc queries and users who runs them as well as to see the first 128 characters of the statements.

cacheobjtype	nvarchar(34)	Type of object in the cache: Compiled Plan Executable Plan Parse Tree Cursor Parse Tree Extended Stored Procedure
objtype	nvarchar(16)	Type of object: Stored Procedure Prepared statement Ad hoc query (Transact-SQL submitted as language events from isql or osql , as opposed to remote procedure calls) ReplProc (replication procedure) Trigger View Default User table System table Check Rule
objid	int	One of the main keys used for looking up an object in the cache. This is the object ID stored in sysobjects for database objects (procedures, views, triggers, and so on). For cache objects such as ad hoc or prepared SQL, objid is an internally generated value.
dbid	smallint	Database ID in which the cache object was compiled.
uid	smallint	Indicates the creator of the plan for ad hoc query plans and prepared plans. -2 indicates the batch submitted does not depend

		on implicit name resolution and can be shared among different users. This is the preferred method. Any other value represents the user ID of the user submitting the query in the database.
usecounts	int	Number of times this cache object has been used since inception.
sql	nvarchar(256)	Procedure name or first 128 characters of the batch submitted.

Conclusion.

Your database may have many stored procedures, tables and views that aren't being used anymore but unless you determine which of your objects fall into this category you will be stuck with them forever. The technique described in the article is not as flawless as it may appear, but it offers a good available way for DBAs to find unused stored procedures, functions and the other objects by checking the database-execution plans without buying any third party tools. The captured information also may offers some great clues about what database objects you need to pay attention.

Upgrading a Database SQL 2000 to SQL 2005

By Sachin Samuel

Introduction

The objective of this white paper is to lay down a step-by-step guide for a SQL Server 2005 upgrade from SQL Server 2000. All the steps, explained in this document, are based on experience with the recent upgrade done for one of our Customer. The database was VLDB (very large database) having size of 470 GB and hosted on an active/passive cluster.

In this document, I will also explain about the tools used for planning the SQL 2005 upgrade, the upgrade process (In-place upgrade), and the rollback strategy. But the most important point is to achieve a smooth and trouble free transition, for which you must devote ample good time in planning and testing of the upgrade.

Why Upgrade to SQL Server 2005

SQL 2005 server contains many new features and improvements as compared to the old version. One or more can be the compelling reason for upgrade. A few of the important features are as below.

1. Online database mirroring
2. Maintaining stronger and more flexible security.
3. Greater manageability for VLDBs.
4. Better development features.
5. Better business development solutions.

You can download complete list of features from this URL:

http://download.microsoft.com/download/2/4/5/2456889a-df87-4def-a553-91f15b4e8c00/SQLServer2005_WhyUpgrade_final.doc

Methods to Upgrade

There are two methods to upgrade SQL Server 2000 to SQL Server 2005.

1. In-place upgrade

2. Side by Side Upgrade

In-place upgrade: SQL Server 2005 gives you ability to automatically upgrade an instance of SQL Server 2000 or 7.0 to SQL Server 2005. The reason it is called in-place upgrade is because a target instance of SQL Server 2000 or 7.0 is actually replaced with a SQL Server 2005 instance. You do not have to worry about coping data from the old instance to new instance as the old data files are automatically converted to new format. This upgrade method is the easiest way to upgrade the database to newer version.

It's because the files are automatically upgraded without any manual intervention. The number of instances and server involved in this type of upgrade is always one. This upgrade method cannot be used if you want to upgrade only a single database.

Side-by-Side Upgrade : In this upgrade method, a new instance is created on the same server or in a new server. In this upgrade method the old database instance runs in parallel to the old legacy database. So as the old instance is untouched during this type of upgrade, the old legacy database is still available and online for the application.

In this upgrade you will have to manually move the data files and other supporting objects (Jobs, DTS packages etc.) to the new instance.

Choosing an Upgrade Method

Lets discuss about the advantages and disadvantages of each method.

In-place upgrade advantages:

1. It's easier and faster, especially in small systems.
2. It's mostly an automated process
3. The instance will be offline for a minimum amount of time.
4. The resulting instance after upgrade will have the same name as the original, as the new setup will replace the older version.
5. No additional hardware is required.

Disadvantages:

1. It's very complex to rollback.
2. Not applicable in scenarios where you want to upgrade a part of system like upgrading just one single databases.
3. You cannot run an upgrade comparison after doing the upgrade.

Side-by-Side Upgrade advantages

1. More control over the upgrade, as you can upgrade the components, which you want to.
2. You can keep you application running even when you are installing SQL 2005 as the old instance or server will be available.
3. Easy to do a rollback as the original database server is untouched.

Disadvantages:

1. You might need additional hardware resources in terms of disk space, CPU and RAM.
2. Manual intervention is required to migrate databases, Jobs, logins.
3. There will be change in configuration settings, which are used by the application to connect to the database.
4. More time is required while moving VLDB to the new version of database.

Preparing for Upgrade Upgrade Advisor

Upgrade advisor is a tool available to find out your database compatibility and blocking issues while doing the migration. You can download this tool from this url:

<http://www.microsoft.com/downloads/details.aspx?familyid=1470E86B-7E05-4322-A677-95AB44F12D75&displaylang=en>

This tool requires .NET framework 2.0. Later in the document we will discuss, how to use this tool and take benefits from it.

Upgrade advisor smoothes the transition from an older version to the newer version by anticipating issues/blocking with legacy databases. The Upgrade Advisor generates reports explaining the upgrade issues and also guidance on how to resolve them. In addition to analyzing databases and objects, the Upgrade advisor can also analyze T-SQL queries, by using SQL trace files. When you start upgrade advisor, you will see the welcome screen below.

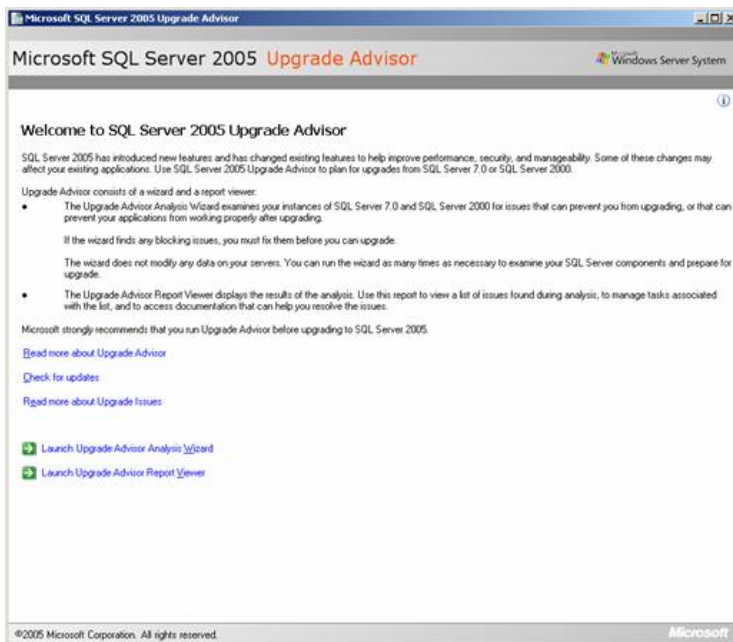


Fig 1.1

You can launch upgrade advisor by clicking on link, "Launch Upgrade Advisor Analysis Wizard". On clicking on this link, you will prompt with a window to select components, which can be analyzed by upgrade advisor. You can select the components or you can click on the detect button. The detect functionality will automatically detect and select the components which need to be migrated. You cannot specify an instance name in the server name box. In the case you want to analyze a server with multiple instance, then you have to specify just the name of the server in the server name box.

Once you have chosen the server name, and have selected the components for analysis, you can click on next button to reach connection parameter screen. Enter the credentials needed to connect to the server.

In this screen, supply the instance name and login credentials. Click next, and the required details are supplied. If the login credentials supplied are correct, you will be re-directed to a new screen as below.

At this point you will be allowed to choose databases, which you want to run against Upgrade Advisor. If you choose to analyze all the databases, wizard will also analyze SQL Server configurations settings. It's always a good practice to choose all the databases in case you are planning to do an in-place upgrade.

You can also run trace files against upgrade advisor. This way you will be able to analyze any adhoc query getting executed from application. The recommended SQL profile template is SQLProfilerTSQL_Replay, as this will have unique number of queries.

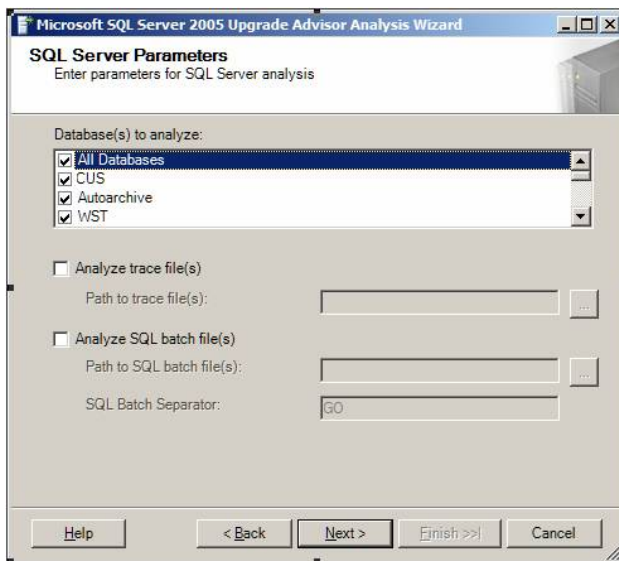
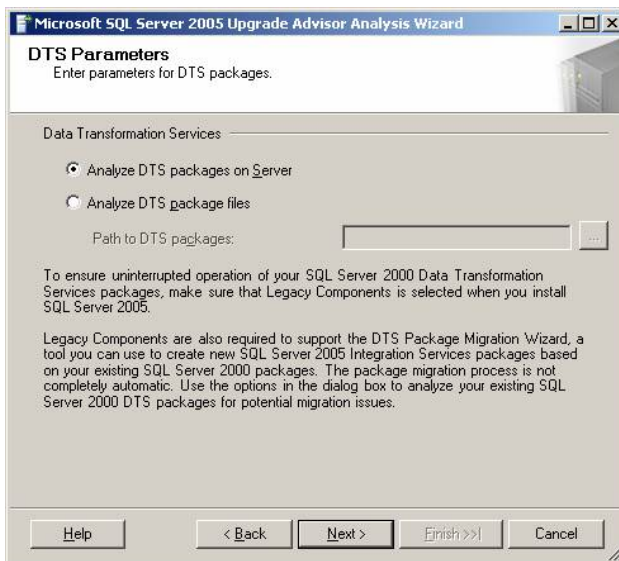


Fig 1.4

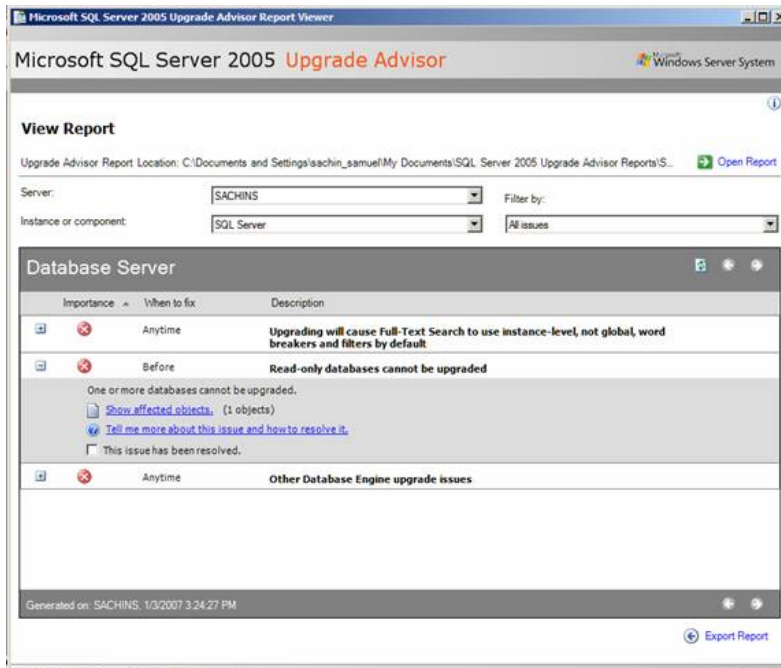
On clicking next, the Upgrade Advisor will prompt you to analyze components you selected. Below is the screen shot. You can analyze either all the DTS packages on the server or any specific DTS packages saved in a file.



At the end of the wizard, you will have a chance to confirm all the options you have selected. Once confirmed by clicking on "Run" button, Upgrade Advisor analyzes each selected database and components and when done, it

saves the report on the default path of logged in user. The reports are saved in XML format, for each component. You can launch the Report viewer to view the reports generated.

Once done, you can launch the report viewer to check the report. You can also apply filter on the report to check analysis reports for a specific component.



Each report has got its own importance level. Red X symbol means that this error needs to be fixed before upgrading. A yellow triangle indicates that additional action needs to be taken once upgrade is done.

The "When to Fix" column indicates when you should address the error. Any error having "Before" as the value in "When to fix" need to be addressed immediately as this error can be blocker for the upgrade. For example in the previous screen shot, there is an error, which says, "Read-only Database cannot be upgraded". This is because if the database is set to read only mode, SQL Server setup won't be able to upgrade it by running scripts on it.

You can expand each report as shown above screen shot to check the objects causing the error and how to resolve it. This can be done by clicking on link, "Show Affected objects"

Things To Do

I will break this section into three parts "Before upgrade", "During upgrade" and "After upgrade". Please carefully note all points in each part.

Before Upgrade

1. As discussed above, you should run upgrade advisor for all databases (including system databases) and fix any issue/blocker for upgrade.
2. Take proper down time before starting the upgrade. The outage required would depend upon the size of the databases and the environment in which database servers are hosted. In my case we took 48 hours of downtime.
3. Once all the issues and blockers are solved, and all applications are closed, take complete backup of all the databases (including master, model and msdb).

4. Also take transactional backup for all user databases and the disable all the jobs including jobs for taking full backups, jobs to defrag and other maintenance plans.
5. It is also recommended to take an image of your database server, this can be very handy in case you will have to rollback.
6. in case you are planning to use Reporting Services in the future, be sure to install IIS on all the nodes of SQL Server. For this you will require the Windows OS CD.
7. If the operating system of the server is Windows 2003, make sure that Windows service pack 1 is installed. This needs to be done on all the nodes of cluster in case the database server is on Failover Cluster environment.
8. You can save upgrade time by installing a few prerequisites in the database server. They are .Net 2.0 framework and SQL Native Client. in case you have servers on cluster, install these setups on each node.
9. Get ready with the new SQL 2005 DVD and the edition you want to install. Also download the latest service pack (SQL Server 2005 SP2) and some cumulative hotfixes. Keeping them ready will save time.
10. Make sure that you have enough space in the drive where SQL Server 2000 is installed in case of in-place upgrade. At least 2 GB will be required.
11. Fulltext indexes applied on tables in SQL 2000 are not compatible with SQL Server 2005. Therefore they should be removed from all the database in case there is any. Once they are removed, execute the below script. This will remove any fulltext index left. It is recommended that you execute the script under a login id having system administrator privileges.

```
exec sp_configure 'allow updates',1
```

```
reconfigure with override
```

```
update sysobjects set ftcatid = 0 where ftcatid <> 0
```

```
declare @a int
```

```
set @a = 1
```

```
while @a > 0
```

```
begin
```

```
    set rowcount 300000
```

```
    delete from sysfulltextnotify
```

```
    select @a = @@rowcount
```

```
end
```

```
update syscolumns set colstat = colstat & ~16
```

```
update sysindexes set status = status & ~33554432
```

```
update sysobjects set status = status & ~200, ftcatid = 0
```

```
exec sp_configure 'allow updates',0
```

```
reconfigure with override
```

12. Once all the Fulltext indexes are removed from the database(s), detach the user databases, before starting the upgrade. This is a good practice, as you have your database untouched during the setup goes on. in case the upgrade fails because of any error you can reattach them on legacy database and continue the production.

During Upgrade

13. Make sure, all the points are taken care from "Before upgrade section".
14. Start installing by clicking SetUp.exe from SQL Server 2005 DVD.
15. The setup program is designed in such a way that it automatically updates the database binaries. It altogether create a different folder "90" to contain its new binaries. 90 specifies the compatibility of the database. It also upgrades the other SQL Servers on clusters automatically but only the database engine and not other components like reporting services or analysis services. This is very important. Therefore you will have to install other components separately in each node.

You can see the progress of installation on installation screen. in case installation is done on cluster, setup will populate all the nodes in a drop down list.

16. Once all the nodes are upgraded with the database engine and all components are installed individually on servers, install the latest SQL Server service pack. This need to be done on all the cluster nodes
17. A reboot may be required to complete the installation due to open file locks during the installation process. If the installation log does not contain any 3010 error, then a reboot is not required.

After Upgrade

18. Once installation is successfully done on all the nodes, attached the databases.
19. Change the compatibility of all the databases attached to 90. If this step is not performed, you won't be allowed to use new features from SQL Server 2005. You can use the below script to achieve this:

```
EXEC sp_dbcmtlevel @dbname = '<Database Name>', @new_cmptlevel = '90'

GO

use [Report]

GO

Alter authorization on database:: '<Database Name>' to sa

Alter database [Report] set parameterization forced

GO
```

20. Recreate all the full text indexes removed from the databases.
21. Update the statistics for all the databases. This step may take time depending on the size of database. It took me 15 hours to update statistics for all the databases.
22. Once it is done, it's very important to test the application with the new database server 2005.

Conclusion

As discussed in the white paper, SQL Server 2005 is complex but can be made quite straight forward by doing lot of testing and study. I will recommend to test multiple times to get confidence and a fair idea about entire process.

Maximum Row Size in SQL Server 2005

By Andy Warren

I've been using SQL 2005 on a part time basis for a while now (those SQL 2000 servers work pretty good you know!) and I'm still intrigued at the things I find that have changed the rules of the game. It's pretty commonly known that SQL pages are 8k, and that the max row size is 8060 bytes (though Steve Jones proved that isn't always the case in his SQLServerCentral article, [What is the Maximum Page Size in SQL Server 2000?](#) But in SQL 2005, the max row size of 8060 bytes is even less true than it was before!

Assuming I'm not the only one that still have a SQL2K instance, create a test table as follows:

```
CREATE TABLE [dbo].[TestTable](  
  
[LargeColumn1] [varchar](8000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,  
  
[LargeColumn2] [varchar](8000) COLLATE SQL_Latin1_General_CP1_CI_AS NULL  
  
) ON [PRIMARY]  
  
GO
```

You should get the following warning:

Warning: The table 'TestTable' has been created but its maximum row size (16025) exceeds the maximum number of bytes per row (8060). INSERT or UPDATE of a row in this table will fail if the resulting row length exceeds 8060 bytes.

The problem is that this table could work fine for a day, a week, or a lifetime, and then boom, you get bitten by your bad table design. Just to prove that really happens, let's inject a row and then make some changes:

```
insert into Testtable (LargeColumn1, LargeColumn2) values ('Blah1', 'Blah2')  
  
update testtable set largecolumn1 = replicate ('0123456789', 800)
```

At this point LargeColumn1 is 8000 bytes, leaving us the 60 or so bytes for column LargeColumn2. Let's see what happens if we try to update it with 100 characters:

```
update testtable set largecolumn2 = replicate ('0123456789', 10)
```

We get this lovely error:

```
Msg 511, Level 16, State 1, Line 1  
Cannot create a row of size 8113 which is greater than the allowable maximum of 8060.  
The statement has been terminated.
```

In fact, the most we can stuff into LargeColumn2 is an additional 47 bytes. Now suppose we try our scenario on a SQL 2005 instance, what do you think might happen? The first thing is you get NO WARNING that you're exceeding the 8060 byte when you run the create table statement . Why? Because it's not a real limit anymore! Here is an excerpt from BOL (search on row-overflow to find the entire detail):

*Surpassing the 8,060-byte row-size limit might affect performance because SQL Server 2005 Database Engine still maintains a limit of 8 KB per page. When a combination of **varchar**, **nvarchar**, **varbinary**, **sql_variant**, or CLR user-defined type columns exceeds this limit, the Database Engine moves the record column with the largest width to*

another page in the ROW_OVERFLOW_DATA allocation unit, while maintaining a 24-byte pointer on the original page. Moving large records to another page occurs dynamically as records are lengthened based on update operations. Update operations that shorten records may cause records to be moved back to the original page in the IN_ROW_DATA allocation unit. Also, querying and performing other select operations, such as sorts or joins on large records that contain row-overflow data slows processing time, because these records are processed synchronously instead of asynchronously.

We can fully populate both columns with no error message:

```
insert into Testtable (LargeColumn1, LargeColumn2) values ('Blah1', 'Blah2')

update testtable set largecolumn1 = replicate ('0123456789', 800)

update testtable set largecolumn2 = replicate ('0123456789', 800)
```

This new behavior affects the entire record rather than a single column, and in the worst case you could have parts of columns bouncing back and forth between in row and out of row storage. Here's another interesting twist; setting the compatibility level to 70 or 80 on a SQL 2005 instance does not change the behavior - so you could update a SQL 2000 server, set SQL2K compatibility, and still use the new behavior. One side affect of that is that if you had designed a table in SQL 2000 where the total width exceeded the maximum and always worried that you might get burned someday this could save you!

I haven't decided if I like the new behavior or not. My Zen side says that there is a time and place for all options, but the more practical side says that sometimes it's good to have some solid rules to work within. One thing I do dislike is that the new behavior is silent. If you've been a DBA for a while you probably rely on that warning message to keep you from creating super wide tables by accident, but now you'll have to work harder and total up the bytes. I think a new version of the warning message would have been nice, something like this:

Feature Change Warning: The table 'TestTable' has been created but its maximum row size (16025) exceeds the maximum number of bytes per row (8060). On instances of SQL 2000 and earlier INSERT or UPDATE of a row in this table will fail if the resulting row length exceeds 8060 bytes. On SQL 2005 instances using any compatibility level there are cases where it is possible to exceed the 8060 byte restriction. Please see row-overflow in BOL for more information.

Dynamic Management Views and Functions in SQL Server 2005

By S.Srivathsani

Introduction

Dynamic Management views and functions are a new add on feature in SQL Server 2005. In previous versions, to find out the real time statistics the user had to issue DBCC commands or run profiler. SQL Server 2005 is shipped with Dynamic management views to ease the troubleshooting process. DMVs and functions have replaced many system tables and system stored procedures. The information returned by DMVs and functions represents the internal state data of the server. It gives us information about the health of that instance, helps us in diagnosing problems and tuning the server accordingly.

DMVs and functions can have server or database scope. The DMVs can be addressed using two, three or four part names. DM functions can be addressed using two or three part names. Both can be addressed using one part names. The DMVs and functions exist in the sys schema. So when using the DMVs and functions, they must be prefixed with sys.

There are around 70 DMVs and functions. Some examples of DMVs and functions are as follows:

Database Related DMVs and functions

Sys.dm_db_file_space_usage : This gives information about the file space usage for each file in the tempdb database. Some of the columns returned by this view are:

1. Database ID
2. File ID
3. Unallocated_extent_page_count - This gives information total number of pages in the unallocated extents
4. User_object_reserved_page_count - Information about the total number of pages allocated from uniform extents for user objects. The user objects can be:
 1. User defined tables and indexes
 2. System tables and indexes
 3. Global and local temporary tables
 4. Table variables
5. Internal_object_reserved_page_count - This gives the total number of pages allocated from uniform extents to the internal objects. The internal objects are applicable only to tempdb database. These objects can be temporary tables used for cursor operations, work files used for hash joins and the temporary tables used for sort operations.

The sys.dm_db_file_space_usage can used to troubleshoot insufficient disk space in tempdb.To find the number of unallocated pages in kb,we can use the sys.dm_db_file_space_usage DMV as follows:

```
SELECT SUM(unallocated_extent_page_count) AS [free pages],  
  
       (SUM(unallocated_extent_page_count)*8) AS [free space in KB]  
  
FROM sys.dm_db_file_space_usage
```

In the similar way one can find out the space used by internal objects and user objects.If there are lots of temporary tables used,it affects the performance.So the user can keep a check on the temporary tables created by using sys.dm_db_file_space_usage DMV.

Execution related DMVs and functions

Sys.dm_exec_sessions - This returns one row per authenticated session established with the SQL Server. Some of the columns returned by dm_exec_sessions are:-

1. Session_ID
2. Login time
3. Host_name - Host name associated with the session.
4. Program_name - Program name associated with the session
5. Login_Name - SQL Login Name of the user of the session
6. Status: - Status of the session. Can be running, sleeping or Dormant
7. Memory_Usage - Memory usage by each session.
8. Transaction_Isolation_level - Isolation level of the transactions in that session. It can be anything from 0 to 5. 0 = Unspecified, 1 = ReadUncommitted, 2 = ReadCommitted, 3 = Repeatable, 4 = Serializable, 5 = Snapshot

Example: To determine the number of sessions established by every login:

```
select  
  
       login_name  
  
       , count(session_id) as session_count  
  
from sys.dm_exec_sessions
```

```
group by login_name
```

Sys.dm_exec_cursors - Returns the cursors that are open in the databases.

```
SELECT * FROM sys.dm_exec_cursors(0)
```

The parameter that is supplied to this DMV is the session id. If 0 is specified, it means the open cursors in all sessions.

Sys.dm_exec_connections - Returns information about the connections made to the instance of the SQL Server and the details of each connection.

```
select a.session_id
       , b.login_name
FROM sys.dm_exec_connections a ,sys.dm_exec_sessions b
WHERE a.session_id=b.session_id
```

The above query gives the connections established by the user along with the session details and login name.

Sys.dm_exec_cache_plans - Returns the query plans which are cached by the SQL Server. The SQL Server caches the execution plans for faster processing of queries. With the help of this DMV, the user can know the memory occupied by the cache objects, the type of the object in the cache, count of the number of cache objects accessing this cache object, the number of times the cache object has been used.

```
select * from sys.dm_exec_cached_plans
```

Some of the columns returned by this query are:

1. BucketID: The ID of the hash bucket in which the plan is cached.
2. Refcounts: Number of cache objects accessing this cache object
3. Usecounts: Number of times this object has been used.
4. Size_in_bytes: The memory occupied by the cache object in bytes.

Index Related DMVs and Functions

sys.dm_db_index_physical_stats - This dynamic management function returns the size and fragmentation information of the index of a specified table or view. It replaces the DBCC SHOWCONTIG statement.

Syntax:

```
sys.dm_db_index_physical_stats (
    {database_id | NULL | 0 | DEFAULT}
    , {object_id | NULL | 0 | DEFAULT}
    , {index_id | NULL | 0 | -1 | DEFAULT}
    , {partition_number | NULL | 0 | DEFAULT}
```

```
, {mode | NULL | DEFAULT}  
)
```

Arguments:

Database_ID: The ID of the database. If NULL | 0 | DEFAULT is specified, then it considers all the databases in that instance.

Object_ID: Object ID of the table or view to which the index belongs to.

Index_ID: ID of the index.

Partition_Number: Partition number in the object

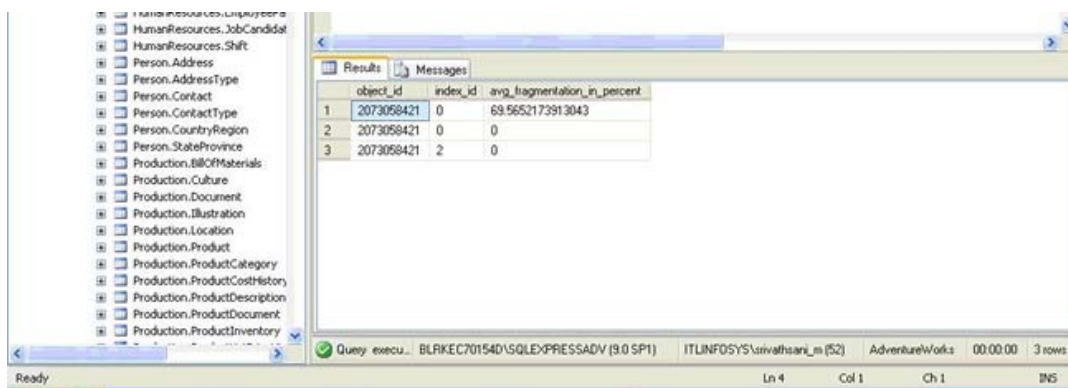
Mode: Specifies the scan level. It can be DETAILED, SAMPLED, LIMITED, NULL or DEFAULT. LIMITED is the fastest. It scans only the parent level pages of an index and not the leaf level pages. The SAMPLED mode returns statistics based on a 1 percent sample of all the pages in the index or heap. If the index or heap has fewer than 10,000 pages, DETAILED mode is used instead of SAMPLED. The DETAILED mode scans all pages and returns all statistics.

Example: To find the information about all the databases in the instance of that SQL Server:

```
select *  
  
from sys.dm_db_index_physical_stats(NULL,NULL,NULL,NULL,NULL)
```

Sys.dm_db_index_physical_stats helps in detecting the fragmentation percentage of the index. The avg_fragmentation_in_percent column returned by Sys.dm_db_index_physical_stats determines the fragmentation level of the index. It should be close to zero for maximum performance. The acceptable limits are between 0 and 10%. If the fragmentation percentage is less than 30%, consider reorganizing the index. If it is greater than 30%, rebuild the index. For example:

```
select object_id  
  
      , index_id  
  
      , avg_fragmentation_in_percent  
  
from sys.dm_db_index_physical_stats(db_id(),object_id('dbo.databaseslog'),NULL,NULL,NULL)
```



	object_id	index_id	avg_fragmentation_in_percent
1	2073058421	0	69.5652173913043
2	2073058421	0	0
3	2073058421	2	0

Now here the avg_fragmentation_in_percent is higher than 30% for index_id 0. So we need to rebuild the index.

Sys.dm_db_index_usage_stats - It returns the number of counts of the different operations like seeks, scans, lookups, user updates etc. and the timings each type of operation was last performed.

Operating System Related DMVs and functions

Sys.dm_os_memory_objects : Returns the memory objects that are currently allocated by SQL Server.

Sys.dm_os_performance_counters : This DMV makes it possible to query a view directly to capture the SQL Server counters related to the instance. Some of the counters associated with SQL Server are:

MSSQL:Locks
MSSQL:Databases
MSSQL:Broker Statistics
MSSQL:Transactions
MSSQL:Memory Manager

This DMV can be used to obtain one SQL Server counters. For counters related to memory, physical disk the user has to run the performance monitor.

Sys.dm_os_wait_stats : This gives information about the threads that waiting for resources in milliseconds. With the help of this information the user can tune the applications for more concurrency and better performance. This DMV can be used instead of DBCC SQLPERF('WAITSTATS'). The columns returned by this DMV are:

1. Wait_type - Name of the wait type
2. Waiting_tasks_count - The number of waiting threads of this wait type.
3. Wait_time_ms - Total time in milliseconds including the signal_wait_time
4. Max_wait_time_ms - Maximum wait time for this wait type
5. Signal_wait_time - Difference between the time the waiting thread was signaled and when it started running in milliseconds

The threads can wait for a resource. Suppose the thread requests to access a resource which is being used by another thread, then the thread has to wait. The waits can also be due to external events such as the SQL Server thread is waiting for an extended stored procedure or linked server query to finish.

This DMV captures around 100+ wait types. They include backup, latches and locks, page latches, Distributed Transactions, Checkpoints etc.

Sys.dm_os_sys_info : Returns information about the computer and the resources available and consumed by the SQL Server. Some of the columns returned are:

1. CPU_count - Number of logical CPUs
2. Physical_memory_in_bytes
3. Virtual_memory_in_bytes
4. OS_Error_mode - The error mode for SQL Server processes
5. OS_priority_class - The priority class for SQL Server processes

Transaction Related DMVs and Functions

Sys.dm_tran_active_transactions : Returns information about the transactions in a SQL Server instance. Some of the columns returned are:

1. Transaction_ID
2. Name
3. Transaction_begin_time
4. Transaction_type - It can be a read-only (2), read/write (1), system (3) or a distributed transaction (4)

Sys.dm_tran_current_transaction : Returns a single row which gives information about the current transaction in the current session. Some of the columns returned are:

1. Transaction_ID
2. Transaction_is_Snapshot - The value is 1 if transaction is started under snapshot isolation level, else 0.

sys.dm_tran_database_transactions : Returns information about the transactions in that database.

Conclusion

Thus the user can make use of this powerful feature to capture real time metrics. For a complete listing of DMVs and Functions, the user can refer to SQL Server 2005 Books Online.

Backing Up a Database with SMO

By Brandi Tarvin

Sure, as a SysAdmin, you can do backups via the Management Studio interface, but what if you've just shut down SSMS for the day and someone comes begging you for a backup? Do you really want to wait for SSMS to pull itself together and get connected to a DB so you can open object explorer or type up the T-SQL required?

Well, okay. Maybe I'm a lazy DBA. I certainly prefer saving code snippets to do all my backup / restores to retyping them from scratch every time I need them. However, when I found out about the new SQL Server SMO object, I just about jumped up and down for joy. Finally, a new toy I could have lots of fun with!

Searching the internet for information on the new SMO object for SQL Server 2005, I found lots of information on how to use the SMO to backup local databases. Unfortunately, what I needed and couldn't find was information on how to backup the databases remotely. After a lot of work, I finally found the solution to my problem and thought I would pass it on to everyone else. Hopefully this will help someone else out. Just make sure the people you give this program to actually have permission to backup the databases or the program will fail.

Important Note: Make sure to save your work intermittently or you may lose everything you're working on if there's a power surge.

Start by creating a project for a new Visual Basic Windows Application. I called mine SMO_BackupDatabase. Okay, so I'm not going to win awards for creative program names, but it gets the point across. @=)

Next, add References to the project by going to Project -> Add Reference. You can add them all at once by using CTRL-Click. The references you need are listed below:

`Microsoft.SqlServer.ConnectionInfo`

`Microsoft.SqlServer.Smo`

Rename Form1 in properties and under the project / solution menu to ChooseEnvironment. Also, change the text on the form to "Choose Environment" so the end users know what they are looking at. **NOTE:** I rename all my objects to descriptive user-friendly names so when a bug occurs, I know exactly where to find it. (See Figure1)

Add a label, rename it to WelcomeLabel, and enter appropriate text as introduction to Program. I used the following:

"Welcome to the SQL SMO Database Backup Program!

Please Select an Environment to get started."

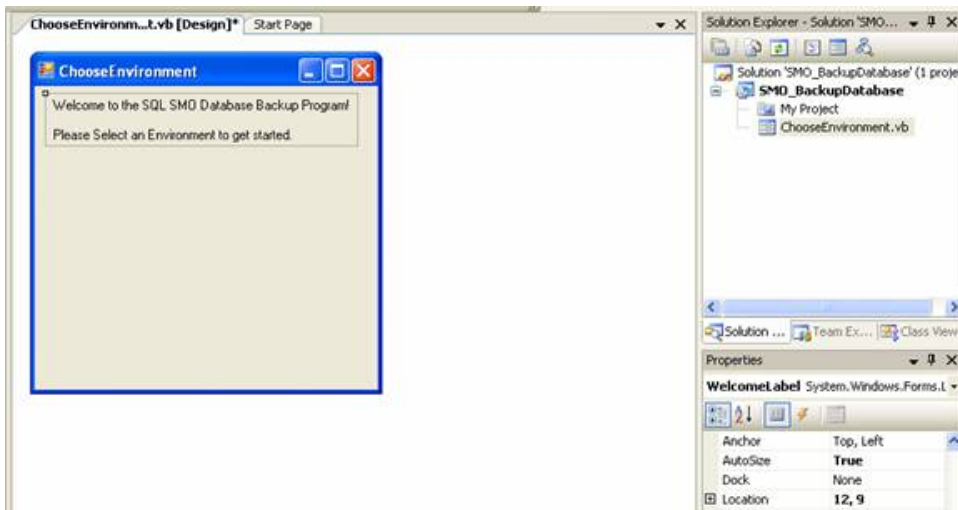


FIGURE 1

Add a GroupBox & as many Radio buttons as you have environments. (I use four, one each for Dev, Test, QC and Production).

Make sure to change your names and the Text properties to something appropriate. I used ServerEnvironment, Development, Test, QC, and Production for mine.

Lastly, add 2 buttons to the bottom of the form. Call the first one ChoseDBButton and set the Text as "Choose Database(s)". Call the second one CancelExitProgramButton and set the Text as "Cancel / Exit" (See Figure 2)

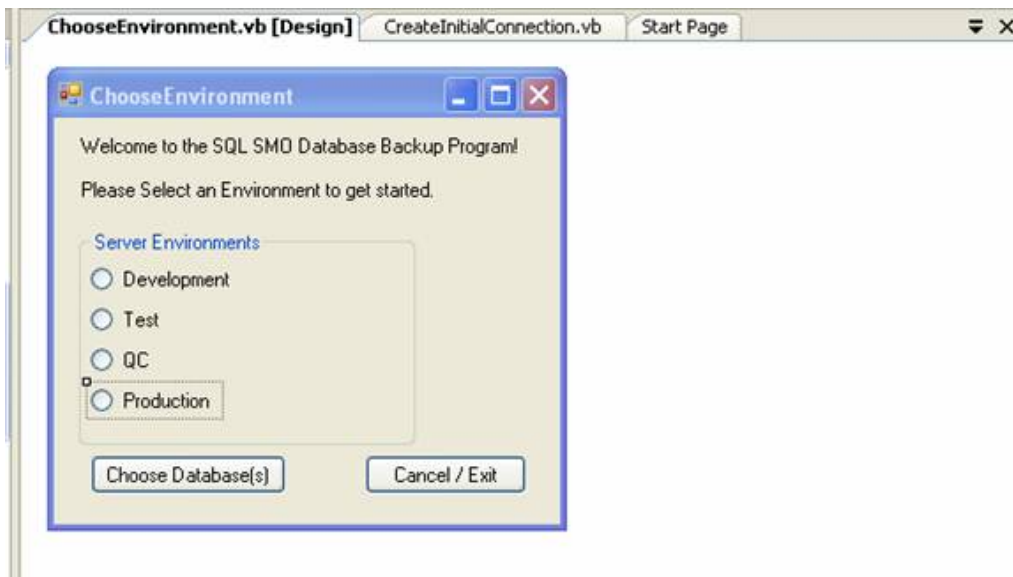


FIGURE 2

Double-click the ChooseEnvironment form. Since it is the initial form you want to start with, set the following code. The FormStartPosition tells the computer where on the monitor to place the form when the program runs.

```
Public Class ChooseEnvironment

    Private Sub ChooseEnvironment_Load(ByVal sender As System.Object _
```

```
        , ByVal e As System.EventArgs) Handles MyBase.Load

    Me.StartPosition = FormStartPosition.CenterScreen

    End Sub

End Class
```

Double-click each radio button, the ChooseDBButton and the CancelExitProgramButton. You want to put in the following code right above the "End Class" of the ChooseEnvironment code.

Editor's Note: This code is available at www.sqlservercentral.com

When this code is created, there will be several underlines indicating that Visual Studio doesn't know what variables and objects you're talking about. That's okay. We'll fix that in a moment.

Next, create a new code module called CreateInitialConnection. (Right click solution name -> Add -> Module). All commented out code is used for debugging to verify I'm connecting to the correct servers. You can delete it if you want to.

First, the following Import statements go above the module name:

```
Imports Microsoft.SqlServer.Management.Smo

Imports Microsoft.SqlServer.Management.Common

Imports System.Data.SqlClient
```

Then it's time to declare the objects and write the module code. You will need a Server object, a ServerConnection object, and 2 Public string objects. I actually force the users to close the program if they want to choose a different environment. You can change this to disconnect and reconnect to the server objects if you want to, but I'm currently leaving it out of the scope of this discussion.

Editor's Note: This code is available at www.sqlservercentral.com

Explanation: For each ServerInstance of the connection object, I set the ServerName and Port# manually. Replace the above with your own ServerName, Port#. This is easier, and faster if you only have a few servers, to code than enumerating your servers. However, if server names change, and they can, you'll have to edit your code and redeploy your program as needed.

Once this code is completed, you will notice all the underlines from the ChooseEnvironment code MyEnv object will disappear.

Add a new module. Call it CloseConnObjects. This code will reset all variables and close all objects when it is called. I do not use the Me.Close() in this because it closes the program the instant I try to open it. Instead, I call this module in my various Exit Program buttons.

```
Module CloseConnObjects

    Public Sub CloseObjects()

        ProdServer = Nothing
```

```
ProdConn = Nothing  
  
MyServerEnv = Nothing  
  
MyEnv = Nothing  
  
End Sub
```

```
End Module
```

Add a new form. Right click the solution name -> Add -> Windows Form. Call it BackupDatabase.

Drag a Listbox over to the new form and enlarge to a size where the user will be able to see all database names easily. Call it DatabaseList.

Add 2 new buttons to BackupDatabase form. Call one BackupDatabaseButton and the other ExitProgramButton. Change the text on the first to "Backup Database" and the text on the other to "Exit Program". Resize buttons as needed. (See Figure 3)

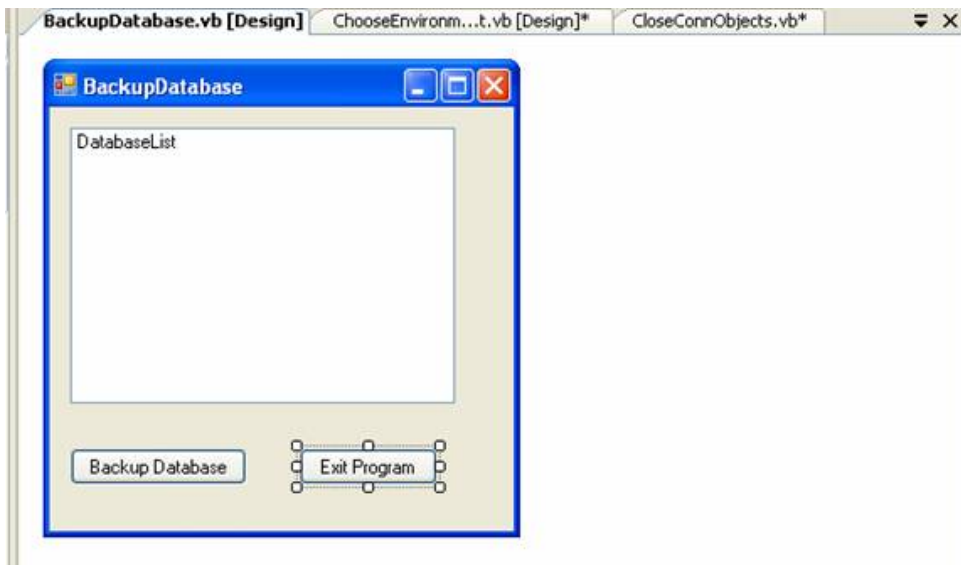


FIGURE 3

Double-click the BackupDatabase form and it will take you to a new code window.

You will need to add the following import statements above your BackupDatabase form code:

```
Imports Microsoft.SqlServer.Management.Smo  
  
Imports Microsoft.SqlServer.Management.Common
```

You will also need to append a last import statement, using the name of your solution and the new module. This will make sure that your CreateInitialConnection actually connects.

```
Imports SMO_BackupDatabase.CreateInitialConnection
```

The Best of SQLServerCentral – Vol. 5

The below code enumerates all the databases in the environment (or instance) that the user has selected. It does not show Snapshots or system databases. *(complete class at www.sqlservercentral.com)*

```
'##### Add list database code here #####

Dim db As Database

For Each db In ProdServer.Databases

If Not db.IsDatabaseSnapshot And Not db.Name = "tempdb" Then

DatabaseList.Items.Add(db.Name)

End If

Next

Try

BackupDatabaseButton_Click(DatabaseList.SelectedItem.ToString, e)

Catch

        MyException As System.NullReferenceException

        'Call MsgBox("You must select a database to continue.")

End Try
```

Above the "End Class", you'll need to add the Backup Database button code. It uses the dating format that SQL Server's database maintenance plans use when backing up the databases. *(complete class at www.sqlservercentral.com)*

Above the "End Class", but below the BackupDatabaseButton code, add the following code for the Exit Program button. This code tells the user it is closing the program and closes all objects as it does so.

```
Private Sub ExitProgramButton_Click(ByVal sender As System.Object _

, ByVal e As System.EventArgs) _

Handles ExitProgramButton.Click

MsgBox("Program terminating as requested.")

CloseObjects()

End

End Sub
```

Then verify in the ChooseEnvironment code that all the underlines have disappeared. If you've done everything correctly, you should be good to go.

Save the program, then go to Debug -> Start Debugging and test it on your servers. I recommend testing on Dev first. It usually takes a moment for the database list to appear, so if you want to stick an additional message box in the ChooseDBButton code that tells the end user to wait, that might help.

The most common error with debugging here is forgetting to substitute your real ServerName\InstanceName and Port # in the CreateInitialConnection and BackupDatabaseButton sections of code. So if you're running into connection errors, those are the first two places you should look.

The way I've got this coded is that you can only backup 1 database at a time. However, you do not have to close the program to backup another database in the same environment. Just keep choosing databases and clicking the "Backup Database" button to your heart's content. When you're done playing with one environment, close the program and re-open to switch to another environment.

Once you've debugged your code, it's time to create the program for distribution. This is a great opportunity to allow Accounting to do a database backup before they run month end or the development team to backup a database before they put in a release.

Building your application installation file

Please note, this is a brief summary intended for those (like myself) who have never done this before. The scope of this article does not include all the possible variations or security options available to you.

Double-click My Project on Solution Explorer. Make sure the Application tab is highlighted and that the Startup Form is listed as ChooseEnvironment.

Using the main menu in VS, navigate to File -> Add -> New Project -> Other Project Types -> Setup and Deployment. The default name of the new project is Setup1 and it will bring up a new page in your project called File System. (See Figure 4)

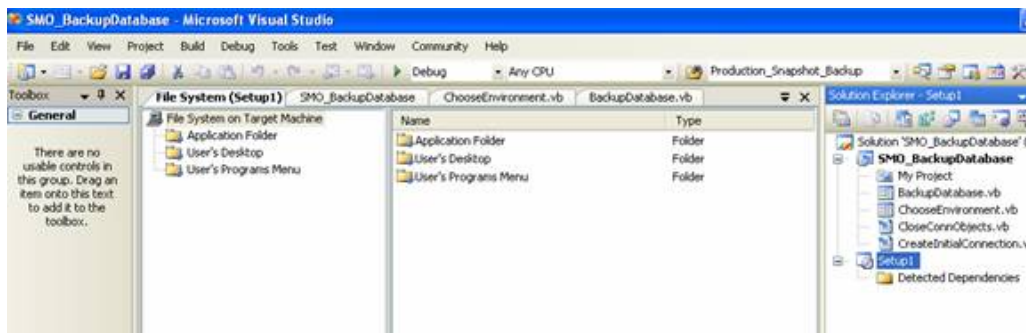


FIGURE 4

Click Setup1 in Solution Explorer. Look at the properties window. Here is where you will enter useful information such as Author's name, version #, localization information and a description of the program. (See Figure 5)

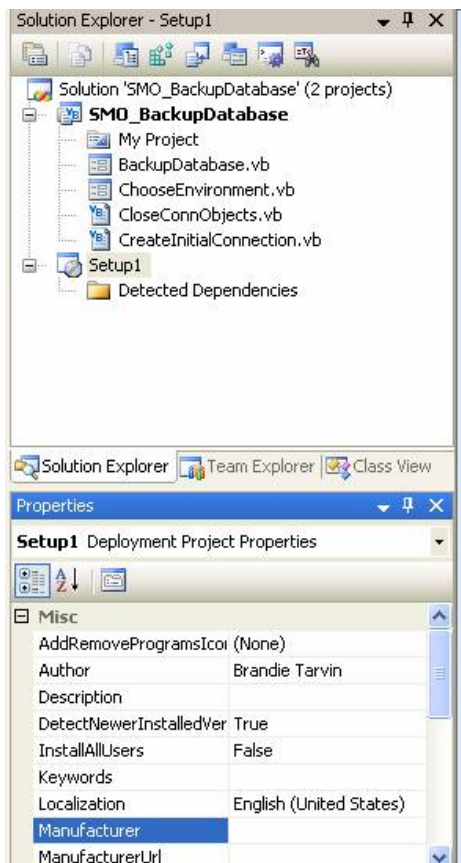


FIGURE 5

Right-click Setup1 and choose Properties. VS will bring up a window telling you the default location of where the .msi file will be created. You can change this if you want, but don't change it too much. Keeping the .msi file with your project will make finding it much easier at a later date. Click Cancel to exit.

Right-click the Application Folder and choose Add -> Project Output. A window will pop up asking what output. You'll want to make sure Primary output and a configuration of Active are chosen. Click OK. (See Figure 6)

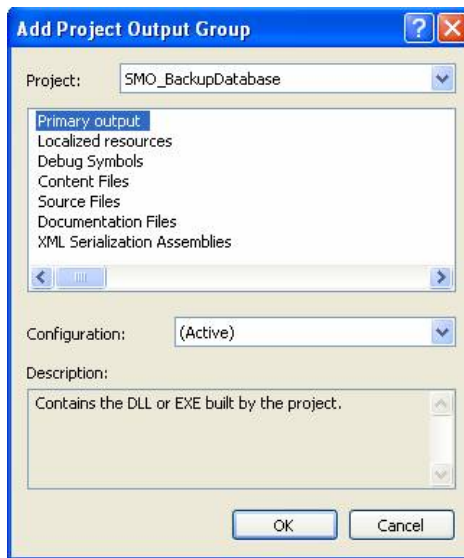


FIGURE 6

Right-click the Primary Output and choose "Create Shortcut to Primary Output..." which will force a shortcut to be created. (See Figure 7) Rename the shortcut to something appropriate and user-friendly. Drag the new item into either the "User's Desktop" or "User's Program Menu" folders. Upon installation, a shortcut in the User's Start -> Program Files menu and/or the User's desktop will be created. If you want one of each, you have to create 2 shortcuts to the application folder and drag them to the appropriate folder.

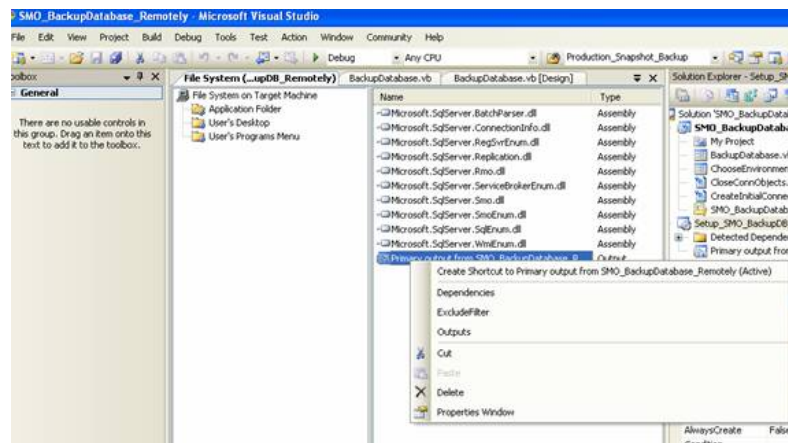


FIGURE 7

Right-click Setup1 and choose Build this time. This will build your .msi file for later distribution. If you left the defaults, you'll find your .msi under SMO_BackupDatabase\Setup1\Debug. You can distribute it to whomever might need the ability to backup the database. Double-clicking the .msi installs it automatically.

When all is said and done, this code should work perfectly for remotely backing up databases, as long as you have permission to do so. I'm still learning VB .Net myself, so there's a lot of things missing from this version of my code (error handling, re-initializing the server connection to switch environments, progress bars, etc.). If anyone has any thoughts on how this could be improved, I'd be glad to hear them.

Having fun coding!

Document Your Database

By Raj Vasant

Introduction

Documentation for the tasks related to a particular project is required for easy maintenance and quick reference. Number of situations has been encountered where we all go through user-manuals or read-me files for getting help/solutions. Most of the applications are having documents that provide in-depth description of the design, code and maintenance related stuff. Most of these documents are for the development team but some of them are also provided to the customers.

Consider a scenario:

You are currently involved in designing a database i.e., created tables, defined constraints, written various scripts, jobs, etc. One fine day, you got an opportunity to work with Microsoft (Nice opportunity, isn't it!!!). The company for which you are working has already hired your replacement. Doing transition or knowledge transfer becomes tuff and boring. So, you try to get rid of it as fast as possible and in turn making the life of your REPLACEMENT miserable.

There will be documents written when the database design was discussed and some of them include the database design in detail. But, some way or the other we miss out on small things that are very crucial, specially the updates or modifications to the database objects. Also, the documents that are created are not concentrated on the complete database design. For example:

- Why a particular table is created?
- What is the need of a particular column?
- What is the use of XYZ stored procedures?
- What is done when a particular job is executed?

These are some mystery questions that will be un-answered. Unless the REPLACEMENT takes the initiative and starts digging in the database, the mystery would not be solved.

How to Document?

There are various tools available that can be used to document your database. The best is Microsoft Word. Instead of writing the overview of the database and storing the sql-scripts in the documents, it is a good practice to write down the details about each table, column, stored procedure, view, job and other database objects. Still, this method demands regular update of the documents as and when the design or functionality changes are made.

Extended properties can also be used to document individual database objects. There are various stored procedures which allow to add and modify details.

Using SQL Server 2005 as your database will be useful, as it provides a good way to add description to all the database objects when they are created. You can specify description for each column [also in SQL Server 2000]. In SQL Server 2005, it is also possible to add description for tables, stored procedures and other database objects. It is helpful because you can get the information instantly without searching a document.

There are tools available that will run through your database and give output as compiled-html file or normal html pages. The output is well categorized and contains all the database objects, with their descriptions. I have tried [ApexSQL Doc](#), and it is really helpful. It provides a good user-interface for selectively documenting various database objects.

Conclusion

Documenting the database perfectly up to the column level is necessary and a good practice so that the design can be better understood. Also, there are some instances when the designer of the database is not sure about the existence of a particular table/column/script. Putting little extra effort by specifying descriptions for the database objects will make life easier for everyone.

A Transactional Replication Primer

By Claudi Rego

Introduction

This tutorial intends to show as you can use a transactional replication without changing the initial snapshot, i.e., without carrying out the initial synchronization between the Publisher and Subscriber. It is presupposed you have knowledge of the replication process, particularly of the configuration of a transactional type replication.

Sometimes it is not feasible to synchronize subscriptions during the transactional replication. For a while I tried to carry out transactional replication but unsuccessfully. Actually, my DB was very heavy with large tables and low network bandwidth.

For such cases there is a solution! It is possible to use copies of the published DB and restore the data in Subscriber. This way we avoid the exchange of the initial snapshot over the network, which is in fact the heaviest part of the process. The following transactions are much lighter.

However, configuring the transactional replication this way, some attention must be paid to the details. Focus on the detailed explanation that will follow of how to carry out this configuration.

Before Starting

In the CLAUDIA server, which will act as Publisher, I have created a new DB. It is termed as cars and it will keep information about cars. It is constituted by the Items table with the design presented in table 1.

Column Name	Data Type	Length	Allow Nulls
Id	int	4	
Model	char	10	Y
Engine	int	4	Y
Fuel	char	10	Y
Price	float	8	Y

Table 1 Design Table Items

The field Id is defined as PK and as Identity (identity seed = identity increment = 1). Data was introduced in the table and its aspect changed into the one presented in table 2.

Id	Model	Engine	Fuel	Price
1	Yaris	1350	Gas	10,95
2	Corolla	2500	Diesel	15,25

Table 2 Data present in Items Table

Now the replication process should be configured.

Configure Transactional Replication

The transactional replication stores and directs to Subscriber transactions in a series. So that this becomes possible it is necessary that the DBs in Subscriber and Publisher are synchronized. In the transactional replication, this synchronization is done sending an initial snapshot to Subscriber.

Doing the synchronization manually, it is possible to reduce the time that the snapshot process takes to the time that making the DB backup takes.

Distributor and Publisher

In this tutorial, the CLAUDIA server is expected to have been already configured as Publisher and Distributor.

Configure Publishing and Distribution

At this point we should indicate the DB which contains the information to be replicated and the destination server of this information.

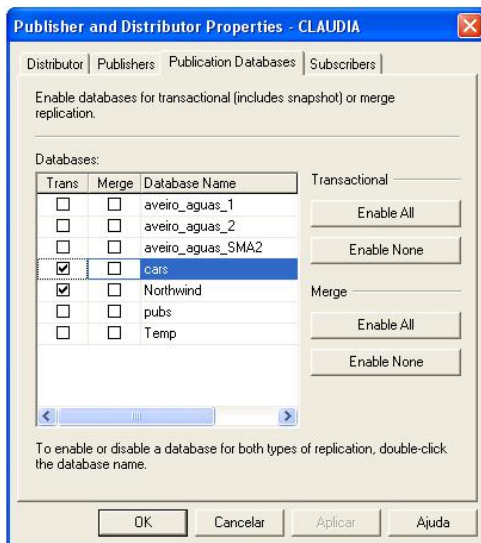


Figure 1 Configure Publishing and Distribution Wizard

Publication Databases

We should start by indicating that the new DB is available to be replicated. This way, using the wizard Configure Publishing and Distribution, you enable the DB cars to do the transactional replication (Trans) in the Publication Database tab. (see Figure 1)

Subscriber

In the same wizard, in the Subscribers tab the TORMENTAS server is qualified as Subscriber. (see Figure 2)

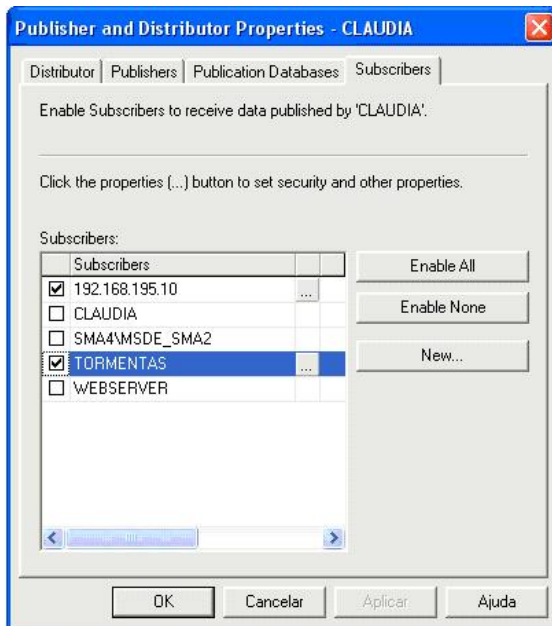


Figure 2 Configure Publishing and Distribution Wizard

We should now indicate that we intend to replicate the data, i.e., to create a new publication.

Create and Manage Publications - New Publication

To create a new publication it was used the wizard Create and Manage Publications represented in figure 3.

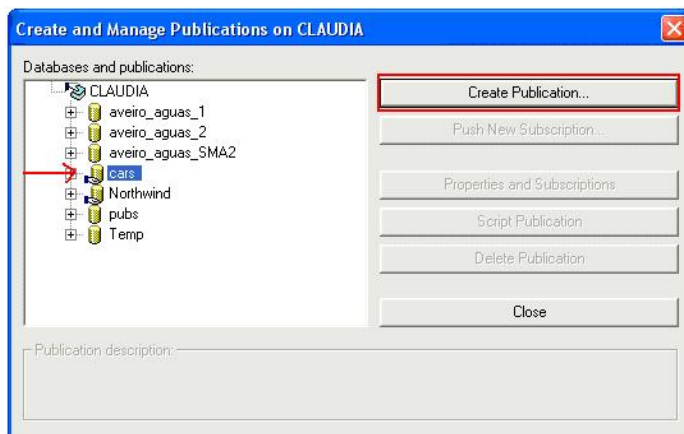


Figure 3 Create Publication Wizard

The new publication was termed as cars_pub as it belongs to the transactional type and with a single article, the Items table.

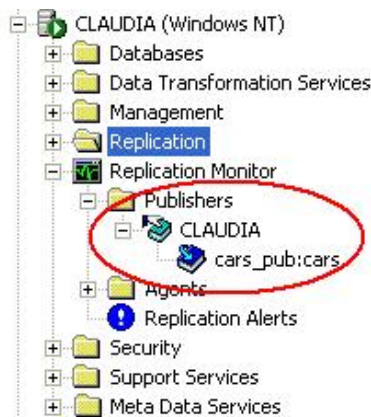


Figure 4 New publication cars_pub

Assure synchronism between Publisher and Subscriber

The DB should be published in the Single User Mode in order to prevent its alteration. This way, it is guaranteed that the Publisher will be synchronized with the Subscriber. Before carrying out the stored procedure `sp_dboption` all the replication agents linked to the DB in question should be stopped. Otherwise this operation cannot be carried out. See Figures 5 and 6.

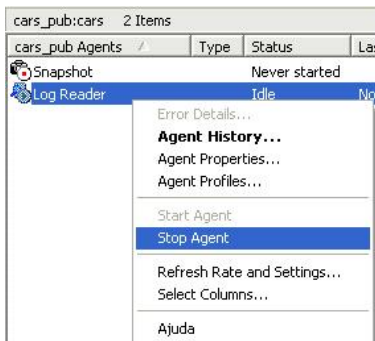


Figure 5 Parando os agents da replicao

The mode of the DB cars should be changed into Single User:

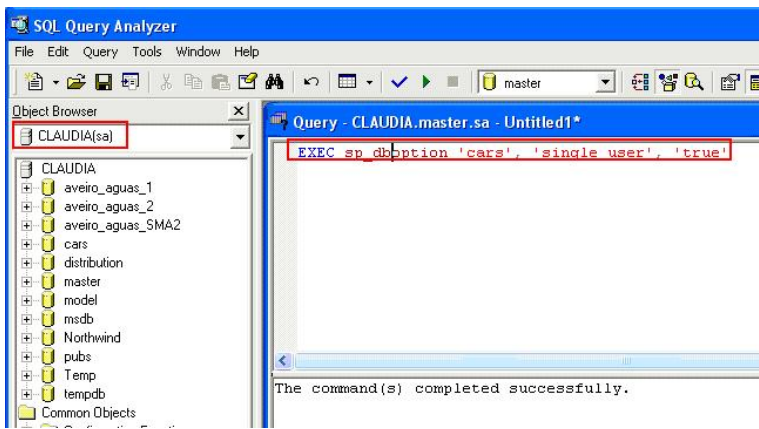


Figure 6 Colocando a DB no Modo Single User atravs do Query Analyzer

After carrying out this operation with success the DB graphically indicates its new Mode. See Figure 7.

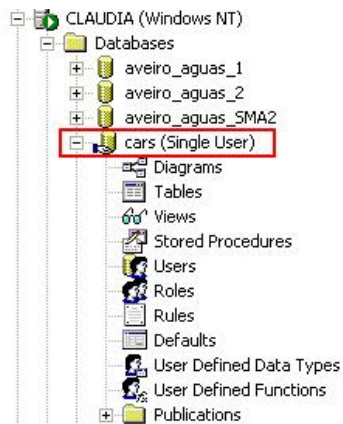


Figure 7 DB no Modo Single User

Backup

There are two methods to make the backup of the published DB. The first method uses the Transaction Log Backup and it assumes that a DB's Full Backup has been already made. The second method uses the Full Backup of the published DB. This is the favorite method when we work with a small DB or if the DB hasn't been configured for the Full Recovery Mode.

Transaction Log

When this method is used, it takes less time for the DB to be in Single User Mode. It also takes less time to make a backup of the Transaction Log than to make a full backup. To use this method, the DB must be using the Full Recovery Mode. This method was not tested in this tutorial.

Full Backup

If the Full Backup is the method being used, a backup of the published DB should be made.

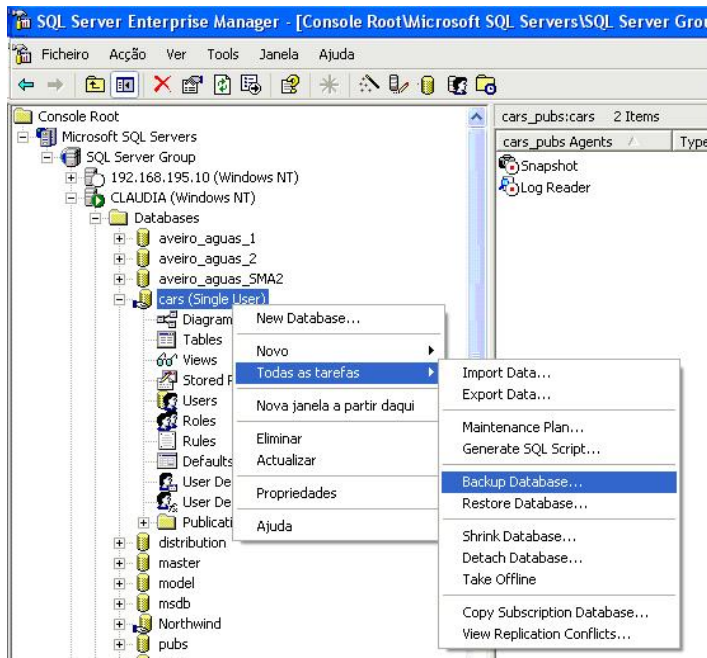


Figure 8 Backup da BD

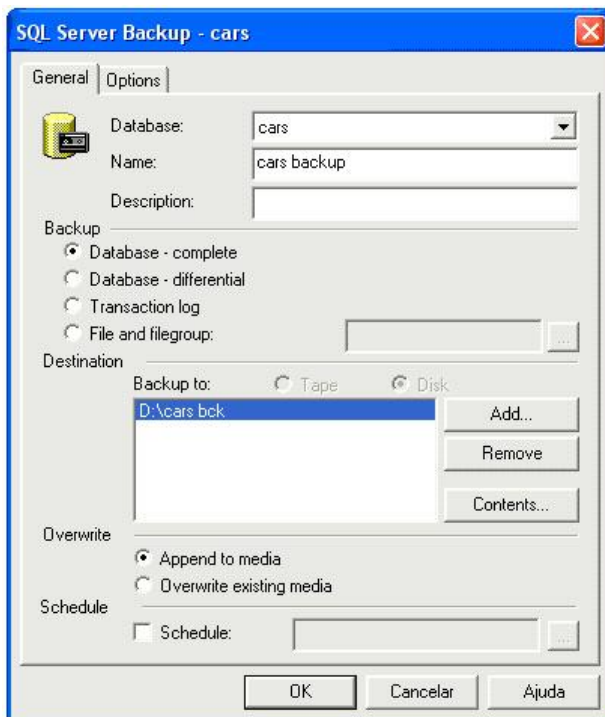


Figure 9 Indicando um backup complete da BD

Subscription - Push Subscription

At this point the data's addressee should be configured. For that a new subscription of the Push type was created, indicating the TORMENTAS server as destination. The destination DB was created at the same as the subscription and it is termed cars as well.

Schedule Distribution Agent

Now the schedule set of the Distribution Agent. It is crucial to schedule it so that the whole configuration process of the replication without initial snapshot is concluded. This way, you should disable a continuous update and schedule this agent. It is suggested that the initial date is tomorrow (15-08-2006). The Figures 10 and 11 show what has been said before.

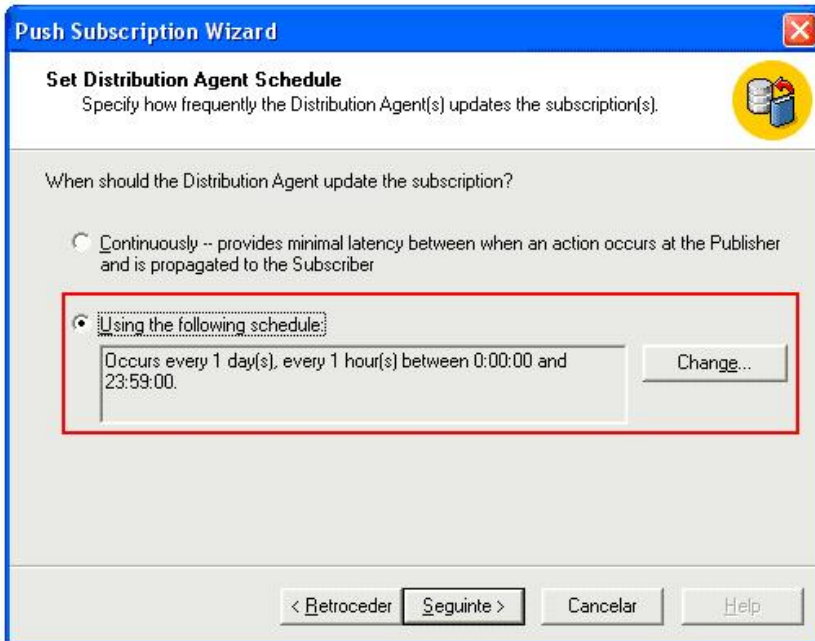


Figure 10 Create Push Subscription Wizard - Set Distribution Agent Schedule

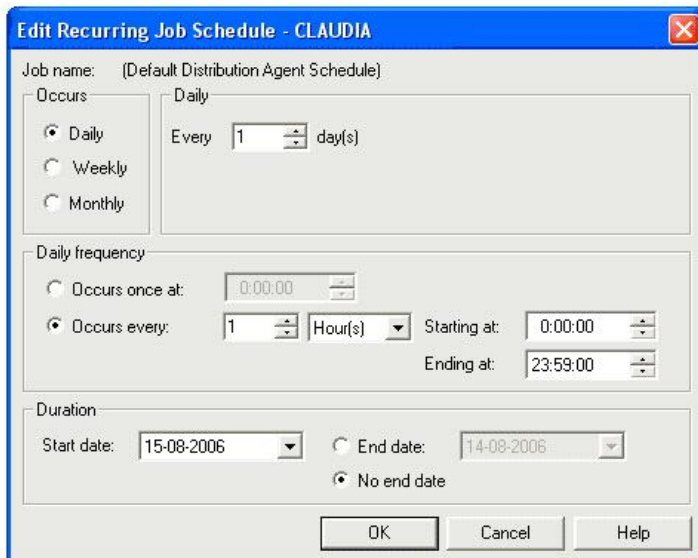


Figure 11 Create Push Subscription Wizard - Edit Recurring Job Schedule

Initialize Subscription

Another fundamental aspect is to indicate the source and destination, i.e., the Publisher and Subscriber are already synchronized and ready for the replication and therefore, there is no need to apply the initial snapshot.

Therefore, on being asked about the initialization of the subscription, you should answer "No, the Subscriber already has the schema and data". See Figure 12.

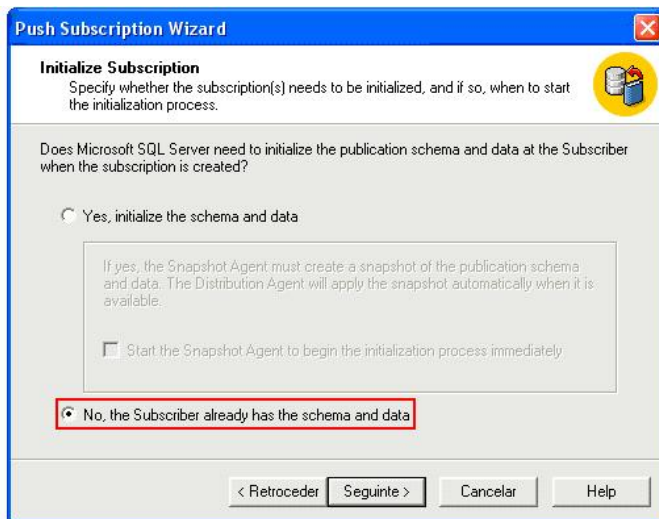


Figure 12 Create Push Subscription Wizard - Initialize Subscription

Replication Agents in Details Pane

Now, in the Details Pane, the following agents for the cars_pubs publications appear.

cars_pubs:cars 3 Items			
cars_pubs Agents	Type	Status	Last Action
Snapshot		Never started	
Log Reader		Succeeded	The process was successfully stopped.
TORMENTAS:cars	Push	Never started	

Figure 13 Details on car_pubs publication

Now the DB can be taken out of the Single User Mode as the process is configured and all the alterations will be directed to the DB distribution.

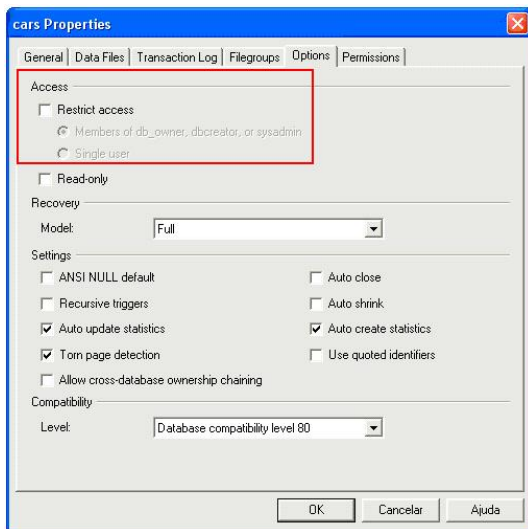


Figure 14 Retirar a BD do Modo Single User atravs das Propriedades

It is verified that now it gives error if you try to use the Query Analyzer.

Restore DB

It is not enough to indicate that the DBs are synchronized in order to obtain a transactional replication without changing the initial snapshot. It is necessary to synchronize them effectively.

As it has been said before, it was made a (full) backup of the DB in question. Now it is necessary to send this file to the Subscriber server machine.

Regarding this tutorial the machines are close to each other and therefore it was feasible only to do a Copy + Paste. If the machines are far from each other, there is the possibility to record the backup in a disk and send it per airmail.

For that: Run

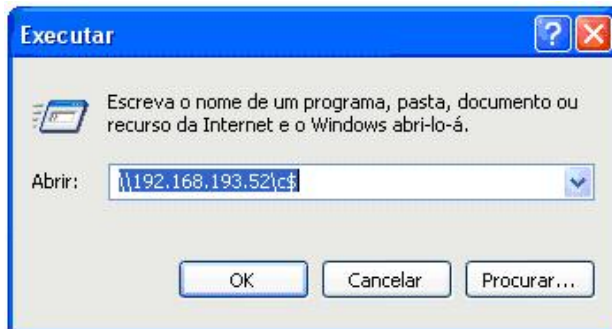


Figure 15 Enviar o ficheiro de backup para a mquina do Subscriber

Simply with a CTRL+C and CTRL+V you will move a copy of that backup file. In order to do the DB's restore, the Distribution Agent should not be running. Otherwise it gives an error.

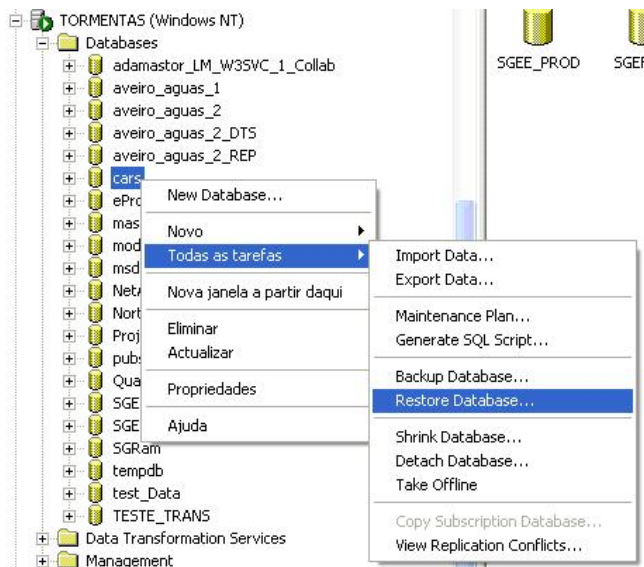


Figure 16 Restore Database no Subscriber

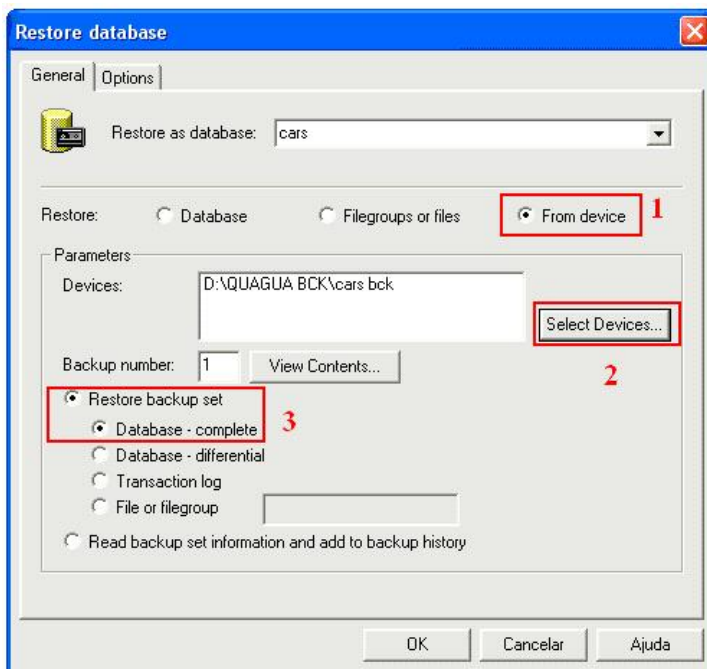


Figure 17 Restore DB as cars

The wizard showed in Figure 17 will appear. It is important to indicate that the restore will be carried out from the disk (Device); afterwards the path to the backup file will be indicated and it should be shown that it is a full backup.

After receiving the message the restore is successfully done you can confirm that in the destination, TORMENTAS server, it was created an Items table, which contains the same information. This way the synchronism between the Publisher and Subscriber will be obtained. It is to highlight that the DB in Subscriber may appear in the Single User Mode. Therefore that should be altered back to multi-user.

Stored Procedures

In a transactional replication, during the initial snapshot, 3 stored procedures per article are generated, one for each type of transaction. Particularly in this tutorial, there is the Items article, and the following stored procedures would be generated:

- sp_MSins_Items (inserts);
- sp_MSdel_Items (deletes);
- sp_MSupd_Items (updates).

As the synchronization process is being done manually, we also have to write the stored procedures or generate them. Fortunately there is already a stored procedure to generate them.

Script custom stored procedures at Publisher

Now it is necessary to create the procedures of Insert, Update and Delete that are used during the replication. For that, in case we are working with a SQL Server 2000 (without immediate updating and queued Subscribers) we only need to run the stored procedure sp_scriptpublicationcustomprocs in Publisher. This stored procedure should be carried out in the published DB.

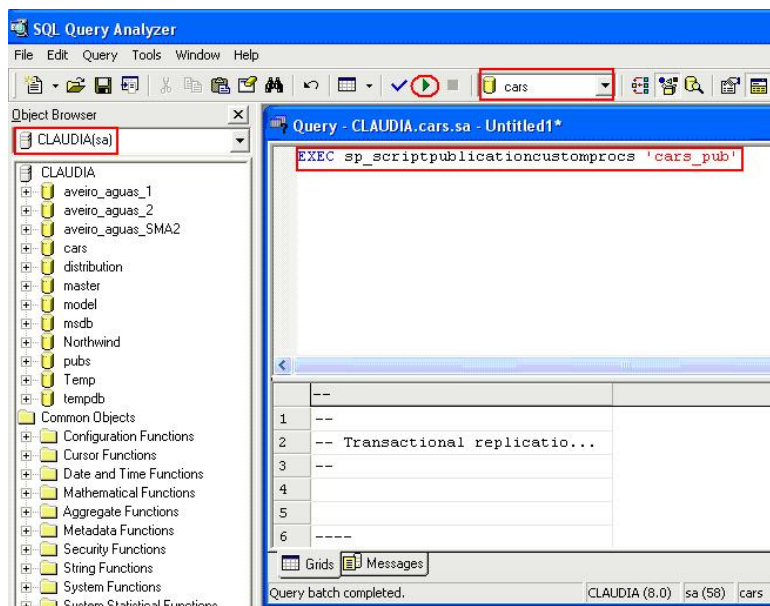


Figure 18 Gerar os precedures Insert, Update e Delete com o stored procedure `sp_scriptpublicationcustomprocs`

It created the code showed in Figure 19.

```
-- Transactional replication custom procedures for publication 'cars_pub' from database
--

-----
----- Replication custom procedures for article 'Items':
-----

if exists (select * from sysobjects where type = 'P' and name = 'sp_MSins_Items') drop
go
create procedure [sp_MSins_Items] @c1 int,@c2 char(10),@c3 int,@c4 char(10),@c5 float
AS BEGIN
    insert into [Items] (
        [Id], [Model], [Engine], [Fuel], [Price]
    )
    values (
        @c1, @c2, @c3, @c4, @c5
    )
end
```

Figure 19 Codigo gerado com sp_scriptpublicationcustomprocs

Applying scripts at Subscriber

To apply the created scripts to the Subscriber we should paste the code created previously in Query Analyzer and execute it on the subscriber. Figure 20 shows this operation. It should be highlighted that this should be carried out over the DB in question.

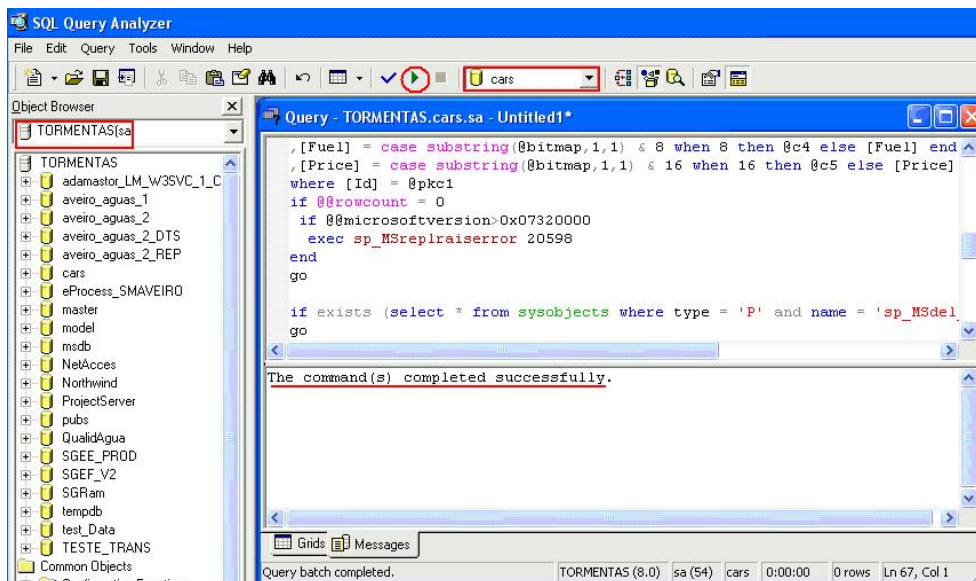


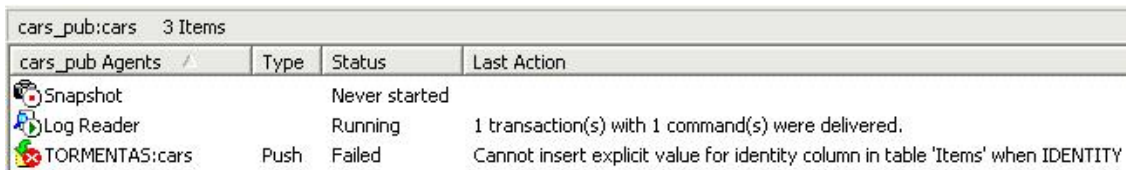
Figure 20 Executando os stored procedures no Subscriber

And that's all! Now all the alterations carried out over the published DB will be reflected in the correspondent DB in the Subscriber.

Last Note

The field Id of the Items table had the enabled Identity property, i.e., yes. As the Items table is precisely equal it will also possess that enabled property. This is due to the method that is being used (backup+restore). It is important to highlight that if there is the delivery of the initial snapshot to the Subscriber, the objects of the DB are not transferred to the Subscriber.

However, The Identity property to Yes, causes error during the replication. See Figure 21.



cars_pub:cars 3 Items			
cars_pub Agents	Type	Status	Last Action
Snapshot		Never started	
Log Reader		Running	1 transaction(s) with 1 command(s) were delivered.
TORMENTAS:cars	Push	Failed	Cannot insert explicit value for identity column in table 'Items' when IDENTITY

Figure 21 Erro gerado pela propriedade Identity

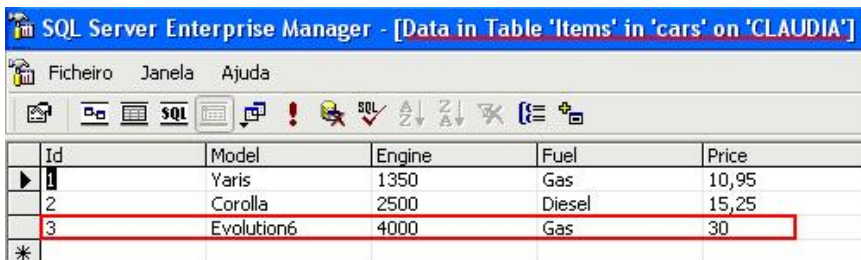
It is necessary to disable that property in Subscriber.

Testing

The following some captures for testing. Three types of transactions carried out in Publisher and reflected in Subscriber will be presented.

Replication Insert

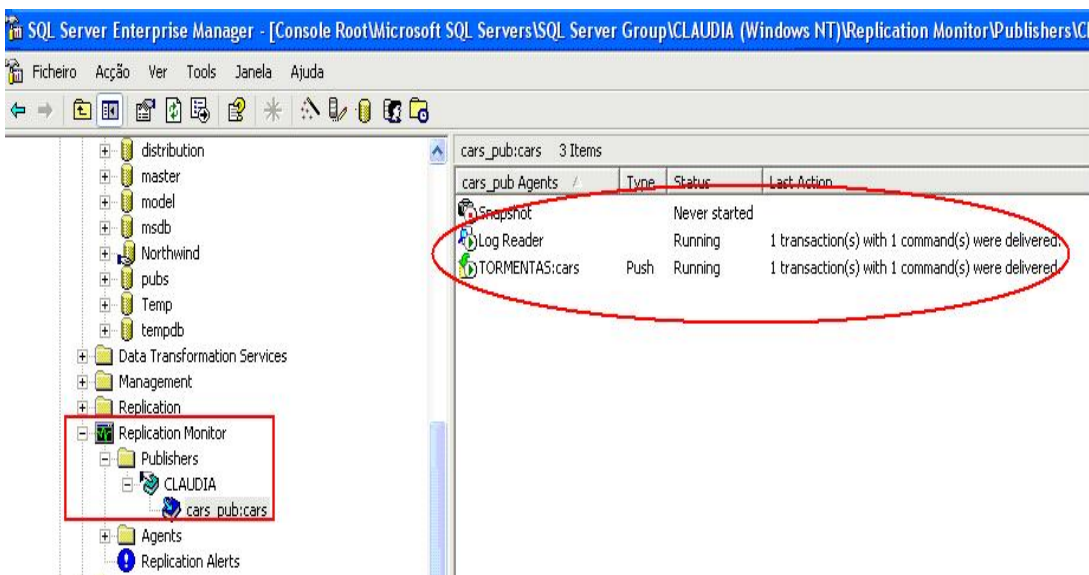
The Figure 23 shows the insert of a row in the Items table in Publisher.



	Id	Model	Engine	Fuel	Price
▶	1	Yaris	1350	Gas	10,95
	2	Corolla	2500	Diesel	15,25
	3	Evolution6	4000	Gas	30
*					

Figure 23 Insert on Publisher

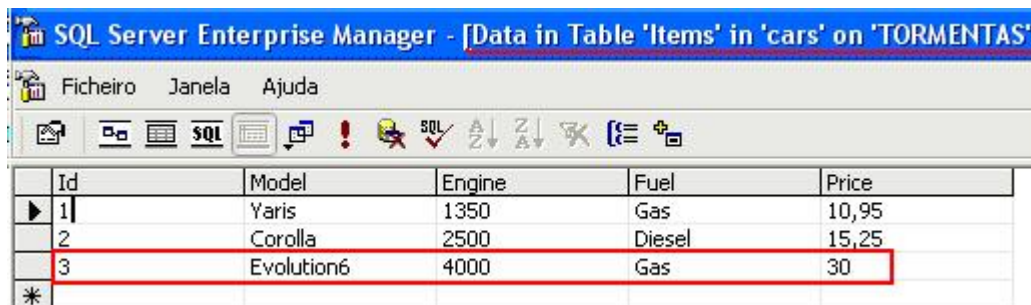
Automatically, the replication agents identify the insert. (see Figure 24)



cars_pub:cars 3 Items			
cars_pub Agents	Type	Status	Last Action
Snapshot		Never started	
Log Reader		Running	1 transaction(s) with 1 command(s) were delivered.
TORMENTAS:cars	Push	Running	1 transaction(s) with 1 command(s) were delivered.

Figure 24 Agentes da replicao detectam o insert

And in the Subscriber, the insert is reflected. (See Figure 25)

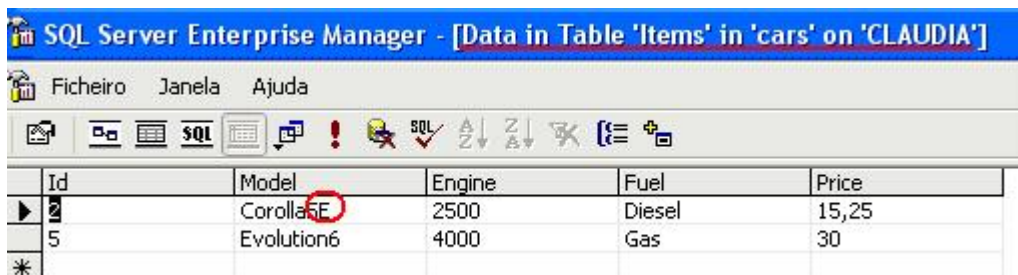


	Id	Model	Engine	Fuel	Price
▶	1	Yaris	1350	Gas	10,95
	2	Corolla	2500	Diesel	15,25
	3	Evolution6	4000	Gas	30
*					

Figure 25 Insert reflected on the Subscriber

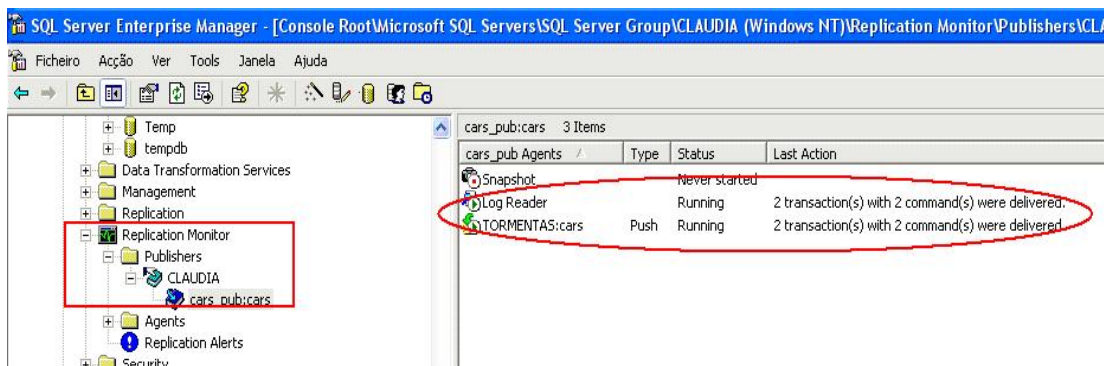
Replication delete and update

The following figures show the other two possibilities, a delete and the update. If you delete (row with Id=1) and update as in Figure 26.



	Id	Model	Engine	Fuel	Price
▶	2	CorollaSE	2500	Diesel	15,25
	5	Evolution6	4000	Gas	30
*					

Figure 26 Delete and Update on the Publisher



cars_pub:cars 3 Items			
	Type	Status	Last Action
Snapshot		Never started	
Log Reader		Running	2 transaction(s) with 2 command(s) were delivered.
TORMENTAS:cars	Push	Running	2 transaction(s) with 2 command(s) were delivered.

Figure 27 Agentes da replicao detectam o delete e o update



SQL Server Enterprise Manager - [Data in Table 'Items' in 'cars' on 'TORMENTAS']

Ficheiro Janela Ajuda

SQL

	Id	Model	Engine	Fuel	Price
▶	2	CorollaSE	2500	Diesel	15,25
▶	3	Evolution6	4000	Gas	30
✱					

Figure 28 Delete and Update reflected on the Subscriber

Conclusion

The experience tells me that the transactional replication is an excellent process to keep one (or more) synchronized copy of a certain DB . With little effort (there is the possibility to configure the whole process resorting to wizards only) it is possible to make the replication work.

However, when it concerns a DB with a considerable dimension or/and the network that links the Publisher and Subscriber has a reduced BW, it becomes complicated to pass the initial snapshot through it. The process showed in this tutorial gets round this situation. Through an easy and quick way we can initiate a replication without submitting the initial synchronization.

Nevertheless source and destination have to be synchronized. This way a manual synchronization is done. For that, the backup of the transaction log or of the DB itself can be used and then applied in the destination. Now we can avoid the network and resort, for instance, to a USB disc. Or even use the network and resort, for example, to the ftp. It is to highlight that despite the fact the ftp is over the network, the backup of a DB can be softer than its snapshot, as this includes, besides the data (.bcp), the schemas (.sch) and constraints (.idx).

T-SQL

The T-SQL language used to be the primary way in which we worked with all aspects of SQL Server. The addition of the CLR, complex scripting to SSIS, and more, may change that, but it seems to me that a strong knowledge of T-SQL is still required for the time being.

This section contains a series of articles that cover some of the more interesting aspects of this query language, including some rather hotly contested ones. A few of these articles are controversial, but that should not prevent you from making your own decisions on what you believe.

In all cases, the code presented here should be considered untested and you should use it at your own risk.

Keyword Searching in SQL Server	106
Recursive Queries in SQL: 1999 and SQL Server 2005	113
Serializing Procedure Calls Without Unnecessary Blocking.....	134
The Truth Table	137
Introduction to Bitmasking in SQL Server 2005.....	142
Self Eliminated Parameters	151
Finding Primes.....	152
CTE Performance.....	155
The Effect of NOLOCK on Performance	160
Converting Hexadecimal String Values to Alpha (ASCII) Strings	163
A Refresher on Joins	166
Using CLR integration to compress BLOBs/CLOBs in SQL Server 2005	172
When To Use Cursors	175
Everybody Reports to Somebody	177
Not In v Not Equal	179
Full Control Over a Randomly Generated Password	183
Performance Effects of NOCOUNT	187
Passing a Table to A Stored Procedure	191
New Column Updates.....	195
The T-SQL Quiz	200
Practical Methods: Naming Conventions.....	205
Large Object Data	208
Row-By-Row Processing Without Cursor	210
Beauty is in the Eye of the Beholder	211

Keyword Searching in SQL Server

By Michael Ahmadi

Introduction

As the concept of search continues to grow in scope, I tend to find myself eschewing multiple levels of categorization in favor of larger groups. I'll give you two examples of what I mean.

Think of your inbox. I used to have folders for different systems, folders for different clients, folders for different colleagues, and so on. Now I tend to leave everything in my inbox and use the various search features of my mail client or other search applications to retrieve messages as I need them. I'm not sure if this is due to the fact that the searching capabilities of these applications are better than they used to be or if I'm just more prone to searching for things since that's what all the hip kids are doing these days.

Think of a data driven web site. A retail site possibly. As you drill down into the categories of goods, you'll usually have a breadcrumb trail somewhere on the page. Do we really drill that far down nowadays or do we immediately go to the search box and type in what we're looking for? If it's the latter case, what's the point of having a bunch of piddly little categories like Electronics > Digital Cameras > Memory Cards > 512MB? Why not just dump everything in one big category and let the search functionality take care of the process of finding something?

I like to be able to find something on the fly when necessary and I don't want to have to remember how I originally categorized it. What's the login info for that FTP site? Categorization is tedious at best, and often times the same person will choose different categorizations for a given item at different times. The following is a simple way to implement basic keyword search functionality using SQL Server and nothing but.

Here are our objects we'll end up with:

- LogEntries (Table)
- Keywords (Table)
- LogEntry_Keyword (Table)
- sp_logentry (Sproc)
- sp_search (Sproc)

Setting up our tables

I'm no cognitive scientist, so I can't tell you exactly how our brains keep track of bits of information. But the things that we need to remember are often very fragmented and defy categorization. I'd venture a guess that our brains are like logs and we make entries into that log. So let's have a table called LogEntries. The most basic column is the actually entry itself - LogEntry varchar(8000):

"Remember that you increased the budget for the Razzmatazz account by \$10K."

And when we make that mental note, it would be nice to remember when we made it - DateEntered datetime:

'YYYY-MM-DD hh:mm:ss'

But when we remember it, do we actually think of the exact phrase we originally thought of? Probably not. We probably remember something more like *"razzmatazz account budget increased"*. So now we have some keywords (or tags) associated with our entry. Tags varchar(200):

'increased budget account'

```
CREATE TABLE LogEntries
(
    EntryId int IDENTITY(1,1) NOT NULL,
    Entry varchar(8000) NOT NULL,
    Tags varchar(200) NOT NULL,
    DateEntered datetime NOT NULL DEFAULT (GETDATE()),
    CONSTRAINT PK_LogEntries PRIMARY KEY CLUSTERED (EntryId ASC)
)
```

Each of the individual tags within the Tags column is a *keyword* describing the data in the LogEntry. So let's have a Keywords table.

```
CREATE TABLE Keywords
(
    KeywordId int IDENTITY(1,1) NOT NULL,
    Keyword varchar(50) NOT NULL,
    CONSTRAINT PK_Keywords PRIMARY KEY CLUSTERED (KeywordId ASC)
)
```

```
CREATE UNIQUE NONCLUSTERED INDEX IX_Keywords ON Keywords
(
    Keyword ASC
)
```

When we make an entry into our log, we'd like to also make inserts into a LogEntry_Keyword table - a table used to calculate the number of occurrences of a keyword or set of keywords in a given entry.

```
CREATE TABLE LogEntry_Keyword
(
    EntryId int NOT NULL,
    KeywordId int NOT NULL,
    Hits int NOT NULL,
    CONSTRAINT PK_LogEntry_Keyword PRIMARY KEY CLUSTERED
(
    EntryId ASC,
```

```
        KeywordId ASC
    )
)
```

Now for the trigger

In most cases, we'd probably prefer to process the tags of our entries by using another application. The `String.Split` function comes to mind. Then we'd hash the keywords where the hash value is the number of occurrences and we'd later do our insert into the `LogEntry_Keyword` table. But we're keeping it simple and want to make everything self contained. Here's how the trigger does that.

```
CREATE TRIGGER trgInsertLogEntry
ON LogEntries FOR INSERT
AS

-----
-- Declarations --
-----

DECLARE @tags      AS varchar(200) -- This will hold the tags string (e.g. 'increase budget such
'n' such account')

DECLARE @keyword   AS varchar(50)  -- An individual keyword from the tags string (e.g.
'increase')

DECLARE @keywordId AS int           -- The keyword id from the keywords table

DECLARE @found     AS int           -- Whether or not the keyword was already in the keywords
table

DECLARE @entryId   AS int           -- The entry id of the logentry being inserted (@@identity)

-----
-- Temp table for current keyword data for the newly inserted LogEntry --
-----

CREATE TABLE #kw
(
    kwid int PRIMARY KEY, -- The id from the Keywords table

    hits int              -- The number of occurrences of this keyword
)
```

```
-----  
-- Data from the newly inserted LogEntry --  
-----  
  
SET @entryId = @@identity          -- Get the newly inserted logentry id  
  
SET @tags     = (SELECT tags FROM INSERTED) -- Get the newly inserted tag  
  
SET @tags     = LTRIM(RTRIM(@tags)) + ' xyz' -- Add a fake keyword to the end that won't get  
inserted  
  
-----  
-- While there are still keywords in the newly inserted LogEntry's tag --  
-----  
  
WHILE (CHARINDEX(' ', @tags) > 0)  
  
BEGIN  
  
    -- Get the next keyword from the tags string  
  
    SET @keyword = SUBSTRING(@tags, 0, CHARINDEX(' ', @tags))  
  
    -- Get the KeywordId from the Keywords table  
  
    SELECT @keywordid = (SELECT KeywordId FROM Keywords WHERE Keyword = @keyword)  
  
    -- Insert the keyword if necessary  
  
    IF (@keywordid IS NULL)  
  
    BEGIN  
  
        INSERT INTO Keywords VALUES (@keyword)  
  
        SET @keywordid = @@identity  
  
    END  
  
    -- See if the keyword id is in the temp table yet  
  
    SELECT @found = (SELECT COUNT(*) FROM #kw WHERE kwid = @keywordid)
```

```
-- If not found insert it

IF (@found = 0)

    INSERT INTO #kw VALUES (@keywordId, 1)

-- If found update the hit count

IF (@found != 0)

    UPDATE #kw SET hits = hits + 1 WHERE kwid = @keywordId

-- Update the tags by lopping off the keyword just processed

SET @tags = substring(@tags, charindex(' ', @tags) + 1, len(@tags) - charindex(' ', @tags))

END

-----

-- End while --

-----

-----

-- Insert the keywords and their occurrences into the LogEntry_Keyword table --

-----

INSERT INTO logentry_keyword

SELECT @entryid, kwid, hits

FROM #kw
```

So that takes care of everything we need to implement our project. Now let's add a few things to make inserting entries easier and finding things easier.

Getting the data in there

The whole purpose behind this exercise is to be able to associate a piece of data with string identifiers. When creating the data we want to be able to say (to ourselves):

Okay, I need to remember that at today's meeting we discussed ways of improving communication within the company. The first way was to blah blah blah...

And then say it more succinctly in keyword form:

meeting ways improve communication within company

That's two things. Two arguments for our sp_logentry function:

```
CREATE PROC sp_logentry(@entry AS varchar(8000), @tags AS varchar(200)) AS
INSERT INTO LogEntries (Entry, Tags)
VALUES (@entry, @tags)
```

sp_logentry 'Okay, I need to remember...blah blah blah...', 'meeting ways improve communication within company'

Getting the data out of there

And now, the last thing we need to do is write our sproc to retrieve our data. This will look quite familiar.

```
CREATE PROC sp_search(@keywords AS varchar(50)) AS
```

```
DECLARE @kws as varchar(50)
```

```
DECLARE @kw as varchar(50)
```

```
DECLARE @kwid as int
```

```
-----
-- Temp table for current keyword data --
-----
```

```
CREATE TABLE #kw
```

```
(
kwid int PRIMARY KEY
)
```

```
-- Add a fake keyword that won't get inserted
```

```
SET @kws = LTRIM(RTRIM(@keywords)) + ' xyz'
```

```
-----
-- While there are still keywords --
```



```
-----  
WHILE (CHARINDEX(' ', @kws) > 0)  
  
BEGIN  
  
    SET @kw = SUBSTRING(@kws, 0, CHARINDEX(' ', @kws))           -- Get the tag from the  
string  
  
    SELECT @kwid = (SELECT keywordid FROM keywords WHERE keyword = @kw) -- Find the keyword id  
  
    IF (@kwid IS NOT NULL)                                         -- If found insert the id  
into the temp table  
  
        INSERT INTO #kw VALUES (@kwid)  
  
    SET @kws = SUBSTRING(@kw, CHARINDEX(' ', @kws) + 1, LEN(@kws) -- Update the keywords  
        - CHARINDEX(' ', @kws) - 1)  
  
END  
  
  
SELECT le.EntryId, Entry, Tags, DateEntered, SUM(Hits) h  
  
FROM LogEntries le  
  
JOIN LogEntry_Keyword lek ON lek.EntryId = le.EntryId  
  
JOIN #kw tkw ON tkw.kwid = lek.keywordid  
  
GROUP BY le.EntryId, Entry, Tags, DateEntered  
  
ORDER BY h DESC  
  
sp_search 'company meeting'
```

Conclusions

Yes, this is probably more of an exercise than something you'll put into practice. But the basic idea is straightforward and useful when expounded upon. If you decide to do more with it here are some ideas as well as some things to look out for.

- Hook it up to a basic UI for a popup notepad.
- Take care to validate your tags string to make sure it's a space delimited string.
- I might also suggest adding a *having count(le.EntryId) >= (select count(*) from #kw)* clause into the search sproc in the event you want to only include entries featuring all of the searched keywords.
- With a little more validation, you can get rid of the Tags column and process the LogEntry column by itself.

Recursive Queries in SQL: 1999 and SQL Server 2005

By Frédéric Brouard

Everybody has at one time in his life, had experience with recursion. When I was young, I was on leave in Paris in an old building in which the corridor had two mirrors facing each other. When I passed between these mirrors my body was reflected *ad infinitum*, and I was very proud, joyfully admiring my image and having a concrete view of what is the infinite. That is it recursion, a process which is able to reproduce itself for some period of time.

In mechanical situations, we do not accept infinite recursion. In the real world, we must have a stopping point because our universe is closed. Waiting for the end of an infinite process, which in fact is eternity, is a hard job! As Woody Allen says: "*eternity is really long, especially near the end ...*"

In computer management, recursion is a special technique that is able, sometimes, to treat complex algorithms with an elegant coding style: a few lines will do a complete job. But recursion has some perverse effects: the resources to do the job are maximized by the fact that every call of the embedded process needs to open a complete environment space, which has the effect of using a large volume of memory. A mathematician, whose name I cannot recall, says that every recursive algorithm can be reduced to an iterative one by the use of a stack!

But our purpose in this article is to speak about RECURSIVE QUERIES in SQL, regarding the ISO standard and what MS SQL Server 2005 has done with it.

The ISO SQL: 1999 standard

Here is the short syntax of a RECURSIVE QUERY:

```
WITH [ RECURSIVE ] <query_alias_name> [ ( <column_list> ) ]  
  
AS ( <select_query> )  
  
<query_using_query_alias_name>
```

Simple! Isn't it? In fact, all the mechanics are inside the <select_query>. We will show first simple, but not recursive queries, and when we understand what we can do with the keyword WITH, we will tear down the curtain to see how sexy recursion is in SQL.

A simple CTE

The use of only the WITH clause (without the keyword RECURSIVE), is to build a Common Table Expression (CTE). In a way CTE is a view build especially for a query and used in one shot: each time we execute the query. In one sense it can be called a "non persistent view".

The basic use of a CTE is to make clear some expression that contains a query twice or more in a complex query. Here is a basic example:

```
-- if exists, drop the table we need for the demo  
  
IF EXISTS (SELECT *  
  
          FROM    INFORMATION_SCHEMA.TABLES  
  
          WHERE   TABLE_SCHEMA = USER  
  
          AND     TABLE_NAME = 'T_NEWS')  
  
DROP TABLE T_NEWS
```

```
GO

-- create the table

CREATE TABLE T_NEWS

(NEW_ID          INTEGER NOT NULL PRIMARY KEY,

 NEW_FORUM       VARCHAR(16),

 NEW_QUESTION    VARCHAR(32))

GO

-- population

INSERT INTO T_NEWS VALUES (1, 'SQL', 'What is SQL ?')

INSERT INTO T_NEWS VALUES (2, 'SQL', 'What do we do now ?')

INSERT INTO T_NEWS VALUES (3, 'Microsoft', 'Is SQL 2005 ready for use ?')

INSERT INTO T_NEWS VALUES (4, 'Microsoft', 'Did SQL2000 use RECURSION ?')

INSERT INTO T_NEWS VALUES (5, 'Microsoft', 'Where am I ?')


-- the traditionnal query :

SELECT COUNT(NEW_ID) AS NEW_NBR, NEW_FORUM

FROM    T_NEWS

GROUP   BY NEW_FORUM

HAVING COUNT(NEW_ID) = ( SELECT MAX(NEW_NBR)

                        FROM ( SELECT COUNT(NEW_ID) AS NEW_NBR, NEW_FORUM

                                FROM    T_NEWS

                                GROUP   BY NEW_FORUM ) T )


-- the result :

NEW_NBR    NEW_FORUM

-----

3          Microsoft
```

This query is one that is very popular in many forums, that is, the one that often has the most of number of questions. To build the query, we need to make a MAX(COUNT(... which is not allowed, and so it must be solved through the use of sub-queries. But in the above query, there are two SELECT statements, which are exactly the same:

```
SELECT COUNT(NEW_ID) AS NEW_NBR, NEW_FORUM

FROM    T_NEWS
```

```
GROUP BY NEW_FORUM
```

With the use of a CTE, we can now make the query more readable:

```
WITH

    Q_COUNT_NEWS (NBR, FORUM)

AS

    (SELECT COUNT(NEW_ID), NEW_FORUM

     FROM   T_NEWS

     GROUP BY NEW_FORUM)

SELECT NBR, FORUM

FROM   Q_COUNT_NEWS

WHERE  NBR = (SELECT MAX(NBR)

              FROM   Q_COUNT_NEWS)
```

In fact, we use the non persistent view Q_COUNT_NEWS introduced by the WITH expression, to write a more elegant version of the solution of our problem. Like a view, you must name the CTE and you can give new names to columns that are placed in the SELECT clause of the CTE, but this is not an obligation.

Note the fact, that you can use two, three or more CTE to build a query... Let us see an example:

```
WITH

    Q_COUNT_NEWS (NBR, FORUM)

AS

    (SELECT COUNT(NEW_ID), NEW_FORUM

     FROM   T_NEWS

     GROUP BY NEW_FORUM),

    Q_MAX_COUNT_NEWS (NBR)

AS (SELECT MAX(NBR)

     FROM   Q_COUNT_NEWS)

SELECT T1.*

FROM   Q_COUNT_NEWS T1

       INNER JOIN Q_MAX_COUNT_NEWS T2

              ON T1.NBR = T2.NBR
```

This gives the same results as the two prior versions! The first CTE, Q_COUNT_NEWS, is used as a table in the second and the two CTEs are joined in the query to give the result. Note the comma which separates the two CTEs.

3 - Two Tricks for Recursion

To do recursion, the SQL syntax needs two tricks:

FIRST: you must give a starting point for recursion. This must be done by a two part query. The first query says where to begin, and the second query says where to go to next step. These two queries are joined by a UNION ALL set operation.

SECOND: you need to make a correlation between the CTE and the SQL inside the CTE (*Inside out, outside in*, was a popular disco song ... isn't it ?) to progress step by step. That is made by referencing the <query_alias_name> inside the SQL that builds the CTE.

4 - First example: a simple hierarchy

For this example, I have made a table which contains a typology of vehicles :

```
-- if exists, drop the table we need for the demo

IF EXISTS (SELECT *
           FROM INFORMATION_SCHEMA.TABLES
           WHERE TABLE_SCHEMA = USER
           AND TABLE_NAME = 'T_VEHICULE')
    DROP TABLE T_VEHICULE

-- create it

CREATE TABLE T_VEHICULE
(VHC_ID          INTEGER NOT NULL PRIMARY KEY,
 VHC_ID_FATHER   INTEGER FOREIGN KEY REFERENCES T_VEHICULE (VHC_ID),
 VHC_NAME        VARCHAR(16))

-- populate

INSERT INTO T_VEHICULE VALUES (1, NULL, 'ALL')

INSERT INTO T_VEHICULE VALUES (2, 1, 'SEA')

INSERT INTO T_VEHICULE VALUES (3, 1, 'EARTH')

INSERT INTO T_VEHICULE VALUES (4, 1, 'AIR')

INSERT INTO T_VEHICULE VALUES (5, 2, 'SUBMARINE')

INSERT INTO T_VEHICULE VALUES (6, 2, 'BOAT')

INSERT INTO T_VEHICULE VALUES (7, 3, 'CAR')

INSERT INTO T_VEHICULE VALUES (8, 3, 'TWO WHEELS')

INSERT INTO T_VEHICULE VALUES (9, 3, 'TRUCK')
```

```
INSERT INTO T_VEICULE VALUES (10, 4, 'ROCKET')
INSERT INTO T_VEICULE VALUES (11, 4, 'PLANE')
INSERT INTO T_VEICULE VALUES (12, 8, 'MOTORCYCLE')
INSERT INTO T_VEICULE VALUES (13, 8, 'BYCYCLE')
```

Usually a hierarchy must be schematized with an auto reference, which is the case here: a foreign key references the primary key of the table. These data can be viewed as:

```
ALL
|--SEA
|  |--SUBMARINE
|  |--BOAT
|--EARTH
|  |--CAR
|  |--TWO WHEELS
|    |--MOTORCYCLE
|    |--BYCYCLE
|  |--TRUCK
|--AIR
    |--ROCKET
    |--PLANE
```

Now let us construct the query. We want to know where the MOTORCYCLE comes from. In other words, what are all the ancestors of "MOTORCYCLE". We must start with the data of the row which contain the motorbike:

```
SELECT VHC_NAME, VHC_ID_FATHER
FROM   T_VEICULE
WHERE  VHC_NAME = 'MOTORCYCLE'
```

We need to have the father ID to go to next step. The second query, which does the next step, must be written like this:

```
SELECT VHC_NAME, VHC_ID_FATHER
FROM   T_VEICULE
```

As you see, there is no difference between the two queries, except that we do not specify the filter WHERE to go to next step. Remember that we need to join those two queries by a UNION ALL, which will specify the stepping method:

```
SELECT VHC_NAME, VHC_ID_FATHER
```

```
FROM    T_VEHICLE

WHERE   VHC_NAME = 'MOTORCYCLE'

UNION ALL

SELECT  VHC_NAME, VHC_ID_FATHER

FROM    T_VEHICLE
```

Now let us place this stuff in the CTE:

```
WITH

    tree (data, id)

AS (SELECT VHC_NAME, VHC_ID_FATHER

    FROM    T_VEHICLE

    WHERE   VHC_NAME = 'MOTORCYCLE'

    UNION ALL

    SELECT  VHC_NAME, VHC_ID_FATHER

    FROM    T_VEHICLE)
```

Now we are close to the recursion. The last step is to make a cycle to execute the stepping techniques. This is done by using the CTE name as a table inside the SQL of the CTE. In our case, we must join the second query of the CTE to the CTE itself, by the chain made with tree.id = (second query).VHC_ID. This can be realized like this:

```
WITH

    tree (data, id)

AS (SELECT VHC_NAME, VHC_ID_FATHER

    FROM    T_VEHICLE

    WHERE   VHC_NAME = 'MOTORCYCLE'

    UNION ALL

    SELECT  VHC_NAME, VHC_ID_FATHER

    FROM    T_VEHICLE V

            INNER JOIN tree t

                    ON t.id = V.VHC_ID)

SELECT *

FROM    tree
```

There is nothing more to do other than make the select as simple as possible to show the data. Now, if you press the F5 button to execute... You will see this:

```
data          id
```

```
-----
MOTORCYCLE      8
TWO WHEELS      3
EARTH           1
ALL             NULL
```

Now have a look back at the relationships that do the stepping, in a graphic view:

```
correlation

      _____
      |                                     |
      | v                                 |
WITH tree (data, id)                       |
AS (SELECT VHC_NAME, VHC_ID_FATHER         |
    FROM   T_VEHICULE                     |
    WHERE  VHC_NAME = 'MOTORCYCLE'        |
    UNION ALL                             |
    SELECT VHC_NAME, VHC_ID_FATHER         |
    FROM   T_VEHICULE V                   |
    INNER JOIN tree t <-----
           ON t.id = V.VHC_ID)

SELECT *

FROM   tree
```

By the way, what stopped the recursive process? The fact that no more chains are possible when arriving with a "NULL" value id, which is the case in this example when we reach "ALL".

Now, you get the technique. Notice that for obscure reasons, MS SQL Server 2005 does not accept the RECURSIVE key word following the WITH introducing CTE. But 2005 is a beta version actually, so we can expect that this will be solved in the final product.

5 - Hierarchical indentation

One more important thing with trees structured data is to view the data as a tree... which means a hierarchical indentation when retrieving the data. Is this possible? Yes, of course. The order need to knows the path, the level for placing *space* characters and the id or the timestamp of the row in case of rows of similar tree placement (multileaves data). This can be done by calculating the path *inside* the recursion. Here is an example of such a query:

```
WITH tree (data, id, level, pathstr)

AS (SELECT VHC_NAME, VHC_ID, 0,
```



```

        CAST(' ' AS VARCHAR(MAX))

FROM    T_VEHICLE

WHERE   VHC_ID_FATHER IS NULL

UNION ALL

SELECT  VHC_NAME, VHC_ID, t.level + 1, t.pathstr + V.VHC_NAME

FROM    T_VEHICLE V

        INNER JOIN tree t

                ON t.id = V.VHC_ID_FATHER)

SELECT  SPACE(level) + data as data, id, level, pathstr

FROM    tree

ORDER BY pathstr, id

```

data	id	level	pathstr
-----	-----	-----	-----
ALL	1	0	
AIR	4	1	AIR
PLANE	11	2	AIRPLANE
ROCKET	10	2	AIRROCKET
EARTH	3	1	EARTH
CAR	7	2	EARTHCAR
TRUCK	9	2	EARTHTRUCK
TWO WHEELS	8	2	EARTHTWO WHEELS
BYCYCLE	13	3	EARTHTWO WHEELESBYCYCLE
MOTORCYCLE	12	3	EARTHTWO WHEELESMOTORCYCLE
SEA	2	1	SEA
BOAT	6	2	SEABOAT
SUBMARINE	5	2	SEASUBMARINE

To do this, we have use a new data type in SQL 2005 which is called VARCHAR(MAX), because we do not know the maximum of chars that will result in an operation a concatenation of a VARCHAR(16) in a recursive query that can be very deep. Notice that it is not a good idea to construct the path with VARCHAR because it can result in some boundaries effects such as concatenating words like 'LAND' and 'MARK' as level 2 of the tree which can be confused as 'LANDMARK' as level 1 in another branch of the same tree, so you must preserve blank space in the concatenated path to avoid such problems. This can be done by casting VHC_NAME as a CHAR SQL data type.

6 - Trees without recursion

But I must say that hierarchical data is not very interesting! Why? Because there are other ways to treat the data. Remember that I told you that a mathematician may say "you can avoid recursion by using a stack". Is this possible in our case?

Yes!

Just put the stack inside the table. How? By using two new columns: RIGHT_BOUND and LEFT_BOUND...

```
ALTER TABLE T_VEHCULE
ADD    RIGHT_BOUND INTEGER
```

```
ALTER TABLE T_VEHCULE
ADD    LEFT_BOUND INTEGER
```

Now, like a magician, I will populate this new column with some tricky numbers:

```
UPDATE T_VEHCULE SET LEFT_BOUND = 1 , RIGHT_BOUND = 26 WHERE VHC_ID = 1
UPDATE T_VEHCULE SET LEFT_BOUND = 2 , RIGHT_BOUND = 7  WHERE VHC_ID = 2
UPDATE T_VEHCULE SET LEFT_BOUND = 8 , RIGHT_BOUND = 19 WHERE VHC_ID = 3
UPDATE T_VEHCULE SET LEFT_BOUND = 20, RIGHT_BOUND = 25 WHERE VHC_ID = 4
UPDATE T_VEHCULE SET LEFT_BOUND = 3 , RIGHT_BOUND = 4  WHERE VHC_ID = 5
UPDATE T_VEHCULE SET LEFT_BOUND = 5 , RIGHT_BOUND = 6  WHERE VHC_ID = 6
UPDATE T_VEHCULE SET LEFT_BOUND = 9 , RIGHT_BOUND = 10 WHERE VHC_ID = 7
UPDATE T_VEHCULE SET LEFT_BOUND = 11, RIGHT_BOUND = 16 WHERE VHC_ID = 8
UPDATE T_VEHCULE SET LEFT_BOUND = 17, RIGHT_BOUND = 18 WHERE VHC_ID = 9
UPDATE T_VEHCULE SET LEFT_BOUND = 21, RIGHT_BOUND = 22 WHERE VHC_ID = 10
UPDATE T_VEHCULE SET LEFT_BOUND = 23, RIGHT_BOUND = 24 WHERE VHC_ID = 11
UPDATE T_VEHCULE SET LEFT_BOUND = 12, RIGHT_BOUND = 13 WHERE VHC_ID = 12
UPDATE T_VEHCULE SET LEFT_BOUND = 14, RIGHT_BOUND = 15 WHERE VHC_ID = 13
```

And here is the magic query, giving the same result as the complex hierarchical recursive query:

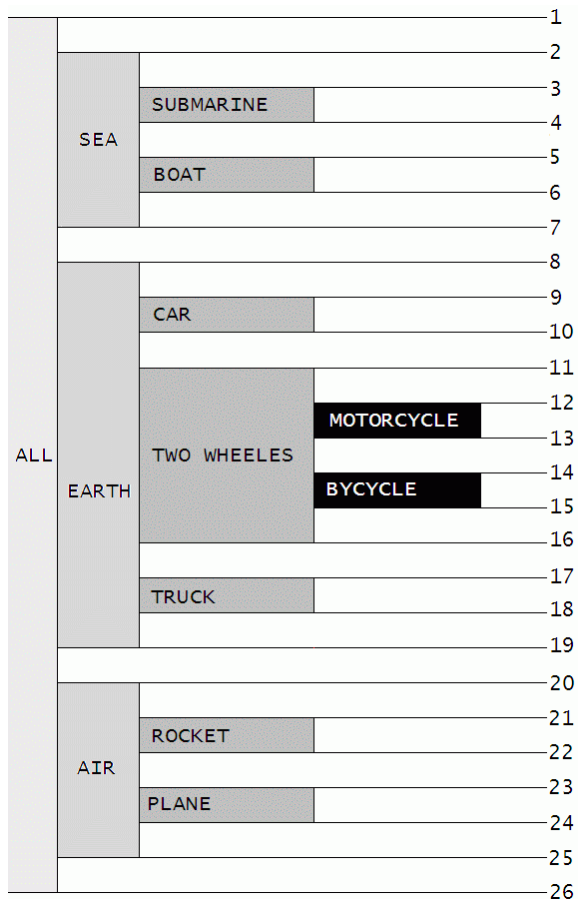
```
SELECT *
FROM    T_VEHCULE
WHERE   RIGHT_BOUND > 12
```

```
AND LEFT_BOUND < 13
```

VHC_ID	VHC_ID_FATHER	VHC_NAME	RIGHT_BOUND	LEFT_BOUND
1	NULL	ALL	26	1
3	1	EARTH	19	8
8	3	TWO WHEELS	16	11
12	8	MOTORCYCLE	13	12

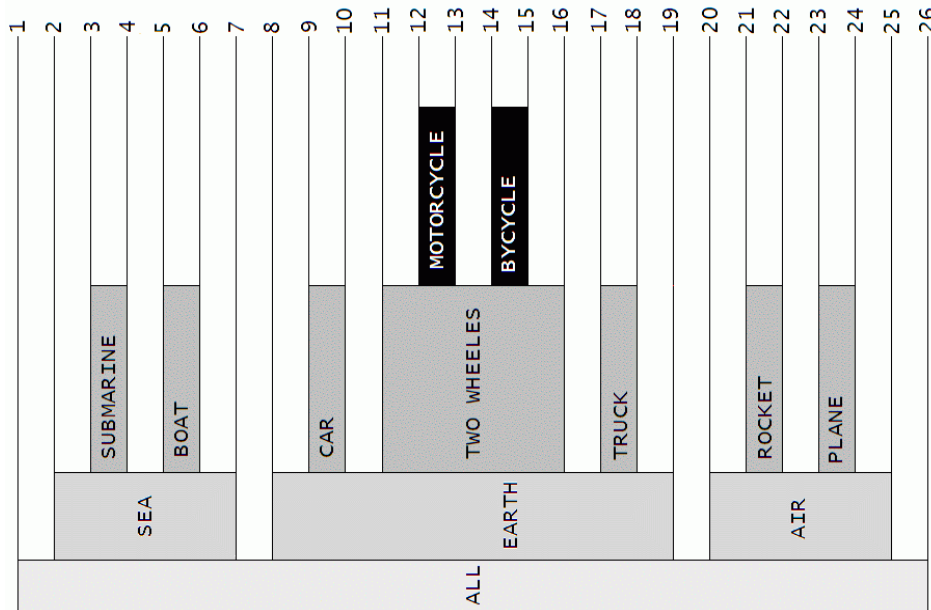
The question is: **what is the trick?**

In fact we realize a stack by numbering the data slices. I make a picture of it:



The only thing I do is to numerate continuously beginning by 1, from the right to the left bounds of all stacks made by each piece of data. Then to obtain the above query, I just take the bounds of the MOTORCYCLE, which are LEFT 12 and RIGHT 13, and place it in the WHERE clause asking for data that have a RIGHT BOUND over 12 and a LEFT BOUND under 13.

By the way, my graphic will be much more clear to understand if we rotate it:



Do you see the stacks? This representation of trees is well known in specialized database literature, especially writings by Joe Celko. You will find every thing you want in his famous book "Trees and Hierarchies" in "SQL for smarties", Morgan Kaufman ed. Another resource if you can read French is to go to my web site in which stored procs are written for MS SQL Server to do all jobs relative to this model:

<http://sqlpro.developpez.com/cours/arborescence/>

Last, can we reproduce the hierarchical indentation as seen in the last query ? Yes of course. This will be much easier by introducing a new column 'LEVEL' to indicate the level of the node. This can be very simple to calculate, because when inserting in the tree, the first node is the root, so the level is 0. Another point to insert in a tree had a level that can simply be calculated with the parent's data : if the point is to insert as a son, the level is the parent level + 1. To insert as a brother, the level is the same as the brother. Here are the ALTER and UPDATE statements that place the levels in the table for our purpose:

```
ALTER TABLE T_VEHICULE
```

```
ADD LEVEL INTEGER
```

```
UPDATE T_VEHICULE SET LEVEL = 0 WHERE VHC_ID = 1
```

```
UPDATE T_VEHICULE SET LEVEL = 1 WHERE VHC_ID = 2
```

```
UPDATE T_VEHICULE SET LEVEL = 1 WHERE VHC_ID = 3
```

```
UPDATE T_VEHICULE SET LEVEL = 1 WHERE VHC_ID = 4
```

```
UPDATE T_VEHICULE SET LEVEL = 2 WHERE VHC_ID = 5
```

```
UPDATE T_VEHICULE SET LEVEL = 2 WHERE VHC_ID = 6
```

```
UPDATE T_VEHICULE SET LEVEL = 2 WHERE VHC_ID = 7
```

```
UPDATE T_VEHICULE SET LEVEL = 2 WHERE VHC_ID = 8
```

```
UPDATE T_VEICULE SET LEVEL = 2 WHERE VHC_ID = 9
UPDATE T_VEICULE SET LEVEL = 2 WHERE VHC_ID = 10
UPDATE T_VEICULE SET LEVEL = 2 WHERE VHC_ID = 11
UPDATE T_VEICULE SET LEVEL = 3 WHERE VHC_ID = 12
UPDATE T_VEICULE SET LEVEL = 3 WHERE VHC_ID = 13
```

Now, the indentation query is:

```
SELECT SPACE(LEVEL) + VHC_NAME as data
FROM   T_VEICULE
ORDER  BY LEFT_BOUND
```

data

ALL

SEA

SUBMARINE

BOAT

EARTH

CAR

TWO WHEELS

MOTORCYCLE

BYCYCLE

TRUCK

AIR

ROCKET

PLANE

Much more simple, isn't it ?

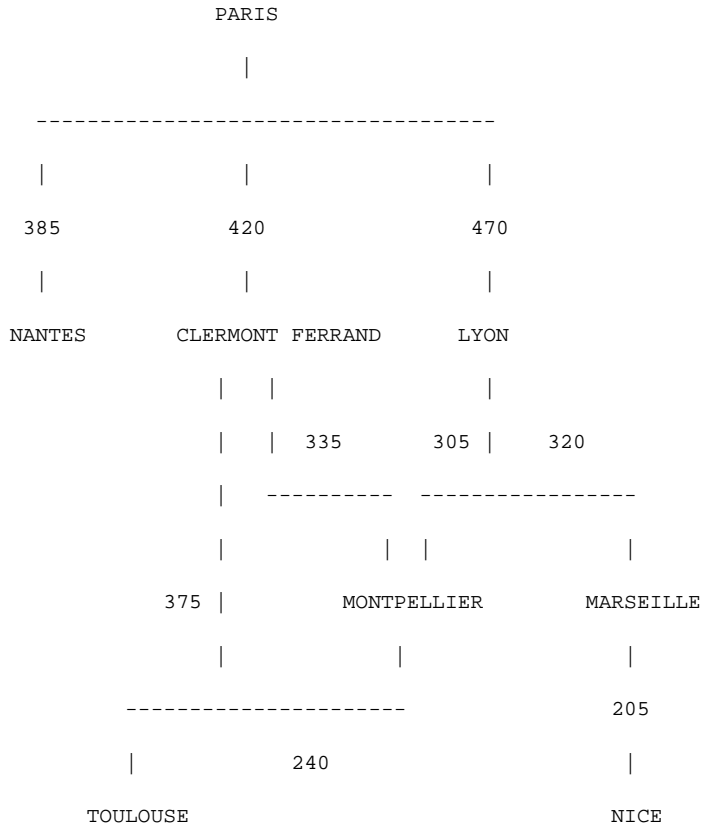
FIRST IMPRESSIONS ...

The only thing to say about these two ways of navigating through hierarchical data, is that the *interval* model is much more efficient and performs better than the one using the SQL: 1999 RECURSIVE query technique. In fact, RECURSIVE queries are not so interesting this way... But another way?... Yes!

7 - Second example: a complex network (a much more sexy query!)

Perhaps you never go to France. So you may be interested by the fact that in Paris, there are beautiful girls, and in

Toulouse a famous dish called *cassoulet*, and a small plane constructor call [Airbus](#). So the problem is to go by car from Paris to Toulouse using the speedway network. I just simplify for you (if you are lost and you do not know the pronunciation to ask people your way to Toulouse, it is simple. Just say "to loose"...):



```

-- if exists, drop the table we need for the demo

IF EXISTS (SELECT *
           FROM INFORMATION_SCHEMA.TABLES
           WHERE TABLE_SCHEMA = USER
           AND TABLE_NAME = 'T_JOURNEY')

DROP TABLE T_JOURNEY

-- create the table :

CREATE TABLE T_JOURNEY
(JNY_FROM_TOWN VARCHAR(32),
 JNY_TO_TOWN   VARCHAR(32),
 JNY_MILES     INTEGER)

-- populate :
```

```
INSERT INTO T_JOURNEY VALUES ('PARIS', 'NANTES', 385)

INSERT INTO T_JOURNEY VALUES ('PARIS', 'CLERMONT-FERRAND', 420)

INSERT INTO T_JOURNEY VALUES ('PARIS', 'LYON', 470)

INSERT INTO T_JOURNEY VALUES ('CLERMONT-FERRAND', 'MONTPELLIER', 335)

INSERT INTO T_JOURNEY VALUES ('CLERMONT-FERRAND', 'TOULOUSE', 375)

INSERT INTO T_JOURNEY VALUES ('LYON', 'MONTPELLIER', 305)

INSERT INTO T_JOURNEY VALUES ('LYON', 'MARSEILLE', 320)

INSERT INTO T_JOURNEY VALUES ('MONTPELLIER', 'TOULOUSE', 240)

INSERT INTO T_JOURNEY VALUES ('MARSEILLE', 'NICE', 205)
```

Now we will try a very simple query, giving all the journeys between towns :

```
WITH journey (TO_TOWN)

AS

    (SELECT DISTINCT JNY_FROM_TOWN

    FROM    T_JOURNEY

    UNION ALL

    SELECT JNY_TO_TOWN

    FROM    T_JOURNEY AS arrival

    INNER JOIN journey AS departure

        ON departure.TO_TOWN = arrival.JNY_FROM_TOWN)

SELECT *

FROM    journey

TO_TOWN

-----

CLERMONT-FERRAND

LYON

MARSEILLE

MONTPELLIER

PARIS

NANTES
```

CLERMONT-FERRAND

LYON

MONTPELLIER

MARSEILLE

NICE

TOULOUSE

MONTPELLIER

TOULOUSE

TOULOUSE

TOULOUSE

NICE

MONTPELLIER

MARSEILLE

NICE

TOULOUSE

MONTPELLIER

TOULOUSE

TOULOUSE

This query is not very interesting because we do not know from which town we came. We just know the towns where we can go, and the fact that we have probably different ways to go to same place. Let us see if we can have some more information... First, we want to start from Paris:

```
WITH journey (TO_TOWN)
```

```
AS
```

```
    (SELECT DISTINCT JNY_FROM_TOWN
```

```
      FROM    T_JOURNEY
```

```
      WHERE   JNY_FROM_TOWN = 'PARIS'
```

```
      UNION  ALL
```

```
      SELECT JNY_TO_TOWN
```

```
      FROM    T_JOURNEY AS arrival
```

```
      INNER JOIN journey AS departure
```

```
          ON departure.TO_TOWN = arrival.JNY_FROM_TOWN)
```

```
SELECT *
```



```
FROM    journey
```

```
TO_TOWN
```

```
-----
```

```
PARIS
```

```
NANTES
```

```
CLERMONT-FERRAND
```

```
LYON
```

```
MONTPELLIER
```

```
MARSEILLE
```

```
NICE
```

```
TOULOUSE
```

```
MONTPELLIER
```

```
TOULOUSE
```

```
TOULOUSE
```

We have probably three ways to go to Toulouse. Can we filter the destination? Sure!

```
WITH journey (TO_TOWN)
```

```
AS
```

```
    (SELECT DISTINCT JNY_FROM_TOWN
```

```
      FROM    T_JOURNEY
```

```
      WHERE   JNY_FROM_TOWN = 'PARIS'
```

```
      UNION  ALL
```

```
      SELECT JNY_TO_TOWN
```

```
      FROM    T_JOURNEY AS arrival
```

```
            INNER JOIN journey AS departure
```

```
              ON departure.TO_TOWN = arrival.JNY_FROM_TOWN)
```

```
SELECT *
```

```
FROM    journey
```

```
WHERE   TO_TOWN = 'TOULOUSE'
```

```
TO_TOWN
```

TOULOUSE

TOULOUSE

TOULOUSE

We can refine this query by calculating the number of steps involved in the different ways:

```
WITH journey (TO_TOWN, STEPS)
```

```
AS
```

```
(SELECT DISTINCT JNY_FROM_TOWN, 0
```

```
FROM T_JOURNEY
```

```
WHERE JNY_FROM_TOWN = 'PARIS'
```

```
UNION ALL
```

```
SELECT JNY_TO_TOWN, departure.STEPS + 1
```

```
FROM T_JOURNEY AS arrival
```

```
INNER JOIN journey AS departure
```

```
ON departure.TO_TOWN = arrival.JNY_FROM_TOWN)
```

```
SELECT *
```

```
FROM journey
```

```
WHERE TO_TOWN = 'TOULOUSE'
```

TO_TOWN	STEPS
---------	-------

TOULOUSE	3
----------	---

TOULOUSE	2
----------	---

TOULOUSE	3
----------	---

The cherry on the cake will be to know the distances of the different ways :

```
WITH journey (TO_TOWN, STEPS, DISTANCE)
```

```
AS
```

```
(SELECT DISTINCT JNY_FROM_TOWN, 0, 0
```

```
FROM T_JOURNEY
```

```
WHERE JNY_FROM_TOWN = 'PARIS'
```

```
UNION ALL
```

```
SELECT JNY_TO_TOWN, departure.STEPS + 1,
       departure.DISTANCE + arrival.JNY_MILES
FROM   T_JOURNEY AS arrival
       INNER JOIN journey AS departure
           ON departure.TO_TOWN = arrival.JNY_FROM_TOWN)

SELECT *

FROM   journey

WHERE  TO_TOWN = 'TOULOUSE'
```

TO_TOWN	STEPS	DISTANCE
-----	-----	-----
TOULOUSE	3	1015
TOULOUSE	2	795
TOULOUSE	3	995

The girl *in* the cake will be to know the different towns we visit by those different ways:

```
WITH journey (TO_TOWN, STEPS, DISTANCE, WAY)

AS

(SELECT DISTINCT JNY_FROM_TOWN, 0, 0, CAST('PARIS' AS VARCHAR(MAX))

FROM   T_JOURNEY

WHERE  JNY_FROM_TOWN = 'PARIS'

UNION ALL

SELECT JNY_TO_TOWN, departure.STEPS + 1,
       departure.DISTANCE + arrival.JNY_MILES,
       departure.WAY + ', ' + arrival.JNY_TO_TOWN

FROM   T_JOURNEY AS arrival
       INNER JOIN journey AS departure
           ON departure.TO_TOWN = arrival.JNY_FROM_TOWN)

SELECT *

FROM   journey

WHERE  TO_TOWN = 'TOULOUSE'
```

TO_TOWN	STEPS	DISTANCE	WAY

TOULOUSE	3	1015	PARIS, LYON, MONTPELLIER, TOULOUSE
TOULOUSE	2	795	PARIS, CLERMONT-FERRAND, TOULOUSE
TOULOUSE	3	995	PARIS, CLERMONT-FERRAND, MONTPELLIER, TOULOUSE

And now, ladies and gentleman, the RECURSIVE QUERY is proud to present to you how to solve a very complex problem, called the *travelling salesman problem*. (One of the operational research problems on which Edsger Wybe Dijkstra found the first efficient algorithm and received the Turing Award in 1972):

```
WITH journey (TO_TOWN, STEPS, DISTANCE, WAY)
AS
    (SELECT DISTINCT JNY_FROM_TOWN, 0, 0, CAST('PARIS' AS VARCHAR(MAX))
     FROM   T_JOURNEY
     WHERE  JNY_FROM_TOWN = 'PARIS'
     UNION ALL
     SELECT JNY_TO_TOWN, departure.STEPS + 1,
            departure.DISTANCE + arrival.JNY_MILES,
            departure.WAY + ', ' + arrival.JNY_TO_TOWN
     FROM   T_JOURNEY AS arrival
            INNER JOIN journey AS departure
                  ON departure.TO_TOWN = arrival.JNY_FROM_TOWN)
SELECT TOP 1 *
FROM   journey
WHERE  TO_TOWN = 'TOULOUSE'
ORDER BY DISTANCE
```

TO_TOWN	STEPS	DISTANCE	WAY

TOULOUSE	2	795	PARIS, CLERMONT-FERRAND, TOULOUSE

By the way, TOP n is a non standard SQL... Dislike it... Enjoy CTE!

```
WITH

    journey (TO_TOWN, STEPS, DISTANCE, WAY)

AS

    (SELECT DISTINCT JNY_FROM_TOWN, 0, 0, CAST('PARIS' AS VARCHAR(MAX))

     FROM   T_JOURNEY

     WHERE  JNY_FROM_TOWN = 'PARIS'

     UNION ALL

     SELECT JNY_TO_TOWN, departure.STEPS + 1,

            departure.DISTANCE + arrival.JNY_MILES,

            departure.WAY + ', ' + arrival.JNY_TO_TOWN

     FROM   T_JOURNEY AS arrival

            INNER JOIN journey AS departure

                  ON departure.TO_TOWN = arrival.JNY_FROM_TOWN),
```

```
short (DISTANCE)

AS
(SELECT MIN(DISTANCE)
 FROM   journey
 WHERE  TO_TOWN = 'TOULOUSE')
SELECT *
FROM   journey j
      INNER JOIN short s
      ON j.DISTANCE = s.DISTANCE

WHERE  TO_TOWN = 'TOULOUSE'
```

8 - What more can we do?

In fact, one thing that is limiting the process in our network of speedways, is that we have made routes with a single sense. I mean, we can go from Paris to Lyon, but we are not allowed to go from Lyon to Paris. For that, we need to add the reverse ways in the table, like:

JNY_FROM_TOWN	JNY_TO_TOWN	JNY_MILES
-----	-----	-----
LYON	PARIS	470

This can be done, by a very simple query :

```
INSERT INTO T_JOURNEY

SELECT JNY_TO_TOWN, JNY_FROM_TOWN, JNY_MILES

FROM T_JOURNEY
```

The only problem is that, previous queries won't work properly:

```
WITH journey (TO_TOWN)

AS

(SELECT DISTINCT JNY_FROM_TOWN

 FROM   T_JOURNEY

 WHERE  JNY_FROM_TOWN = 'PARIS'

 UNION ALL

 SELECT JNY_TO_TOWN

 FROM   T_JOURNEY AS arrival

      INNER JOIN journey AS departure

      ON departure.TO_TOWN = arrival.JNY_FROM_TOWN)

SELECT *

FROM   journey
```

```
TO_TOWN
```

```
-----
```

```
PARIS
```

```
NANTES
```

```
CLERMONT-FERRAND
```

```
LYON
```

```
...
```

```
LYON
```

```
MONTPELLIER
```

```
MARSEILLE
```

```
PARIS
```

```
Msg 530, Level 16, State 1, Line 1
```

```
The statement terminated. The maximum recursion 100 has been exhausted before statement completion.
```

What happened? Simply, you are trying all ways including cycling ways like Paris, Lyon, Paris, Lyon, Paris... ad infinitum... Is there a way to avoid cycling routes? Maybe. In one of our previous queries, we have a column that gives the complete list of stepped towns. Why don't we use it to avoid cycling? The condition will be: do not pass through a town that is already in the WAY. This can be written as:

```
WITH journey (TO_TOWN, STEPS, DISTANCE, WAY)
```

```
AS
```

```
(SELECT DISTINCT JNY_FROM_TOWN, 0, 0, CAST('PARIS' AS VARCHAR(MAX))
```

```
FROM T_JOURNEY
```

```
WHERE JNY_FROM_TOWN = 'PARIS'
```

```
UNION ALL
```

```
SELECT JNY_TO_TOWN, departure.STEPS + 1,
```

```
departure.DISTANCE + arrival.JNY_MILES,
```

```
departure.WAY + ', ' + arrival.JNY_TO_TOWN
```

```
FROM T_JOURNEY AS arrival
```

```
INNER JOIN journey AS departure
```

```
ON departure.TO_TOWN = arrival.JNY_FROM_TOWN
```

```
WHERE departure.WAY NOT LIKE '%' + arrival.JNY_TO_TOWN + '%')
```

```
SELECT *
```

```
FROM journey
```

```
WHERE TO_TOWN = 'TOULOUSE'
```

TO_TOWN	STEPS	DISTANCE	WAY
TOULOUSE	3	1015	PARIS, LYON, MONTPELLIER, TOULOUSE
TOULOUSE	4	1485	PARIS, LYON, MONTPELLIER, CLERMONT-FERRAND, TOULOUSE
TOULOUSE	2	795	PARIS, CLERMONT-FERRAND, TOULOUSE
TOULOUSE	3	995	PARIS, CLERMONT-FERRAND, MONTPELLIER, TOULOUSE

As you see, a new route occurs. The worst in distance, but perhaps the most beautiful!

CONCLUSIONS

A CTE can simplify the expression of complex queries. RECURSIVE queries must be employed where recursivity is needed. If you make a bad query with MS SQL Server, don't be afraid, the cycles of recursions are limited to 100. You can overcome this limit by fixing OPTION (MAXRECURSION n), with n as the value you want. The OPTION clause must be the last one in a CTE expression. But remember one thing: MS SQL Server 2005 is actually a beta version!

Last but not least, ISO SQL:1999 had some more syntax options that can allow you to navigate in the data DEPTH FIRST or BREADTH FIRST, and also to all the data contained in the steps (in an ARRAY of ROW which must be of "sufficient" in dimension to cover all cases!).

Serializing Procedure Calls Without Unnecessary Blocking

By Robert Cary

Introduction

There are several situations where you might want to marshal calls to a particular PROC. For me, the most common situation is where we have several instances of an application, running in parallel, that need to work with distinct data sets supplied by a stored procedure. These solutions are also useful in any situation where you occasionally encounter race-conditions but do not want to block other processes by acquiring restrictive locks on the objects in use or using a SERIALIZABLE transaction isolation level. Below are various approaches to this problem and reasons why using sp_getapplock is usually the most preferable.

One approach that I've seen used in the past is to create a surrogate locking table that is only referenced by the PROC you wish to marshal. By acquiring an exclusive table lock on the surrogate table within in the proc, you are ensuring that only one instance of that proc can run at a time. Subsequent calls are effectively queued and thereby serializing execution of the PROC.

```
/****** Create Table *****/  
  
CREATE TABLE dbo.SurrogateLockTable  
(
```

```
        ID INT PRIMARY KEY CLUSTERED
    )
GO

INSERT INTO dbo.SurrogateLockTable(ID)
VALUES(1)
GO

/***** Proc Code *****/
CREATE PROC dbo.LockingTest
AS

BEGIN TRAN

    SELECT ID

    FROM SurrogateLockTable WITH(HOLDLOCK,TABLOCKX)

    /*your code goes here*/

COMMIT
```

This technique can be extended to lock an individual key, allowing either greater granularity when marshalling calls or to allowing multiple PROCs to utilize the same lock table and thus avoiding a lock table for every PROC you wish you marshal.

```
/***** Create Table *****/
CREATE TABLE dbo.SurrogateLockKeyTable
(
    KeyVal VARCHAR(200) PRIMARY KEY CLUSTERED -- VARCHAR for flexibility, but could be anything
                                                -- depending on what your keys will be
)
GO

/***** Proc Code *****/
CREATE PROC dbo.KeyLockingTest
AS
```



```
/* CREATE Record in table if it doesn't already exist */

IF NOT EXISTS(SELECT 1

FROM SurogateLockKeyTable WITH(NOLOCK) --NOLOCK hint allows you to read the table

                                -- without being blocked

WHERE KeyVal = @KeyVal)

BEGIN

    INSERT INTO SurogateLockKeyTable(KeyVal)

    VALUES(@KeyVal) -- This can cause a race condition, if two identical calls

                    -- are made simultaniuosly.

END

BEGIN TRAN

    SELECT @rand = Id

    FROM [SurogateLockKeyTable] WITH (HOLDLOCK,ROWLOCK,XLOCK)

    WHERE KeyVal = @KeyVal

    /*Code goes here*/

COMMIT
```

This allows for higher concurrency by only blocking calls that would affect the same key. Naturally, key could be a proc name, a table key, a table etc.

An alternative and more favorable approach would be to utilize `sp_getapplock` and `sp_releaseapplock`. `SP_getapplock` is a wrapper for the extened procedure `XP_USERLOCK`. It allows you to use SQL SERVERs locking mechanism to manage concurrency outside the scope of tables and rows. It can be used you to marshal PROC calls in the same way the above solutions with some additional features.

- By using `sp_getapplock`, you do not need to create and manage a surrogate table. `Sp_getapplock` adds locks directly to the server memory. Also, if you look at the second example, there is a flaw where a race-condition could still occur. `Sp_getapplock` removes this danger.
- Second, you can specify a lock timeout without needing to change session settings. In cases where you only want one call for a particular key to run, a quick timeout would ensure the proc doesnt hold up execution of the application for very long.
- Third, `sp_getapplock` returns a status which can be useful in determining if the code should run at all. Again, in cases where you only want one call for a particular key, a return code of 1 would tell you that the lock was granted successfully after waiting for other incompatible locks to be released, thus you can exit without running any more code (like an existence check, for example).

The synax is as follows:

```
sp_getapplock [ @Resource = ] 'resource_name',  
  
    [ @LockMode = ] 'lock_mode'  
  
    [ , [ @LockOwner = ] 'lock_owner' ]  
  
    [ , [ @LockTimeout = ] 'value' ]
```

An example using sp_getapplock that is equivalent to the second example:

```
/****** Proc Code *****/  
  
CREATE PROC dbo.GetAppLockTest  
  
AS  
  
BEGIN TRAN  
  
    EXEC sp_getapplock @Resource = @key, @Lockmode = 'Exclusive'  
  
    /*Code goes here*/  
  
    EXEC sp_releaseapplock @Resource = @key  
  
COMMIT
```

Conclusions

All the approaches described will allow you to marshal calls to a proc in situations where you don't want to simply acquire exclusive locks on the tables you are using. Overall, I believe using sp_getapplock is cleaner, more elegant, and more flexible for most situations.

I know it goes without saying, but when using this or any of the other of the locking examples, always wrap them in a transaction with XACT_ABORT on, or checks in code to ensure a ROLLBACK where required.

The Truth Table

By Yousef Ekhtiari

Background

As we all know, we use a logical operator in a WHERE clause. It means each of us are consciously or unconsciously familiar with propositional logic. In propositional logic, we only have two values for each variable: True or False (1 or 0), therefore, any logic statement can be analyzed using a table which lists all possible values of the variable: a Truth Table. Since each variable can take only two values, a statement with "n" variables requires a table with 2^n rows which is difficult to construct when the number of variables are more than 4.

Truth Table

In this article I will show you how SQL can help to construct the Truth Table. In order to construct the Truth Table I wrote a stored procedure:

```
create PROC Usp_BuildTruthTable ( @variables xml,
                                @expressions xml)

AS

DECLARE @docHandle int,
        @SELECT varchar(8000),
        @FROM varchar(8000),
        @SQL nvarchar(4000)

EXEC sp_xml_preparedocument @docHandle OUTPUT, @variables

SELECT @FROM=isnull( @FROM +char(13)
        +' cross join (select cast(0 as bit)union all
        select cast(1 as bit) ) as '+value+'('+value+')'+char(13)
        , ' (select cast(0 as bit) union all select  cast( 1 as bit) ) as '
        +value+'('+value+')')

FROM OPENXML(@docHandle, N'/Variables/var')
with (value char(1) )

EXEC sp_xml_removedocument @docHandle

--constructing the Select Clause

EXEC sp_xml_preparedocument @docHandle OUTPUT, @expressions

SELECT      @SELECT=isnull(@SELECT+' ',['+col+']='+col, '['+col+']='+col)

FROM OPENXML(@docHandle, N'/expressions/exp')

WITH (col VARCHAR(8000) '@val')

SET @SQL='select *,'+@SELECT+ ' from '+char(13) +@FROM

EXEC( @SQL)
```

```
EXEC sp_xml_removedocument @docHandle
```

As you can see it accepts two parameters which are declared as XML data type that is new to SQL Server 2005. If you are not familiar with the XML procedure which I used in the Usp_BuildTruthTable, you can refer to BOL documentations.

The main core of stored procedure is the statement "(select cast(0 as bit)union all select cast(1 as bit))", which assigns all the possible values to the variables and the CROSS JOIN that is used to produce all possible combinations of variables. The @SELECT will evaluate the expression, and the logic behind this is simple: SQL Server has a bitwise operator, so it can evaluate bitwise expressions. This is the reason that I converted zero and one to the bit data type. There are 4 bitwise operators in SQL SERVER:

Operator	Meaning
~	NOT
&	AND
	INCLUSIVE OR
^	EXCLUSIVE OR

Now I want to show you how to use these bitwise operators in logical connectives:

Logical operator	Expression	Bitwise
Not	P	~P
And	P AND Q	P & Q

Inclusive or	$P \text{ OR } Q$	$P \mid Q$
Exclusive or	$P \text{ XOR } Q$	$P \wedge Q$
Implies	$P \text{ IMP } Q$	$\sim P \mid Q$
Equivalence	$P \text{ EQU } Q == (P \text{ IMP } Q) \& (Q \text{ IMP } P)$	$(\sim P \mid Q) \& (\sim Q \mid P)$

You have to pass variables and expressions with the following format:

<pre> <Variables> <var value="VAR1" /> . . . <var value=" VARn" /> </Variables> </pre>	<pre> <expressions> <exp val="EXP1" /> . . . <exp val="EXPn" /> </expressions> </pre>
--	---

Note: you have to use "& amp;" (remove th space between the & and amp) instead of & or else you get the following error:

```

.Net SqlClient Data Provider: Msg 9421, Level 16, State 1, Procedure Usp_BuildTruthTable,
Line 0
XML parsing: line 3, character 15, illegal name character

```

To test the stored procedure, run the following snippet:

```
exec usp_BuildTruthTable '<Variables>

<var value="a" />

<var value="b" />

</Variables>' ,

'<expressions>

<exp val="~a" />

  <exp val="a & amp; b" />

<exp val="(a | b)" />

<exp val="(a ^ b)" />

<exp val="(~a | b)" />

<exp val="(~a | b)& amp; (~b | a)" />

</expressions>  '
```

Note: Remove the space between the & and amp in the code.

Here is the result:

a	b	~a	a & b	(a b)	(a ^ b)	(~a b)	(~a b)& (~b a)
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	0	0
1	1	0	1	1	0	1	1

Let's check a complicated expression:

```
((P imp Q) and (Q imp R)) imp (P imp R) == ~ ((~P | Q) & (~Q | R)) | (~P | R)
```

We have three variables and an expression, so we invoke the stored procedure as follows:

```
exec usp_BuildTruthTable

'<Variables>

<var value="P" />

<var value="Q" />

<var value="R" />
```

```
</Variables>' ,  
'<expressions>  
<exp val="~((~p |Q) & amp; (~Q |R)) | (~P|R)" />  
</expressions>    '
```

And here is the result:

P	Q	R	$\sim((\sim P Q) \& (\sim Q R)) (\sim P R)$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	0	1
0	0	1	1
0	1	1	1
1	0	1	1
1	1	1	1

Introduction to Bitmasking in SQL Server 2005

By Lee Everest

Bitmasking and bitwise operations are low level programming techniques used for turning bits off and on - the manipulation of bits provides a way to make representations in a computer. The goal of bitmasking in SQL Server 2005 is to combine two or more attributes, normally stored into a single column in a table, into one hexadecimal value to make representations as well. Achieving this goal has many possibilities, as we will discover later, and introduces many computer science concepts into the relational database. Part 1 of this series is an introduction to counting and bitmasking in SQL Server. In Part 2, we will discover its practical application in SQL Server. Finally, in Part 3, we will explore programmability using bitmasking techniques.

Why Bitmask?

Suppose, for instance, that you have been assigned to capture 50 *Request.Browser* properties (Figure 1) from your website and maybe 50 keys in the *Request.ServerVariables* collection off of the web to store in SQL Server. Are you going to create 100 columns in a table to hold these values? Sure, you can and that will work very well, but bitmasking several values to one value is a cleaner, neater, and more efficient approach in my opinion.

Table 1: Bit terminology

Numbers can be represented via a series of bits in combinations that are either on or off. Your computer represents things as numbers as well, such as memory locations, printer ports, devices, etc. When you think about it in this context, we can certainly represent things with numbers of our choosing if we are inclined to in the relational database - permissions, cities, states, hair color, gender can be assigned a numerical value. And by doing such, we are representing entities, objects, or things just as your computer does. Let us now look at Base 10 and Base 2 notation. This will allow us to become acquainted with numbers and their representations. We'll first pick an easy number, and then a larger one as our working examples. If you remember from mathematics, our numbering system is centered on the Decimal System, or Base 10. Let's look at some Base 10 representations of our numbering system. Understanding the following is critical to bitmasking in SQL Server 2005.

Base 10

Check out Table 2 for a refresher on simple Base 10 counting. We will build off of this concept later. Remember that we can count numbers in a variety of ways. Follow each line in the graphical representation to get (re)acquainted with counting numbers in this manner. At this point, a calculator might be handy to verify some of what I am deriving.

	A	B	C
1	Base 10		
2	Example	Representation	
3	5	= 1 * 5	= 5
4		= 10 ^ 0 * 5	= 5
5		= SELECT POWER (10,0) * 5	= 5
6		(remember that anything to the zero power is 1)	
7			
8	4458	= 4000 + 400 + 50 + 8	= 4458
9		= (1000 * 4) + (100 * 4) + (10 * 5) + (1 * 8)	= 4458
10		= 10*10*10*4 + 10*10*4 + 10*5 + 1*8	= 4458
11		= 10^3*4 + 10^2*4 + 10^1*5 + 10^0*8	= 4458

Table 2: Counting numbers with Base 10

We certainly could have expanded line 11 in our table even further, but I think you get the point. The number 10 is the base value of our final calculation here, and we can express final results based on 10 raised to a power times another value.

Base 2

As Base 10 is synonymous with the Decimal system, so is Base 2 to the *Binary System*.

Because bits are used to represent numbers in a computer, we find it easier to translate our numbering to bit notation (we previously pointed out that the core of computing is based on only two values). Rather than using Base 10, we will use a Base 2, as it provides to us a much easier way to represent our values.

A very important concept, Base 2 allows us to translate from hexadecimal to decimal and back very easily. Mapping values using base two is much easier than trying to keep up with counting all of the zeroes and ones when using Base 10.

	A	B	C	D	E
1	Base 2				
2	Example		Value	Status	
3	Four Bits	0000	0	All off	
4	...	0 0 0 0	0	All off	
5					
6	Eight Bits	0000 0000	0	All off	
7		1 1 1 1 1 1 1 1	255	All on	
8					
9					
10		128 64 32 16 8 4 2 1			
11					

Table 3: Base2 mapping

The above table first shows us that one nibble and one byte, all zeros, has a value of zero. A byte, all 1's (all bits turned on) has a value of 255. How did we come up with this?

1 1 1 1 1 1 1 1

$$2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0 = 255$$

+ + + + +

$$128 64 32 16 8 4 2 1 = 255$$

+ + + + +

$$240 15 = 255$$

Let's break this down piece by piece. Don't worry all of this will come together for you soon. Each bit position represents an ordinal position within a byte. Starting from right to left (we could have started from left to right it's personal preference), position 0 is represented by two to the zero power, which equals one. Moving to the left one position, we find two to the first power, or 2^1 , which gives us the value of two. (Notice the increasing value of the exponent by one, starting at zero, each iteration). Again using base two, our Base 2 value next is raised to the second power, and then two raised to the third power, etc. Since all the bits are on in the first nibble of this byte, what is the value of the sum of the first four numbers? Fifteen, you are correct!

Now let's examine our value of 15 further, and place it in our windows scientific calculator. Make sure that you enter 15 and that it is reading in Dec, short for decimal notation. Now, change it to Hex, short for Hexadecimal notation.

What the heck? Our value is F. Where did this come from? Remember that our visual representation is easily changed to Hex from Base 2.



Display 1: Our value 15 as Hexadecimal

Review Table 4. This mapping allows us to successfully convert our numbering scheme from base two to Hex notation. This final representation of numbers is what we will actually use in our masking far easier than trying to convert base 10 or base 2 to something visual that we can work with.

Decimal	Bin	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Table 4: A conversion table for bits

Study this concept for a moment, and try to come up with some values of your own. For practice, jot down a few numbers between 0 and 20 and try to figure out their binary value representation.

Let's look at another example, this one is more difficult. Let's attempt to map the hex value of our previous, 4458. Before we can do this, we have to calculate its value in Base2. We must first figure out how to get there. Here is what I come up with:

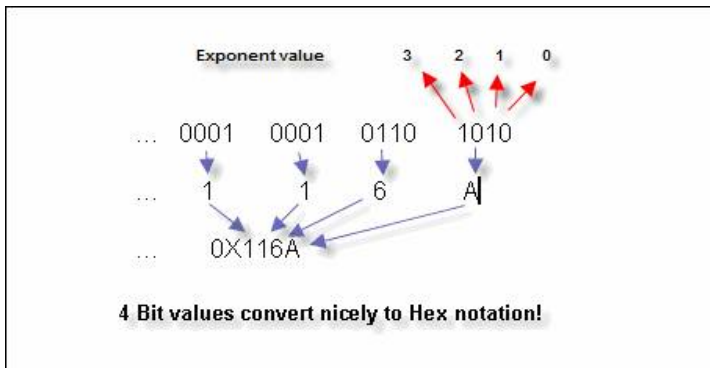
$$4458 = 2^{12} + 2^8 + 2^6 + 2^5 + 2^3 + 2^1$$

$$= 4096 + 256 + 64 + 32 + 8 + 2$$

If we now map each nibble, we end up with the following:

...	0001	0001	0110	1010
...	1	1	6	A
...	0X116A			

Notice how each set of 4 bits translates nicely into our first beginning value of Base2 exponential notation. Let's look at it again:



We treat each nibble as if they were stand alone the first nibble is the sum of $2 + 8$, since the second and forth bits are on, giving us a value of 10 (or the value A!). For the second nibble, the second and third bits are on, summing to give a value of 6 ($2 + 4$). Yes you got it!

Again, review this concept carefully before moving on. Pick a few numbers larger than, say, 100, and come up with 1) its binary representation and 2) its hex representation. Understanding this exercise, converting from decimal to binary to hexadecimal, is critical to successful bitmasking. Let's now begin with a fictitious problem and solve it via bitmasking.

Our Problem

Given our refresher, we are ready for a challenge. We have been tasked to bitmask 3 values that are generated each time a visitor comes to our website. We are to record 1) the operating system, 2) the browser used, and 3) whether or not cookies are turned on in the client environment. We wish to combine these three values into 1 bitmasked value and store this in the database.

Preparing to code our mask

Before we begin coding T-SQL, I first find it easy to mask values into a single value by creating a key, or legend, that will guide me to the correct bit placement of my values. Below is a useful way to prepare a legend as a map to 1)

Define the number of bytes needed, 2) Keep track/calculate bit values, and 3) Position of the bytes where you wish to store your values. Simply add one of these as a comment in your code so that you have a reference to go back whenever you like to remind yourself of the mask positions for your values. Examine the key in Figure 1 and study the explanation of the key carefully.

```
/*
Key
Reason & # of Bits      Code values to store
-----
Allow 2 bits for OS      Up to 3
Allow 3 bits for Browser Up to 7
Allow 1 bits for Cookies Up to 1

Assignment      Bit Position
-----
Byte   :        1
Bit Position:    8765 4321
Mask   :          CB B000
*/
```

Figure 2: Creating a key to map bitmask values

Explanation of the key

In our example, we will use only 1 byte to represent three columns. (If you were masking several columns, you would need more bytes for storage). First, I assign the placeholders for the bit values: 2 bits for the operating system, 3 bits for the browser type, and 1 bit to notify if cookies are turned on or off. The second portion of the key outlines the assignment for your mask. We are using 1 byte which has 8 bit positions. Each position will be assigned bit values. For O (operating system), I previously defined in the key that we will allow three values, so I'll need 2 bytes. For Browser B storage, I'll need 3 bits (Remember why 3 bits allows for 7 storage values? If you forgot, go back and review counting numbers. You'll get it!), and for Cookie C values just 1 bit. Note that I map them in the Mask line above, each of the 6 bits that I'm using assigned to a corresponding value for my mask. This is very important, since you will actually be placing numerical representations of your data within these bits. It does not matter that I do not use all 8 bits that I have allocated in our example we can leave the remaining bits zero without disrupting our storage or retrieval process.

Coding the Mask

Below is the code that will mask our values. Review this script, and then look at the explanation below.

```
DECLARE @mask varbinary(1)
, @Os int
, @Browser int
, @Cookie int
, @BitshiftBrowser int
, @BitshiftCookie int
, @OSName varchar (10)
, @BrowserName varchar (10)
, @CookieStatus varchar (3)

--*** Code the mask
SET @OSName = 'Mac'
SET @BrowserName = 'Firefox'
SET @CookieStatus = 'Off'

SELECT @BitshiftBrowser=4 ,@BitshiftCookie=32

SELECT
  @Os=
    CASE WHEN @OSName = 'Windows' then 1
          WHEN @OSName = 'Linux' then 2
          WHEN @OSName = 'Mac' then 3
          ELSE 0
    END
  , @Browser=
    CASE WHEN @BrowserName = 'IE' then 1
          WHEN @BrowserName = 'Firefox' then 2
          ELSE 0
    END
  , @cookie=
    CASE WHEN @CookieStatus='ON' THEN 1
          ELSE 0
    END

SELECT @mask = @Os | @Browser*@BitshiftBrowser | @Cookie*@BitshiftCookie
SELECT @mask as BitMaskedValue
```

1. Binary data type

2. Assign mask value

3. Assign shift values

4. Assign code to values

5. Mask values

Figure 3: Code to mask our data into one value

Let's briefly review the above steps for coding the mask.

Steps to mask values

Step 1) Binary data type at this point, you choose the binary or varbinary data type that you will use. The more values that you have, the larger the data type necessary to store values. I use a varbinary (1) data type for no reason other than I normally use varbinary instead of binary (length 1 doesn't vary, obviously).

Step 2) Assign Mask Values values that we wish to mask. They could come from a web page, a rich client, web service, or whatever. For this initial demo, we're just running a script, so hard-coded values are adequate.

Step 3) Assign the shift values these variables store the multiplier to allow you to shift, within the byte or bytes, the beginning point for you storage attributes. For example, the first variable, @BitshiftBrowser=4, will allow us to skip the first two bits (our designated holding spot for the operating system) in order to get to the location where we are going to store our browser settings.

Step 4) Assign code to values in order to store our values, remember back to our previous paragraph on object representations in the computer. Here we simply assign a number to our value via the [CASE](#) statement.

Step 5) Mask the values use the bitwise OR operator to place the integer value that represents our textual value inside of the binary data type at the outlined location. (For further reading, see http://en.wikipedia.org/wiki/Bitwise_OR#OR).

Step 6) Masked value our end result. We have successfully collapsed 3 attributes into 1 value (Figure 3); we could have put hundreds of them into one value. A powerful concept? I agree!

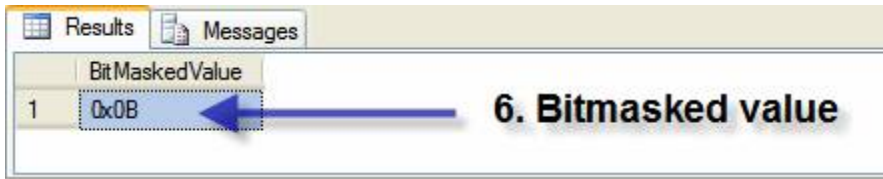


Figure 4: Bitmasked value final result

Decoding the Mask

Just as we code the mask, similarly we must decode the hexadecimal value (Figure 5). Let's look at the steps to accomplish this.

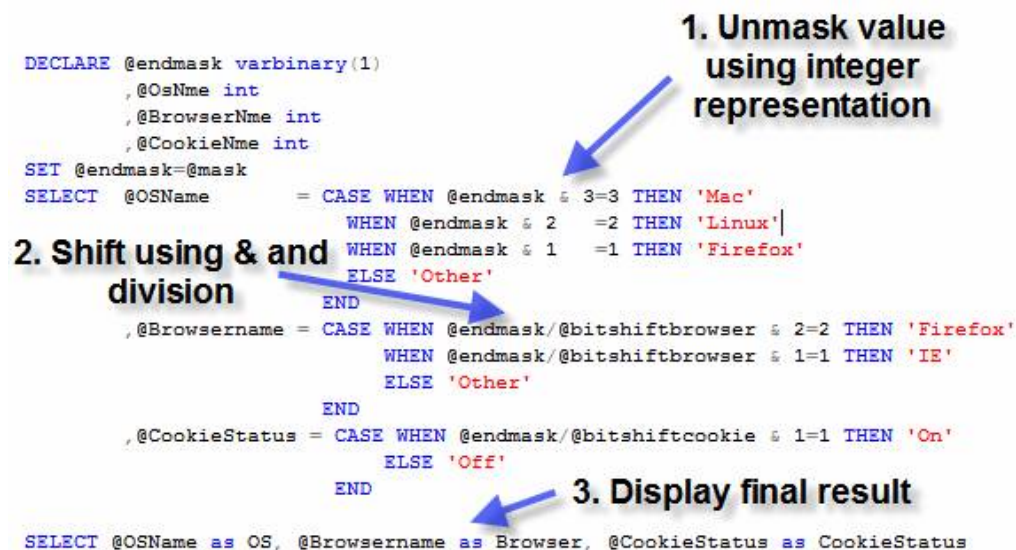


Figure 5: Decode the masked value

Steps to decode mask values

Step 1) Unmask value using integer representation This is a simple decode. The values that you previously assigned to your textual representation are reversed. For the first bit, you perform a bitwise AND (http://en.wikipedia.org/wiki/Bitwise_AND#AND) to recapture the values.

Step 2) Shift using & and division As we shifted to the right to place the values into the mask, we must divide by the same bitshift variable to decode the second and subsequent values out of the mask. The mask value/shift integer *anded* with the storage value assigned in the code mask steps is found again by the **CASE** statement.

Step 3) Display final result



The screenshot shows a SQL Server Results window with two tabs: 'Results' and 'Messages'. The 'Results' tab is active, displaying a table with three columns: 'OS', 'Browser', and 'CookieStatus'. There is one row of data with the values 'Mac', 'Firefox', and 'Off' respectively. The row is highlighted with a blue selection bar.

	OS	Browser	CookieStatus
1	Mac	Firefox	Off

Figure 6: Final decoded value

Conclusion

Bitmasking in SQL Server is fun, powerful, and a neat way to store data in a relational database. While it shouldn't be used to overpower or overcomplicate solving a particular task, as it is with every tool in our SQL Server toolkit arsenal, there is a time and a place for everything. Keep this technique in mind next time you have to store many attributes into a table. In Part 2, we will look at how you might build objects in the database that will allow implementation of this concept. We'll also look at some queries and indexing the bitmasked value, and look at performance implications for a bitmasked value in SQL Server.

About the Author

Lee Everest is a Sr. Consultant for Software Architects, and a part-time SQL instructor at North Lake College in Irving, Texas. He can be reached at tsql-northlake@dcccd.edu.

Self Eliminated Parameters

By Alex Grinberg

There are many common tasks that require parameters which may be optional. One common example is a search that has many different ways to locate the data such as name, age, address, gender, etc. Another example is selecting groups of data via filtering. Again, this task can have multiple criteria to select the data to be presented.

In both cases, not all of the criteria would be specified each time the procedure is called. Coding a SQL statement with one or more parameters missing in the WHERE clause causes a problem. A common solution is the dynamic SQL and IF statement which allows the building of a filter based on passing values, or in the case of fewer parameters using an IF statement to call a query which a statically predefined filter where a specific parameter has value.

In this article I would like to show a different technique, let's call it a smart filter, which means - the WHERE clause skips empty parameter(s) and excludes them from the query by themselves.

If you are not familiar with how the EXEC() and sp_executesql procedures work, which are the dynamic SQL options, I would recommend you read an article written by Erland Sommarskog [The Curse and Blessings of Dynamic SQL](http://www.sommarskog.se/dynamic_sql.html). This article has a comprehensive explanation for dynamic SQL. It can be found at: http://www.sommarskog.se/dynamic_sql.html.

Now let's focus on the issue of how to avoid dynamic SQL for this case. We need to let the WHERE clause know when the parameter equals the default value do not use this as a filter condition.

To do that, you must use this syntax: (COLUMN_NAME = @PARAM_NAME OR @PARAM_NAME = DEFAULT_VALUE)

Below I have provided you with a code sample based on the Northwind database. Lets take one line from the sample and see how the filter works.

The filter line is (Products.ProductName = @prodname OR @prodname Is Null). Break filter line down in two parts. The first part would be "Products.ProductName = @prodname attempt to retrieve rows from result set and filter by ProductName." The second part would be "If the value in the row does not match then parameter equal to default which is always true." This way if the parameter remains as the default value filter, look at the second part and exclude all parameter(s) with the default values from the WHERE clause.

Note: In my coding techniques I tried to limit my usage of the OR operator as much as possible, the dynamic SQL would not be the better choice from the possibilities that are given. In the final analysis, this technique could be one more choice for you to use.

Code samples: Code is available at www.sqlservercentral.com

Finding Primes

By Kathi Kellenberger

Introduction

I do not get a chance to read much popular literature, but my daughter recently gave me the book [The Curious Incident of the Dog in the Night-Time](#) by Mark Haddon. Once I start reading a novel, I get so involved that I just keep reading until the book is finished. Luckily, this book contained just over 200 pages, and I completed the entire book one Sunday.

The interesting thing about this book is that the narrator and protagonist of the story is a 15 year old autistic boy. He is extremely gifted in math and physics but is unable to understand most human emotions or connect with the people close to him in the ways one would normally expect. The premise of the story is that the young hero is writing the book himself about solving the murder of a neighbor's dog.

Because the teen is so obsessed with math, he decides to number the chapters with prime numbers. A prime number is only evenly divisible by itself and one. He brings up prime numbers and the search for new prime numbers several times throughout the story. Since I am so passionate about SQL Server, I wondered how difficult it would be to create a list of prime numbers using T-SQL and how efficient the algorithm would be.

How to find prime numbers

The author described the search for primes as listing all possible numbers in order and then eliminating the numbers divisible by two, then by three, then by four, then by five, then by six, etc. I knew that I would only have to check to see if a prime number candidate was divisible by other prime numbers. For example, why check to see if a number is divisible by six? If a number is not divisible by two or three, then it will not be divisible by six, either.

I also needed to determine the maximum divisor to check for each prime number candidate to avoid doing extra work. My first thought was that I could stop testing divisors at one-third the value of the prime number candidate. My reasoning was that I would only consider odd numbers, and the first divisor to consider was three. When dividing by

three I was also effectively testing one-third of the prime number candidate. After doing a bit of research about [how to determine small prime numbers](#), I found that I could stop at the square root of the prime number candidate. This made sense to me when I realized that as the divisor increased, the quotient decreased. The two numbers would meet at the square root of the dividend. Consider the example in Listing 1.

Operation	Result
149 / 2	74 r 1
149 / 3	49 r 2
149 / 5	29 r 4
149 / 7	21 r 6
149 / 11	13 r 6
149 / 13	11 r 6
149 / 17	8 r 13

Listing 1: Division operations to determine if 149 is prime

Notice that the quotient decreases in value as the divisor increases. At the point that the divisor reaches 13, the quotients are now in the range of divisors that have already been checked. The square root of 149 is between 12 and 13 so it is unnecessary to test additional divisors past 13. Since none of the remainders are zero, 149 is prime.

The modulo operator (%) returns the remainder from a division operation. When a zero is returned from a modulo operation, the dividend is evenly divided by the divisor. Conversely, if a non-zero value is returned from a modulo operation, the dividend is not evenly divided by the divisor. For a number to be prime, all modulo operations must return zero. If a non-zero value is returned by each and every modulo operation for a particular number, that number is prime.

The T-SQL Solution

The first step in my solution is to create a table (Listing 2) to hold the prime numbers and populate it with the first two primes: two and three.

```
CREATE TABLE [dbo].[Primes](
    [Prime] [int] NOT NULL,
    CONSTRAINT [PK_Prime] PRIMARY KEY CLUSTERED
(
    [Prime] ASC
)WITH (PAD_INDEX = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
```

```
) ON [PRIMARY]

Insert into Primes (Prime) values (2)

Insert into Primes (Prime) values (3)
```

Listing 2: The Prime table

The script, a stored procedure called `usp_FindPrimes`, determines the first number to check: two more than the maximum prime number in the table. Another variable is used to control how many prime numbers to locate each time the proc runs.

Because SQL Server works best with sets of data, I believed it would be more efficient to write a query comparing the current prime number candidate against the list of prime numbers already discovered. Each time through the loop, the prime candidate is incremented by two so that only odd numbers are considered. If all of the modulo operations return 0, the number is inserted into the prime table.

Editor's Note: The code is available at www.sqlservercentral.com

I was surprised at how quickly the proc ran, in about two seconds, on my laptop. Because the script always begins with a number two more than the maximum prime number in the table, running the proc additional times continues to build the prime number list. The duration increased very slowly as the table grew in size.

Is the solution accurate?

Even though I had an efficient stored procedure that seemed to work, I wondered if it was really accurate. By importing the [list of the smallest 1000 known prime numbers](#) and comparing them to my first set of results, I was able to verify that my result set matched the first 1000 known prime numbers. My table contained 1002 rows after running the procedure once (remember it was seeded with two and three) and 1000 rows were returned when I joined on the table of known primes. The KnownPrime table is included in the downloadable file. Listing 4 shows the query which returned 1000 rows.

```
select Prime, KnownPrime

from Primes join KnownPrimes
on Prime = KnownPrime
```

Listing 4: Checking the results

What about a cursor?

I was curious to see if there would be a big loss of performance by using a cursor based approach. To test the idea, I created a second stored procedure, `usp_FindPrimes_Cursor`. This procedure uses a cursor to perform each modulo operation individually instead of all of them at once for each prime number candidate. Maybe I should not have been surprised when the cursor method took 15 seconds the first time it ran on my laptop. Subsequent trials took even longer as more rows were added to the Primes table and the cursor had more rows to loop through. Once again, the set based approach consistently performed much better than a cursor. The code for `usp_FindPrimes_Cursor` is also included in the [download](#) file.

Conclusion

T-SQL is a very rich language that can be used to solve many interesting problems besides retrieving and manipulating data. While this example will not help discover any previously unknown prime numbers, the largest known prime numbers are thousands of digits long and some mathematicians devote their careers to the task, it

demonstrates a creative approach to solving a problem that would be difficult to solve manually. It also demonstrates that SQL Server typically performs best when operating on sets of data versus the use of a cursor.

CTE Performance

By Peter He

Hierarchy structures are widely used in data model and SQL Server implementation for real world entities like manageremployee relation, organizational structures, regional structures etc. In order to get all the descendants for a node in the hierarchy, we need to recursively query the children of the node, and the children of the children, and so on until we reach the leaf level. In SQL Server 2000, this is achieved by loops traversing all the levels under the node. SQL 2005 provides the new feature Common Table Expression (CTE), which can be used to solve this request.

Introduction

When CTE is used in recursive query, it consists of at least one anchor member and one recursive member, e.g., in the following sample that gets all the employees under the "Vice President of Production",

- [1] is the anchor member that queries the record for "Vice President of Production";
- [2] is the recursive member by referencing the CTE name AllEmps;
- [3] is the execution of the CTE;

```
USE AdventureWorks;
```

```
GO
```

```
WITH AllEmps (ManagerID, EmployeeID, ManagerTitle, EmpTitle, [Level])
```

```
AS
```

```
(
```

```
--[1]
```

```
SELECT E.ManagerID, E.EmployeeID, F.Title, E.Title, 0 AS [Level]
```

```
FROM HumanResources.Employee AS E,HumanResources.Employee AS F
```

```
WHERE E.Title='Vice President of Production' AND E.ManagerID=F.EmployeeID
```

```
UNION ALL
```

```
--[2]
```

```
SELECT E.ManagerID, E.EmployeeID, A.EmpTitle, E.Title, A.Level + 1
```

```
FROM HumanResources.Employee AS E
```

```
INNER JOIN AllEmps AS A ON E.ManagerID = A.EmployeeID
```

```
)
```

```
-- [3]

SELECT ManagerID, EmployeeID, ManagerTitle, EmpTitle, [Level]

FROM AllEmps

ORDER BY [Level], ManagerTitle, EmpTitle

GO
```

Though CTE provides a simple and neat solution for recursive queries in SQL 2005, we need to know its performance implication before we migrate existing code.

Performance Test

Hierarchy structures can be level intensive, i.e. the tree has many levels with fewer children under each node, or sub-node intensive, i.e. the tree has fewer levels with many children under each node. Tests are performed for these two extreme scenarios.

Test Environment

- Hardware: Desk top PC with P4 1.60GHz CPU, 1GB RAM, 200GB IDE hard drive without software or hardware RAID.
- Software: SQL server 2005 developer edition with SP2 on Windows 2003 Standard Edition;
- Table schema and the stored procedures ([script](#)):

The script creates one table and stored procedures used by the test. The table schema is as follow:

```
CREATE TABLE dbo.Groups

( GroupID int identity NOT NULL,

  GroupName nvarchar(100),

  ParentGroupID int NULL,

  [Description] nvarchar(150)

)

ALTER TABLE dbo.Groups ADD CONSTRAINT PK_Groups PRIMARY KEY CLUSTERED (GroupID)

CREATE INDEX IX_Groups_ParentGroupID ON dbo.Groups (ParentGroupID)
```

The stored procedures used for descendants queries are `dbo.GetDescendantsCTE` and `dbo.GetDescendantsLoop`, which query all the descendant groups for a given group by CTE and loop respectively.

The stored procedures used for ancestors queries are `dbo.GetAncestorsCTE` and `dbo.GetAncestorsLoop`, which query all the ancestor groups for a given group by CTE and loop respectively.

Test Case Design

Tests to query descendants:

	Script to populate data	Roots	Children	Total levels	Total records
Test 1	CTETestPopulateData1.txt	2	2	10	2046
Test 2	CTETestPopulateData2.txt	2	3	10	59048
Test 3	CTETestPopulateData3.txt	2	12	4	3770

For each test case, nodes of different levels in the tree are selected and its descendants are queried by the two stored procedures respectively.

Test to query ancestors:

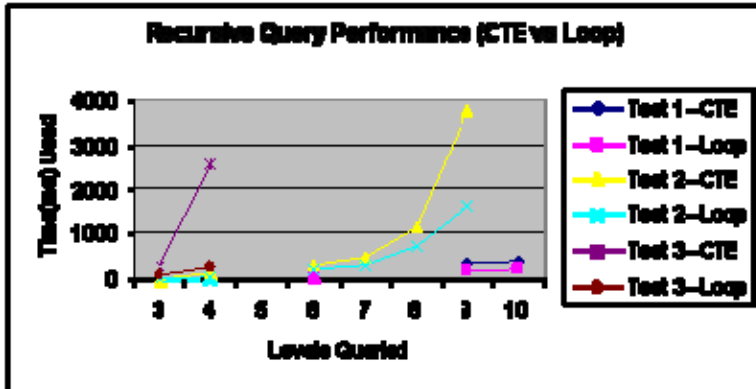
	Script to populate data	Roots	Children	Total levels	Total records
Test 5	CTETestPopulateData2.txt	2	3	10	59048

Test Results

Results are shown in the following table and chart.

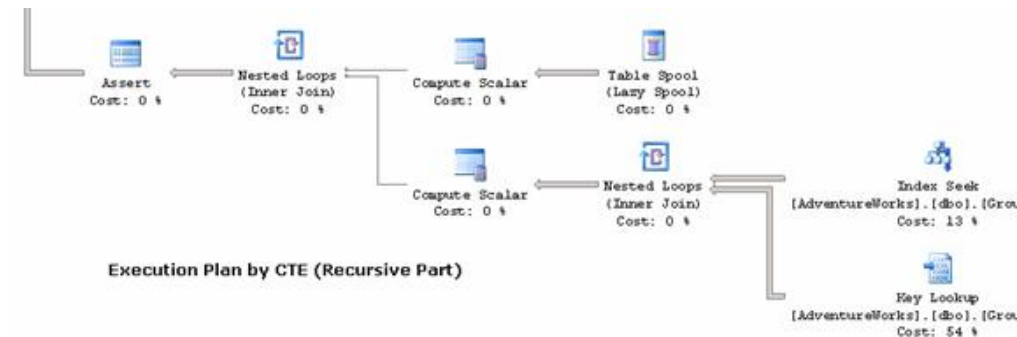
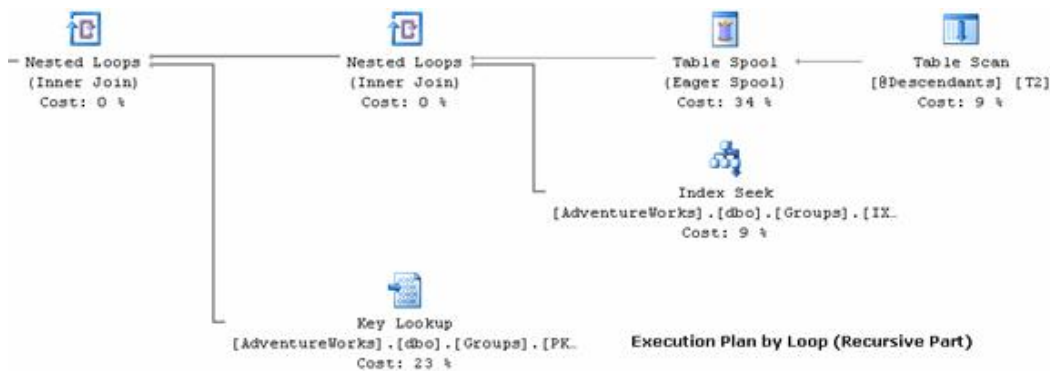
Tests	Levels	Rows Returned	ByCTE(ms)	ByLoop(ms)	CTE:Loop
Test 1	10	1023	359	214	1.6776
	9	511	317	166	1.9096
	6	63	90	63	1.4286
Test 2	9	9841	3743	1601	2.3379
	8	3280	1136	703	1.6159
	7	1093	467	306	1.5261
	6	364	291	203	1.4335
	4	40	125	46	2.7174
	3	13	<3	<3	1.0000

Test 3	4	1885	2526	255	9.9059
	3	157	258	77	3.3506



The results indicate that CTE is much slower than loop. When the number of children increases under each node, the performance of CTE gets worse quickly.

The following images are the actual execution plans of the recursive part for both methods. The plans do not give any clue to their performance difference. However, both plans have expensive key lookup operator due to the index seek operator in the plans.



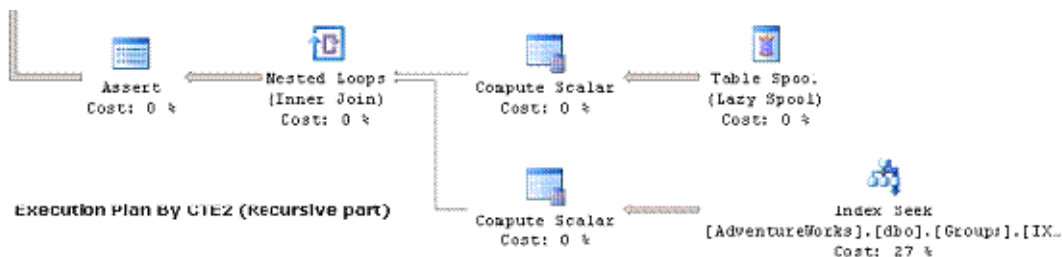
To investigate whether key lookup on CTE and loop recursive queries has different behavior, I created two new stored procedures (dbo.GetDescendantsCTE2 and dbo.GetDescendantsLoop2) to keep only the key column

The Best of SQLServerCentral – Vol. 5

GroupID of the index and [Level] in the CTE and the table variable. Other output columns are retrieved by join the CTE (table variable) with the table in the select statement, where clustered index seek is used:

```
SELECT G.GroupID, G.ParentGroupID, G.GroupName, P.GroupName AS ParentGroupName, D.[Level]
FROM Descendants D INNER JOIN dbo.Groups G ON D.GroupID=G.GroupID
LEFT JOIN dbo.Groups P ON G.ParentGroupID=P.GroupID
ORDER BY [Level],ParentGroupID,GroupID
```

Run [CTETestPopulateData2.txt](#) to populate test data, and run the "Test4" in both [CTETestByCTE.txt](#) and [CTETestByLoop.txt](#) – all available from sqlservercentral.com. The following is the new recursive part execution plan for CTE. Both plans for CTE and loop eliminated the key lookup operator.



The results are showed below. The performance of both methods is improved by removing the key lookup.

Levels	Rows Returned	Test#	ByCTE(ms)	ByLoop(ms)	CTE:Loop
9	9841	Test2	3743	1601	2.3379
		Test4	2056	1402	1.466
		Improvements	82%	14%	
8	3280	Test2	1136	703	1.6159
		Test4	823	547	1.5046
		Improvements	38%	28%	

In the query ancestor test, run [CTETestPopulateData2.txt](#) to populate the data, and run the "Test5" in both [CTETestByCTE.txt](#) and [CTETestByLoop.txt](#). The result shows that performance of CTE is as good as, if not better than, traditional loop methods. I didnt test trees with more than 10 levels.

Tests	Levels	Rows Returned	ByCTE(ms)	ByLoop(ms)	CTE:Loop
Test 5	10	10	<3	<3	1.0

A test (script not included) to query descendants on a 10 level tree with one child per node shows that the performance of CTE and loop is also comparable.

Conclusion

CTE is a neat way to implement recursive queries. However its performance is generally more than 50% slower than the recursive query by the traditional loop method if recursive part of the query returns more than 1 record for each node like the query for descendants of a node in a tree. When the number of children under a node is increasing, the performance of CTE degrades even more quickly in the mentioned scenario.

If the recursive part returns one record for each node, e.g. queries for ancestors of a node in a tree, the performance of CTE is comparable as loop method.

In the recursive part of a CTE or loop method, if index seek is used in the execution plan, only including columns covered by indexes of the tables involved in the recursive query can improve performance.

Before migrating the recursive queries to CTE in your database, you need to understand the scenarios how data is queried and how heavy the recursive query load is in your system. In a busy OLTP system, 10ms degradation for highly concurrent calls can become bottleneck of the whole server.

The Effect of NOLOCK on Performance

By Wayne Fillis

Updated with an author's note

When this article was first published, it produced some lively debate. It was believed by some that the article misled readers into thinking that NOLOCK should be a tool for performance tuning, which it is not. There is a divided opinion on the appropriate use of NOLOCK, and therefore I am adding this foreword to my article in an effort to clarify matters.

NOLOCK is a query optimizer hint. It has advantages, and disadvantages, and it potentially breaks one of the fundamental rules of databases – data integrity through the use of a locking mechanism. In a nutshell, NOLOCK does not take locks on the data it reads. This provides benefits for concurrency and performance, but data integrity suffers.

I can't speak for Microsoft or the MVP's, but if Microsoft offers an ISOLATION LEVEL which does the same thing as NOLOCK, then there must be an appropriate place for the use of NOLOCK. We've discussed the advantages of NOLOCK, so let's take a look at some of the disadvantages.

Firstly, when reading with NOLOCK you are reading uncommitted data. This means the data has not been committed to the database, and could be rolled back (undone) after you have read it. So, you may find your application is processing invalid data. This is not so much of a problem with Name and Address data, but is particularly problematic with Accounts, Finance and Sales data. This is where your data integrity would suffer.

Secondly, as noted by one of the SQLServerCentral.com forum posters, Itzik Ben-Gan demonstrated to the UK SQL Server User Group that NOLOCK can return duplicate rows when reading data. To quote the forum posting: "Tony Rogerson's blog has code which demonstrates this. <http://sqlblogcasts.com/blogs/tonyrogeron/archive/2006/11/16/1345.aspx>".

This article was intended as a clinical approach to the effect of NOLOCK on performance, without advocating you use NOLOCK in an attempt to speed up your queries. If you do decide to use it that way, I would like you to know the risks.

That's all I have to add, folks. I hope you read on and enjoy the article!

Cheers, Wayne Fillis

Introduction - How locking works

Using the NOLOCK query optimiser hint is generally considered good practice in order to improve concurrency on a busy system. When the NOLOCK hint is included in a SELECT statement, no locks are taken when data is read. The result is a Dirty Read, which means that another process could be updating the data at the exact time you are reading it. There are no guarantees that your query will retrieve the most recent data.

The advantage to performance is that your reading of data will not block updates from taking place, and updates will not block your reading of data. This means that you will have improved concurrency on your system, and more tasks can be performed at the same time. Without NOLOCK each access to the database results in a lock being made on the row being read, or the page on which the data is located. SELECT statements take Shared (Read) locks. This means that multiple SELECT statements are allowed simultaneous access, but other processes are blocked from modifying the data. The updates will queue until all the reads have completed, and reads requested after the update will wait for the updates to complete. The result to your system is delay, otherwise known as Blocking.

Blocks often take milliseconds to resolve themselves, and this is not always noticeable to the end-user. At other times, users are forced to wait for the system to respond. Performance problems can be difficult to diagnose and resolve, but blocking is often easily resolved by the use of NOLOCK.

One solution to blocking is to place the NOLOCK query optimiser hint on all select statements where a dirty read is not going to cause problems for your data integrity. You would generally avoid this practice in your Finance stored procedures, for example.

Faster reading of data with NOLOCK

Logically, a select using NOLOCK should return data faster than without, as the taking of locks invokes an overhead. Ever curious, I decided to investigate the impact of NOLOCK on the performance of a SELECT query. In order to prove this I needed data - lots of it. The following example shows code I ran to create two tables: Products and Orders, and populate them with 5 million rows each.

You will need to create a database called NOLOCKPerformanceTest, and I recommend you allocate 400MB of initial space for the data file. This code could take over an hour to complete, depending on your system. I am running this on an Intel Dual Core 2Ghz laptop, with a 7200rpm SATA drive and 2GB RAM. The database edition is SQL Server 2005 Developer, with SP1 and hotfix 2153 installed.

I decided to use very large tables, as the longer it takes my test to run, the more noticeable any differences.

Editor's Note: The code is available at www.sqlservercentral.com

I started the exercise by running a simple join query between the two tables, while checking the runtime with NOLOCK, and without NOLOCK. I flush the data buffer before each execution, and I drew an average over 5 executions.

The Results

The results for execution times are listed below. It takes a while to run, as 310,001 rows are returned. If you are running these queries, execution times will vary depending on the specification of your server. The format for these results is hh:mm:ss.nnn, where nnn is milliseconds.

Without NOLOCK:

Run1: 00:00:29.470

Run2: 00:00:30.467

Run3: 00:00:28.877

Run4: 00:00:29.123

Run5: 00:00:29.407

Average: 00:00:29.469

With NOLOCK:

Run1: 00:00:25.060

Run2: 00:00:25.157

Run3: 00:00:25.107

Run4: 00:00:25.140

Run5: 00:00:25.893

Average: 00:00:25.271

This test shows the average execution time is less when using NOLOCK. The average time saved is 00:00:04.197, which equates to a saving in elapsed time of approximately 14%.

My next step was to check the amount of CPU and IO used in both queries. I added the following code immediately before the first SELECT in examples 3 and 4, and ran both queries again:

```
SET STATISTICS IO ON
```

```
SET STATISTICS TIME ON
```

The execution times were the same as before, and the results for the SET STATISTICS options were as follows:

Without NOLOCK:

```
Table 'Orders'. Scan count 3, logical reads 27266, physical reads 10, read-ahead reads 24786, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

```
Table 'Products'. Scan count 3, logical reads 27119, physical reads 78, read-ahead reads 24145, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

```
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

```
SQL Server Execution Times:
```

```
CPU time = 13844 ms, elapsed time = 27480 ms.
```

With NOLOCK:

```
Table 'Orders'. Scan count 3, logical reads 24845, physical reads 0, read-ahead reads 24826, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

Table 'Products'. Scan count 3, logical reads 24711, physical reads 0, read-ahead reads 24689, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

SQL Server Execution Times:

CPU time = 13813 ms, elapsed time = 24999 ms.

While not being an expert at IO, I nevertheless noticed lower physical and logical reads for the NOLOCK version. I ran both queries several times, and noticed the same pattern. Using NOLOCK definitely results in lower IO.

The CPU time is similar between the versions, though you can see that once again the elapsed time is lower when using NOLOCK.

Changing the date in the WHERE clause (see examples 3 and 4) to "01 Dec 2006" results in no rows being returned. However, the execution time for the without NOLOCK example shows an average of about 24 seconds. With NOLOCK it runs for approximately 22 seconds each time the query is run.

Conclusion

This article has hopefully proven that using NOLOCK when querying data results in lower IO and faster response times. I have executed these queries at least a dozen times each over several weeks, and the expected result is always the same.

If you can suggest any additional queries I can put to the test, please don't hesitate to contact me on willis@gmail.com. I would be happy to test your scenarios, and present the results in a future article.

Converting Hexadecimal String Values to Alpha (ASCII) Strings

By Stephen Lasham

Overview

On investigating some data, I came across a text column containing hexadecimal string information. A hexadecimal string consists of consecutive pairs of data written in base-16, with the symbols 09 and AF, where A equals 10, B equals 11, etc, up to F equals 15, i.e. "416E6E656C6F75697361".

It is possible to take each pair, convert it to decimal, and then to its ASCII character equivalent. Simple mathematics allows the conversion to decimal. As it is base-16, multiply the second column in from the right (sixteenths column) by 16, remembering A to F equals 10 to 15, and add on the value of the column on the right (unit column).

$6E = (6 * 16) + 14 = 110$

$6F = (6 * 16) + 15 = 111$

Next convert the decimal value to its ASCII equivalent using a conversion table, or in SQL the operation CHAR.

Select CHAR(110) = n

Select CHAR(111) = o

My string data consisted of ten pairs of hex data per row, which I required to convert to ASCII to make it readable in English. Investigating the Internet showed ways to convert character data to hexadecimal but not a lot the other way. I considered going through the string character pair by character pair and execute the above maths, but thought perhaps there may be a better way.

The online examples happily showed me information for converting a single hex character, where the hexadecimal two-character string is preceded by a 0x and is defined with a type of *varbinary*.

```
select char(cast(0x57 as varbinary)) -- = "W"
```

Unfortunately, the above example shows the hex pair as a constant not a variable, and on attempting to set a variable into the select statement, it failed as shown below.

```
Declare @pair as char(2)
```

```
Set @pair = '57'
```

```
select char(cast('0x' + @pair as varbinary)) -- = null
```

The only way I could see round this was to build the select into a statement string and execute it.

```
Declare @stmt nvarchar(500)
```

```
Declare @pair as char(2)
```

```
Set @pair = '57'
```

```
Set @stmt = 'select char(cast(0x' + @pair + ' as varbinary))'
```

```
execute sp_executesql @stmt -- = "W"
```

This actually works, but handling only one character (hex pair) at a time, thus I needed to do a pass of my sample table, and then do a character-by-character conversion of each string. The resulting code follows.

My Solution

As this is an example, it uses a sample table created below. This table contains three columns, a record id to sequence the row, the hex string to be converted, and an alpha string to hold the converted results. This is loaded with some sample data for conversion, and then follows the code to convert it.

Editor's Note: Code available at www.sqlservercentral.com

If all is well, you will have a list of names following execution of this script.

Some Analysis

The biggest pain of this is to extract the characters one at a time. This required the need to create a temporary table (#result) to hold a record for each character in the string. Now my knowledge of SQL is sadly lacking, as I would have preferred to have only a single record or even a variable to hold the resulting string to which I could just concatenate each new successive character.

i.e. instead of

```
Set @stmt = 'insert into #result values(char(cast(0x' + @pair + ' as varbinary)))'
```

I would have preferred the result column to be 10 characters long and apply a statement as follows

```
Set @stmt = 'update #result set result = result + '  
            + 'values(char(cast(0x' + @pair + ' as varbinary)))'
```

This however did not work, and if anyone can enlighten me as to why, I would love to know.

Instead, my code generates a #result table for each hex string, which looks like this.

For hex string 416E6E61737461736961

```
1, A  
2, n  
3, n  
4, a  
5, s  
6, t  
7, a  
8, s  
9, i  
10, a
```

In this form, it is of little use and needs joining back into a single string. I used the unique identifier along with the max and case operations to concatenate the row values into a single string.

```
Update #HexToAlpha
```

```
set alphastring = (Select max(case when recordid = 1 then result else '' end)  
+ max(case when recordid = 2 then result else '' end)  
+ max(case when recordid = 3 then result else '' end)  
+ max(case when recordid = 4 then result else '' end)  
+ max(case when recordid = 5 then result else '' end)  
+ max(case when recordid = 6 then result else '' end)  
+ max(case when recordid = 7 then result else '' end)  
+ max(case when recordid = 8 then result else '' end)  
+ max(case when recordid = 9 then result else '' end)  
+ max(case when recordid = 10 then result else '' end)
```

```
from #result) where recordID = @recordcount
```

This provided me with the desired result, in this case "Annastasia". An obvious problem presents itself with this solution, in that it is limited to fixed length hexadecimal strings, in this case 20 characters of hex into 10 characters of text. Increasing this means adding lines to the max/case statements above. My logic only needed to cater for the above lengths so this suited me fine.

Conclusion

I enjoyed the excursion away from the daily coding routines I normally work with and hope this code proves useful to others. I look forward to seeing alternative methods, including ones that can handle variable length strings. If you only have one string to convert, a good online translator is available at <http://www.defproc.co.uk/toys/hex.php>

A Refresher on Joins

By Jambu Krishnamurthy

In this article we will look at JOINS. We will primarily focus towards beginners, but this may be a refresher for the experienced. We will see how each type of JOIN works. Specifically we will discuss these: INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, SELF JOIN and then wind up with CROSS JOINS.

Note there are so many other terminologies used, like equi join, non-equi join, inner join etc, but there are only the 6 types of joins as mentioned above.

Let's start by creating the tables/data required to understand JOINS.

Step - 1

Use the following script to create the table and data. You can just cut and paste this code into Query Analyzer and execute it. (Editor's note: The code is at www.sqlservercentral.com)

You will now have data like this in the three tables. Please note that the empty spaces in rows are only to show a pictorial mapping. There are only 10 records in t1, 10 records in t2 and 5 records in t3 (no empty rows or NULL values in any field).

t1	t2	t3
f1	f1	f1
--	--	--
1		
2		2
3		3
4		
5		
6	6	6
7	7	

```
8      8
9      9
10     10
      11
      12      12
      13      13
      14
      15
```

Step - 2

Try these queries

```
select * from
t1 JOIN t2
on t1.f1 = t2.f1
```

```
select * from
t1 INNER JOIN t2
on t1.f1 = t2.f1
```

```
select * from
      t1,t2
where
      t1.f1 = t2.f1
```

All the three queries above are essentially the same. The last one is referred to as the equi-join. What if you want all rows in t1 that are not in t2 (non-equi joins?). Will this query work?

```
select * from
      t1,t2
where
      t1.f1 <> t2.f1
```

It did not quite do what you expected, huh? Here is one way to achieve this:

```
select * from t1
```



```
where t1.f1 not in (select t2.f1 from t2)
```

Step - 3 - LEFT OUTER JOIN

Now suppose you want all the records from t1, whether they have a corresponding entry in t2 or not. Why will we ever want to do this? A simple situation could be that, you want a list of all the departments (t1) whether there are employees or not in that department (t2). For this example assume t1.f2 holds department ids, and t2.f1 as department ids fk'd into t1.f1. Here is the query

```
select t1.f1, t2.f1
from t1 LEFT OUTER JOIN t2
on (t1.f1 = t2.f1)
```

This is exactly similar in functionality to the above.

```
select t1.f1, t2.f1
from t1 LEFT JOIN t2
on (t1.f1 = t2.f1)
```

t1.f1	t2.f1
=====	=====
1	NULL
2	NULL
3	NULL
4	NULL
5	NULL
6	6
7	7
8	8
9	9
10	10

Step - 4 - RIGHT OUTER JOIN

Ok, now the reverse, assume we want all the employees, whether or not they have been assigned to any departments. Both the queries below are similar again

```
select t1.f1, t2.f1
from t1 RIGHT OUTER JOIN t2
on (t1.f1 = t2.f1)
```

```
select t1.f1, t2.f1
from t1 RIGHT JOIN t2
on (t1.f1 = t2.f1)
```

```
t1.f1 t2.f1
```

```
=====
```

```
6      6
```

```
7      7
```

```
8      8
```

```
9      9
```

```
10     10
```

```
NULL   11
```

```
NULL   12
```

```
NULL   13
```

```
NULL   14
```

```
NULL   15
```

Step - 5

JOINS make much sense between two tables, but can obviously be extended to more than two tables. However, just to demonstrate, as to how complicated/confusing they can become, here is a FULL OUTER JOIN example between t1, t2 and t3.

```
select a.f1, b.f1, c.f1
from t1 a
FULL OUTER JOIN t2 b on (a.f1 = b.f1)
FULL OUTER JOIN t3 c on (a.f1 = c.f1)
```

```
t1.f1 t2.f1 t3.f1
```

```
=====
```

```
6      6      6
```

```
7      7      NULL
```

```
8      8      NULL
```

```
9      9      NULL
```

10	10	NULL
NULL	11	NULL
NULL	12	NULL
NULL	13	NULL
NULL	14	NULL
NULL	15	NULL
5	NULL	NULL
4	NULL	NULL
3	NULL	3
2	NULL	2
1	NULL	NULL
NULL	NULL	13
NULL	NULL	12

Observe the query and output carefully...and then see if you can get the following output, which you will agree makes more sense.

t1.f1 t2.f1 t3.f1

=====

6	6	6
7	7	NULL
8	8	NULL
9	9	NULL
10	10	NULL
NULL	11	NULL
NULL	12	12
NULL	13	13
NULL	14	NULL
NULL	15	NULL
5	NULL	NULL
4	NULL	NULL
3	NULL	3
2	NULL	2

```
1      NULL  NULL
```

That's it for OUTER JOINS for now.

Step - 6 - SELF JOINS

When the same table is used in a query with two different aliases, such a join is referred to as a self-join. Let us see this case with an example. Create the following table and data.

```
create table jk_Workers(Worker int, Manager int)
```

```
insert into jk_Workers values(111,NULL)
```

```
insert into jk_Workers values(222,111)
```

```
insert into jk_Workers values(333,222)
```

```
insert into jk_Workers values(444,222)
```

```
insert into jk_Workers values(555,222)
```

```
insert into jk_Workers values(666,111)
```

```
insert into jk_Workers values(777,111)
```

```
insert into jk_Workers values(888,333)
```

```
insert into jk_Workers values(999,222)
```

All the workers with their respective managers are stored in this table. And if we wish to list out the managers and the workers working under them, we could use a query similar to the following.

```
select b.Manager,b.Worker
```

```
from jk_Workers a, jk_Workers b
```

```
where a.Worker = b.Manager
```

```
or b.manager is null and a.Manager is null
```

```
order by b.Manager
```

```
Manager Worker
```

```
NULL      111
```

```
111       222
```

```
111       666
```

```
111       777
```

```
222       999
```

```
222       333
```

222 444

222 555

333 888

Step - 7 - CROSS JOINS

A cross join is referred to as a Cartesian product, which means, for each element in set-A pick all the values from set-B

```
select t1.f1, t2.f1
```

```
from t1 cross join t2
```

```
select t1.f1, t2.f1
```

```
from t1,t2
```

Both the above queries are same. The output of this query will be 100 rows. This is because, for each row in t1, all the rows in t2 would be matched. Unless you know what you are doing, you got to be careful in building your queries, to ensure you are not working on Cartesian products.

I am not sure of a real business need for a Cartesian product (I can be wrong as ever), but I know mathematically it is valid to have the same y values for different values of x ($m=0$), basically a horizontal line on the Cartesian plane. Joins could be as complicated/confusing in real world scenarios, but if your fundamental understanding of how they behave is clear, you should be able to handle any situation.

Thankz for reading!

Using CLR integration to compress BLOBs/CLOBs in SQL Server 2005

By Yoel Martinez

Introduction

Manipulating Binary Large Objects (BLOBs) and Character Large Objects (CLOBs) has always been difficult in SQL Server. The new SQL Server 2005 provides new data types (NVARCHAR(MAX), VARCHAR(MAX) and VARBINARY(MAX)) for large object storage (up to 2 GB) allowing better manipulation as well as the ability to process these data types using CLR procedures and functions.

This article shows how to create CLR functions to seamlessly compress and decompress large data objects with minimum performance impact using compression algorithms included in .NET Framework 2.0. Code samples included in this article can be used in any database implementation but do not cover all possible scenarios. For large implementations or mission critical applications consider using third party products like [SQLCompres.NET](#) (it is free).

The new data types

SQL Server 2005 provides three new data types to store and manipulate large objects. VARCHAR(MAX) can store CLOBs, NVARCHAR(MAX) does the same but allows Unicode characters and VARBINARY(MAX) can store BLOBs. Books Online states that max indicates that the maximum storage size is $2^{31}-1$ bytes. These new data types can be used with a wide range of T-SQL commands and behave much like traditional VARBINARY(n), VARCHAR(n) and NVARCHAR(n).

The new data types should replace the TEXT and IMAGE types from previous versions. As per SQL Server Books Online TEXT and IMAGE columns should not be used in new development and legacy applications should be changed to use the new types.

These new types, as opposed to TEXT and IMAGE, can be used as variables and function parameters and can be returned by CLR (or T-SQL) scalar-value functions. These new characteristics make them great candidates for compression. Previous attempts to add CLOB and BLOB compression to SQL Server involved using extended procedures, a difficult and risky business. Using the CLR integration capabilities introduced with SQL Server 2005 makes such implementation more secure and stable (and sometimes faster than their extended procedure counterparts).

CLR's procedures and functions parameters can receive and process these data types as SQLChars and SQLBytes. SQLChars can be used to pass VARCHAR(MAX) and NVARCHAR(MAX) parameter while SQLBytes is used to pass VARBINARY(MAX) types. CLR uses only Unicode characters and passing VARCHAR(MAX) as a parameter implies converting it to Unicode before parameters are passed.

Get the code from www.sqlservercentral.com

Compressing a BLOB

SQL Server 2005 allows CLR code to be registered as functions and stored procedure. Developers can now extend SQL Servers functionality using a broad array of programming languages from managed C++ to Visual Basic. How the CLR is hosted inside SQL Server goes beyond the scope of this article. For those who worry about enabling CLR integration, suffice to say that Microsoft has made a conscious effort to keep this integration as safe and secure as possible.

Lets use VARBINARY(MAX) for simplicity sake, since data can be converted between BLOB and CLOB types this articles code can be extended using T-SQL functions. Listing 1 contains a function to compress BLOBs in SQL server, the function receives a VARBINARY(MAX) or IMAGE as a SQLBytes and compresses it using the DeflateStream class provided in .NET Framework 2.0. SQLBytes represents a mutable type that wraps either an array or a stream. We are going to assume it wraps an array to avoid complicated code, and get the data from the Buffer property. Using this property might fail with larger BLOBs causing the CLR to throw an out of memory exception (but dont worry, unlike extended procedures errors, CLR exceptions should not crash your SQL Server).

Listing 1: Compression function – Available at www.sqlservercentral.com

Compressing a BLOB in SQL Server 2005 is as easy as passing a SQLBytes parameter, reading its content and writing it to a compression stream. The compression stream writes to a MemoryStream that is later used to create a new SQLBytes object that can be returned to SQL Server applications or directly to the client. There is only one caveat: Microsofts implementation of DeflateStream requires the stream to be closed before it writes the last compressed bytes, flushing is not enough.

Listing 2 loads an assembly in SQL Server (a process called cataloging where assemblies are verified for security and reliability) and creates the compression function. Listing 3 shows how to use the function in a T-SQL update. This usage makes compressing columns a seamless process that would require only server side adjustments.

Listing 2: Loading assembly and creating CLR function

```
CREATE ASSEMBLY [BlobCompression]

        FROM 'D:\Development\BlobCompression.Dll'

        WITH PERMISSION_SET = SAFE


CREATE FUNCTION [fn_decompress]          (

        @compressedBlob varbinary(MAX))

        RETURNS varbinary(MAX)

        AS      EXTERNAL NAME [BlobCompression].[UserDefinedFunctions].[fn_decompress];
```

Listing 3: Compressing data

```
create table #temp (

        blob_col      varbinary(max));


insert into #temp

values(convert(varbinary(max), 'To run your project, please edit the Test.sql file in your
project. This file is located in the Test Scripts folder in the Solution Explorer.'));


drop table #temp;
```

Decompressing a BLOB

Listing 4 contains a function to decompress a BLOB. This function follows the same principles used in compression but now reads from a stream returning a decompressed block that can be used to create and return a decompressed SQLBytes object.

Listing 4: Decompression function – Available at www.sqlservercentral.com

Listing 5 loads an assembly and creates a compression and decompression CLR function. Compression and decompression can be tested using listing 6, it creates a table and add some values to it, a compression update is run followed by a select statement that returns uncompressed data.

Listing 5: Loading assembly and creating CLR functions

```
CREATE ASSEMBLY [BlobCompression]

        FROM 'D:\Development\BlobCompression.Dll'

        WITH PERMISSION_SET = SAFE


CREATE FUNCTION [fn_decompress]          (
```

```
        @compressedBlob varbinary(MAX))

    RETURNS varbinary(MAX)

    AS      EXTERNAL NAME [BlobCompression].[UserDefinedFunctions].[fn_decompress];

CREATE FUNCTION [fn_compress](

        @blob varbinary(MAX))

    RETURNS varbinary(MAX)

    AS      EXTERNAL NAME [BlobCompression].[UserDefinedFunctions].[fn_compress];
```

Listing 6: Testing functionality

```
create table #temp (

    blob_col    varbinary(max));

insert into #temp

values(convert(varbinary(max), 'To run your project, please edit the Test.sql file in your
project. This file is located in the Test Scripts folder in the Solution Explorer.'));

update #temp

set blob_col = master.dbo.fn_compress(blob_col);

select convert(varchar(1000), master.dbo.fn_decompress(blob_col))

from #temp;

drop table #temp;
```

Limitations

The code included in this article allows column level compression in SQL Server 2005 but it lacks functions for consistency check and will not work very well with large objects (5 MB or more depending on configuration). It is intended to show how to use CLR integration in SQL Server to extend the engines functionality and provides an overview on what can be done with the new BLOB/CLOB data types.

When To Use Cursors

By Andy Warren

Most DBA's will tell you that cursors are bad, or worse. That stems from developers in the early days of SQL using cursors to accomplish tasks the same way they did in the programming language of their choice - looping. My standard line is that you should try to solve the problem in a set based way first and reserve cursors for these situations:

- Multiple database maintenance type tasks, where you need to run through many (or at least more than one) databases
- You just cannot figure out a set based solution. That may sound simple and/or lame, but we get paid to solve problems. If you (and your DBA) can't figure out a set based solution in 30 minutes, use the cursor. That solves the problem and often you gain a deeper understanding of the problem that may lead you back to a set based solution.
- You want to leverage logic you've already encapsulated in a stored procedure.

I hope you'll agree with the first point, imagine you'll vilify me for the second point, and maybe scratch your head about the third. The goal today is to get you to rethink your views on cursors and if enough interest, I may follow up with a deeper discussion. We'll focus on the third point as a starting place.

First, some background. I believe in using stored procedures for all data access with the exception of things that intrinsically require dynamic SQL like searches. I also believe that putting some business logic in a stored procedure is worth doing if it reduces round trips to the client machine. The limiting factor to stored procedures is that it's hard to pass in the equivalent of an array so that you can use that encapsulated logic for one record (request) or for many.

One example from my background required parsing a string containing order information (for books, shirts, etc) and splitting it into parent/child tables. Orders were received via an HTTP post, each post containing all the information relevant to the order. The post was inserted as a single string into a table so that it completed quickly, then a job ran frequently to process the new records. It was an ugly bit of parsing and I fell back on point #2 above, using a cursor to handle the parsing because there were some special cases I needed to handle. So, in pseudo code, I had something like this:

```
create proc usp_ParseOrder @WebOrderID

as

begin trans

some ugly cursor stuff here

commit trans
```

Possibly I could have solved it set based, but for the sake of discussion let's say that it performed well enough and had the advantage of being easy to understand if maintenance was needed that no further effort was warranted. For the purposes of this discussion it's how I solved the problem inside the stored procedure but rather that it was complicated to express.

So the job that processed new records looked something like this (again, pseudo code):

```
open cursor

for each record in cursor

    exec usp_ParseOrder @WebOrderID

next

close cursor
```

That's the flaw of building procedures designed to handle a single request (think of users adding/editing records, etc). They are fast and clean, but if you want to reuse the logic you either call the procedure repeatedly, refactor the procedure to handle 1 to unlimited records, or you duplicate the logic for batch processing.

The flaws are offset by a some nice wins:

- Duration of each transaction should be short compared to a large batch
- Growth of the log file has less chance of getting out of hand. Logging 100 small transactions provides the chance for the log to roll over where doing one big transaction may require the log to grow
- That complex logic is in one place and is pretty easy to work with
- If something goes wrong you roll back one very small transaction

Those potentials wins should also be considerations when you're coding. It's easy to think batch when you're processing 100 records or even a 1000, but what about 10k? 100k? Ever roll back a million record transaction?

I think looping to call a stored procedure multiple times to leverage logic is a valid and useful technique. Be practical and pragmatic about it's application and you'll do well. I look forward to the discussion!

Everybody Reports to Somebody

By Craig Hatley

Have you ever thought about the logistics of the organizational structure of a company? Do you know how many layers of management exist from the top to the bottom? I was recently forced to consider these very questions to support a new application that required managers to be able to access data associated with any subordinate employee. This could be an employee that directly reports to the manager or an employee that is several layers below the manager in the overall organizational structure.

For example, if my function received Bob's User ID, it should return the following people based on the sample organizational chart (See Figure 1) . . .

- Bob
- Sue
- Todd
- Mary
- Sandy
- Megan
- Randy

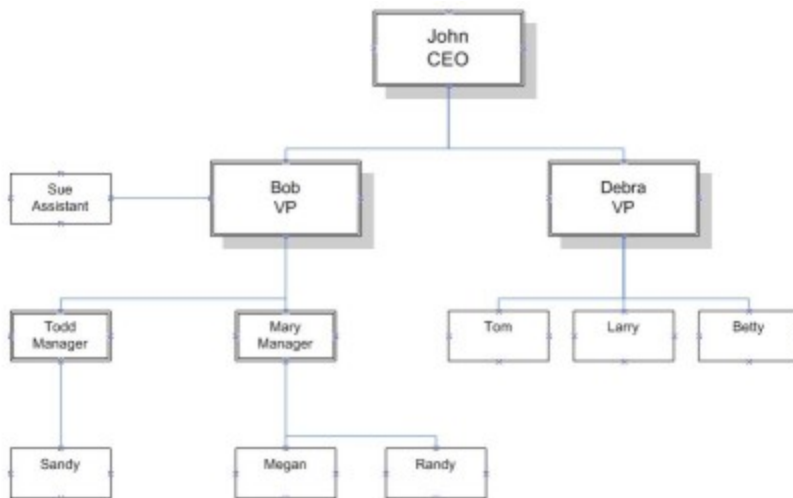


Figure 1

The table that stores our employee data (see figure 2) includes a 'User ID' field which is the primary key for the table and a 'Manager User ID' field which contains the User ID of the manager that each employee directly reports to (everybody reports to somebody philosophy). If Bob has a User ID of '2' in our sample above, Sue, Todd, and Mary would all have a value of '2' in their 'Manager User ID' field indicating that they report directly to Bob (see figure 3).

Column Name	Data Type	Length
UserID	int	4
FirstName	varchar	50
LastName	varchar	50
ManagerUserID	int	4

Figure 2

UserID	FirstName	LastName	ManagerUserID
1	John	Smith	0
2	Bob	Johnson	1
3	Debra	Burton	1
4	Sue	Fields	2
5	Todd	Jacobs	2
6	Mary	Sanders	2
7	Sandy	Teeter	5
8	Megan	Morris	6
9	Randy	Binks	6
10	Tom	Sims	3
11	Larry	Wright	3
12	Betty	Morgan	3

Figure 3

Now for the tricky part . . . One of the objectives of our stored procedure is that it should not assume any

foreknowledge of the organizational structure including the number of management layers. As I started this project I knew that my code would be required to perform some type of recursion, so I found a great article on the Microsoft web site that discussed recursion in SQL. After spending a little time reading the article, I discovered a couple of potential significant limitations in that SQL recursion is limited to 32 loops (to prevent infinite loop conditions) and most of the examples for passing data back up through the recursive loop chain used a data type that was restricted by a maximum length. Although our company doesn't currently have 32 layers of management hierarchy, I simply didn't want to settle for a 'work around' solution because I suspected it would eventually be an issue for us.

This caused me to start looking for alternatives which after much prayerful consideration yielded the following stored procedure ... (**Editor's Note:** The code is at www.sqlservercentral.com)

The stored procedure basically starts off by returning a list of the User IDs that report directly to the specified manager User ID. It then runs in a loop checking for User IDs that report to the list of User IDs that were returned in the previous loop until no more User IDs are found. The User IDs returned from each successive loop are added to the temporary table called #Result which is ultimately used to return a list of User ID values representing the subordinate employees.

We typically use the User ID list returned from the previous stored procedure in conjunction with another stored procedure (see below) to return a list representing the employees that report to the specified manager either directly or indirectly.

```
CREATE PROCEDURE select_subordinate_names_by_manager
    @UserID int
As
Create Table #Subordinates (UserID int)
Insert Into #Subordinates (UserID)
Exec ('dbo.select_subordinates_by_manager ' + @UserID)
Select e.UserID, e.FirstName, e.LastName
    From Test..Employee as e
    Join #Subordinates as s on s.UserID = e.UserID
    Order By e.FirstName, e.LastName
```

After looking at the code and the extensive use of temporary tables you may be concerned about how well this solution will perform. As a baseline, I measured the overall execution time of this technique in a test environment (see environment details below) with a single organizational structure branch that represents 50 layers of management and more than 4500 employees. Needless to say I was pleasantly surprised to discover that the overall execution time was only 64 ms.

Test Environment

- SQL Platform : Microsoft SQL Server 2000 (Developer Edition)
- Hardware Platform : Workstation Class PC (Single Pentium 4 3.2Ghz/3GB RAM)

Not In v Not Equal

By Ken Johnson

A coworker recently asked me which was "more efficient - a bunch of <> or a NOT IN clause?" The answer, I assumed, like almost all things relating to databases, is "it depends." The nice thing about that answer is that it not only camouflages my vast stores of ignorance, it is also quite often true.

This time, however, I thought I'd do some investigation to see if, in fact, we should prefer one method over the other in this case. The answer this time was a bit unexpected. It turns out that they are actually the same query, and it should make absolutely no difference. This is because SQL is a declarative language, meaning: you tell the computer what you want, not how to get it ^[1].

The query engine takes both of these queries and performs them with the exact same sequence of events. In actuality, I did find that one of the queries typically outperformed the other. I also discovered several other methods (written by people much smarter than myself) that were significantly quicker than either of the two methods I set out to test (the quickest method I tested was a full 35% faster than the quicker of the original two).

With all that in mind, let's set off to find out if NOT IN is quicker than a "bunch of <>" and see a few methods that improve in their performance. The original question was accompanied by a snippet of a WHERE clause evaluating the inequality of a column and four integer values:

```
s.Status_SV <> 214

and

s.Status_SV <> 215

and

s.Status_SV <> 216

and

s.Status_SV <> 217
```

Now that we know we're dealing with integer data, we'll set up a table with an integer column, then we'll set up a timing and iteration framework for tracking our query times over multiple executions. Just to make our queries lengthy enough to be nicely measurable, I arbitrarily chose to insert 1,000,000. The test results were less conclusive at 10,000 rows (with one notable exception), but the trends we see at 1,000,000 rows are clearly discernable by the time we have 100,000 rows.

Let's set up our table: (Code at www.sqlservercentral.com)

Now we'll write two queries to exercise our original question (NOT IN vs. <>). We'll start with the NOT IN query, then follow up with the AND <> query:

```
SELECT @results = count(filterCriterion_sv)

FROM tbl_IN_VS_AND

WHERE filterCriterion_sv NOT IN (214, 215, 216, 217)


SELECT @results = count(filterCriterion_sv)

FROM tbl_IN_VS_AND

WHERE filterCriterion_sv <> 214
```

```
AND filterCriterion_sv <> 215

AND filterCriterion_sv <> 216

AND filterCriterion_sv <> 217
```

The NOT IN() certainly wins for conciseness. Before running a lot of iterations, let's look at snippets of the query plan text so we know SQL is doing the same thing under the covers for each query.

NOT IN():

```
      |--Compute Scalar(DEFINE:([Expr1003]=CONVERT_IMPLICIT(int,[globalagg1005],0)))
      |--Stream Aggregate(DEFINE:([globalagg1005]=SUM([partialagg1004])))
      |--Parallelism(Gather Streams)
      |--Stream Aggregate(DEFINE:([partialagg1004]=Count(*)))
      |--Clustered Index

Scan(OBJECT:([master].[dbo].[tbl_IN_VS_AND].[PK__tbl_IN_VS_AND__3B4BBA2E]),
WHERE:([master].[dbo].[tbl_IN_VS_AND].[filterCriterion_sv]<>(214) AND
[master].[dbo].[tbl_IN_VS_AND].[filterCriterion_sv]<>(215) AND [
```

AND <>:

```
      |--Compute Scalar(DEFINE:([Expr1003]=CONVERT_IMPLICIT(int,[globalagg1005],0)))
      |--Stream Aggregate(DEFINE:([globalagg1005]=SUM([partialagg1004])))
      |--Parallelism(Gather Streams)
      |--Stream Aggregate(DEFINE:([partialagg1004]=Count(*)))
      |--Clustered Index

Scan(OBJECT:([master].[dbo].[tbl_IN_VS_AND].[PK__tbl_IN_VS_AND__3B4BBA2E]),
WHERE:([master].[dbo].[tbl_IN_VS_AND].[filterCriterion_sv]<>(214) AND
[master].[dbo].[tbl_IN_VS_AND].[filterCriterion_sv]<>(215) AND [
```

It turns out that SQL likes the AND <> so much, it converts the NOT IN() clause into a series of AND <> clauses. After 100 executions of each query, it seems that execution times for the AND <> query tend to be lower than those for the NOT IN query (conversion overhead, maybe?):

Beginning first test run...

"NOT IN" ET: 46170 ms

Beginning second test run...

"AND <>" ET: 42326 ms

I ran the same series of tests on another occasion and the NOT IN query consistently outperformed the AND <> query. The results regularly go back and forth, much like the heads and tails of a coin toss. So, I have managed to convince myself that, despite the two execution times listed above, these two queries are, indeed, the same query as far as SQL Server is concerned -- at least on this day, on this server, for these queries (I still gravitate toward the comforting ambiguity of "it depends").

I mentioned earlier that there were several queries that outperform our basic AND <> and NOT IN queries (on this server on this day). Let's take a look at some of those queries and their execution results. The first alternative technique doesn't use a WHERE clause to filter out our integer values. It places the integer values into a UNION query and does a LEFT OUTER JOIN against that to filter out unequal rows. Here is what that query looks like:

```
SELECT @results = count(filterCriterion_sv)

FROM tbl_IN_VS_AND

LEFT OUTER JOIN (

    SELECT 214 AS filterValue_val UNION

    SELECT 215 UNION

    SELECT 216 UNION

    SELECT 217 ) AS tbl

ON tbl_IN_VS_AND.filterCriterion_sv = tbl.filterValue_val

WHERE tbl.filterValue_val IS NULL
```

It definitely feels odd, placing things you would normally put in a WHERE clause into a derived table then looking for absent values, but the performance benefit gives us a compelling reason to consider doing this. On this test run of 100 executions, this odd query was consistently outperforming the quicker of our original two queries by about 19%:

```
Beginning fourth test run...

"derived UNION table LEFT OUTER JOIN" ET: 34360 ms
```

Our last query happened to be our best performing (on this server, on this day). Like the previous query, this one uses a derived table. However, it takes it one step further and nests that inside an IF NOT EXISTS(). Let's take a look at it:

```
SELECT @results = count(filterCriterion_sv)

FROM tbl_IN_VS_AND

WHERE NOT EXISTS(SELECT * FROM

    (

        SELECT 214 AS filterValue_val UNION ALL

        SELECT 215 UNION ALL

        SELECT 216 UNION ALL

        SELECT 217 ) AS tbl

    WHERE tbl.filterValue_val = tbl_IN_VS_AND.filterCriterion_sv )
```

And here is the time it took for 100 executions:

```
Beginning seventh test run...
```

```
"NOT EXISTS from derived UNION table" ET: 27920 ms
```

On this day, on this server, this query runs a full 35% faster than the quicker of our two original queries. Are there even faster ways to run this query? I would say the odds are pretty good that there are. However, I must admit that we have exhausted the current depth of my knowledge and experience, and that was only thanks to the derived table and NOT EXISTS techniques I discovered in some old newsgroup postings ^[2]^[3]. I hope you'll be able to successfully adapt these techniques to your environment and always remember to do a little experimenting and draw performance conclusions based on testing with your servers and your data.

Full Control Over a Randomly Generated Password

By Peter Larsson

Every DBA needs to generate passwords for various purposes and here is a stored procedure you can use to generate customized passwords. It is very easy to change to include more character groups or lessen the present characters in a group. The default is 4 groups of characters; Upper case, Lower case, Numbers and Special characters.

You call the stored procedure with the number of characters you want from each group with a parameter. There is an extra feature built-in the code! If you do not want duplicate characters from a group but still want three characters from Numbers, use a negative parameters with the value of -3.

Full control over a customized random-generated password seems a contradiction, but it is not. With this stored procedure you have full control over the creation process with no effort!

Let us go through how this procedure works. First of all, create the stored procedure header.

```
CREATE PROCEDURE dbo.uspCreatePassword
(
    @UpperCaseItems SMALLINT,
    @LowerCaseItems SMALLINT,
    @NumberItems SMALLINT,
    @SpecialItems SMALLINT
)
AS
```

The stored procedure accepts four parameters, one for each group of significant characters. In this stored procedure I have included UpperCase items, LowerCase items, Number items and Special items. You can, if you want to, change these groups to other items to better fit your purposes.

The reason these parameters are SMALLINTs is that you can pass the value 2, as well as -2. If you pass a positive value of 2, you will get 2 characters from that group. You will have no control over if duplicate characters are output, such as double E. If you do not want duplicate characters from a group, pass a negative value such as -2. Then a double E combination is not possible.

Next, we tell SQL Server to not output the number of affected rows

```
SET NOCOUNT ON
```

Now we need some local variables. We need four variables to hold the groups of significant characters to use. We also need to declare a temporary variable to hold the selected characters from each group as well as a loop counter (@i) and a character variable (@c) and a position variable (@v).

```
DECLARE @UpperCase VARCHAR(26),  
  
        @LowerCase VARCHAR(26),  
  
        @Numbers VARCHAR(10),  
  
        @Special VARCHAR(13),  
  
        @Temp VARCHAR(8000),  
  
        @Password VARCHAR(8000),  
  
        @i SMALLINT,  
  
        @c VARCHAR(1),  
  
        @v TINYINT
```

With these variables set, we now need to set the default characters for each group. We also initialize the @Temp and @Password variable to an empty string.

```
-- Set the default items in each group of characters  
  
SELECT @UpperCase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',  
  
        @LowerCase = 'abcdefghijklmnopqrstuvwxyz',  
  
        @Numbers = '0123456789',  
  
        @Special = '!@#$$%&*()_+-=',  
  
        @Temp = '',  
  
        @Password = ''
```

If, for some reason, the wanted number of characters for a group is set to a very high number, limit the number to a maximum.

```
-- Enforce some limits on the length of the password  
  
IF @UpperCaseItems > 20  
  
    SET @UpperCaseItems = 20  
  
  
IF @LowerCaseItems > 20  
  
    SET @LowerCaseItems = 20
```

```
IF @NumberItems > 20  
  
    SET @NumberItems = 20
```

```
IF @SpecialItems > 20  
  
    SET @SpecialItems = 20
```

We need to do the selection for UpperCase items. Since the parameter can be either positive or negative, set the loop counter to the absolute value of characters wanted from that group.

```
-- Get the Upper Case Items  
  
SET @i = ABS(@UpperCaseItems)
```

As long as the loop counter is greater than zero, loop. But only as long as there are characters left to choose from!

This second condition is only needed for negative values, because this insures that no duplicate characters can be chosen.

```
WHILE @i > 0 AND LEN(@UpperCase) > 0  
  
    SELECT @v = ABS(CAST(CAST(NEWID() AS BINARY(16)) AS BIGINT)) % LEN(@UpperCase) + 1,  
  
        @c = SUBSTRING(@UpperCase, @v, 1),  
  
        @UpperCase = CASE WHEN @UpperCaseItems < 0 THEN  
  
STUFF(@UpperCase, @v, 1, '') ELSE @UpperCase END,  
  
        @Temp = @Temp + @c,  
  
        @i = @i - 1
```

The first SELECT gets a random number with the help of NEWID() function. A NEWID() value returns a signed 128-bit binary number, formatted with hexadecimal numbers. We need to CAST that value to binary and then CAST again to BIGINT in order to do calculations with it.

What we do next is to take the absolute value with ABS function and take the modula value with the MOD function denoted with % in SQL Server! The reason for taking the modula value for the random number is that it returns a value between 0 and the length of the numbers available in that group. And we add 1 to that value since strings of characters in SQL Server begins with position 1. The second SELECT is to get the character selected.

The third SELECT is the trick! If a negative number is passed to the stored procedure, we now need to discard the selected character with the STUFF function, so that the character can not be chosen again.

The fourth SELECT simply concatenates the @Temp variable with the chosen character. The fifth and last SELECT is to decrease the loop counter.

For the next group of characters, Lower Case items, do exactly as with the Upper Case items.

```
-- Get the Lower Case Items  
  
SET @i = ABS(@LowerCaseItems)
```

```
WHILE @i > 0 AND LEN(@LowerCase) > 0

    SELECT @v = ABS(CAST(CAST(NEWID() AS BINARY(16)) AS BIGINT)) % LEN(@LowerCase) + 1,

        @c = SUBSTRING(@LowerCase, @v, 1),

        @LowerCase = CASE WHEN @LowerCaseItems < 0 THEN STUFF(@LowerCase, @v, 1, '') ELSE @LowerCase
    END,

    @Temp = @Temp + @c,

    @i = @i - 1
```

For the next group of characters, Number items, do exactly as with the Upper Case items.

```
-- Get the Number Items

SET @i = ABS(@NumberItems)

WHILE @i > 0 AND LEN(@Numbers) > 0

    SELECT @v = ABS(CAST(CAST(NEWID() AS BINARY(16)) AS BIGINT)) % LEN(@Numbers) + 1,

        @c = SUBSTRING(@Numbers, @v, 1),

        @Numbers = CASE WHEN @NumberItems < 0 THEN STUFF(@Numbers, @v, 1, '') ELSE @Numbers END,

        @Temp = @Temp + @c,

        @i = @i - 1
```

For the next group of characters, Special items, do exactly as with the Upper Case items.

```
-- Get the Special Items

SET @i = ABS(@SpecialItems)

WHILE @i > 0 AND LEN(@Special) > 0

    SELECT @v = ABS(CAST(CAST(NEWID() AS BINARY(16)) AS BIGINT)) % LEN(@Special) + 1,

        @c = SUBSTRING(@Special, @v, 1),

        @Special = CASE WHEN @SpecialItems < 0 THEN STUFF(@Special, @v, 1, '') ELSE @Special END,

        @Temp = @Temp + @c,

        @i = @i - 1
```

The drawback right now with this algorithm is that all the Upper Case items are first, Lower Case items are second, Number items are third and Special items are fourth.

Now we need to reposition these chosen characters. This is done with the code above. First get a random character from the @Temp string and add that to the @Password string. Next, remove the selected character and repeat the process until there are no more characters to reposition!

```
-- Scramble the order of the selected items

WHILE LEN(@Temp) > 0

    SELECT @v = ABS(CAST(CAST(NEWID() AS BINARY(16)) AS BIGINT)) % LEN(@Temp) + 1,

    @Password = @Password + SUBSTRING(@Temp, @v, 1),

    @Temp = STUFF(@Temp, @v, 1, '')
```

The last thing to do, is to output the randomly generated password

```
SELECT @Password
```

The complete stored procedure is available at www.sqlservercentral.com

Performance Effects of NOCOUNT

By David Poole

Although I have known that there was a performance benefit in using SET NOCOUNT ON and qualifying objects with their owners I had never actually established benchmarks against either of these two practices. SET NOCOUNT ON gives a performance boost to action queries by suppressing the "(n row(s) affected)" message that results from running a query.

Qualifying an object with its owner boosts performance because SQL does not have to work out where if there is a user specific version of the same object. It also gives benefits in the caching of execution plans. For more details on the affects of [not qualifying and object with its owner](#), see Chris Hedgate's article from the "Worst Practices" series.

Andy Warren and Steve Jones also contributed articles and I recommend reading all the articles in the series.

- [Encrypting Data](#)
- [Making on the fly changes](#)
- [Not using Primary Keys and Clustered Indexes](#)
- [Blank passwords](#)
- [Assigning user rights](#)
- [Depending on the GUI](#)
- [Connection Strings and SysProcesses](#)
- [Sorting by ordinal](#)
- [Spaces in object names](#)
- [Case sensitivity](#)
- [Bad comments](#)

This brief article describes my experiments with each of these two settings.

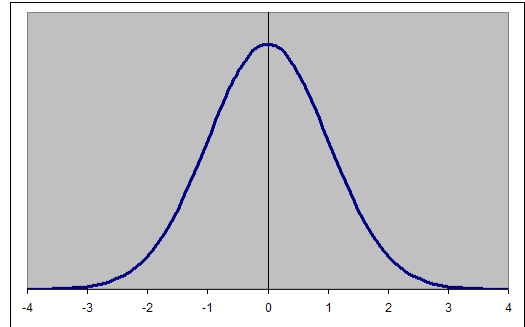
Lies, damn lies and statistics

The ability to provide an experiment that can be repeated and produce similar results is the corner stone to a good scientific method.

As there are many criteria for measuring SQL performance and each have their positive and negative points I am including my methodology to enable you to test my results in your own environment. One thing you will observe is that any set of benchmarks should not be taken at face value and must be tested in your own particular environment to determine their relevance to your set up.

When I first tried to create a set of benchmarks against SQL Server, not just for this article, I assumed that repeating a particular action would result in times for those actions would fall in the classic bell shaped curve with the average time forming the top of the bell. This is known as the central limit theorem.

I quickly found that this wasn't the case and that although the performance generally fell within a range of values there appeared to be no particular pattern to the results.



Methodology

My experiment was carried out on a stand-alone server using SQL2000. There was no network connectivity to that server and the experiments were carried out using SQL Management Studio. My first task was to set up two tables with separate stored procedures to populate them as follows:

```
CREATE TABLE dbo.TestTable(  
  
    TestId int IDENTITY(-2147483648,1) NOT  
    NULL  
  
    CONSTRAINT PK_TestTable  
    PRIMARY KEY CLUSTERED,  
  
    TestDesc varchar(50) NOT NULL  
  
)  
  
GO  
  
CREATE PROC dbo.AddTestWithNodbo  
  
    @TestDesc VARCHAR(50)  
  
AS  
  
SET NOCOUNT ON  
  
INSERT INTO TestTable(TestDesc)  
  
VALUES (@TestDesc)
```

```
CREATE TABLE dbo.TestTable2(  
  
    TestId int IDENTITY(-2147483648,1) NOT  
    NULL  
  
    CONSTRAINT PK_TestTable2  
    PRIMARY KEY CLUSTERED,  
  
    TestDesc varchar(50) NOT NULL  
  
)  
  
GO  
  
CREATE PROC dbo.AddTestWithdbo  
  
    @TestDesc VARCHAR(50)  
  
AS  
  
INSERT INTO dbo.TestTable2(TestDesc)  
  
VALUES (@TestDesc)
```

```
RETURN @@ROWCOUNT
```

```
GO
```

```
CREATE PROC dbo.AddTestWithNoCount
```

```
    @TestDesc VARCHAR(50)
```

```
AS
```

```
SET NOCOUNT ON
```

```
INSERT INTO TestTable(TestDesc)
```

```
VALUES (@TestDesc)
```

```
RETURN @@ROWCOUNT
```

```
GO
```

```
RETURN @@ROWCOUNT
```

```
GO
```

```
CREATE PROC dbo.AddTestWithoutNoCount
```

```
    @TestDesc VARCHAR(50)
```

```
AS
```

```
INSERT INTO dbo.TestTable2(TestDesc)
```

```
VALUES (@TestDesc)
```

```
RETURN @@ROWCOUNT
```

```
GO
```

As you can see the procedures are virtually identical differing only in whether they use SET NOCOUNT ON or whether they qualify objects with their owners.

All generate identical execution plans.

My next task was to set up a script that would execute a fixed number of iterations of each stored procedure and measure the time taken to carry out those iterations.

```
DBCC DROPCLEANBUFFERS
```

```
DBCC FREEPROCCACHE
```

```
TRUNCATE TABLE dbo.TestTable
```

```
TRUNCATE TABLE dbo.TestTable2
```

```
GO
```

```
DECLARE @StartTime1 DATETIME ,
```

```
        @EndTime1 DATETIME
```

```
@Loop INT ,
```

```
@MaxLoop INT
```

```
SET @Loop=0
```

```
SET @MaxLoop=10000
```

```
SET @StartTime1=GETDATE()
```

```
WHILE @Loop<@MaxLoop
```

```
BEGIN

        exec dbo.AddTestWithNoCount 'AAAA'

        SET @Loop=@Loop+1

END

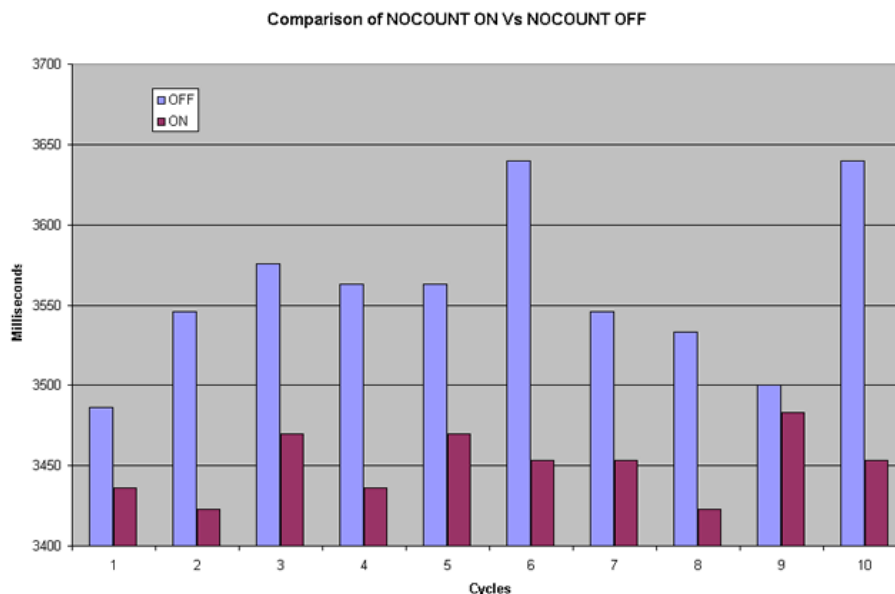
SET @EndTime=GETDATE()

SELECT DATEDIFF(ms,@StartTime,@EndTime) AS TestEnd
```

I ran the script above 10 times recording the execution time in milliseconds each time. I then replaced `exec dbo.AddTestWithNoCount 'AAAA'` with `exec dbo.AddTestWithoutNoCount 'AAAA'` and reran the script a further 10 times.

The results are shown in the graph below. As you can see SET NOCOUNT OFF is consistently slower than SET NOCOUNT ON. All in all results for this experiment reveal a 3% performance advantage in having SET NOCOUNT ON.

I repeated the experiment several times, sometimes starting with iterations of `exec dbo.AddTestWithoutNoCount 'AAAA'`, sometimes with iterations of `exec dbo.AddTestWithNoCount 'AAAA'` to ensure that I was not introducing bias into the experiment. Overall the performance advantage remained around the 3% margin.



Points to consider

The performance boost is due to the few bytes of information that make up the "(1 row(s) affected)" message not being transmitted to the client application.

With this in mind I should consider the following points

- Communication between the database and the client application on a stand-alone machine will be as fast as it is possible to get. If your front end application had it's own clock and you recorded the time from submitting the

query to the time when the client finally received the full results I would expect that transmitting results across a network to be slower.

- In this experiment we are carrying out a single simple insert statement. If your procedure carries out multiple operations the performance boost will be more pronounced.
- For queries that retrieve data the performance boost will be less simply because the size of the "(1 row(s) affected)" message is small compared to the volume of data being returned.
- In .NET applications an ExecuteNonQuery command returns the number of records affected by the operation. Set NOCOUNT ON means that the value that this call returns is always zero.

Qualified objects vs non-qualified objects

For my experiment to compare qualified and non-qualified objects I used the same methodology that I used for my SET NOCOUNT experiment however I explicitly logged onto my server with a non-dbo user with rights to execute both stored procedures.

When calling the dbo.AddTestWithNodbo stored procedure I deliberately ran
exec AddTestWithNodbo
rather than
exec dbo.AddTestWithNodbo.

My initial results showed a performance improvement of 0.3% when qualifying an object with an owner. Given the non-committal results I reran the experiment several times. In some cases the difference in performance was virtually nothing, in others there was a slight bias towards qualified objects and once the unqualified procedure ran slightly faster.

In short my personal view is that qualifying objects with their owner is something that should be done for reasons of being explicit in what you are asking SQL to do rather than for performance reasons.

Conclusions

When considering qualifying objects with their owners the main point to consider is how many different logins access objects in your database. In my experiment I had a single non-dbo user, but a more realistic environment may have multiple users.

Chris's article mentions benefits gained by accessing cached execution plans. With multiple users hitting the system I suspect that the benefits of this would be more pronounced.

In my SET NOCOUNT experiment the difference in performance between SET NOCOUNT ON/OFF over 10,000 iterations was measurable in milliseconds rather than seconds. If your system is not busy then this is small beer however in highly stressed systems every millisecond consumed by one process is a resource denied to another. As I said earlier, my results were gained by measuring multiple iterations of a single operation. The performance gain on more complicated procedures may be more pronounced.

Passing a Table to A Stored Procedure

By Jacob Sebastian

Introduction

Most of us would be very specific in designing the database code (Stored procedures, functions, views etc) in a reusable and manageable manner. It is particularly important when the application is large. Code for common functionalities should be identified and moved to a function that can be called from different parts of the application.

The same may be done with views, stored procedures etc. Designing the code in such a manner increases the manageability of the code as well as provides greater re-usability, and thus better productivity and lesser bugs.

Some times, while attempting to achieve the above, we would come across certain hurdles due to the limitations of TSQL. At times we feel that TSQL does not really give us enough freedom like other application development platforms. In this article, I am trying to present such a case where a re-usable function is created to which a table can be passed as an argument.

The Problem

Let us say, we are working on an Inventory Management System. When a transaction (sales order, invoice, receipt of goods, inventory adjustment etc) takes place, we need to update the available inventory of the items affected by the transaction. We already have a stored procedure to save/update each transaction. Each of those stored procedures needs to update the inventory of all the items affected by the current transaction.

Please note that, the word 'Transaction' above, does not refer to Database Transactions. They refer to the various Inventory Operations supported by the application.

Since the inventory needs to be updated from different places, it makes sense to move that part of the code to a separate stored procedure. Then this new stored procedure needs to be called from different places from where the inventory is to be updated. So far it looks simple. But the difficult part is to pass the items to be updated.

A TABLE variable would look to be the ideal solution. If we could pass a TABLE variable containing the list of items to be updated, then the complexity can be reduced to a great extent. But SQL Server does not allow to pass a TABLE variable as a parameter to a stored procedure. So what is the next option?

In this article, I am trying to present a solution to the above scenario by using XML as the format to pass a table to a stored procedure. The CALLER can transform the table (Query result) to an XML variable and pass to the stored procedure. The CALLEE can either convert the XML parameter back to a TABLE variable or directly use XQuery on the XML variable.

The Caller

The CALLER should transform the table to an XML variable. The DATA may come from a table or a query. The [following example](#) shows how to create an XML variable from the results of a query.

```
1  /*
2     Let us first create sample table.
3  */
4
5  CREATE TABLE [dbo].[OrderDetails](
6      [OrderDetailID] [int] IDENTITY(1,1) NOT NULL,
7      [ItemNumber] [varchar](20) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
8      [Qty] [int] NULL
9  ) ON [PRIMARY]
10
11 /*
12     Populate the sample table with values
13 */
14 INSERT INTO OrderDetails(ItemNumber, Qty)
15     SELECT 'A001', 10
16     UNION SELECT 'A002', 20
17     UNION SELECT 'A003', 30
18 /*
19     The query below returns the results in XML format.
20 */
21
22 SELECT ItemNumber, Qty FROM OrderDetails FOR XML RAW('item'), ROOT('items')
23
```

```
24 /*
25 OUTPUT:
26
27 <items>
28   <item ItemNumber="A001" Qty="10" />
29   <item ItemNumber="A002" Qty="20" />
30   <item ItemNumber="A003" Qty="30" />
31 </items>
32 */
```

In the article [Advanced XML Processing - II](#), I have presented a few detailed examples which demonstrate the different ways to generate and format query results as XML.

Now, let us assign the resultant XML value to an XML variable. [\[code\]](#)

```
1 -- Declare the variable
2 DECLARE @x XML
3
4 -- store the results of the Query to XML variable
5 SET @x = (SELECT ItemNumber, Qty FROM OrderDetails FOR XML RAW('item'), ROOT('items'),
TYPE)
6
7 -- select the values from the XML variable (to make sure that we did it correctly)
8 SELECT
9     x.item.value('@ItemNumber[1]', 'VARCHAR(20)') AS ItemNumber,
10    x.item.value('@Qty[1]', 'INT') AS Qty
11 FROM @x.nodes('//items/item') AS x(item)
```

At this stage, we have an XML variable ready, which we could pass to a child procedure/function. The XML variable contains the values that we want the child procedure/function to process/update. The child procedure can either transform the XML variable back to a TABLE or it can directly read the values from the XML variable.

The Callee

So far, we have seen how to create an XML variable from the results of a query. This XML variable can be passed to another stored procedure which can update the inventory data based on the item information passed to the procedure. The simplest way is to create a wrapper view around the XML variable and use it as if it is a table.

Let us create another sample table, *Inventory*, which will be updated with the information passed through the XML parameter. The following [script](#) will create the sample table.

```
1 CREATE TABLE [dbo].[Inventory](
2     [InventoryID] [int] IDENTITY(1,1) NOT NULL,
3     [ItemNumber] [varchar](20) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
4     [Stock] [int] NULL
5 ) ON [PRIMARY]
6
7 INSERT INTO Inventory (ItemNumber, Stock)
8     SELECT 'A001', 0
9     UNION SELECT 'A002', 0
10    UNION SELECT 'A003', 0
```

The following [sample code](#) shows the implementation needed at the side of the 'callee'.

```
1 CREATE PROCEDURE [dbo].[UpdateInventory1]
2 (
3     @x XML
4 )
5 AS
6
7 SET NOCOUNT ON
```

```
8
9  /*
10   The code below creates a wrapper view around the XML variable and updates the
11   "inventory" table with the information.
12  */
13
14  UPDATE Inventory SET
15     stock = stock + v.Qty
16  FROM Inventory inv
17  INNER JOIN (
18     SELECT
19         x.item.value('@ItemNumber[1]', 'varchar(20)') AS ItemNumber,
20         x.item.value('@Qty[1]', 'INT') AS Qty
21     FROM @x.nodes('//items/item') AS x(item)
22 ) v ON (v.ItemNumber = inv.ItemNumber)
23
24  RETURN
```

Execute

Let us execute the procedure now. Run the following [code](#).

```
1  -- Declare the variable
2  DECLARE @x XML
3
4  -- store the results of the Query to XML variable
5  SET @x = (SELECT ItemNumber, Qty FROM OrderDetails FOR XML RAW('item'), ROOT('items'),
TYPE)
6
7  -- execute the stored procedure
8  EXECUTE UpdateInventory1 @x
9
10 -- review the results
11 SELECT * FROM inventory
```

Updated Procedure

The [sample code](#) above, creates a wrapper view around the XML variable. This is a pretty simple and straightforward approach. You could still access the values as if it is coming from a table/view. The complexity of XML processing is absorbed in the inner view. The [example](#) below, demonstrates another syntax, which updates the table directly from the XML variable.

```
1  CREATE PROCEDURE [dbo].[UpdateInventory2]
2  (
3     @x XML
4  )
5  AS
6
7  SET NOCOUNT ON
8
9  /*
10   This version of the stored procedure has a slightly enhanced version of the
11   TSQL code. This version updates the table directly from the XML variable,
12   rather than converting the XML data to a view.
13  */
14
15  UPDATE Inventory SET
16     stock = stock + x.item.value('@Qty[1]', 'INT')
17  FROM Inventory inv
18  INNER JOIN @x.nodes('//items/item') x(item) ON
19     (x.item.value('@ItemNumber[1]', 'varchar(20)') = inv.ItemNumber)
20
21  RETURN
```

Conclusions

In the past few years, several times I came across the situation where I needed a way to pass a table to a function or stored procedure. Since SQL Server does not allow to pass a TABLE variable to a function or stored procedure, the only way I could make it work is by using the approach presented above. There may be other ways to get this done too. It is apparent that there will be a small performance penalty by doing this. I did not do extensive tests to see if there is a performance problem. I did not notice any performance issues in my applications so far.

New Column Updates

By Bimal Fernando

What is the fastest way to update a newly added column in to a very large table?

Recently I had to add a new column to a table which has more than 35 million very wide (1000+ bytes per row) rows and populate with data. Using the recommended method by various people the best speed I could get was 20 hours. I will first describe the failed attempt and then I will explain the method I came up with to increase the speed of this operation significantly.

Table Schema:

```
CREATE TABLE [dbo].[LargeTable](
    [ID] [int] IDENTITY(1,1) NOT NULL,
    [InvID] [decimal](11, 0) NOT NULL,
    [JNumber] [tinyint] NOT NULL,
    [JType] [nchar](1) NOT NULL,
    [NameKey] [nvarchar](341) NOT NULL,
    [SystemID] [decimal](12, 0) NOT NULL,
    [BDate] [datetime] NOT NULL,
    [EDate] [datetime] NOT NULL,
    [DateTimeModified] [datetime] NOT NULL,
    [RowTimestamp] [timestamp] NULL,
    [Designation] [nvarchar](200) NOT NULL,
    [Capacity] [nvarchar](20) NOT NULL,
    [Agreement] [nvarchar](50) NOT NULL,
    [DateQ] [nvarchar](10) NOT NULL,
    CONSTRAINT [LargeTable__PK] PRIMARY KEY NONCLUSTERED
(
    [ID] ASC
```

```
)ON [PRIMARY]

) ON [PRIMARY]
```

```
CREATE CLUSTERED INDEX [LargeTable__IE1] ON [dbo].[LargeTable]

(

    [NameKey] ASC

)ON [PRIMARY]
```

Note that the clustered index is on a very wide column.

The recommended method to add a new column and update it is as follows. There was nothing wrong with the method to add a new column to such a large table.

Step 1: Add new column

```
alter table MyTable

add NewZip char(10) NULL
```

Step 2: Set Batch Size for update

```
--- (2) set piecemeal batch size

set nocount on

SET ROWCOUNT 1000
```

Next, the piecemeal update is repeatedly done until there are no NULL values in NewZip. This example sets all rows using the char(5) ZipCode column. Alternately, you can selectively set the value of an added column using a lookup or join. Step 3: De-coupled Piecemeal Update

```
declare @rowcount int, @batchcount int, @rowsupdated int

select @rowcount = 1, @batchcount = 0, @rowsupdated = 0

while @rowcount > 0

begin

    begin tran

        update MyTable

        set NewZip = ZipCode

        where NewZip is NULL

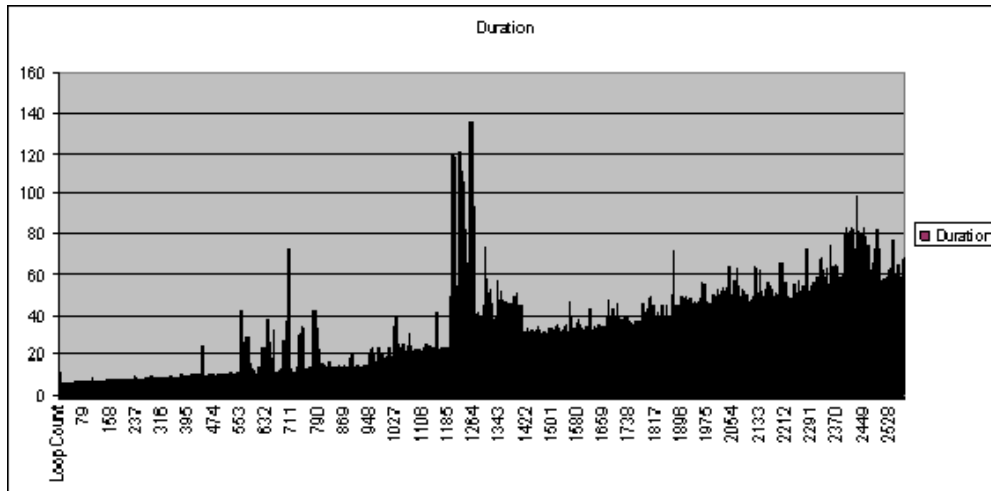
        set @rowcount = @@rowcount

        select @rowsupdated = @rowsupdated + @rowcount

        if @rowcount > 0 select @batchcount = @batchcount + 1
```

```
commit  
  
end
```

When I followed the above method I realized that the performance is degrading exponentially with the iteration count. See below graph for the actual statistics during my test load. Duration is in seconds and the batch size was 10,000.



As you can clearly see when it comes to iteration 2000+ the time to update 10,000 records has gone up to 80 seconds and with this behavior it took more than 20 hours for me to update my 35 million records table.

The reason for this is the cost to find the next 10,000 to update and it becomes costlier with each iteration because of lesser and lesser availability of records where the NewColumn is null and they are scattered everywhere in the clustered index. You would think why not we then add an index on this new column so that its easier to locate the null records, but it doesnt help because it still has to do a costly bookmark lookup to find the actual data page and it even has to update this new index with each update.

OK, so what's the solution to this issue? We know that we cannot do this update in one shot because that'll fill up the log file and if something goes wrong then we need to do it from the beginning again. Thats why we still need to do this in batch by batch basis.

Then I realized that actually SQL server has to load the data pages (Clustered index) to update this new column anyway and if I can find a way to go through the same order as the clustered index key column(s) then it would be much faster because then SQL server optimizer can easily predict the next set of pages (read-ahead) because they are physically ordered in the clustered index key order. And for that reason it will not have to jump everywhere in the clustered index to find the next set of records to be updated since we are doing it in the same order as they are stored physically.

If I had a clustered index on an IDENTITY column my life would have been much easier because then I can iterate thorough it from start to finish advancing the loop by the batch number.

But since my clustered index is an Alpha Name column which contains values from the ASCII table (<http://www.neurophys.wisc.edu/comp/docs/ascii.html>) I had to use a different approach. I had to find a way to iterate though the Clustered index from A to Z but in batches. My batch count will not be consistent with this approach but if you smart enough you can break the ASCII table in to several ranges to break the clustered index key values in to manageable batches. Dont think you can do that easily with some simple ranges like A-B, B-C; This is a huge table and A to B range it self may contain millions of records. Thats why I had to do something extra to break in to smaller batches.

To achieve this I did the following trick.

1) Create a table to hold the different Range keys from the ASCII table.

```
create table _CLUSTKEY (ID int Identity(1,1) primary Key, Key1 nvarchar(2), Key2 nvarchar(2))
```

2) Generate the ASCII key ranges and insert them in to the above created table. (If its difficult to understand the algorithm, first run this and then look at the results. Then you will be able to understand the logic easily)

```
declare @var as char(1), @ascii as int, @count as int

set @ascii = 65

set @count = 0

while(@ascii<90)

begin

    insert into _CLUSTKEY

    select char(@ascii) + char(ascii('A')+@count) , char(@ascii) + char(ascii('A')+@count+1)

    if (char(ascii('A')+@count+1) = 'Z')

    begin

        insert into _CLUSTKEY

        select char(@ascii) + char(ascii('A')+@count+1) , char(@ascii+1) + char(ascii('A'))

        set @ascii = @ascii + 1

        set @count = 0

    end else set @count = @count + 1

end
```

3) Insert beginning and end ranges to the above table manually

```
insert into _CLUSTKEY
```

```
select '', 'AA'
```

```
insert into _CLUSTKEY
```

```
select 'ZA', CHAR(ASCII('Z')+1) Your end criteria may be different to this depending on the data you have on your clustered key column.
```

Once populated the above table will look like this:

ID	Key1	Key2
----	------	------

1	AA	AB
2	AB	AC
3	AC	AD
...		.
...		
650	YZ	ZA
651		AA
652	ZA	[

4) Add the new column to my very large table

```
Alter table LargeTable Add AlphaName_CS int null
```

5) Start the update in batches (Batch boundaries are defined by the table I have created and populated above)

```
-- Run this Script to create a table to record the progress so that you can see how far you are  
from the completion
```

```
create table __progress(RecCount int, LoopCount int, DateT datetime)
```

```
-- Actual Update Script
```

```
declare @Key1 as nvarchar(2), @Key2 as nvarchar(2),@count as int, @rowcount as int, @MaxLoopCount  
as int
```

```
select @MaxLoopCount = max(ID) from _CLUSTKEY
```

```
set @count = 1
```

```
while(@count<=@MaxLoopCount)
```

```
begin
```

```
    -- Get the start Key and End key value for the current batch in to variables
```

```
    select @Key1 = Key1, @Key2 = Key2
```

```
    from _CLUSTKEY
```

```
    where ID = @count
```

```
    -- Do the update for the above Key range
```

```
    update dbo.LargeTable
```



```
set AlphaName_CS = CHECKSUM(NameKey)

where NameKey >= @Key1 and NameKey < @Key2

select @rowcount = @@rowcount

-- insert a record to this progress table to check the progress

insert into __progress select @rowcount, @count, getdate()

set @count = @count + 1

end
```

35 million rows were updated within less than 35 minutes..!!!

The T-SQL Quiz

By Grant Fritchey

I was reading the latest [Simple-Talk](#) email that linked me to an article on [Coding Horror](#), "Why Can't Programmers - Program?" In the article they talked about a simple test that they gave developers to verify their abilities and decide whether or not to continue the interview. Here's the test: Write code that counts from 1 to 100 For each number evenly divisible by 3, substitute 'Bizz' For each number evenly divisible by 5, substitute 'Buzz' For each number divisible by both substitute 'BizzBuzz'.

I decided to try it out in TSQL. Once I had a solution (it took about five minutes, it would have been about two but I kept getting syntax errors on the CASE statement). I had so much fun that I sent it out to our team to see who else could meet the requirements and I added one more: no cursors are allowed.

Before you read any further, try it out for yourself. It's not hard. A solution should take you less than 10 minutes.

Here are the various solutions that we came up with on our team.

First up, mine:

```
DECLARE @i VARCHAR(3) ;

DECLARE @s VARCHAR(8) ;

SET @i = 1 ;

WHILE @i < 101

BEGIN

    SELECT  @s = CASE WHEN ( ( @i % 3 = 0 )

                        AND ( @i % 5 = 0 )
```

```
        ) THEN 'BizzBuzz'

    WHEN ( @i % 3 = 0 ) THEN 'Bizz'

    WHEN ( @i % 5 = 0 ) THEN 'Buzz'

    ELSE @i

END ;

PRINT @s ;

SET @i = @i + 1 ;

END ;
```

I didn't need to use the @s variable for the print statements, but overall, first pass, it was simple and worked fine. I'm assuming I'd still get interviewed although my pay scale may have dropped a grade because I introduced parameters that weren't really necessary.

The next submission came in from Chris:

```
declare @i int

set @i = 1

while( @i < 101)

begin

    if @i%3 = 0 print 'Bizz'

    else if @i%5 = 0 print 'Buzz'

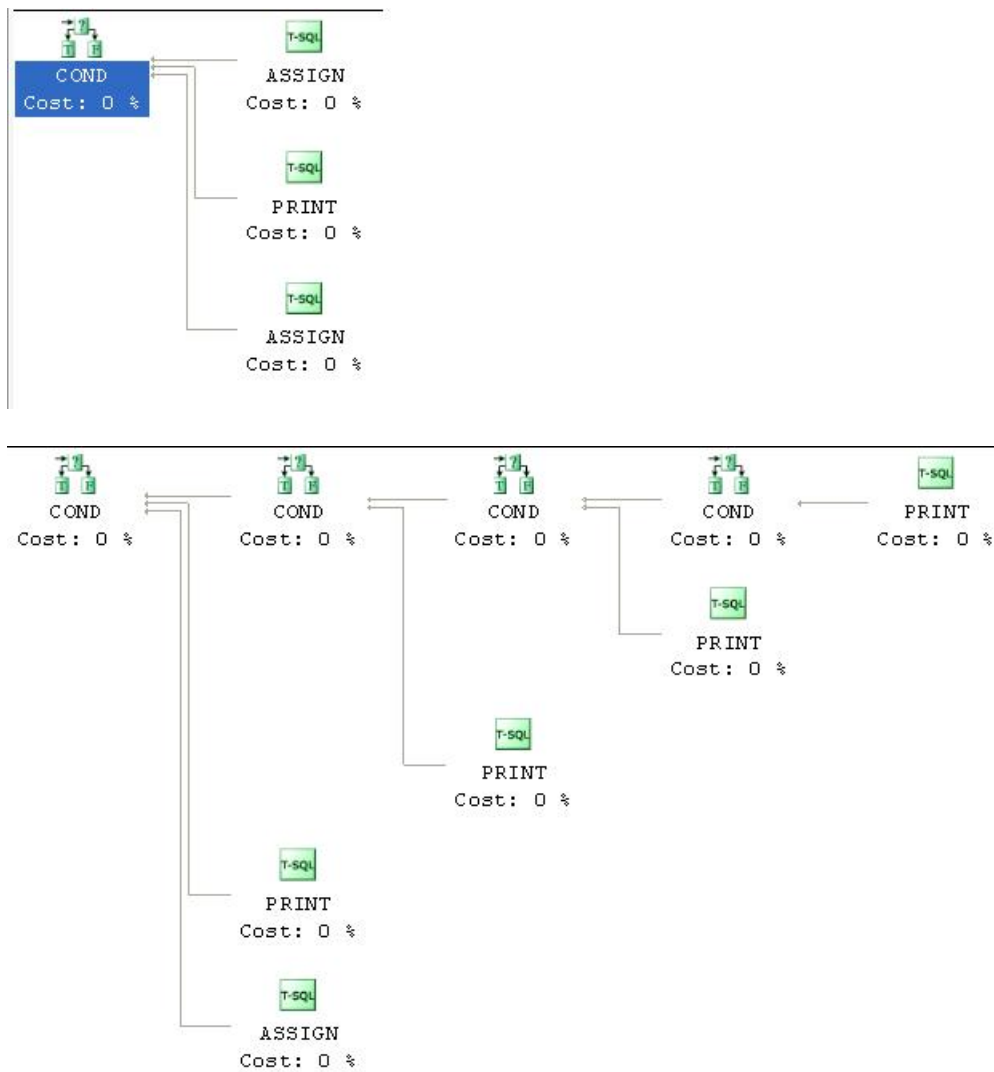
    else if @i%3 = 0 and @i%5 = 0 print 'BizzBuzz'

    else print @i

    set @i = @i+1

end
```

He fixed my problem with the string, but he introduced a new one. Go back, reread the code and see if you can spot it. Give up? Because he checked for 3 then 5 then the combination, none of his code found the combination of 3 and 5. End of interview. Sorry Chris. It's also worth mentioning that the CASE statement resolves in a single pass where as the IF statements check each condition. Take a look at the estimated query plans:



Next came one from Det:

```
CREATE TABLE Nums ( num int primary key )  
  
GO  
  
DECLARE @i int  
  
SET @i = 1  
  
WHILE @i <= 100  
  
    BEGIN  
  
        INSERT Nums ( num )  
  
        VALUES ( @i )  
  
        SET @i = @i + 1  
  
    END
```

```
END
```

```
SELECT CASE WHEN num % 3 = 0
           AND num % 5 = 0 THEN 'BizzBuzz'
        WHEN num % 3 = 0 THEN 'Bizz'
        WHEN num % 5 = 0 THEN 'Buzz'
        ELSE CAST(num AS nvarchar)
END
FROM   nums
```

Det's worked very well, but he created a permanent table and didn't include a drop statement. When I went back to look at the query plan, I got an error. We'll continue the interview, but Det shouldn't count on getting a top spot on the team. He submitted a second query as well:

```
DECLARE @i int
SET @i = 1

WHILE @i <=100
BEGIN
    SELECT CASE WHEN @i % 3 = 0 AND @i % 5 = 0 THEN 'BizzBuzz'
              WHEN @i %3 = 0 THEN 'Bizz'
              WHEN @i %5 = 0 THEN 'Buzz'
              ELSE CAST(@i AS nvarchar) END
    SET @i = @i + 1
END
```

This worked in a repeatable fashion and instead of printing output lines to the message window, it returned actual results. 100 separate result sets, but even so, it worked. Points for perseverance.

Next came Scott Abrants:

```
DECLARE @int AS INT;
DECLARE @immdResult AS VARCHAR(100);

SET @int = 1;

WHILE @int < 101
```

```
BEGIN

    SET @immdResult = CASE WHEN @int % 3 = 0

        AND @int % 5 <> 0 THEN 'Bizz'

    WHEN @int % 5 = 0

        AND @int % 3 <> 0 THEN 'Buzz'

    WHEN @int % 3 = 0

        AND @int % 5 = 0 THEN 'BizzBuzz'

    END ;

    PRINT 'The number is ' + CONVERT(VARCHAR(10), @int);

    IF LEN(@immdResult) > 0

        BEGIN

            PRINT @immdResult;

        END

    SET @int = @int + 1;

END;
```

Not at all surprisingly, Scott's is dead on accurate and extremely thorough. He reintroduced the extra parameter that I had, but instead of a crutch, he makes real use of it. At this point Scott's at the head of the class.

Freely admitted, this is a silly little test and doesn't really demonstrate much in the way of coding skill, or more especially TSQL skills. However, it spawned a lot of discussion within the team. We started running the queries to see what performance looked like. Since most of them didn't do anything approaching data manipulation, there really weren't query plans to look at. Performance was very similar to a degree:

Chris - 15ms
Grant - 31ms
Scott - 46ms
Det 1 - 62ms
Det 2 - 93ms

One of the discussions that came up was, shouldn't there be a simple way to do this with a CTE? So after lunch, I set out to try it. This is what I came up with:

```
WITH Nbrs(n) AS (

    SELECT 1

    UNION ALL

    SELECT 1 + n FROM Nbrs WHERE n < 100)

SELECT CASE WHEN n%5=0 AND n%3=0 THEN 'BizzBuzz'

    WHEN n%3 = 0 THEN 'Bizz'
```

```
        WHEN n%5 = 0 THEN 'Buzz'

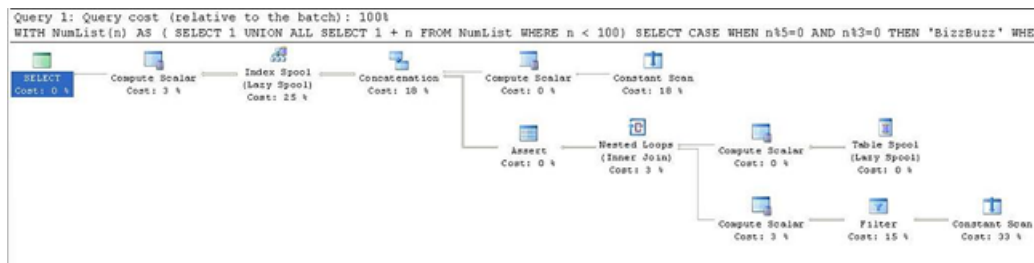
        ELSE CAST(n AS VARCHAR(8))

    END

FROM Nbrs

OPTION (MAXRECURSION 100);
```

This worked well, looks neat, and it ran in 15ms. The only problem with it was, it created one scary looking query plan:



That's it. We had some fun writing these queries and I decided to share them. The results speak for themselves. There is no great and magnificent truth revealed here. It's simply a starting point for conversation and a method of verifying a basic level of knowledge along with the ability to apply logic. There are probably much more elegant ways to do this. I looked up some methods for generating counting data that could be used for something like this or for something a heck of a lot more sophisticated like a Numbers table and found a nice summary of information [here](#). If you have a better way to skin this cat, please share it. If you're applying at my company, you might want to anticipate this as a question at the start of the interview.

Practical Methods: Naming Conventions

By Michael Lato

Naming conventions are used to streamline development when multiple developers are working on a single system, as well as to simplify ongoing maintenance during the Development Lifecycle. There are many different naming conventions for SQL Server and no single convention is necessarily right or wrong. However, a single consistent naming convention should be followed within each database to reduce confusion and to enhance usability. Ideally, follow a single naming convention throughout all SQL Servers in an organization.

Databases

- Single database applications may use any simplified name. Multiple database applications should use a prefix followed by the database category.
- Samples:
 - "Finance" for Financial Operations
 - "Operations" for Operations
 - Prefix of "HR" for Human Resources
 - "HRData" for the primary database
 - "HRImport" for data import holding tables and procedures
 - "HRExport" for data export holding tables and procedures

Backup Files

- Prefix all backups with the database name. Use an underscore and add the date and time of the backup.
- Samples:
 - Full backup: dbname_200601011800.bak
 - Differential backup: dbname_200601011800.dif
 - Transaction Log: dbname_200601011800.trn

Users and Logins

- Match all database user names to the mapped login. This simplifies security audits.
- No user accounts should be shared among logins. Use database roles to achieve continuity instead.

Tables

- Prefix all tables with "t".
- Complete the name with the primary entity stored in the table in a singular form.
- Name tables that rely on other tables in sequence using the primary table name as a starting point.
- Name many-to-many tables with both primary tables listed alphabetically. Separate the names with an underscore for clarity.
- Holding tables for temporary data should be prefixed with "temp".
- Samples:
 - tCompany, tCustomer, tProduct, tInvoice
 - tCompanyAddress, tCustomerAddress, tInvoiceDetail
 - tRole_User, tPermission_Role
 - tempCustomerBackup
- Comments:
 - The prefix used here helps ensure that no keywords are used as a table name. For example, "user" is a common desired table name that is also a keyword, but "tUser" is not.
 - Never, ever link a stored procedure to a table with the "temp" prefix. It is a virtual guarantee that this will lead to a "temp" table being used in production. Sooner or later someone will delete this "temp" table and break your production system.

Columns

- Name columns according to the information they contain. Primary keys should use the table name plus the suffix "ID".
- Samples:
 - CustomerID, CustomerName, CustomerNumber, Address, City, Country
 - FirstName, LastName, Phone
 - CreateOn, CreateBy, EditOn, EditBy, DeleteOn, DeleteBy
- Comments:
 - Be consistent with column names between tables. Don't refer to a primary key as "CustomerID" in one table and as "CustID" in another.
 - Don't prefix columns with their data type. This is unnecessary and makes for extra typing.

Indexes

- Use the table name as a prefix plus the first indexed column name. If creating more than one index starting with this column, use as many as necessary to uniquely identify the index.
- Always explicitly name your indexes rather than allowing SQL Server to generate names. This makes indexes easier to trace and to understand just by looking at the name.
- Samples:

- tCustomer_CustomerID, tCustomer_CustomerName_Address, tCustomer_CustomerName_CustomerNumber

Constraints

- Use the table name as a prefix plus the constrained column name.
- Always explicitly name your constraints rather than allowing SQL Server to generate names. This makes constraints easier to trace and to understand just by looking at the name.
- Samples:
 - Primary Key: tCustomer_CustomerID
 - Foreign Key: tInvoice_CustomerID
 - Unique: tInvoice_InvoiceNumber

Views

- Prefix all views with "v". This prefix helps ensure that no keywords are used as a view name.
- Complete the name with the primary entity displayed by the view in a singular form.
- For views that merge entities, use a combined name starting with the primary table.
- Samples:
 - vCustomerDetail, vCustomerAddress, vInvoiceHeader, vInvoiceDetail, vCustomerInvoiceDetail
- Comments:
 - The prefix used here helps ensure that no keywords are used as a view name. For example, "user" is a common desired view name that is also a keyword, but "vUser" is not.
 - Although consistency in naming between tables and views allows them to be used interchangeably in accordance with ANSI standards, I prefer a clear difference provided by the prefix. This is because I do not allow direct access to either tables or views to the users of my systems (all access is provided through stored procedures that feed various reports).

Stored Procedures

- Prefix all stored procedures with "p". Complete the name with the primary table affected, then the job performed. This will group all procedures for a given table in one location alphabetically.
- Use the prefix "r" for stored procedures that directly generate a report.
- Samples:
 - pCustomerList, pCustomerSearch, pCustomerCreate, pCustomerRead, pCustomerUpdate, pCustomerDelete, pCustomerPurge
 - rCustomersByCountry, rCustomersBySales
- Comments:
 - Never prefix a stored procedure with "sp_". This will cause a performance hit against your system as SQL Server always searches the Master database for these stored procedures first.

User-Defined Functions

- Prefix all user-defined functions with "f". Add a shortened description of functionality.
- Samples:
 - fSplit, fMixedCase

Triggers

- Prefix all triggers with "tr". Add the table name and trigger type.
- Samples:
 - trCustomerInsert, trCustomerUpdate

General Notes

- Never use reserved words as a name for any database object. The SQL Server Books Online help file contains a list of reserved words under the title "Reserved Keywords (Transact-SQL)".
- Only use letters, numbers and underscores in the names of database objects. Specifically, never use a space as these can create many problems in other applications.
- Avoid extremely long names, but don't oversimplify past the point of readability. Too many acronyms makes it difficult for new developers to follow the design.
- Use mixed case rather than underscores (in most cases) to indicate word breaks . Use "pCustomerAddressCreate" instead of "pcustomer_address_create".
- Use singular names rather than plural. This is often debated - singular naming saves a character when coding.

Behind The Scenes

I was once faced with the task of "cleaning up" a database with more than 1200 tables and 2000 stored procedures. 80% of these tables and procedures were not in use and needed to be removed. A standard naming convention would have made the cleanup work much faster and would have allowed new developers to learn the system much faster as well.

In addition, a standard naming convention would have allowed for directed text searches to trace specific stored procedures and tables. This would have allowed consolidation of some redundant procedures without having to resort to a "change it and see what breaks" testing methodology.

References and Additional Reading

- Narayana Vyas Kondreddi
- Joe Celko's SQL Programming Style (a book available on Amazon.com)

About "Practical Methods"

I have written the "Practical Methods" series as a guide for database developers and administrators that are starting out with SQL Server. The articles are intended to serve as a quick reference and starting point but are not necessarily comprehensive. The articles are written for SQL Server 2005 but most will also apply to SQL Server 2000.

This information has been pulled directly from my day-to-day experience. I hope that you find it useful - Michael Lato

Large Object Data

By Simon Sabin

Large Object Data can be confusing for many DBAs. How to load it, retrieve it, and optimize queries on your server. New author and MVP Simon Sabin brings us a look at whether or not you should store this data in a row or on separate pages.

Introduction

I have just received the latest copy of SQLMagazine and read with interest the article on "Varbinary(max) tames the BLOB" it discusses in passing the fact that the new max data types hold the data in the row (same page) if possible and only if the data is too large, move it off the page. This is the opposite to the older LOB data types, TEXT, NTEXT

and IMAGE which only hold the data in row if you set an option. (Kalen Delaney goes into more depth in another article)

I have a word of caution around the inclusion of any of the max data types in a table with other columns (except the PK). The reason for this is that because the max data types are stored "in row", this means that if you have to read other data that requires the row being accessed i.e. accessing a column that is not in an index you will be reading this large column of data as well.

Imagine a person table

```
create table Person (  
  
    PersonId int  
  
    identity(1,1) Primary Key,  
  
    Forename varchar(50),  
  
    Surname varchar(50),  
  
    DateOfBirth datetime,  
  
    Avatar varbinary(max))  
  
  
Create Index IX_Person_DateOfBirth  
  
    On Person(DateOfBirth)
```

Let's assume the average size of the Avatar is 4000 bytes (20x20 logo). If you then run the following query

```
Select Forename, Surname  
  
From Person  
  
Where DateOfBirth Between '1 Aug 2006' and '1 Sep 2006'
```

To get the Forename and Surname fields a bookmark lookup that has to go to the main row of the table. Unfortunately this is bloated with the Avatars and so you only get 1 row per page. So if the query returned 80 rows you will probably have had to read at least 80 pages of data.

If the Person table was set to have "**large value types out of row**" on then each page would contain about 80 rows because only a pointer to the Avatar is stored in the row (it would be > 80 if the forename and surname weren't always 50 characters). So your query would then read 2 or 3 pages (its > 1 due to the index pages being read)

There are other ways around this by adding the extra columns onto your indexes by using the new "INCLUDE" clause, but this would have to be done on all indexes on your table.

Conclusion

My preference is to store images and other LOB data on tables with columns that have a common selection policy, the LOB column(s) will always be selected when the other columns are selected i.e. if in the above situation the Avatar was always returned when any of the surname, forename or dateOfBirth columns were selected then having the Text In Row would be a benefit.

So bottom line is to plan carefully whether to include LOB data in row, taking note of the other columns on the table. You can also read about the new .write method available for the max data types that greatly improve update performance

Simon is a database architect for Totaljobs Group based in the UK. He specialises in performance SQL Server systems and recently on search technologies.

Row-By-Row Processing Without Cursor

By Amin Sobati

INTRODUCTION

Cursors exist because there are situations that row-by-row processing is inevitable. However as they are resource intensive, developers always try to refine their codes and bypass cursors using T-SQL tricks.

Not all scenarios are capable for this replacement, but I have seen many of them! One of the common usages of cursors is inside triggers for processing the Deleted and Inserted tables. At the time of writing this article, SQL Server does not support any mechanism to fire a trigger for each row separately when a DML statement affects more than one row.

As an example, an Insert command is inserting 3 records (from a query) into a table. Your trigger needs to retrieve each PK value from Inserted table and send it to a particular Stored Procedure for some processes. Iterating through a cursor made of PKs might seem the first option. However in this case we can use the power of variables that live within a query, then Dynamic T-SQL Execution finalizes the trick! Let's create necessary objects:

```
CREATE TABLE Books (
    BookCode VARCHAR(5),
    BookDesc VARCHAR(100))

-- The SP which processes new BookCode during insertion
CREATE PROC usp_Process
    @BookCode VARCHAR(5)
AS
-- Do something useful with each BookCode. We simply print it
Print 'SP is processing... ' + @BookCode
GO
```

All we need to do in the trigger is to construct a string of T-SQL commands that contain EXEC usp_Process for each BookCode:

```
CREATE TRIGGER tr1 ON Books AFTER INSERT AS
```

```
DECLARE @sql VARCHAR(8000)

SET @sql=''

SELECT @sql=@sql+'EXEC usp_Process ''' + BookCode + '''; ' FROM Inserted

PRINT 'Trigger is preparing Dynamic T-SQL: ' + @sql -- Just o see @sql

EXEC (@sql)

GO
```

Now let's try to see how it works:

```
INSERT Books

SELECT 'A','Book desc 1' UNION

SELECT 'B','Book desc 2' UNION

SELECT 'C','Book desc 3'

Trigger is preparing Dynamic T-SQL: EXEC usp_Process 'A'; EXEC usp_Process 'B'; EXEC usp_Process 'C';

SP is processing... A

SP is processing... B

SP is processing... C

(3 row(s) affected)
```

This manner can be used even if you need to send more parameters to the SP. Please be careful with @sql here because your dynamically-built string cannot exceed 8000 characters. Fortunately SQL Server 2005 developers can benefit from MAX length of VARCHAR and NVARCHAR data types!

CONCLUSION

Use the power of T-SQL and SELECT statement whenever you can. They are flexible enough to help us to perform some sort of row-by-row processing faster without bothering the hardware. Experts think about a Cursor as the last option!

Beauty is in the Eye of the Beholder

By Stephen Hirsch

Shakespeare probably didn't realize he was talking about dynamic SQL when he wrote those lines, but he did. There have been a number of articles on this site telling developers that dynamic SQL is "bad". After commenting on the articles a few times, I'd like to detail an application space where dynamic SQL is "good".

The quotes around the words "good" and "bad" are not just the stylistic affectations of an English major. The question of what you need to optimize for should be the first question asked in the design process. The answer to the question depends on so many factors, whether the software to be developed is embedded or not, whether it is internal to an organization or not, whether it is transaction or reporting focused, etc., that I am loathe to pass judgement right away.

After all, which direction does a clock move, clockwise or counterclockwise? The answer depends on your perspective...

For most of the developers reading this site, and indeed for most of us working in and around relational databases, the default optimization strategy is to optimize for speed of execution. This is not without reason. If you need to optimize for speed, then dynamic SQL will slow you down to some extent, since the SQL will need to be compiled before it can be run. In that case, dynamic SQL is "bad".

However, there are application spaces that use relational databases extensively, where speed of execution is not the primary optimization path. I worked in such an application space for many years, clinical trial applications.

Clinical trials are simply put, giant, extremely costly experiments. Each clinical trial is its own application, with its own data structures, programs, etc. Clinical trials tend not to have huge amounts of data (a massive trial had about a gigabyte of raw data), but since we had over 250 trials in production at one time, we had an enormous amount of metadata.

Since clinical trials come under the purview of the FDA, there is a tremendous overhead of validation for each distinct piece of software you write. Validation here does not mean testing your software, but rather documenting the testing of your software. Whether it is a good use of anyone's time is a discussion for another day, but the requirement is there, and it is onerous to one degree or another. Dynamic SQL is incredibly useful in reducing the regulatory overhead. Below are two areas where it was used.

Two important notes. One, these were internal, server-only apps with no real direct end user involvement. Therefore, SQL injection was not something we had to deal with. If your app needs to be concerned with SQL injection, that could definitely limit your use of dynamic SQL. Two, ClinTrial, the clinical trial programming tool we used, has a well thought out and extensive metadata repository. Dynamic software, whether dynamic SQL or some other language, practically requires a metadata repository.

The first area that dynamic SQL was used was the creation of generic data validation scripts. Instead of having to write the same scripts over and over (static SQL couldn't be used because each trial had different table names, and if they did have the same table names, they often had different column definitions), the tool read the metadata repository to dynamically create the validation routines. Days upon days spent creating validation scripts were eliminated.

There certainly was a time-to-execute performance hit, but it really wasn't that awful, and certainly worth the savings in development and validation time.

The second area for dynamic SQL was one where I was directly involved. We were tasked with creating the first ever clinical data warehouse. Among other things, this meant loading literally thousands of heterogeneously structured tables into 10 standard structures. A new set of tables would typically be created each week, and we had to allow for the fact that table structures could change in the middle of a trial, since they were, after all, experiments.

Finally, we had to make allowances for bizarre behavior on the part of individual trials. While you could certainly make an argument that the initial trial design process should have been more closely regulated (i.e., don't think outside the box until you look inside the box first), that was the environment we had to deal with.

With our validation requirements, there was no way in the world to do that without dynamic SQL. Here's a very short version of how we accomplished this.

1) For each individual target entry, we created a "feed table". This feed table was populated from the metadata repository and a configuration table we called the "metameta" table.

2) From the feed table, we created an enormous SELECT statement, full of UNION ALLs, that extracted and transformed the individual trial's data to the standard target, as well as handled the change data capture logic. We stored this SELECT statement as a VIEW.

3) An Informatica program would load the data by invoking the VIEW and moving its result set into the staging target table.

In some ways, you could think of the whole feed table apparatus as a giant, complicated text replacement program, or keystroke macro; for validation purposes, that's exactly what we did, since we considered the VIEW to be the E and T of ETL. Validating that was trivial; just do `SELECT * FROM new MINUS SELECT * FROM old` and `SELECT * FROM old MINUS SELECT * FROM new` statements to compare the new and the old views. If you got the results you wanted, you were done. That was our story, and we stuck with it.

In the second example, there was no performance hit at all. Dynamic SQL turned what could have been an onerous task, a working in the coal mines kind of task, into one that didn't harsh anybody's mellow once trained.

Security

As DBAs and developers we have seen security become a bigger and bigger concern over the last decade. Despite the fact that, these days, more and more breeches aren't due to technical errors, we must still work harder to ensure that our code can stand the scrutiny of hackers and other malicious users.

This section highlights a few articles that pertain to security in SQL Server in a general sense and will help you better understand how this platform can work for you.

SQL 2005 Symmetric Encryption	215
Ownership Chaining	222
Preventing Identity Theft Using SQL Server.....	223

SQL 2005 Symmetric Encryption

By Michael Coles

Introduction

One of the most exciting new features of SQL Server 2005 is the built-in encryption functionality. With this new version of SQL Server, the SQL Server Team has added encryption tools, certificate creation and key management functionality directly to T-SQL. For anyone who makes their living securing data in SQL Server tables because of business requirements or regulatory compliance, these new features are a godsend. For those trying to decide whether to use encryption to secure their data, the choice just got a lot easier. This article describes how the new encryption tools work, and how you can use them to your advantage.

T-SQL now includes support for symmetric encryption and asymmetric encryption using keys, certificates and passwords. This article describes how to create, manage and use symmetric keys and certificates.

Because of the amount of information involved, I've divided this article into three sections:

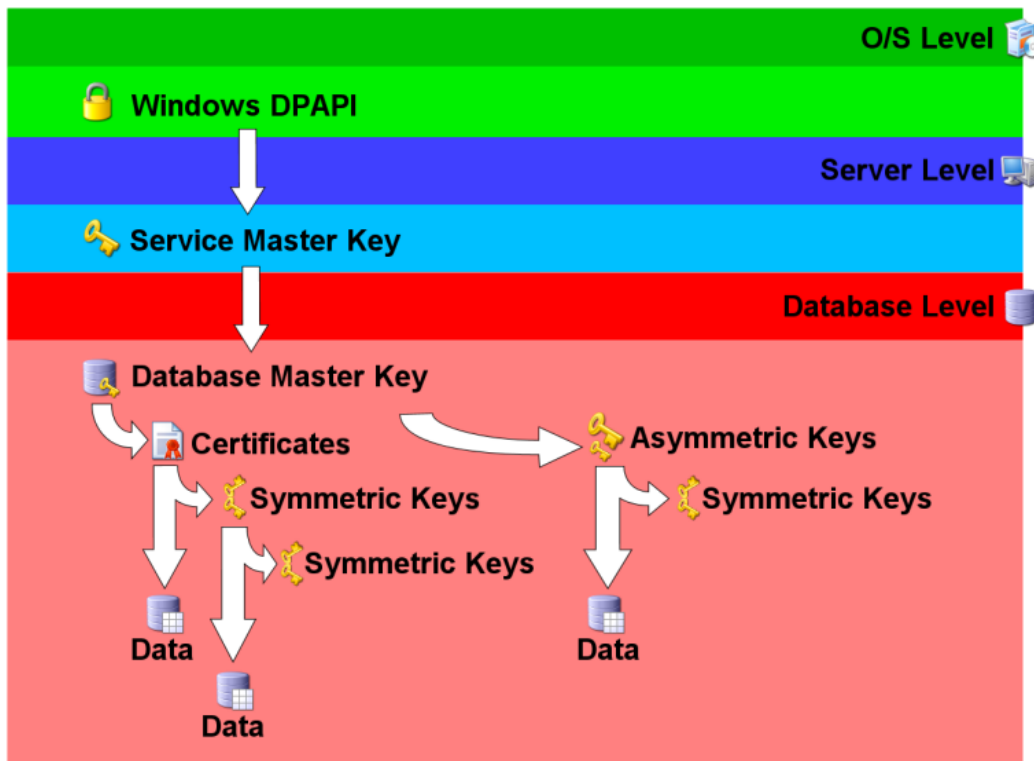
- Part 1: Service and Master Keys
- Part 2: Certificates
- Part 3: Symmetric Keys



Part 1: Service and Database Master Keys

The SQL 2005 Encryption Hierarchy

SQL Server 2005 encryption functionality uses a hierarchical model that looks like this:



SQL Server 2005 Encryption Hierarchy

Service Master Key

Each SQL Server 2005 installation has exactly one Service Master Key (SMK), which is generated at install time. The SMK directly or indirectly secures all other keys on the server, making it the "mother of all SQL Server encryption keys." The Windows Data Protection API (DPAPI), at the higher O/S level, uses the SQL Server service account credentials to automatically encrypt and secure the SMK.

Because it is automatically created and managed by the server, Service Master Keys require only a few administrative tools. The SMK can be backed up via the `BACKUP SERVICE MASTER KEY` T-SQL statement. This statement has the following format:

```
BACKUP SERVICE MASTER KEY TO FILE = 'path_to_file'
```

```
    ENCRYPTION BY PASSWORD = 'password'
```

Path_to_file is the local path or UNC network path to the file in which the SMK will be backed up. Password is a password which is used to encrypt the SMK backup file.



You should backup your Service Master Key and store the backup in a secure off-site location immediately after installing SQL Server 2005.

Should you ever need to restore the Service Master Key from the backup copy, you can use the `RESTORE SERVICE MASTER KEY` statement:

```
RESTORE SERVICE MASTER KEY FROM FILE = 'path_to_file'

    DECRYPTION BY PASSWORD = 'password' [FORCE]
```

The *path_to_file* is the UNC or local path to the backup file. Password is the same password previously used to encrypt the backup. When restoring the SMK, SQL Server first decrypts all keys and other encrypted information using the current key. It then re-encrypts them with the new SMK. If the decryption process fails at any point, the entire restore process will fail. The `FORCE` option forces SQL Server to ignore decryption errors and force a restore.



If you have to use the `FORCE` option of the `RESTORE SERVICE MASTER KEY` statement, you can count on losing some or all of the encrypted data on your server.

If your Service Master Key is compromised, or you want to change the SQL Server service account, you can regenerate or recover the SMK with the `ALTER SERVICE MASTER KEY` statement. The format and specific uses of the `ALTER SERVICE MASTER KEY` statement are available in Books Online.

Because it is automatically generated by SQL Server, there are no `CREATE` or `DROP` statements for the Service Master Key.

Database Master Keys

While each SQL Server has a single Service Master Key, each SQL database can have its own Database Master Key (DMK). The DMK is created using the `CREATE MASTER KEY` statement:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'password'
```

This statement creates the DMK, encrypts it using the supplied password, and stores it in the database. In addition, the DMK is encrypted using the Service Master Key and stored in the master database; a feature known as "automatic key management." We'll talk more about this feature later.

Like the Service Master Key, you can backup and restore Database Master Keys. To backup a DMK, use the `BACKUP MASTER KEY` statement. The syntax is analogous to backing up a Service Master Key.

```
BACKUP MASTER KEY TO FILE = 'path_to_file'

    ENCRYPTION BY PASSWORD = 'password'
```

Restoring the Database Master Key requires that you use the `DECRYPTION BY PASSWORD` clause, which specifies the password previously used to encrypt the backup file. In addition you must use the `ENCRYPTION BY PASSWORD` clause, which gives SQL Server a password to encrypt the DMK after it is loaded in the database.

```
RESTORE MASTER KEY FROM FILE = 'path_to_file'

    DECRYPTION BY PASSWORD = 'password'

    ENCRYPTION BY PASSWORD = 'password'

    [ FORCE ]
```

Like restoring the Service Master Key, the DMK restore statement has a `FORCE` option which will ignore decryption errors.



It is recommended that you immediately create backups of Database Master Keys and store them in a secure off-site location immediately after creating them. Also, the `FORCE` option of the `RESTORE MASTER KEY` statement can result in encrypted data loss.

To drop a DMK, use the `DROP MASTER KEY` statement:

```
DROP MASTER KEY
```

This statement drops the Database Master Key from the current database. Make sure you are in the correct database before using the `DROP MASTER KEY` statement.

Automatic Key Management

When you create a Database Master Key, a copy is encrypted with the supplied password and stored in the current database. A copy is also encrypted with the Service Master Key and stored in the master database. The copy of the DMK allows the server to automatically decrypt the DMK, a feature known as "automatic key management." Without automatic key management, you must use the `OPEN MASTER KEY` statement and supply a password every time you wish to encrypt and/or decrypt data using certificates and keys that rely on the DMK for security. With automatic key management, the `OPEN MASTER KEY` statement and password are not required.

The potential downfall of automatic key management is that it allows every sysadmin to decrypt the DMK. You can override automatic key management for a DMK with the `DROP ENCRYPTION BY SERVICE MASTER KEY` clause of the `ALTER MASTER KEY` statement. `ALTER MASTER KEY` and all its options are described in full detail in Books Online.



Part 2: Certificates

Creating Certificates

Once you have your Service Master Key and Database Master Key configured, you're ready to begin making certificates. SQL Server 2005 has the ability to generate self-signed X.509 certificates. The flexible `CREATE CERTIFICATE` statement performs this function:

```
CREATE CERTIFICATE certificate_name [ AUTHORIZATION user_name ]
```

```
{ FROM <existing_keys> | <generate_new_keys> }
```

```
[ ACTIVE FOR BEGIN_DIALOG = { ON | OFF } ]
```

```
<existing_keys> ::=
```

```
ASSEMBLY assembly_name
```

```
| {
```

```
  [ EXECUTABLE ] FILE = 'path_to_file'
```

```
[ WITH PRIVATE KEY ( <private_key_options> ) ]  
}  
  
<generate_new_keys> ::=  
[ ENCRYPTION BY PASSWORD = 'password' ]  
WITH SUBJECT = 'certificate_subject_name'  
[ , <date_options> [ ,...n ] ]  
  
<private_key_options> ::=  
FILE = 'path_to_private_key'  
[ , DECRYPTION BY PASSWORD = 'password' ]  
[ , ENCRYPTION BY PASSWORD = 'password' ]  
  
<date_options> ::=  
START_DATE = 'mm/dd/yyyy' | EXPIRY_DATE = 'mm/dd/yyyy'
```

There are a lot of options associated with the `CREATE CERTIFICATE` statement. Fortunately few are needed most of the time. The following statement will create a certificate encrypted by password:

```
CREATE CERTIFICATE TestCertificate  
  
    ENCRYPTION BY PASSWORD = 'thisIsAP@$$w0rd'  
  
    WITH SUBJECT = 'This is a test certificate',  
  
    START_DATE = '1/1/2006',  
  
    EXPIRY_DATE = '12/31/2008';
```

If you leave off the `ENCRYPTION BY PASSWORD` clause, the Database Master Key is used to encrypt the certificate. Leaving the `START_DATE` out will result in the current date being used as the default start date for your certificate.

You can also use the `CREATE CERTIFICATE` statement to import an existing certificate into your SQL Server.

In addition to `CREATE CERTIFICATE`, SQL Server provides additional statements to manage certificates. These include `DROP CERTIFICATE`, `ALTER CERTIFICATE`, and `BACKUP CERTIFICATE`.



There is no `RESTORE` statement for certificates. Use the `CREATE CERTIFICATE` statement to restore a backed-up certificate.

Encryption and Decryption by Certificate

Certificates can be used to encrypt and decrypt data directly by using the built-in `EncryptByCert`, `DecryptByCert` and `Cert_ID` functions. The `Cert_ID` function returns the ID of the certificate with the specified name. The format of the `Cert_ID` function is:

```
Cert_ID ( 'cert_name' )
```

The `'cert_name'` is the name of the certificate. The `EncryptByCert` function requires the Certificate ID and has the following format:

```
EncryptByCert ( certificate_ID , { 'cleartext' | @cleartext } )
```

The `certificate_ID` is acquired by using the `Cert_ID` function. `'Cleartext'` is the clear text string to encrypt. The clear text can be a char, varchar, nchar, nvarchar or wchar value. The `EncryptByCert` function returns a varbinary result of up to 8,000 bytes.

The `DecryptByCert` function is used to decrypt data that was previously encrypted by certificate. The format for `DecryptByCert` looks like this:

```
DecryptByCert (certificate_ID,  
               { 'ciphertext' | @ciphertext }  
               [ , { 'cert_password' | @cert_password } ]  
               )
```

Like `EncryptByCert`, `certificate_ID` can be obtained using the `Cert_ID` function. `'Ciphertext'` is the previously encrypted text. If you created your certificate with the `ENCRYPT BY PASSWORD` clause, `'cert_password'` must be the same password you used when you created the certificate. If you did not use `ENCRYPT BY PASSWORD` to create the certificate, leave out `'cert_password'`.

The following sample script creates a Database Master Key, a test certificate and demonstrates how to encrypt/decrypt data using the certificate. (Complete code is at www.sqlservercentral.com)



Part 3: Symmetric Keys

Creating Symmetric Keys

You can use certificates to create symmetric keys for encryption and decryption within the database. The `CREATE SYMMETRIC KEY` statement has the following syntax:

```
CREATE SYMMETRIC KEY key_name [ AUTHORIZATION owner_name ]  
  
    WITH <key_options> [ , ... n ]  
  
    ENCRYPTION BY <encrypting_mechanism> [ , ... n ]
```

```
<encrypting_mechanism> ::=  
  
    CERTIFICATE certificate_name |  
  
    PASSWORD = 'password' |  
  
    SYMMETRIC KEY symmetric_key_name |  
  
    ASYMMETRIC KEY asym_key_name
```

```
<key_options> ::=  
  
    KEY_SOURCE = 'pass_phrase' |  
  
    ALGORITHM = <algorithm> |  
  
    IDENTITY_VALUE = 'identity_phrase'
```

```
<algorithm> ::=  
  
    DES | TRIPLE_DES | RC2 | RC4 | DESX | AES_128 | AES_192 | AES_256
```

Like the CREATE CERTIFICATE statement, CREATE SYMMETRIC KEY is very flexible. In most situations you will probably use a small subset of the available options. As an example, this statement creates a symmetric key and encrypts it with the Test Certificate created in the previous section:

```
CREATE SYMMETRIC KEY TestSymmetricKey  
  
    WITH ALGORITHM = TRIPLE_DES  
  
    ENCRYPTION BY CERTIFICATE TestCertificate;
```

Symmetric keys can be secured via other symmetric keys, asymmetric keys and passwords, as well as by certificates. SQL Server also provides ALTER SYMMETRIC KEY and DROP SYMMETRIC KEY statements to manage your symmetric keys. Specific syntax for these statements can be found in Books Online.



When dropping keys and certificates, the order is important. SQL 2005 will not allow you to DROP certificates or keys if they are being used to encrypt other keys within the database.

Symmetric Key Encryption

SQL Server provides a set of functions to encrypt and decrypt data by symmetric key. These functions are EncryptByKey, DecryptByKey and Key_GUID. The Key_GUID function returns the unique identifier assigned to a specific symmetric key. The format of the function is:

```
Key_GUID( 'Key_Name' )
```

The EncryptByKey function requires a reference to the symmetric key GUID in order to encrypt data. The format of the EncryptByKey function is:

```
EncryptByKey( key_GUID, { 'cleartext' | @cleartext }  
  
    [ , { add_authenticator | @add_authenticator }  
  
    , { authenticator | @authenticator } ]  
  
    )
```

The *key_GUID* is the symmetric key GUID, '*cleartext*' is the plain text to be encrypted. *Add_authenticator* and *authenticator* are optional parameters that can help eliminate post-encryption patterns from your data.

The *DecryptByKey* function performs the reverse of *EncryptByKey*. This function decrypts your previously encrypted data. The format for *DecryptByKey* is:

```
DecryptByKey( { 'ciphertext' | @ciphertext }  
  
    [ , add_authenticator  
  
    , { authenticator | @authenticator } ]  
  
    )
```

'*Ciphertext*' is the encrypted text. *Add_authenticator* and *authenticator*, if present, must match the values used in the *EncryptByKey* function. The *DecryptByKey* function doesn't require you to explicitly specify the symmetric key GUID. The symmetric key used previously to encrypt the data must be open, however. The *OPEN SYMMETRIC KEY* statement is used to open a symmetric key.

Here is a sample T-SQL script demonstrating encryption and decryption by symmetric key: (code at www.sqlservercentral.com)

Conclusions

SQL Server 2005 includes several new functions to securely create, manage and use encryption keys and certificates to secure sensitive data. Taking advantage of this new functionality can greatly enhance your database and application security.

Ownership Chaining

By Rob Farley

Ownership chaining can be really cool. Many of us would have granted a restricted user access to a stored procedure or function called *dbo.GetInfo*, but denied access to the underlying tables. The user can get access to the data through the stored procedure, but can't get at the tables underneath at all. Makes life nice and safe. But why does this work?

The answer is ownership chaining.

What happens is that the system sees the request come in for the stored procedure (or could be a function, I'll say 'module' to make life easier). It checks to see if the user has permission to execute the module, and it does! Great. So then it enters the module and executes what's in there. Now within the module, the system tries to access a table, and it needs to work out if the user should have access to it.

Of course, the answer should be no, because the user doesn't have access to that table. But here's where the ownership chaining kicks in. The system says "I'm in the context of a stored procedure which is owned by dbo (or whoever). This table I'm trying to access is owned by them too, so clearly I must have access to it!" It's a bit sneaky, I know. Heck of an assumption to make, but it's what happens, and it makes life quite handy for us developers.

What many developers think is that a stored procedure has some special kind of super-access to access stuff because it runs under a different context or something, but it's not the case. The security check is just skipped on the underlying tables if the owner is the same.

"But I can use 'execute as' within a module, and that changes all that" I hear you say, and I'm going to ignore you for a minute.

Cross-database ownership chaining is worth being aware of. You might have read about what I've just written and suddenly realise why various things just worked before. You might also have suddenly wondered why that didn't work in some other situations, particularly when you've tried to access a table in a different database.

Cross-database ownership chaining is just like ownership chaining, but it works across databases. Let's suppose we have two databases, we'll call them db1 and db3 (haha - get it?). A stored procedure in db1 tries to access a table owned by the same login on db3. But it's denied. That's because cross-database ownership chaining is turned OFF by default. Let me explain why.

Barry and I each have a database on a server. I don't have access to his, and he doesn't have access to mine. We both have full access to our own databases, and I don't let Barry see my sensitive data. But somehow, Barry has persuaded the DBA to turn on cross-database ownership chaining. I'm not worried, am I?

Next thing I know, all my data is compromised, and I've lost my job. Why?

Turns out evil Barry created a stored procedure in his own database that tried to access my data. It didn't work. But then he created a user in his database for my login. Yeah! He didn't even tell me. But even if I had known, it wasn't as if that user could do anything in his database. Except that he then changed the owner of that dodgy stored procedure to be that login of mine. Yeah great, so I could access a stored procedure on his machine.

But because of cross-database ownership chaining, when Barry accessed that stored procedure, the system let him see my sensitive data. It had confirmed that he had access to the stored procedure in his own database (that module that was owned by me!), and everything was fine. But when that module tried to access my tables, it noticed that the owner was, well, me - the same owner as the module it was running. So it skipped the check and granted access.

So cross-database ownership chaining is bad, unless you have full control over the whole server. Many of us do, of course, but not always. Be aware of that.

Thanks for reading...

Oh sorry - you were asking about 'execute as'. Yes. Execute as is a great way of avoiding this. Stick all your tables into a schema owned by someone else, and tell your modules to run as a particular user, who presumably has access to the right tables in whichever database is appropriate. Now you can avoid all the ownership chaining, and feel like you have proper control over your security.

Preventing Identity Theft Using SQL Server

By Yaroslav Pentsarskyy

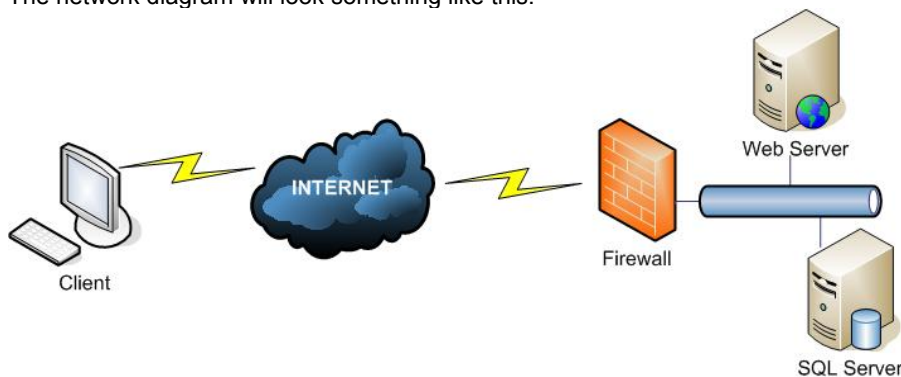
When it comes to security of an online business so often we are so used to the concept of an attacker hitting our web server as a main target that we may forget about other important elements of our network, such as our database

server. There are many techniques and layers of defense available to protect the external perimeter of the network and web applications. We cannot have too many tools, however, that will protect our data on a database server when an attacker has direct access to the system. That implies that we'll always rely on some other mechanisms to protect our database system, but what if those mechanisms are compromised?

In this example I would like to share one solution that will demonstrate how the database system can be used in detecting the potential unauthorized alteration of the data.

Let's assume we have an online gambling business that relies on our clients accessing web application (an e-casino) which communicates with the backend database. The web application handles user authentication and account management. The database holds information about customer's account: username, encrypted password, name, address, email, and account balance. As a part of our service we offer our clients the ability to associate their accounts with an online payment processing system (such as PayPal) so they can withdraw their funds. To associate the account with PayPal we have to have their email, and name matching PayPal account information.

The network diagram will look something like this:



Assuming that our web application has the best security practices applied when developed, and our firewall has all the restrictive rules implemented we might think that there is nothing to worry about. However, there is a constant threat of a blackhat community exploiting certain server vulnerabilities and access the database engine directly bypassing our web application. It may be as "simple" as exploiting another buffer overflow vulnerability, which we hear about so many times these days. In this case an attacker may access our database records and modify certain data that will let her impersonate one of the users by replacing her name and email with the original account information.

For example, in the table below, we have 3 users with their corresponding information:

	ID	username	password	name	address	email	balance
1	1	ChesterJ	0xD41D8CD98F00B204E9800998ECF	Chester J	123 Lans Rd.	chesj@wp4tv.com.com	469.00
2	2	JimK	0xC4CA4238A0B923820DCC509A6F7	Jim K	435 Shell St.	jim@businessinvent.com	1027.00
3	3	ElsieM	0xC81E728D9D4C2F636F067F89CC1	Elsie M	5th Ave.	spentsarsky@webtv.net	327.00

Assuming that "JimK" is a legitimate user, and "ChesterJ" is a phony user created by an attacker "ChesterJ" can replace information used by Payment Processing System to become the owner of the legitimate balance or change his balance with the bogus amount and withdraw funds.

In case of our database system being attacked directly using system specific vulnerability we won't have any signs of a compromise from the web application part. We can use restrictions and triggers, but those can be deleted or altered in no time once an attacker gains access to our DBMS.

The solution to a problem includes simple modifications to our web application and the database.

As we have designed our web applications we know that the only system that is allowed to access the database is an application and its respective user interface.

For example, to create a user we have to use the same old registration screen that is duplicated practically on any website. The same situation is with a system updating user's balance no one can access balance management system unless certain events in an application triggered the change of that balance (for example user won a jackpot etc).

Knowing who's allowed to access certain parts of the database - we can develop custom signatures for our applications that modify user's account. That will include adding one more column to the database that will store unique signature of a modifying application.

For instance, the only time username can be changed is when the user is created through our registration form. Therefore, the registration form will add its hashed signature to the column corresponding to our username. The hash will also include some other important parts of the user's account being unique for every user. Here are the steps to generate user specific hash in a database:

```
The registration screen will take down the following information:

date user is created (dd/mm/yyyy), username, email, name, and address.

The following string is an example of user identification information generated
from the input in the registration form:

"29/08/2006, 12:39:2368*JimK*Jim K*435 Shell St.*jim@businessinvent.com".

Where "*" is a separator.

The string is hashed using SHA1 algorithm (you can use algorithm of your choice)

The hashed result (in our case: "0xBB4E951901FF5DCECE446162D2E5B39BE589363F")

is recorded to the table along with the rest of the data.
```

Now that we have created a unique hash for our records any account changes to the information that were not supposed to be made by an authorized application will mean that the data in the database has been maliciously modified. But how do we know that the data has been modified?

There are many approaches to this. We can create a schedule for the user records to be scanned and verified against hash that is in our special column. The scanning is better to be done by another independent system so that when our database is compromised an attacker cannot figure out the hashing algorithm we used along with the string that is expected as an input.

With SQL Server 2005 we can have a scheduled job running any time we prefer comparing original hash value with the one calculated using HASHBYTES (SHA1', @input) function. If the result of our dynamic hash doesn't match the recorded value in a user row the alert is sent to an administrator using Notification Services.

Another approach is to have the integrity of our records checked when it is required. Going back to our e-casino application, whenever user wants to withdraw funds we double check the hash of his current information with the one created earlier.

Of course this solution will not (and should not) replace other security practices available out there. However, it is a nice addition to our multilayer security system.

Feel free to post your comments and feedback here or at spentsarsky@aim.com.

ETL and BI

More and more often, we are called to move data between systems and, oh yeah, can you fix these few little problems. Three days later, after massaging the flies in Excel or T-SQL, you're ready to pull your hair out. The process of extracting data, transforming it, and loading it back into a system (ETL), is one that has created employment for a great many people. Business Intelligence (BI) seems to deal mainly with reporting.

In SQL Server 7, we got DTS, which was a great tool for moving data around in a limited fashion. After many complaints on the problems with complex transformations, SQL Server 2005 brought us Integration Services (SSIS), a very rich programming environment.

This came at a cost, however, in that this environment requires true programming skills and has a steep learning curve. This year we have a special section just for our ETL articles.

A Common Architecture for Loading Data	227
Overview of SSIS	232
SSIS - Transfer SQL Server Objects Debugged	240
SSIS Is Not Just for SQL Server	244
SSIS Programming.....	247
Loading a 24x7 Data Warehouse	250
More Problems with Data Warehousing	253
Storage Modes in SSAS 2005.....	256
Dynamic Connection Strings in Reporting Services 2005	258
Populating Fact Tables.....	260
Reporting Services 2005 101 with a Smart Client	272
Data Driven Subscriptions for Reporting Services (2000 and 2005)	283

A Common Architecture for Loading Data

By Marck Balasundram

It is always a welcome challenge in a big organization to write ETL application, that runs quite efficiently. As we all know during the nightly batch process the time is a constraint and the allocated time must be wisely used. Recently I was given the task of designing a service that loads data into a sql server database. The extracts come in pipe separated text files. There are eighteen extracts; The challenge was to create one loader which can handle all eighteen extracts at the same time and must handle any totally new feeds without any new major development. The design I came up with is very flexible that any structural changes to source files was accommodated very easily without changing the frame work code. The initial requirement was to load only eight extracts, but the project ended up having eighteen extracts. I accepted the new extracts without changing the framework code.

It is important to pay attention to performance and scalability. My goal was to create an ETL application that can load multiple files with good performance and should be able to add new extracts without modifying the existing framework code. So the design was a no brainer, so I chose "Plug in" Architecture as the solution. This methodology provides increased flexibility for any future enhancement.

The challenge was to create one loader which can handle all eighteen extracts at the same time and must handle any totally new feeds without any new major development. The design I came up with is very flexible that any structural changes to source files was accommodated very easily without changing the frame work code. The initial requirement was to load only eight extracts, but the project ended up having eighteen extracts. I accepted the new extracts without changing the framework code.

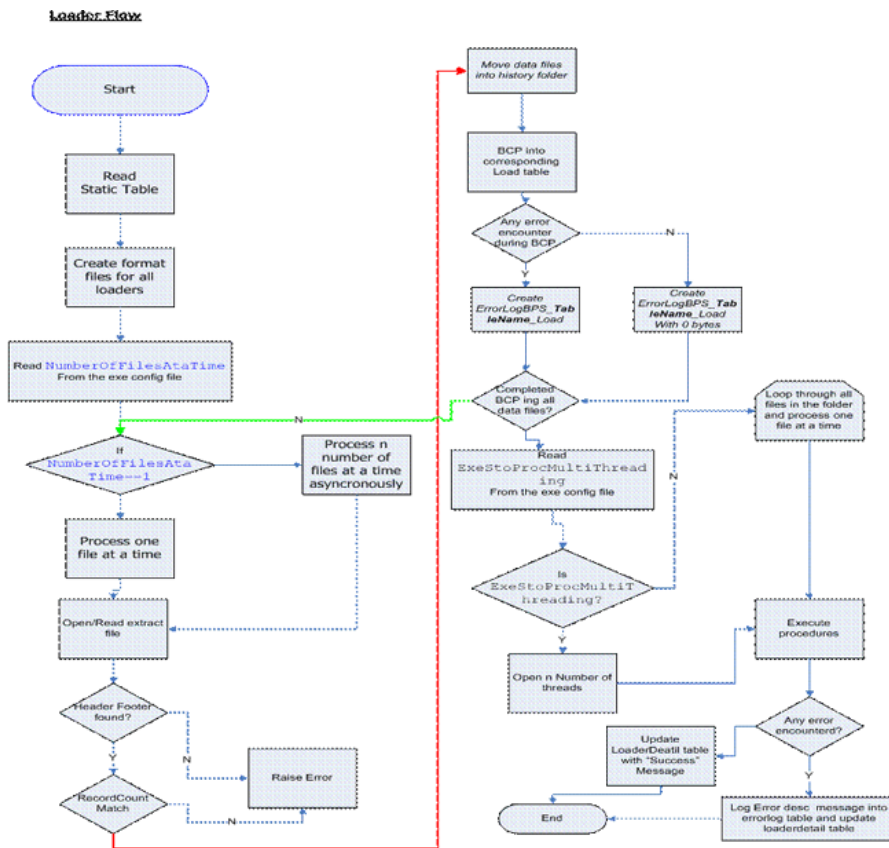
Framework functionalities

- Finding Extract files
- Verifying Extract files integrity
- Loading into staging tables
- Creating dynamic format files
- Running component (Eg Multithreading)
- Integrating component results in the database
- Unloading component

The advantage of the "Plug In" approach is that it simplifies the future addition using only metadata, this table contains unique names of extracts, names of stored procedures for uploading data from staging to production tables, unique key position in the data file and so on. The flow diagram is shown below.

It is always interesting to create a high performance application, so I decided to go with a combination of using BCP (Bulk Copy Program) asynchronously and uploading data from staging tables into production tables with the use of many threads. BCP is a part of the SQL Server 2000 client. It is a very powerful tool to upload data.

I used System.Diagnostics namespace to run BCP asynchronously to upload data into the staging tables. This means that while application is loading data into a staging table, another process reads the source folder and prepares a data file to be uploaded.



It is easier to write synchronous code solutions. However, in using asynchronous multithreaded programming, one of the biggest gains is to avoid bottlenecks. We all know that synchronous programming can cause unnecessary dependencies. There are many more reasons for to choose asynchronous multithreaded programming over synchronous programming. More information can be found at <http://msdn.microsoft.com/msdnmag/issues/05/08/concurrency/default.aspx>.

So I created two C# components

1. An executable which acts as Loader framework
2. A common class(objLibrary) which contains all common activities such as error logging, encryption decryption and connection to DBs.

The database is SQL Server 2000.

Here are the functionalities of the loader framework

1. Read source folder for data files to be loaded. Multiple source files can be loaded at the asynchronously using System.Diagnostics.Process based on the flag value in the executable configuration file.

Example: Here are the variables from the config file; application uses these variables to decide the course of action.

```
<add key="NumberOfFilesAtATime" value="18" />
```

Please include System.Diagnostics in the C# code. The following code is executed within a loop using Directory.GetFiles and then processing all the files in the specified directory.

```
System.Diagnostics.Process p = null;

//Process multiple files or process one file at a time

if(System.Configuration.ConfigurationSettings.AppSettings["NumberOfFilesAtaTime"]=="1")
{
    p.WaitForExit();
}

//This executed after execution of the
loop, ensure the all extracts are
loaded

if(System.Configuration.ConfigurationSettings.AppSettings["NumberOfFilesAtaTime"]!="1")
{
    p.WaitForExit();
}
```

2. Read the static table. This static table contains the names of the tables, primary key information, and names of the stored procedures used for uploading data from staging tables into production tables. Here is a sample DDL for static table.

```
CREATE TABLE [dbo].[LoaderDetails](
    [L_LoaderName] [varchar](50) NOT NULL,
    [L_StoredProcedureName] [varchar](80) NULL,
    [L_LoaderKeyColumnNumber] [int] NULL,
    [L_KeyIDColName] [varchar](50) NULL,
    [L_LoaderDescription] [varchar](50) NULL,
    [L_LastUpdatedTime] [datetime] NULL,
    [L_LastStatusMessage] [varchar](500) NULL,
    [L_BPSKeyLoader] [bit] NULL
) ON [PRIMARY]
```

In this application format files are used very heavily through BCP to upload data. I use format files to selectively pick columns from source file to be exported into sql server table. For example if source files contain twenty columns and I need data from columns 3,7,15 and 16 then format files are extremely useful to complete this task. The format files are created on a nightly basis, and the primary reason for creating format files regularly is to accommodate any table structure changes. I use the schema of the staging tables to do this.

Create format files using the schema of staging tables. The format files are created every night before the batch begins to ensure all changes to tables are included during data load. These table names are stored in the static table. For example if the staging table *ABC* has six columns then a format file with six fields is created dynamically using *SqlDataAdapter*, *StreamWriter* classes.

```
8.0
7
1 SQLCHAR 0 0 "\n" 0 " " "
2 SQLCHAR 0 0 "|" 1 a0 Finnish_Swedish_CS_AS
3 SQLCHAR 0 0 "|" 2 a1 Finnish_Swedish_CS_AS
4 SQLCHAR 0 0 "|" 3 a2 Finnish_Swedish_CS_AS
5 SQLCHAR 0 0 "|" 4 a3 Finnish_Swedish_CS_AS
6 SQLCHAR 0 0 "|" 5 a4 Finnish_Swedish_CS_AS
7 SQLCHAR 0 0 "\r" 6 a6 Finnish_Swedish_CS_AS
```

4. Use Bulk Copy Program (BCP) with the created format file to upload data from data files into a staging table. BCP also produces an output file and an error file.

I added the following node in my config file to find the location of BCP.

```
<add key="BCPexePath" value="C:\Program Files\Microsoft SQL Server\80\Tools\Binn\BCP.EXE"/>
<add key="BCPSwitches" value=" -n -t | -c -b 50000"/>
```

Here is the code that uses to execute BCP to load data.

```
//The file that will be use to capture errors thrown by BCP
strOutPutFileName=ConfigurationSettings.AppSettings["FormatFilePath"]+"RecordsProcessed"
    +ConfigurationSettings.AppSettings["ErrorFileNamePrefix"] + strLoaderName+".txt";
strBCPErrorFileName=ConfigurationSettings.AppSettings["FormatFilePath"]
    +ConfigurationSettings.AppSettings["ErrorFileNamePrefix"] + strLoaderName + ".txt";

proc.Arguments=strLoaderName + @" IN " + strReceivedFileName.ToString() +
ConfigurationSettings.AppSettings["BCPSwitches"].Replace("|","\"|\")+
objLibrary.Decrypt(objLibrary.Read("CONNECTION_BCP"))

+ " -o" + strOutPutFileName + " -e " + strBCPErrorFileName + " -L " + dblRecordCount+1 + " -f"

+ proc.UseShellExecute=false;

proc.RedirectStandardOutput=false;

proc.ErrorDialog=false;

//The file that will be use to capture errors thrown by BCP
strOutPutFileName=ConfigurationSettings.AppSettings["FormatFilePath"]+"RecordsProcessed"
```

```
+ConfigurationSettings.AppSettings["ErrorFileNamePrefix"] + strLoaderName+".txt";

strBCPErrorFileName=ConfigurationSettings.AppSettings["FormatFilePath"]

+ConfigurationSettings.AppSettings["ErrorFileNamePrefix"] + strLoaderName + ".txt";

proc.Arguments=strLoaderName + @" IN " + strReceivedFileName.ToString()

+ ConfigurationSettings.AppSettings["BCPSwitches"].Replace("|","\\")

+ objLibrary.Decrypt(objLibrary.Read("CONNECTION_BCP"))

+ " -o" + strOutPutFileName + " -e " + strBCPErrorFileName + " -L " + dblRecordCount+1

+ " -f" + proc.UseShellExecute=false;

proc.RedirectStandardOutput=false;

proc.ErrorDialog=false;

//Begin process asynchronously

p = System.Diagnostics.Process.Start(proc);
```

5. Another set of format files are created to upload all failed records into an exception management table. The static table contains the primary column number for all the staging tables, which is used to create these format files. For example if the primary key is the 6th column then the following format file is created. The execution is same as above except this time all failed records are loaded into exception table.

5.6.

8.0

9

```
1 SQLCHAR 0 0 "\n" 0 " " "
2 SQLCHAR 0 0 "\t" 0 a0 Finnish_Swedish_CS_AS
3 SQLCHAR 0 0 "\t" 0 a1 Finnish_Swedish_CS_AS
4 SQLCHAR 0 0 "\t" 0 a2 Finnish_Swedish_CS_AS
5 SQLCHAR 0 0 "\t" 0 a3 Finnish_Swedish_CS_AS
6 SQLCHAR 0 0 "\t" 0 a4 Finnish_Swedish_CS_AS
7 SQLCHAR 0 0 "\t" 0 a5 Finnish_Swedish_CS_AS
8 SQLCHAR 0 0 "\t" 1 a6 Finnish_Swedish_CS_AS
9 SQLCHAR 0 0 "\r" 0 a8 Finnish_Swedish_CS_AS
```

6. Use Bulk Copy Program (BCP) with the created format file to upload failed data from error log files into exception management table

```
//load error records
```



```
System.Diagnostics.ProcessStartInfo proc = new System.Diagnostics.ProcessStartInfo(strBCPPath);

proc.Arguments="FAILEDRECORDS " + @" IN " + ConfigurationSettings.AppSettings["FormatFilePath"]

    + "Log"+oTable.Rows[j].ItemArray.GetValue(0).ToString()+"_Load.txt"

    + ConfigurationSettings.AppSettings["BCPErrorLoadSwitches"]+

@" ""\t"" "

    + objLibrary.Decrypt(objLibrary.Read("CONNECTION_BCP"))

//+ " -o" + strOutPutFileName + " -e " + strBCPErrorFileName + " -L " + dblRecordCount+1 +

// " -f" + Get_Create_Format_File(SQLCon,strLoaderName);

    + " -f" + strFormatFilename;
```

7. After loading all data files into the staging and exception management tables (All failed records), a Normalizing process cleans up the staging tables using the exception management table. My requirement is that any failed record must be taken out from all extracts before loading them into production table. The reason is to ensure that no partial records are loaded. So another process reads FailedRecords and then delete records from all staging tables containing these values. This is another way for normalizing before uploading into production tables.
8. Finally, we read the static table and execute stored procedures to load data from staging table into production table System.Threading used for multithreading. (Code available at www.sqlservercentral.com)

Conclusions

My objective was to create an application that is very flexible, good performance and scalable. Although the above design sounds complicated, it accomplished all required goals. Now the daily feeds process all eighteen files at the same time and loads them into production tables very efficiently.

Overview of SSIS

By Amit Lohia

There are many articles on SSIS focusing on different aspects of SSIS. When I started learning SSIS I came across many great articles and found a solution to almost all of the problems I faced. This article is nothing more than compiling all the blog/help files and providing a comprehensive view of SSIS.

Let's start

SSIS is not an upgrade to DTS. It is a replacement for DTS. SSIS should not be considered just a part of SQL Server. It is very powerful and should be treated with respect. SSIS Service is installed as a **single service** on the server even if we have multiple **instances** of SQL Server. In other words it is a shared service across multiple instances. We can manage the SSIS service via SQL Server Configuration Manager



SQL Server Management Studio (SMS)

- Migrates DTS packages into SQL Server 2005
- Run/ Schedule Packages
- Assign Package Security
- View Packages

BIDS ("SQL Server Business Intelligence Development Studio") - This is nothing but Visual Studio. Imagine the number of hours spent in a meeting in a conference room at Microsoft to come up with this name.

- Manage, Develop and Edit Package
- Deploy Package

SQL Server Configuration Manager

- To manage SSIS service

Accessing SSIS via SQL Server Management Studio (SSMS)

We can connect to SSIS Service for various reasons via SSMS. To connect to SSIS we have to mention the server name without the instance of SQL Server. SSMS will connect to the default instance of SQL Server (if specified, else it would error out).

Now the question is how can we manage packages on different Instances? For that we need to change a configuration file. By default, the file is located in the folder, Program Files\Microsoft SQL Server\90\DTS\Binn, and the file name is MsDtsSrvr.ini.xml. The default configuration file contains the following settings:

- For Integration Services in SMS (Object Explorer) are the MSDB and File System folders.
- The packages in the file system that the SSIS Service manages are located in Program Files\Microsoft SQL Server\90\DTS\Packages.

You can modify the configuration file to display additional folders in Object Explorer, or to specify a different folder or additional folders in the file system to be managed by SSIS service. The example below shows how to configure the MsDtsSrvr.ini.xml to use more than one MSDB database, which are stored in different database instances.

Here is configuration file after the changes.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<DtsServiceConfiguration xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StopExecutingPackagesOnShutdown>true</StopExecutingPackagesOnShutdown>

  <TopLevelFolders>

    <Folder xsi:type="SqlServerFolder">

      <Name>MSDB DEV01</Name>

      <Server Name>\<Instance Name 1></ServerName>

    </Folder>

    <Folder xsi:type="SqlServerFolder">

      <Name>MSDB DEV02</Name>

      <Server Name>\<Instance Name 2></ServerName>

    </Folder>

    <Folder xsi:type="FileSystemFolder">

      <Name>File System</Name>

      <StorePath>..\Packages</StorePath>

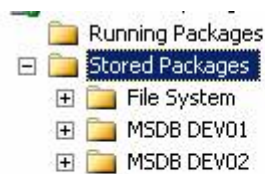
    </Folder>

  </TopLevelFolders>

</DtsServiceConfiguration>
```

Note: We have to restart the Integration Service after the changes are done to the file

You will now notice that we can see 2 different folders for MSDB



With this, we can manage multiple instances of SQL Server for SSIS. Check this link :

<http://bloggingabout.net/blogs/mglaser/archive/2007/03/01/multiple-sql-server-integration-services-ssis-database-instances-on-one-machine.aspx>. This can also give us the advantage to manage the Packages for different instance from a different location.

To access SSIS remotely we should have RPC port open (Port 135). Here are few links which can help in case you are not able to connect remotely:

http://sqljunkies.com/WebLog/knight_reign/archive/2006/01/05/17769.aspx and
http://www.sqlteam.com/forums/topic.asp?TOPIC_ID=62541

Executing the SSIS package from Command Prompt

The **dtexec** command prompt utility is used to configure and execute SQL Server 2005 Integration Services (SSIS) packages. The **dtexec** utility provides access to all the package configuration and execution features, such as connections, properties, variables, logging, and progress indicators. The **dtexec** utility lets you load packages from three sources: a Microsoft SQL Server database, the SSIS service, and the file system.

The command of dtexec is very simple, just a matter of getting used to it. Most commonly used will be the /set command which is used to assign values to the properties of the package / task or to assign values to variables. Need more on dtexec utility? Try this link: <http://msdn2.microsoft.com/en-us/library/ms162810.aspx>

To get a better idea of the variables and properties and how to assign them from the command line we will go through a small example. Let's add a new data source. Right Click Data Source -> New Data Source -> Complete the Wizard (Use SQL server Authentication). Go to the properties window for the newly created data source

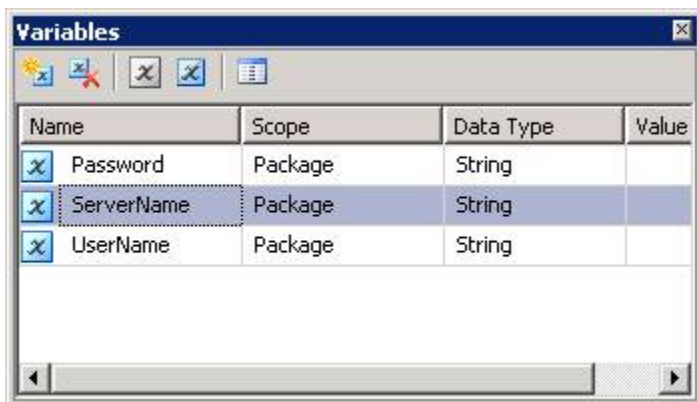
Note: To view properties window use "F4"

In the expression properties of the data source add ConnectionString as property and the following line in the expression.

```
"Data Source=" + @[User::ServerName] + ";User ID=" + @[User::UserName]  
+ ";Password=" + @[User::Password] + ";Initial Catalog=msdb;Provider=SQLOLEDB.1;"
```

You will notice we have 3 variables in the expression. We will add the above variable and then assign the values from command line. To add variables open the variable window (2 ways to open variables window)

- SSIS -> Variables
- View -> Other Window -> Variables



Now in the variables window add the above variables, just keep the scope as default ("Package") and change the data type to "String". Now we can change the value of the variable from the command prompt, which will change the connectionstring of our data source.

Syntax for command line

```
dtexec /f "D:\PacakageName"
```

```
/set \Package.Variables[User::ServerName].Properties[Value];< ServerName>
```

```
/set \Package.Variables[User::UserName].Properties[Value];<UserName>
```

```
/set \Package.Variables[User::Password].Properties[Value];< Password>
```

Other method of changing the connectionstring without the variables or expression is to manipulate the property of our data source directly. Here is the command line syntax:

```
dtexec /f " D:\PacakageName"
```

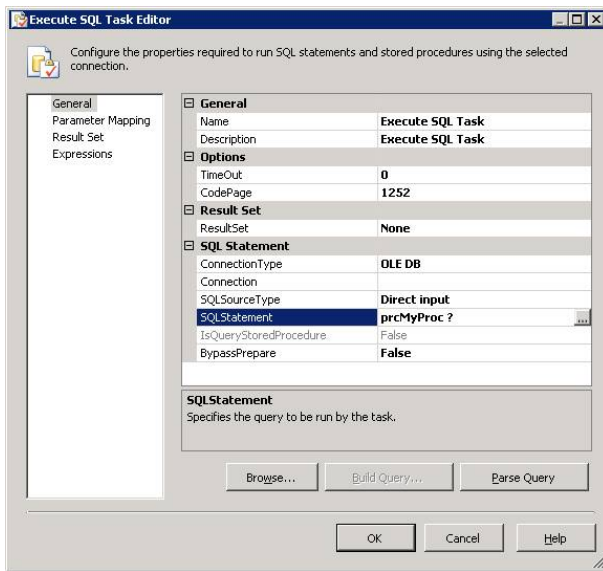
```
/set \Package.Connections[DataSourceName].Properties[ServerName];<ServerName>
```

```
/set \Package.Connections[DataSourceName].Properties[UserName];<UserName>
```

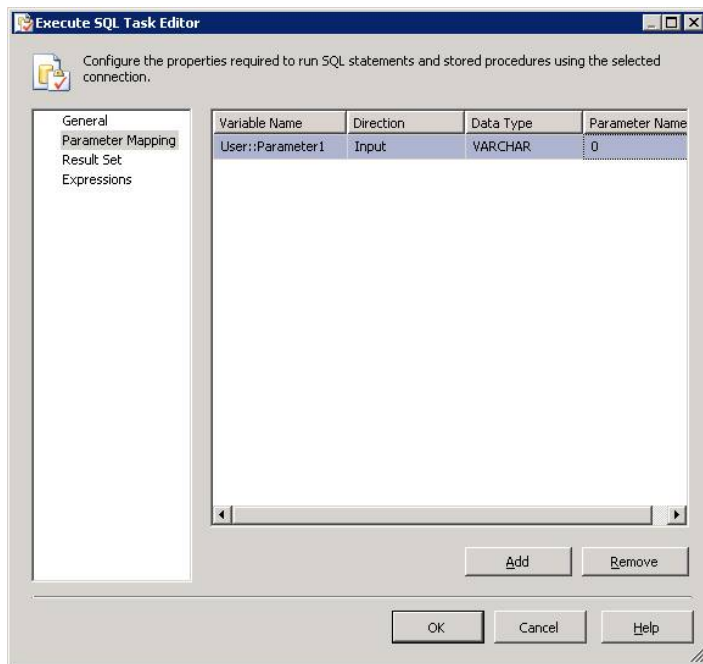
```
/set \Package.Connections[DataSourceName].Properties[Password];<Password>
```

Passing variable as a parameter in a stored procedure

Add a new SQL Server Task and go to the property window (F4 key).



Add the proc name in SQLStatement and add "?" for the parameter and then go to parameter Mapping (make sure you have the connection set in the previous screen).



Add a new parameter Mapping, change to the appropriate Data Type and the Parameter Name to 0 (this determine the ordinal position). For more information on Parameter Mapping: <http://msdn2.microsoft.com/en-us/library/ms187685.aspx>. The best way to find the property path of a property is to create an XML configuration file.

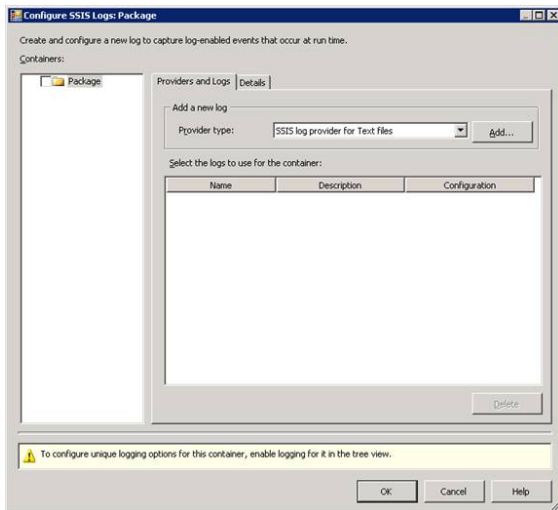
Logging Package Information

SSIS comes with some predefined ways to log package information. We can log the information to the following 5 providers (excluding the log window)

- Text File
- SQL Server Profiler
- SQL Server (Table)
- Windows Event Log
- XML File.

Logging is disabled by default. We have to enable it at the package level. We can log the information at the package level or at the task level.

To enable logging go to the property window of package and change the LoggingMode to enable. You can change the loggingMode at the task level based on your requirement. Now to select the destination for logging information we need to right click on the control flow (anywhere) and then click Logging, which will open the following window

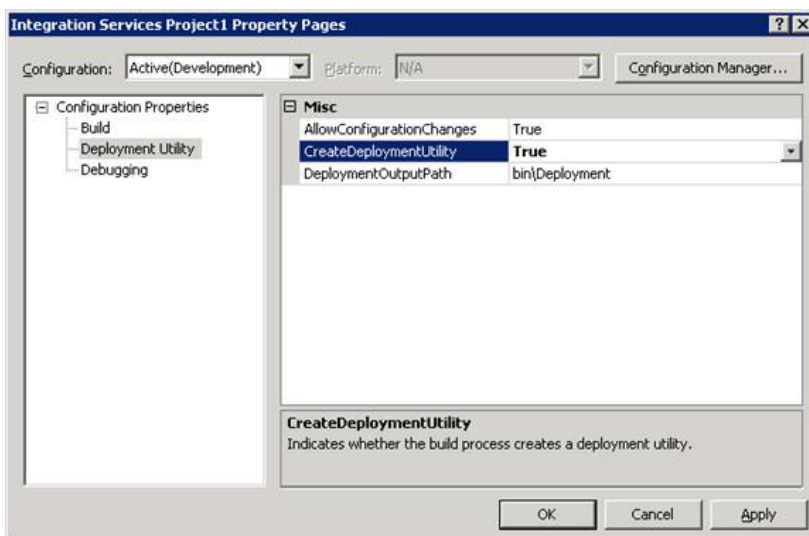


Choose the provider type and then Click Add. Complete the wizards for that provider and the logging will happen. If you click on the details tab you will notice below we can select the events and the information which we need to capture. We can save this configuration in XML by clicking the save or can apply the configuration based on previously saved XML file by clicking Load.

One thing confusing, was the name of the table which logs the information? The name of the table is sysdtslog90 (it will be created in the database which was specified in the connection string). If the table is already created it will append the information.

Deploying the package

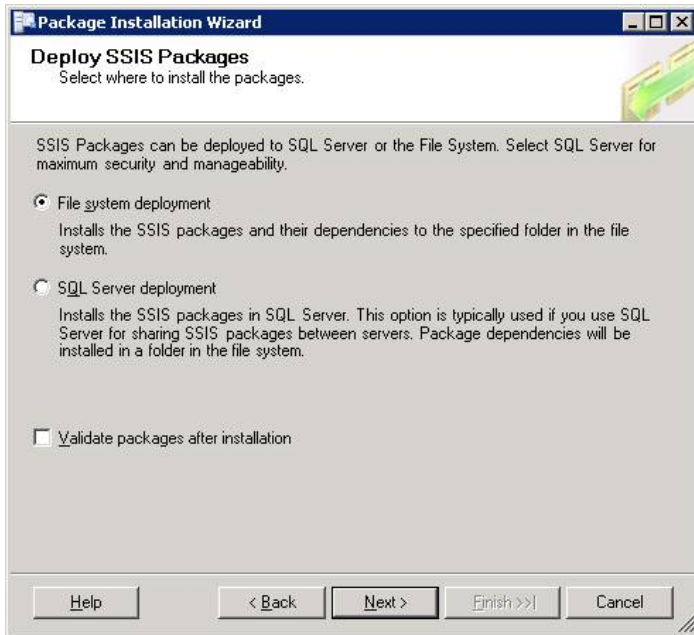
Once the package is ready for production we can deploy using the following steps. Go to the package Property Projects -> Project Properties which will open the following window.



By default the CreateDeploymentUtility will be false, change it to True and hit ok. Now Build the Project Build -> Build <Project Name>. Now go to the location where you have the package (not sure where it is?) Check the property window of the package (press F4) and the location Path. Go to that folder from window explorer and you will see a

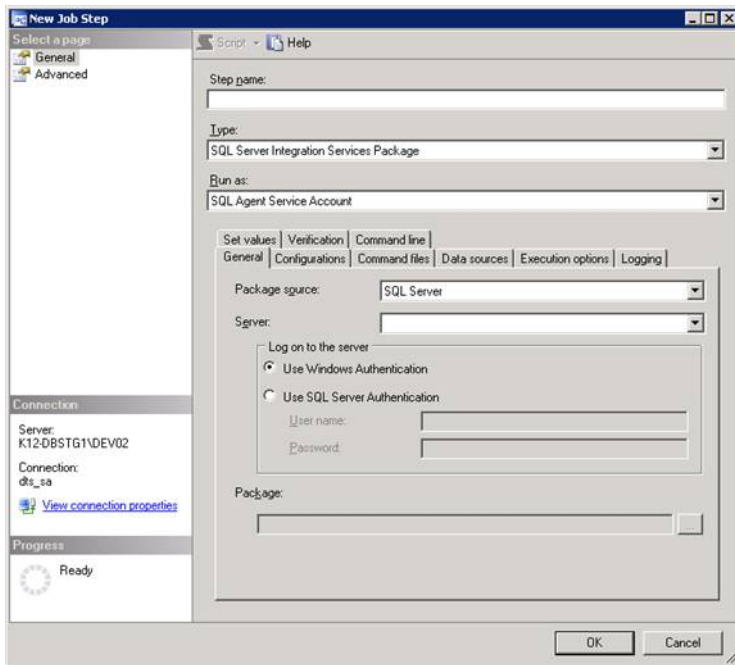
folder as Bin and under that you will see the Deployment folder (this is the same path which you see in DeploymentOutputPath in the above screen).

You will see a file name (your project name); file type will be Development Manifest. Double click on the file and it will open the Deployment wizard. After the welcome screen, the next screen will give you 2 options, select the one where you want to deploy and complete the wizard.



Running the package from a SQL Agent job

While adding a new step in the job now we have a new type as SSIS package



Complete the above screen and we are set to go.

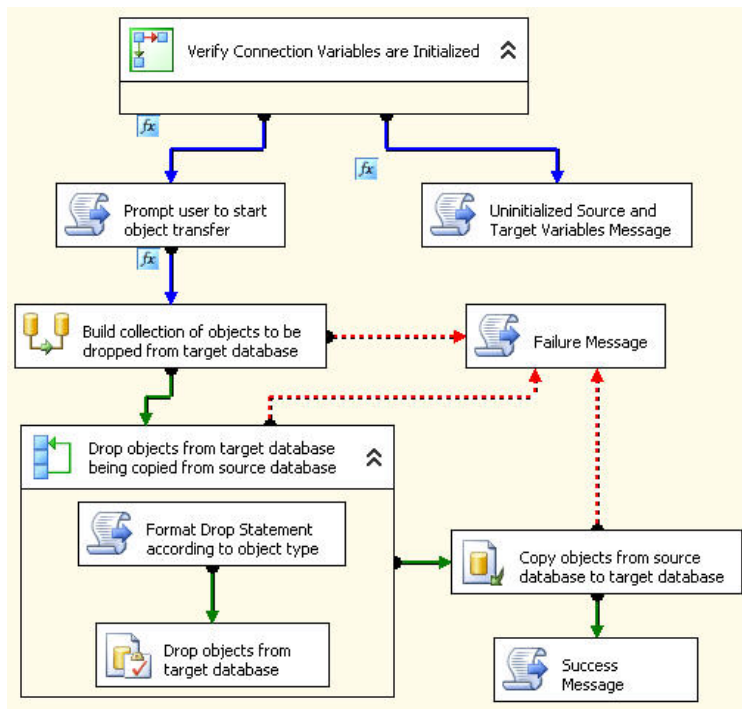
SSIS - Transfer SQL Server Objects Debugged

By Stephen Laplante

I have recently received SSIS training and have had the opportunity to write my first package. As a software developer, I know how hard it is to write perfect code but I ran into some rather significant problems that I find hard to believe an organization with the talent and resources of Microsoft could have missed. I am quite impressed with the *published* capabilities of SSIS but after writing my first internally important package (which ultimately failed), I was quite embarrassed.

I'm not the type to point a finger very quickly as I have made my fair share of mistakes in the past. In an effort to make my experience beneficial to others, I thought I would put something together to document my experience, showing how I went about identifying each problem and ultimately resolving them. I have also included two SSIS packages to help others who are trying to create a package to perform a similar task. Let me begin by explaining the purpose of my package and the general concept of how it works.

The package is responsible for copying Stored Procedures and User Functions from one Database to another. I saw that Microsoft had provided a "Transfer SQL Server Objects" task which appeared it would be able to do what I needed all by itself. After completing the package, my first test resulted with no errors, but also did not copy any objects to the destination database either. So I reached into the developer's bag of tricks and pulled out the single most valuable tool in the diagnostic arsenal, Google. After a half hour of reading articles, I found what I was looking for; I needed to set the "CopySchema" property to True. Why Microsoft gives me a property that if not set causes the task to do nothing is beyond me, but easy enough to fix.

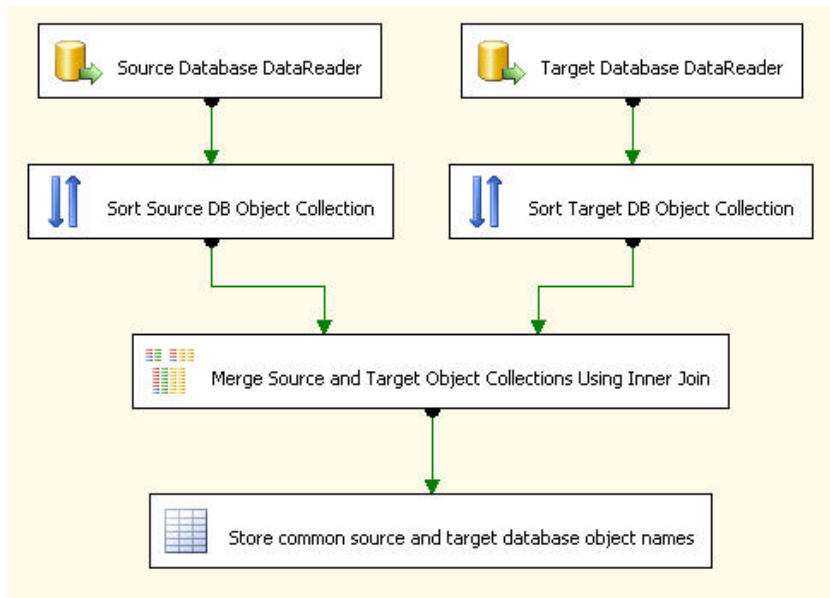


Package Control Flow Diagram

So I deleted all the Stored Procedures and User Functions from the destination database and ran the package again. My second test also ended in failure giving me an error telling me that Procedure "xxx" already existed at the destination database. With a quick look at the Task Editor, the "DropObjectsFirst" property jumped out at me and I thought, OK, that error makes sense; you need to drop the procedures with the same name from the destination database before you can copy them over from the source database. I figured if I set this property to True that would resolve the problem. When I ran the package for a third time, I got an error telling me that Procedure "yyy" did NOT exist at the destination database.

So I had discovered the first problem I was not sure how to resolve. It made sense to me to get an error when the procedure already existed at the destination database and the 'DropObjectsFirst' property is set to 'False'. When you set it to 'True', you would expect the behavior to be that any procedures being copied from the source database that did not exist on the destination database should just be silently transferred. After all, the whole point of setting the property to 'True' is to ensure that procedures don't exist at the destination database that are being transferred from the source database. I would expect it would be common occurrences to have new procedures on the source database you were trying to copy over to the destination database. You would think Microsoft would have thought of such a simple test right?

I went back to Google to see if anyone else had seen this problem and what ideas were floating out there to work around it. What I found was a suggestion to write a script task to handle dropping the procedures from the destination database before running the Transfer SQL Server Objects task. I figured it wasn't that much work to write some code to handle this so I rolled up my sleeves and got busy writing a Data Flow task to get a collection of procedures and functions from each database and merge the collections through an inner join to derive a collection of objects common to both databases. I passed this collection to a For Each Loop task and then inside the loop, used a script task to create the Drop statement which was sent to an "Execute SQL" Task.



Data Flow Task to Build Collection of Common Database Objects

Everything seemed to be working perfectly until I reached the "Transfer SQL Server Objects" task in the control flow. This time I had successfully transferred 25 procedures and on the 26th procedure I got an error telling me the procedure already existed at the destination database. I thought how can this be? I started with an empty destination database; there were no procedures in the database to begin with so how can there be a duplicate procedure?

Well, after a couple of days of Googling, testing and debugging, I finally found the problem after running a trace on the destination database. The trace file revealed that the Transfer SQL Server Objects task was attempting to create 2 procedures with the same name at the destination database. I thought how is this possible? You can't have 2 procedures with the same name on the same database instance. Right away I suspected I was not connecting to the source database I thought I was. I verified all my server and database settings and was truly at a diagnostic dead end.

For a sanity check, I exported all the stored procedures from the source database into a single file and using a text search, I looked for the procedure name that was being created twice. The name was only found in the file once so that ruled out the possibility of a procedure creating another from within itself with this procedure's name. So if there is only one create statement for this procedure, then where is the second procedure coming from?

A little more digging in the trace file revealed the answer; the task was trying to create two versions of the same procedure on the destination database. A careful examination of the text in each of the procedures (which was fortunately available in the trace file) showed that one of these versions was in fact the previous version that had originally been created on the source database but was later replaced by the second version of the procedure.

At first, I thought the underlying database was somehow keeping a version history of procedures that had been copied and renamed so I made every attempt to locate both versions of the procedure in the source database by looking at every view I could find relating to the stored procedure text. There was only the one, current version of the procedure visible to me on the source database. So now I knew how it was happening but didn't really know 'what' exactly was happening. My thought was that this couldn't be a SQL database bug since the views available to me show only the correct version of the procedure. That suggested to me that however the Task was getting the procedure list and associated text, must not be using the views I had available to me.

Assuming I was dealing with a bug internal to the Task, I thought I would just set the "MaximumErrorCount" property to something ridiculous like 999999 and the task would just ignore cases where the procedure already existed and

continue transferring the remaining procedures. Unfortunately, the "MaximumErrorCount" property had absolutely no effect on the operation of the task. As soon as it got the first error, it threw an exception and aborted the transfer.

Just to be sure I didn't have something funky in my database, I tried this against another database and got some surprising results, this time, I got 3 procedures transferred to the destination database that don't even exist on the source database!!! To be sure I didn't make a mistake, I deleted all the procedures from my destination database, ran the package (which would work fine with "this" database if I would delete all procedures first) and sure enough, there were 3 procedures that did not exist on the source database.

Suspecting that however unlikely it could be that I had two bad databases, I decided to create two fresh test databases and then using the script file I had created earlier, I loaded all the procedures from the first source database I had been working with into my new test source database. I ran the package on the 2 new test databases and it worked perfectly 10 out of 10 times! Then I decided to keep pointing at my test destination database, but changed the source database to point to each of the other two source databases that had previously failed. This resulted in the same failures from both of them as I had observed before.

I thought to myself, this task is fundamentally flawed and absolutely unusable. How could Microsoft miss such basic and fundamental error checking? If I would have released a piece of software this flawed to my organization, I would have been flogged and publicly humiliated. This is just plain shoddy work that should have never passed into the release phase of development. With all the money Microsoft pulls in, I think it's high time they started testing their own software instead of dumping it on the development community to do for them (*for free).

Now I set out to write my own task to handle the object transfer between databases. I didn't know it at the time, but this would prove to be the silver bullet in identifying the real cause of the problem I was having and it wasn't a bug in the Transfer SQL Server Objects task as I had originally suspected (although the task does have some of those as well).

Once I completed coding my task and ran the package, I got the exact same results as the Microsoft task did. I was quite surprised by this but this time, I could get under the hood of the task with debugging tools and see what was really going on. I discovered it wasn't that an older version of the procedure was somehow being stored in the underlying database; it was that the name of the procedure and the name referenced in the procedure text did not match. The name of the procedure in the definition text, which is used to create the procedure, was in fact the same as another procedure so it was in fact a duplicate procedure.

Now I could see 'what' was happening but 'why' was still a bit of a mystery. In the case I was investigating, it just so happened that the older version of the procedure I had seen in the trace file was in fact an older version of the procedure that had been renamed from 'xxx' to 'xxx_old'. This is what sent me down the wrong path, just some dumb coincidence. When I looked in the sys.objects view from the database system views, I found the procedure named 'xxx_old' as expected but after adding the join with the sys.sql_modules view to get at the procedure definition text, I saw the name in the "Create Procedure" line was 'xxx'. That was the problem! The procedure 'Name' and procedure name referenced in the definition text did not match.

Since I had a procedure named 'xxx' and a procedure named 'xxx_old' being transferred from the source database, when the 'xxx_old' procedure script was executed, it was actually creating a procedure named 'xxx'. This is what threw me off when I was debugging the Microsoft task; they used the name of the procedure being created (from the definition text) rather than the visible name of the procedure being transferred in the error message. The error message made it appear that a duplicate procedure was being copied from the source database. I don't fault Microsoft for writing the error message the way they did, I agree with their reasoning, it was just an unfortunate scenario that prevented me from being able to correctly identify the problem.

So now that I knew what the problem was and why it was happening, I set out to duplicate the problem myself. So I went back to my test databases that worked just fine and made a copy of one of the procedures and renamed it in the Object window using the right mouse rename menu item. As expected, the name was changed in the sys.objects view 'name' column but in the sys.sql_modules view 'definition' column, the old procedure name was still in the create statement.

Now that I had identified the problem and was able to successfully duplicate the problem, it was now time to figure out a way to solve the problem so I could make this package work. The solution that came to mind was to add code to the script task that was creating the drop statement to search the definition text for the 'Create Procedure' line and parse out the procedure name. Then I compared the name from the definition text to the name stored in the sys.objects table and if there was a mismatch, the definition text procedure name was changed to match the name from the sys.objects view.

This time when I ran the package, I was able to successfully transfer all the stored procedures and user functions from both of the databases I was not able to get to work using the Microsoft Transfer SQL Server Objects task. I was also able to confirm that this time, the procedure name and the name referenced in the definition were identical.

You can download some [packages and code here](#).

For Microsoft to allow you to rename a stored procedure in the object window of the SQL Server Management Studio and not make the necessary changes to all tables in the underlying database is a major oversight and should be corrected immediately. One of the potential problems that could occur as a result of this bug would be that if you simply renamed an older version of a procedure in the Object window of SQL Server Management Studio, the old version could potentially overwrite the current version because the internal names matched. Now you have the old version of a procedure masquerading as the new version.

I hope you found this article helpful and hope the packages will help you work around the current limitations of this particular task as well as provide a base package you can modify and tweak to meet your own needs. If nothing else, the bugs have been exposed and the work around has been made public. Now we can continue to push the limits of the SSIS technology to meet the ever increasing demands of our profession.

SSIS Is Not Just for SQL Server

By Tim Mitchell

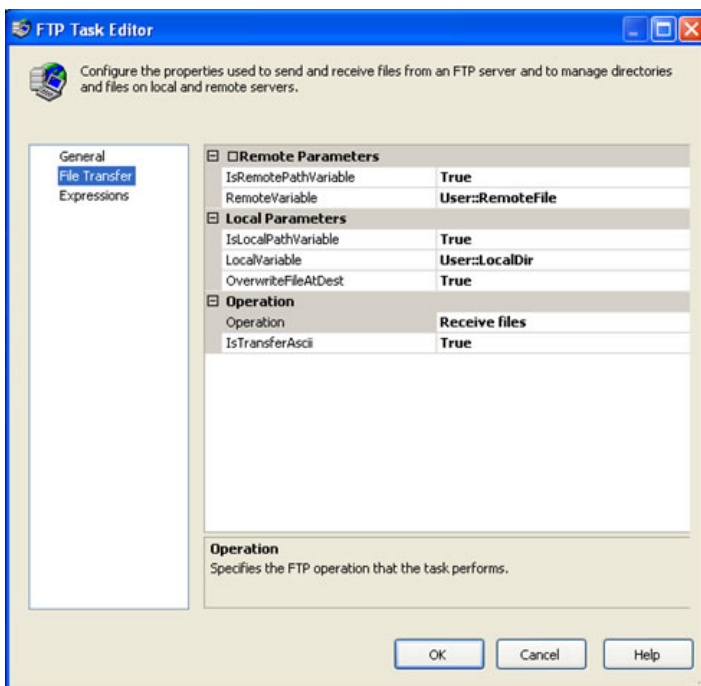
It's been said that when all you have is a hammer, everything looks like a nail. SQL Server Integration Services is just such a tool, and can turn lots of difficult or troublesome tasks - some of which are unrelated to SQL Server data storage and retrieval - into nothing more than simple 16 penny nails. Since I got started with SQL Server Integration Services (SSIS) over a year ago, I've found a number of uses for this product that would have required hours of coding or batch scripting. This article will share a simple use of SSIS for data flow that does not even involve SQL Server proper.

A little background here... I ran into a task that required me to download web log files from a remote web server each day, run an executable to process each file, then archive the log file locally. This was being done manually before the SSIS hammer got involved. Even though this information is external to SQL Server, the nature of the task seemed to be a perfect fit for SSIS. Automating this process was a quick and easy task with Integration Services.

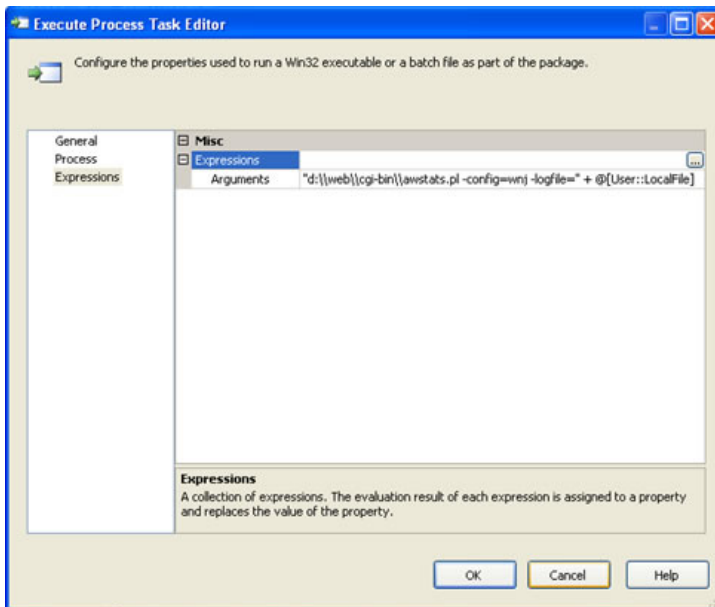
I start off by creating a simple script task whose purpose is to set a couple of package variables. The file name we are to download is for the previous day and is named according to the date. The following snippet shows the logic to set those variables:

```
Public Sub Main()  
    ' Use yesterday's date to process the previous day's report  
    Dim yesterday As DateTime = DateAdd(DateInterval.Day, -1, DateTime.Now)  
  
    Dts.Variables("DateToProcess").Value = yesterday.ToString("yyMMdd")  
    Dts.Variables("LocalFile").Value = Dts.Variables("LocalDir").Value.ToString() &  
    & "ex" & Dts.Variables("DateToProcess").Value.ToString() & ".log"  
    Dts.Variables("RemoteFile").Value = Dts.Variables("RemoteDir").Value.ToString() &  
    & "ex" & Dts.Variables("DateToProcess").Value.ToString() & ".log"  
  
    Dts.TaskResult = Dts.Results.Success  
End Sub
```

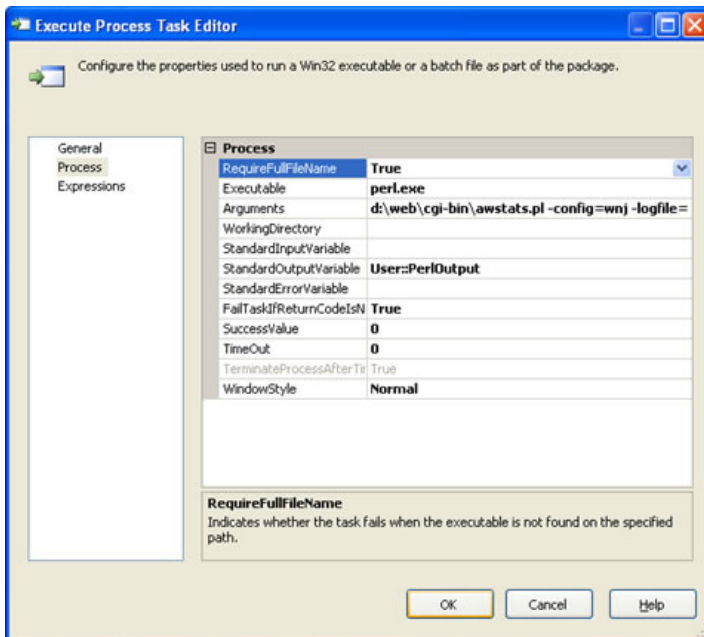
Next, I'll use another simple task, the FTP Task, to download the source file from the FTP server. The RemoteFile name set in the script task above is used to download yesterday's file. You can see that I am using the variables referenced in the above script task to dynamically set the file names in the FTP Task:



The next step is to create an instance of the very useful Execute Process task. This element allows you to call an external executable from within SSIS, and also allows you to specify arguments for those calls. In this case, I need to call the perl.exe executable and pass the name of the Perl script to run along with some other values (including the local file name set earlier). The settings here are relatively straightforward; the only thing that requires a little tweaking is to set the argument properly, since this will need to be dynamic. See below where I use an expression to set the name of the Argument to the local file I need to process:



You can also see that I am capturing the output of the call to perl.exe below, to allow me to review the statistics generated by the script. As shown below, the StandardOutputVariable can be used to capture the command line output and send it to another package variable:



Lastly, I use a Send Mail task to e-mail the output message in the PerlOutput variable to the operator to verify record counts.

Conclusions

SQL Server Integration Services is a very powerful data processing tool, and its uses with SQL Server are almost limitless. Even more, this article has shown that SSIS can be used apart from SQL Server to simplify data processing in heterogeneous environments.

SSIS Programming

By U.K Padmaja

Overview

Most of us have been using Data Transformation Services (DTS) in the previous versions of SQL Server in order to manipulate or move data. With the introduction of SQL Server 2005, Microsoft has introduced a completely re-written version of DTS with a lot more new features which is now known as Integration Services. This new ETL platform is nothing but a set of utilities, applications, designers, components, and services all wrapped up into one powerful software application suite.

The most attracting feature of SSIS is that the data movement and transformation is separate from the package control flow and management. There are two different engines that handle these tasks. Integration Services Data Flow engine takes care of the data movement and transformation whereas the Integration Services Run-time engine manages the package control flow.

The motivation behind this article is that the SSIS API is not well documented in MSDN though you get an overall idea of what is going on. This article assumes that the reader is aware of the fundamental components of an SSIS Package. To know more about what SSIS is all about, check the MSDN Books Online <http://msdn2.microsoft.com/en-us/library/ms141026.aspx>

DTS vs. SSIS Programming

There are two fundamental approaches one can have for SSIS programming. One is to use the SSIS Designer to create and run packages. The other approach is to use the API to create, configure and run packages from own applications.

Those who are familiar with the DTS programming can see that there is very little code from DTS remaining in Integration Services. Some of the similarities are listed here.

DTS	SSIS
Control flow and Data flow on the same design surface	Control flow and Data flow have been separated
Datapump (supported only one source, transform, and destination)	Data Flow Task or Pipeline (can perform extremely complex and advanced data processing within one pipeline)
Step	Taskhost (An expanded version of Step) which schedules the task execution
Precedence Constraints	Still remains with added features
Connections	Connection Managers

Getting Started

Let us get started with some actual coding in VB.NET, my favorite!

The following table lists the assemblies that are frequently used when programming Integration Services using the .NET Framework (<http://msdn2.microsoft.com/en-us/library/ms403344.aspx>).

Assembly	Description
Microsoft.SqlServer.ManagedDTS.dll	Contains the managed run-time engine.
Microsoft.SqlServer.RuntimeWrapper.dll	Contains the primary interop assembly (PIA), or wrapper, for the native run-time engine.
Microsoft.SqlServer.PipelineHost.dll	Contains the managed data flow engine.
Microsoft.SqlServer.PipelineWrapper.dll	Contains the primary interop assembly (PIA), or wrapper, for the native data flow engine.

Consider the example of transferring data from a flat file to a database table in SQL Server 2005. Create the destination table in SQL Server prior to running the package. This can be done using a simple T-SQL statement.

The programmatic approach would include the following steps:

1. Create an SSIS Package - <http://msdn2.microsoft.com/en-us/library/ms135946.aspx>
- b. Create Control flow, which would be as simple as adding a new Data Flow task in our case. - <http://msdn2.microsoft.com/en-us/library/ms135997.aspx>
- c. Add two connection managers. We need a Flat file connection manager for the source and an OleDb or SQL Server connection manager for the destination.

Creating OleDb connection manager is not hard.

```
Dim oleDbConn As ConnectionManager = myPackage.Connections.Add("OLEDB")  
  
oleDbConn.Name = "MyOLEDBConnection"  
  
oleDbConn.ConnectionString = "Provider=SQLOLEDB.1; Integrated Security=SSPI;"  
  
& "Initial Catalog=<db>; Data Source=" & SS_INSTANCE & ";
```

The following code shows creating a flat file connection manager and setting up the formatting information for the same. (code available at www.sqlservercentral.com)

- d. Create Data flow which does the actual job of copying the data from flat file to table. This would involve creating a Flat file source and an OleDb destination. This is fairly standard. We give it for the sake of completeness. (code at www.sqlservercentral.com)
- e. Connect Data flow components - <http://msdn2.microsoft.com/en-us/library/ms136086.aspx>
- f. Map source and destination columns

```
Dim input As IDTSInput90 = DFDestination.InputCollection(0)  
  
Dim destinationInputID As Integer = CInt(input.ID)  
  
Dim vInput As IDTSVirtualInput90 = input.GetVirtualInput()  
  
  
For Each vColumn As IDTSVirtualInputColumn90 In vInput.VirtualInputColumnCollection
```

```
' This will create an input column on the component.

DestInst.SetUsageType(destinationInputID, vInput, vColumn.LineageID, DTSUsageType.UT_READONLY)

' Get input column.

Dim inputColumn As IDTSInputColumn90 =
input.InputColumnCollection.GetInputColumnByLineageID(vColumn.LineageID)

' Getting the corresponding external column.

' Ex : We will use the column name as the basis for matching data flow columns to external
columns.

Dim externalColumn As IDTSEExternalMetadataColumn90 =
input.ExternalMetadataColumnCollection(vColumn.Name)

' Tell the component how to map.

DestInst.MapInputColumn(destinationInputID, inputColumn.ID, externalColumn.ID)

Next
```

g. Validate and execute the package

```
Dim pkgStatus As DTSExecResult = myPackage.Validate(Nothing, Nothing, Nothing, Nothing)

If pkgStatus = DTSExecResult.Success Then

    Dim pkgResult As DTSExecResult = myPackage.Execute()

End If
```

Notes

1: For debugging purposes it is a good idea to save the SSIS package created programmatically which can then be exported to the Designer. This would allow us to check whether the properties of the connection managers etc are set properly. See the link for different options - <http://msdn2.microsoft.com/en-us/library/ms403347.aspx>

2: To get a list of data types that SSIS uses, check <http://msdn2.microsoft.com/en-us/library/ms141036.aspx>

3: In case a subset of columns from the flat file needs to be transferred to the destination table, then one can identify such columns in Step c, and then delete them from Output Collection of the source in Step d before creating the external columns

[Download the code](#)

Conclusion

SQL Server Integration Services, the new ETL platform in SQL Server 2005, is the successor to DTS, which was there in previous versions of SQL Server. SSIS with all its new features is a huge topic in itself and we tried to look at some of the programming aspects of it. We have looked at how we can programmatically create a package to transfer data from a Flat file source to a database table. Interested readers can look at the MSDN for more information on SSIS. A couple of books that I have come across are also worth reading

1. Microsoft SQL Server 2005 Integration Services by Kirk Haselden
 2. Professional SQL Server 2005 Integration Services by Brian Knight et al
 3. Microsoft SQL Server 2005: A Developers Guide by Michael Otey et al.
-

Loading a 24x7 Data Warehouse

By Leo Peysakhovich

Extract-Transform-Load (**ETL**) is a process that is used to take information from one or more sources, normalize it in some way to some convenient schema, and then insert it into some other repository.

A common use is for data warehousing, where regular updates from one or more systems are merged and refined so that analysis can be done using more specialized tools. Typically, the same process is run over and over as new data appears in the source application(s). Many data warehouses also incorporate data from non-OLTP systems, such as text files, legacy systems, and spreadsheets; such data also requires extraction, transformation, and loading. In its simplest form, ETL is the process of copying data from one database to another. This simplicity is rarely found in data warehouse implementations. ETL is often a complex combination of process and technology that consumes a significant portion of the data warehouse development efforts and requires the skills of business analysts, database designers, and application developers.

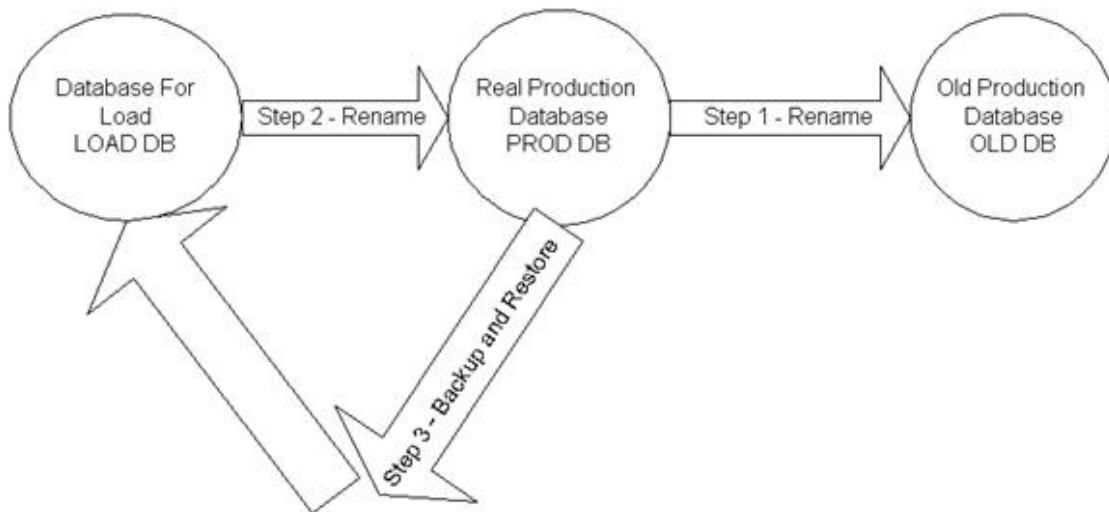
ETL process is not a one-time event; new data is added to a data warehouse periodically. Many companies have data warehouses that are loaded nightly and used as READ ONLY databases for the applications during regular business hours. ETL processes must be automated and documented. The data warehouse is often taken *offline during update operations*. But what if data warehouse database used 24*7 even traffic at a load time is very low? It means that the database can't be taken offline during the load! This is the reality of my company and this presents some challenges for the DBA group to setup processes with following criteria:

- database must be online 24*7
- data (tables) can't be locked
- data must be consistent at any time. E.g. it can't be time when data is loaded partially
- If load is failed then the previous day data must be returned

Also, there are some other restrictions that require special architecture for the nightly ETL process.

Let's start with the fact that the ETL job itself is very complicated and consists of 60+ individual steps. If at least one step fails the whole job should fail and database should keep the previous day's data. The challenge to control data includes the fact that the load process can't be transactional because the data warehouse database used 24*7 even traffic at night time is very low. This means that the database can't be locked or placed offline. At the same time, load can't leave partially loaded data in case of error and/or partially loaded data for users while load is running. It requires mentioning that the ETL process usually runs for 30-60 minutes based on the number of daily made changes.

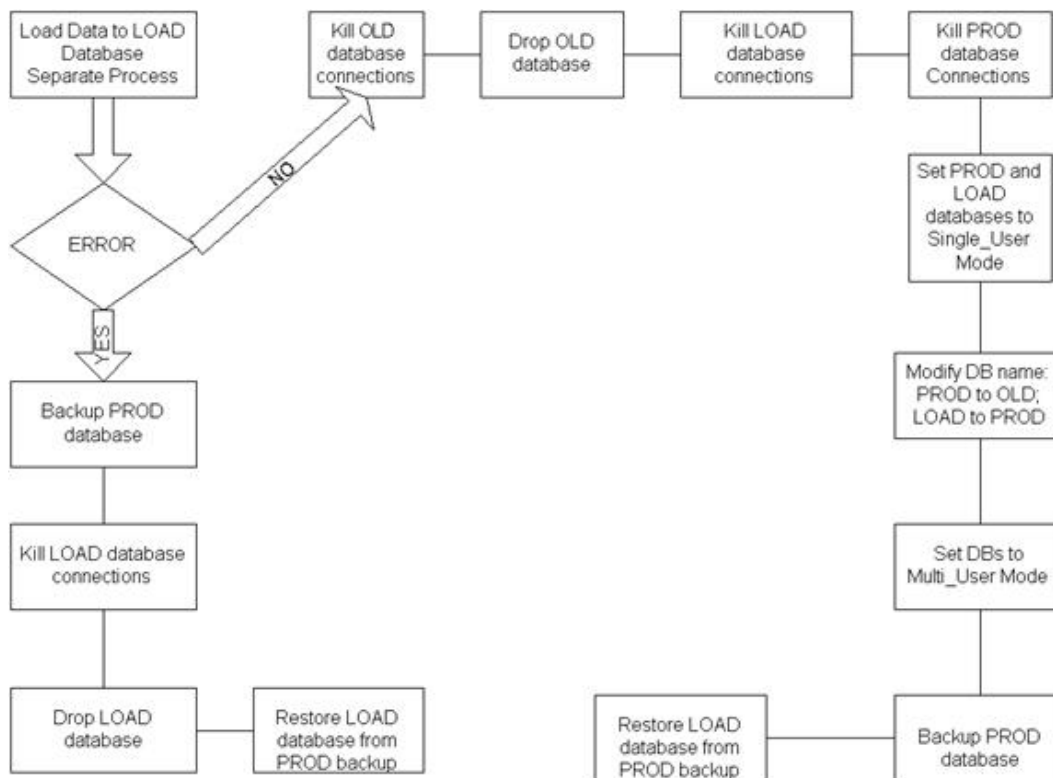
After many hours of thinking I came up with this idea for the load based on the fact that this is the data warehouse and data changes ONLY with ETL process once a day. I decided keeping 2 the same databases on the server. Let's call them LOAD and PROD databases.



LOAD and PROD databases are the same at the time ETL process is started. ETL starts loading data to the LOAD database. If load is successful then LOAD database keeps the new data and PROD keeps the previous day's data. The next two steps will be done consecutively with no delay time in between.

Step 1 and 2 rename production database to the OLD database and rename LOAD database to PROD database. The rename process takes less than a second. Step 3 backup production database and restore LOAD database from PROD backup to prepare for the next day's load. At the end we have the previous day's data in OLD database, current day's data in PROD database, and current day's data ready for the next day's load.

If you can't afford to keep 3 databases because of drive space restrictions or some other factors, then OLD database can be dropped. You don't need to restore LOAD database until the next load is started and it can be the first step for the ETL process. The picture below shows the whole process logic.



Let's see the code for the parts of the process.

Step - Kill database connections. (Code at www.sqlservercentral.com)

Step Set database to Single User Mode

SINGLE_USER | RESTRICTED_USER | MULTI_USER controls which users may access the database. When SINGLE_USER is specified, only one user at a time can access the database. MULTI_USER returns the database to its normal operating state

```
ALTER DATABASE PROD SET SINGLE_USER with rollback immediate
```

You need to remember that ALTER DATABASE permissions default to members of the **sysadmin** and **dbcreator** fixed server roles, and to members of the **db_owner** fixed database roles. These permissions are not transferable.

ROLLBACK IMMEDIATE specifies whether to roll back after the specified number of seconds or immediately. If the termination clause is omitted, transactions are allowed to commit or roll back on their own. Remember that this database is a data warehouse and used as READ ONLY source.

Step Modify database name

```
Alter database PROD modify name = OLD
```

Step Set database to Multiuser mode

```
ALTER DATABASE PROD set MULTI_USER
```

Step backup database to the database device

Backup database PROD to PROD_BAK with INIT

The step to restore LOAD database from backup file presents some challenge because each time databases are renamed or restored the physical file names should have unique name's.. The code below illustrate the example of the step restore from device LOAD_BAK and files named based on the database name LOAD, OLD, and PROD (code at www.sqlservercentral.com)

This is the logic if load is successful. If ETL process fails then PROD database must be backed up and the LOAD database must be restored from PROD backup (see picture above). You may notice from the process that the user's requests will be cut from the database during renaming process. This is true, but remember, it takes less than a second to switch the database's names and the user has to have active request during this time. Most of our requests are very short ,2-3 seconds, and traffic is very low at night time. During our tests we prove that the worst case scenario for the user will be an application message stated that user has to hit "Refresh" button to get the result.

Conclusion

The process may have some additional logic. For example, if you would like to keep the LOAD database when load fails, then the steps to rename LOAD to ERR database can be added to the process. It will allow you to make the load analysis and to find an answer for the question "why the load failed" easier. In our case, we added some additional data verification and analysis steps to the process after the ETL is completed to verify the data integrity and business rules. If the fatal data rules violation is found then ERROR part of the process is started and loaded database renamed to ERR. LOAD database is restored from the PROD database. Next day DBA is analyzing the data issues from ERR database. But the process of data verification and analysis is the topic for my next article.

We using this architecture for 6 months and there are no issues or user's complains.

More Problems with Data Warehousing

By Richard Gardner

I was interested in the sheer number of people who had read Janet Wong's recent article, [Problems In Building a Data Warehouse](#). Reading into a couple of the [replies](#), it seems that one thing which people were interested in were the "gotchas" from implementing a data warehousing system, and having the most technophobic user base in the universe I thought I could highlight a few.

The most important thing to bear in mind is clean data. You have to remember that Data Warehousing is like a pyramid, your base data makes up the base of the pyramid, and you need to fine hone this to get to the point. In getting to the point I found a few problems.

Make Sure You Have a Universal Primary Key Across All Data Sources

Obvious isn't it? But in my case I was a couple of weeks in before I realized one of the systems I was integrating didn't have one (it looked like it did, but actually there were subtle differences). Luckily there was a text field that I could use with the SSIS fuzzy logic task to match for my pilot proof. In the end it took 2 months for the users to go back and change their systems to fix this for good.

Make Sure You Know How Reliable your Base Information is

This can't be stressed enough. No system is 100% accurate, and bear in mind that a 97% accurate report combined with a 97% accurate report will probably not make a 97% accurate report; most likely it will be 94% accurate.

If you find information in two places, work out which is the most reliable and try to get rid of the other one. It's kind of like having two cigarette lighters, front door keys or pairs of glasses. Because you have a spare you don't expend as much energy keeping track of either of them as you do when you only have one; it's just human nature. Also, try and work out if one of the systems has checks and balances which help to keep things up to date.

As an example we have an ink system which records the cost of the ink we buy. We also have a finance system where we raise purchase orders to buy ink. It makes far more sense to get the ink cost from the finance system because if we get the price wrong in the finance system the invoices from the suppliers don't match the purchase order and it soon gets sorted out. If the price is wrong in the ink system nothing bad happens and nobody cares.

Until your data warehousing project fails. Then it's your fault.

Some systems are just not right; the data is bad and they are poorly looked after. Of course, a re-implementation project can help to get these straightened out, but you are likely to be quite a way into your warehousing project before you realise just how not right they are. Also, re-implementation helps going forward, but what about the historical data? Do you just throw it away?

The key here is to look for the person with the spreadsheets who is responsible for getting meaningful data about the part of the business this system controls. Chances are he drags a report from the system into Excel then uses a combination of algorithms to come up with his figures. For example if this value is 0 this record is clearly rubbish; if this is greater than 10 then this is clearly rubbish. If you're a glorified hacker like me you can use a series of very basic statistical techniques and where clauses to match this guys logic and get to a pretty good average. If you're a stats genius then you'll probably do a much better job.

Compare Like For Like

Even in something as simple as a CRM system the report "Show me all customers that bought X in the last 3 months" can be inaccurate when combined with other data are you talking about the invoice date, invoice payment date, delivery date or order date?

Picking one out of thin air is not the way forward. This is a difficult topic and one you have to work out for yourself. Suffice it to say that your sales team will usually want you to go with purchase order date, but your finance team will probably only be interested in invoices raised or even paid invoices.

If you start reconciling a sales report with a finance report you'll see anomalies in the data, sales will consider a product as being sold in January whilst finance will consider it as sold in February.

Errors are Obvious

In my opinion this is the single most frustrating thing you can experience as an IT professional. You are rolling out information that has never been rolled out before, people are going to look at it. There will be errors.

You need to know how many. A guy far more experienced than me always told me you should never sell a data warehouse on 100% accuracy, because you will never get there. The problem you have is that even if management have had this information in the past through manual reporting, they have probably never had this level of visibility, and they will almost certainly never have been able to cross check it.

The nature of computerised reports is such that generally the details are 100% accurate, but there may be some data which falls outside your logic, even when your logic is perfect you'll usually lose something because of system or entry errors, and therefore there will be whole rows of data missing. The nature of manual reports is that generally all the rows are there, but the details in the rows are incorrect (typing errors rather than omissions).

As a rule nobody checks manual systems. Apart from me. I've spent weeks doing it, and I can tell you that cumulative errors from complex manual systems seem to manage a maximum of 90% accuracy. Omission errors from computerised reports seem to average about 95% accuracy (once you've ironed out the basic problems).

The problem being that omission errors are easy to spot, and they can see your report is "only" 95% accurate. This is a tricky one, and of course there will be a combination of bad data and compromises you have made to get your data onto a level playing field, which may affect accuracy. But 95% ain't bad (excepting for simple CRM type apps where you can probably do better)

I've found almost without exception that the department who produce these manual systems are inordinately proud of it and boast of their 100% ISO compliance and traceability blah blah blah, they usually have a pretty well established marketing machine and everyone parrots this as gospel.

So you need to check their manual systems yourself, they're not going to do it, and prove your 95% is better than their 90%. Then you have to convince everyone else that you've checked it, 'cos they ain't going to check your checks.

This is a hard sell.

Don't be Nasty About Spreadsheets

At least try. I know it's hard sometimes, but when your DW will be replacing the pinnacle of somebody's technical achievement they can get pretty emotional about it. Try and engage the user, tell them they've got a lot of very useful information in their spreadsheet and ask them to explain what they're doing. This will tell you a lot about what you're trying to achieve (remember they may have algorithms which weed out bad data which you can use, but they almost certainly won't know that they are algorithms).

Remember everyone knows something you don't. Their spreadsheet may well be a complete waste of time, energy and duplicated effort, but don't be negative about it; just deliver a proven (ultra-efficient) replacement for it.

Manage your Stored Procedures

You'll find that some data cannot be compared directly straight away, two bits of data will need to be manipulated to get it compared on a like for like basis. This means you will end up with a lot of stored procedures that transform your data so it is on a level playing field with all your other data. If your warehouse is anything like mine you'll end up taking data from various places and performing several transforms on it to get it into your warehouse.

It is absolutely imperative that you name your stored procedures properly. Use numbers to denote sequence. What you will find is that you have made some mistakes in your transforms along the way, and you have to keep going back over the transforms to get it just right.

Try and name the transform path sensibly, and I would suggest numbering it. For example, I have several data paths for the data in our printing firm, I have an ink recipe which I need to combine with our paper usage, here are my transforms :

Ink Recipe

1. DW_Sp_IR_Get Ink Recipe by Job 1
2. DW_Sp_IR_Get Ink Cost From Finance per Gramme 2
3. DW_Sp_PJ_Calculate Job Size 3
4. DW_Sp_IR_Calculate total Ink Cost for job 4

Printing Job

1. DW_Sp_PJ_Get Paper use per ink job 1
2. DW_Sp_PJ_Get Paper cost from Finance 2
3. DW_Sp_PJ_Calculate Job Size 3
4. DW_Sp_PJ_Calculate total paper cost for job 4

Combine

1. DW_Sp_Nexus1_Combine Printing Job with Recipe

You might be able to see from the above that there is more than one relationship between those two sequences. Draw a flow diagram and work out which procedures you run where. If you go at it from a trial and error point of view you're going to get stuck when something isn't quite right, you'll go back and change it. There is no real mechanism in SQL Server to document the interdependency of these transforms (Unless you're using SSIS in SQL 2005, which is pretty good for this sort of thing, incidentally).

Don't try and document it straight away in any detail, you won't know half of what you need to do, try and do it when you understand each particular "Module" of your project.

Conclusion

OK, so there are a lot of barriers to the completion of a Data warehousing project, most of them are not technical. I think the most important thing is educating the business (and yourself) about the accuracy of their current figures, as the quickest way to derail a data warehousing project is to set the business expectations too high.

Remember the point of a DW project is to get the best information you can to the top level as quickly and painlessly as possible. You are probably fighting against a series of manual reports which have been produced possibly for many years, probably with completely spurious but completely accepted statements of accuracy. The people who provide this information may well see you as a threat to their job and try and derail you themselves unless you can engage them in the process, never forget they know stuff you don't and they might be quite proud of their nonsense reports, or they might not care, or they might know or suspect they're nonsense and worry that you'll prove it.

Even if your Warehouse is attempting to deliver completely new information it is still going to be based on a backbone of accepted business reporting, so there should be some way to reconcile your information against something you know to be true (but remember that "know" is sometimes subjective, so try not to take anyone's word for it)

Bear in mind that the accuracy of data is related to the number of checks made against it, for example a Sales Order system is pretty accurate, because a customer will almost always question an incorrect invoice and a salesman will usually take care with an order, whereas a production reporting system may reconcile material flow against material purchased, but there will be an acceptable variance for error.

You are going to have to go back and do it again, you'll get some stuff wrong on the first attempt. You may have to go right back to your initial query and change the whole flow from beginning to end. Make sure you document the data flow and name your procedures sensibly or you'll get lost.

Storage Modes in SSAS 2005

By Yaniv Mor

Overview

In SSAS 2005, a cube, a partition within a cube or a dimension can store data and aggregations in numerous ways, each with its own pros and cons. While Microsoft has retained the storage modes available in AS 2000 (MOLAP, HOLAP and ROLAP), it has added several new storage modes, which make an SSAS implementation a more robust and flexible solution in a production environment. A new concept: proactive caching has been introduced, and as the name implies, using this new feature enables the administrator to better control the frequency and the way cube updates are being performed.

What exactly do we store?

SSAS 2005, being an OLAP engine, stores cumulative data of the measures defined in the cube. For example, if we have a cube with a customer dimension and a time dimension, we might have a measure like: sales amount which will indicate the dollar figure of the sales per customer in a certain date and time. This data is stored as part of the cube. On top of the measures data, each cube can have its own aggregations. Aggregations are pre-calculated summaries of data for various combinations of dimension slicing. SSAS 2005 needs to store both the measures data and the aggregation data on disk. This enables the OLAP engine to retrieve answers to users queries faster. In the remainder of this article, I'll try to explain what types of storage modes SSAS 2005 offers to store this data.

1. MOLAP (Multi dimensional OLAP)

This is the most common storage mode. It implies that the cubes data and aggregations will be stored in a multidimensional format. This is the most efficient way to store data and will provide the user with optimal query performance. The downside with using this storage mode is that it creates a completely offline data set. Once the cube retrieved the data from the underlying relational database and processed it, there is no further connection to the relational database. Any subsequent updates in the database will not be reflected in the cube, unless it is re-processed. Now, for most data warehouse solutions, a daily or even a weekly re-process of the cubes is more than enough. For environments like these, it is highly recommended to use the MOLAP storage mode.

2. Scheduled MOLAP

This is somewhat similar to the MOLAP storage mode, in that it stores its data in a multidimensional format. However, the content of the cube/partition cache will be automatically updated every 24 hours. In a way, using this storage mode already utilizes the proactive caching feature. It is important to note that there's no need to create a SQL Agent job or an IS package to process the cube. The cubes cache will be refreshed automatically every 24 hours.

3. Automatic MOLAP

Again, data is stored in a multidimensional format. There is a basic change, though, when compared with the previous 2 storage modes. Using an automatic MOLAP storage mode means that Analysis Services needs to listen to notifications from the underlying relational database. Once an update has occurred in the database, an event is raised and sent to Analysis services. This type of messaging system can be achieved by using Notification Services, for example. When the Analysis Services Server has received the notification, it will automatically refresh the cubes cache, to reflect the new changes in the database. Users will still be able to access the cube while the cache is being refreshed, but they will be using the old cache. Once the cache is refreshed, users will be redirected to the new data. This type of storage provides the users with an almost up-to-date cube (the target latency is 2 hours from update to process). This obviously depends on the cube size and the magnitude of changes in the underlying database.

4. Medium Latency MOLAP

Things are starting to get interesting as we approach the latency variance (which we touched briefly in the previous section). Initially data is stored in a multidimensional format. The Analysis Services server is listening to notifications from the underlying relational database. When an update is performed in the database, the cube switches to a Real Time ROLAP storage mode. We will touch on this storage mode shortly, but I will briefly note that this storage mode means: no multidimensional storage at all and the cubes data is being retrieved directly from the underlying relational database. The timing of the switch to ROLAP is determined by the latency variable. By default, cache will be updated after 4 hours from the time the relational database was updated. This switch should not last for long though, as in parallel, the cubes MOLAP cache gets processed to reflect the latest changes. Once the cache is processed, the cube switches back to the MOLAP storage mode. You would want to use this type of storage when you realize that the underlying database goes through updates occasionally, and users do want to have an up-to-date cube. However, they still require performance to be reasonable. Having the cube reflecting data which is accurate up to the last 4 hours is reasonable for the users and provides the correct trade-off between performance and real-time data. If the users agree to these terms, then this is the preferred storage mode to use.

5. Low Latency MOLAP

This mode is similar in its behavior to the Medium Latency MOLAP mode, the only change is (yes, you may have guessed it by now) the latency. Instead of allowing a latency of 4 hours, this mode will allow only up to 30 minutes of latency. This implies a more frequent switch to the ROLAP mode and even poorer query performance, as the cube is being processed more often. The users do get to see a more updated version of the data though.

6. Real Time HOLAP

HOLAP stands for hybrid OLAP. So far we have discussed having the data and aggregations stored in a multidimensional format. In the hybrid storage mode, data is maintained in its relational format (i.e. data resides in the relational database), while aggregations are stored in a multidimensional format. What this means is that the data is always real-time data. The Analysis Services server still listens to notifications, and when changes in the relational database occur, the aggregations (which are still stored in a multidimensional format) are refreshed. During this process, the cube is switched to the infamous ROLAP mode, until aggregation processing is complete. It is easy to see that users are enjoying real-time data with the added benefit of MOLAP aggregations to improve query performance. Still, performance is not as good as it used to be when compared to data stored in a multidimensional format, as well as the aggregations.

7. Real Time ROLAP

This is the last storage mode available and we are actually quite familiar with it already. Data and aggregations are stored in a relational format. This means zero latency for users, who will always be able to access the cube and retrieve real-time data. Query performance is the poorest here though as no MOLAP objects are available at all.

Just a bit more before finishing

You can apply the various storage modes on cubes, partitions and dimensions. Some of the features will not be available on some of the objects. When using proactive caching, there are several settings which you can manually tune, like the latency, the listening intervals and the notification types. You can also fine-tune the processing options and instruct the SSAS engine whether to fully process a partition or to incrementally process it.

Conclusion

The MOLAP, HOLAP and ROLAP storage modes were available in the Analysis Services 2000 version. The important addition to the 2005 version is the proactive caching feature, which enables the user to have a real-time version of the data while still enjoying the query performance benefits of the MOLAP storage mode.

Dynamic Connection Strings in Reporting Services 2005

By Bilal Khawaja

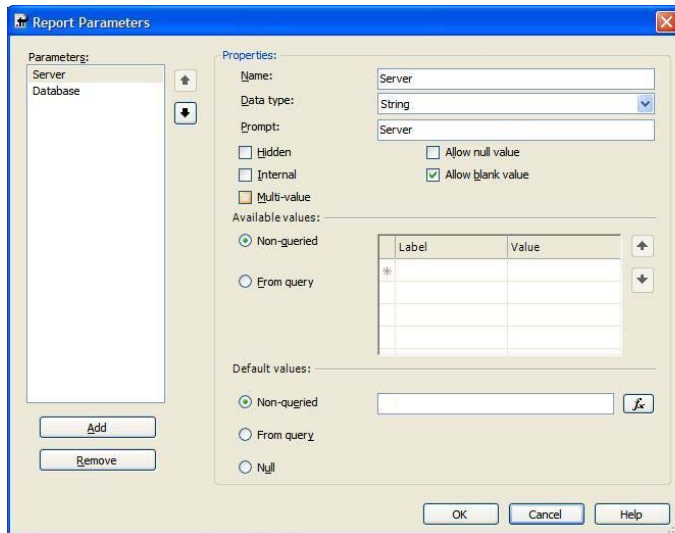
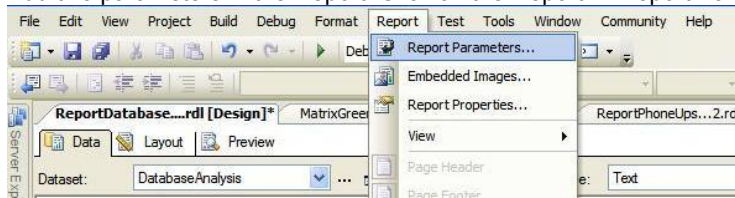
Wouldn't it be nice to have one report and can display data from different databases residing on different servers? Instead of creating a same report multiple times to be deployed on different servers, we can have one report and it can generate data from any database on any server.(if you are using stored procedures, make sure the same stored procedure exists in every database you are running the report against). So in the URL we can pass in the database name and server name for different clients for their specific data.

Example

If your connection string looks like this:

```
Data Source=XYZ333;Initial Catalog=ABC01
```

Add two parameters in the Report. Click on the Report -> Report Parameters.



Since the prefix for DataSource is XYZ and Prefix for Database is ABC, you can hard code the "XYZ" and "ABC" in your connection string and just pass in "333" for Server and "01" for Database and have your connection string like this:

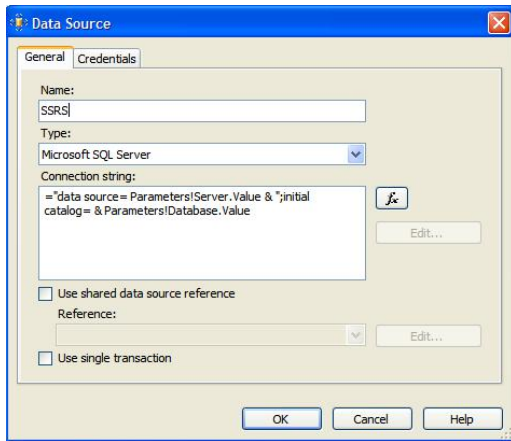
```
= "data source=XYZ" & Parameters!Server.Value & ";initial catalog=ABC" &  
Parameters!Database.Value
```

If you want to pass in the full ServerName and full Database Name do the following:

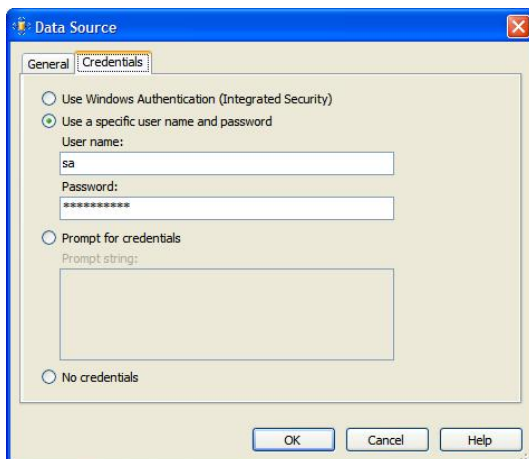
```
= "data source=" & Parameters!Server.Value & ";initial catalog=" & Parameters!Database.Value
```

Steps

1. First build your Report by hard coding a connection string
2. Data Source=XYZ333;Initial Catalog=ABC01;uid=sa;pwd=dynamic
3. Test and Preview the report to make sure you see the results you want.
4. Add the two parameters to the Report as i mentioned above with the screen shots.
5. If everything looks good, go back to your Data Tab and select your DataSet and choose edit.
6. Replace your existing connection string with this dynamic expression based connection string.



Choose either the Windows Authentication or you can supply user name and password in the Credentials Tabs:



7. Once you did this, do not preview the report since, it will not work and give you an error.
8. Right click on the report and select deploy to the ReportServer. You can right click on the project and go to properties and TargetServerURL: Make sure the the path of the server is correct. For example it should be: `http://localhost/ReportServer` or the server name you are deploying it to.
9. Once it is deployed, run the report and it will ask you to enter the server name and database name to run the report against.
10. Enter that and it will generate the report.

Hope you liked the Article :)

Populating Fact Tables

By Vincent Rainardi

Fact tables are normally loaded from transaction tables, such as order tables or from transactional files, such as web logs. Hence the number of rows to update in a load is much larger than in dimensions. The core of loading fact tables

is to change the natural keys into surrogate keys. Let's start with the basic steps, then continue with a few of important consideration such as loading partitioned fact tables, slim lookup tables, deduplication, loading snapshot and delta fact tables, and dealing with fact table indexes

Basic Steps

When loading a transaction table from OLTP into a fact table in the data warehouse, the value columns on the transaction table become fact table measures, the primary key(s) on the transaction table become degenerate dimension column on the fact table, and the alternate primary keys become dimensional key columns on the fact table.

As rows of fact tables are loaded, each of the key columns is converted from the natural key. This is why all the dimension tables must be populated first before we populate the fact tables: because we need the surrogate keys on the dimension tables to translate the fact table natural keys. This is the very basic and the very heart of data warehouse loading, so it is important to understand it. Perhaps the best way to describe this concept is using an example. Consider a simplified order detail table with the following columns: order_id (primary key), order_date, product_id, quantity and price.

Source table: order_detail

order_id	order_date	product_id	quantity	price	last_update
352	15/07/2006	BGCKQ	1	12.99	30/10/2006
410	30/10/2006	KMSCG	2	7.99	30/10/2006

In the data warehouse let us assume we have an over simplified star schema consisting of a date dimension, a product dimension and a sales fact table, as describe below.

Date dimension table: dim_date

date_key	date	day	month
2388	15/07/2006	Saturday	July
2485	30/10/2006	Monday	October

Product dimension table: dim_product

product_key	product_id	product_name	description
53076	BGCKQ	Aike IDE HD	case USB 2
92184	KMSCG	Sony BP71	VIAO CG7 battery

Fact table: fact_sales

fact_key	date_key	product_key	order_id	quantity	price	load_time
830923	2388	53076	352	1	11.99	15/07/2006

Notes:

Note that order_id 352 already exists in the data warehouse but the price has changed. When it was loaded on 15/07/2006 the price was 11.99 but now the price is 12.99.

In reality date dimension would contains many more other attributes, such as day of the week (2 columns: name and number), ISO date (YYYY-MM-DD), Julian date (number of elapsed days since the beginning of the year), SQL date (in SQL Server datetime format), day of the month, calendar week / month / quarter / year (3 columns for month: full name, short name and number), fiscal week / period / quarter / year, weekday flag, statutory holiday flag, last day of the month flag, etc. Plus day and month names in other languages if your data warehouse is internationalised.

In reality the dimension tables would have standard dimension table columns such as load_time and SCD attributes but for this simplified case these columns are not displayed here.

It is a common practice not to have a fact_key column in the fact table, with the argument being the combination of all the dimension keys will make the record unique. In this case, the primary key of the fact table is a composite key. When loading this kind of fact table we will need to compare all dimensional key columns to identify the correct row when doing updating (see step 3 below).

The basic steps in loading data warehouse fact tables are described below. It is not that different from the steps in loading dimension tables. In fact, they are almost the same. In this example, it is assumed that the source table, order_detail on the OLTP, has been loaded incrementally into a stage table called order_detail. The stage database name is stg. So what we need to do now is to load the fact table records from the stage into the data warehouse.

Step 1. Create The Temp Table

```
SELECT * INTO #fact_sales FROM dw.dbo.fact_sales WHERE 1 = 0
```

The temp >Step 2. Populate The Temp Table

```
SET IDENTITY_INSERT #fact_sales ON
```

```
INSERT INTO #fact_sales
```

```
(date_key, product_key, quantity, price, load_time)
```

```
SELECT
```

```
ISNULL(f.fact_key, 0),
```

```
ISNULL(d.date_key, 0),
```

```
ISNULL(p.product_key, 0),
```

```
ISNULL(s.quantity, 0),
```

```
ISNULL(s.price, 0),
```

```
@current_load_time
```

```
FROM stg.dbo.order_detail s

LEFT JOIN dw.dbo.dim_date d ON s.trans_date = d.sql_date

LEFT JOIN dw.dbo.dim_product p ON s.product_id = p.product_id

LEFT JOIN dw.dbo.sales_fact f ON d.date_key = f.date_key

AND p.product_key = f.product_key

WHERE s.load_time BETWEEN @last_run AND @current_run
```

The temp table after it is populated:

fact_key	date_key	product_key	order_id	quantity	price	load_time
830923	2388	53076	352	1	12.99	30/10/2006
0	2485	92184	410	2	7.99	30/10/2006

Source table: order_detail

order_id	order_date	product_id	quantity	price	last_update
352	15/07/2006	BGCKQ	1	12.99	30/10/2006
410	30/10/2006	KMSCG	2	7.99	30/10/2006

Date dimension table: dim_date

date_key	date	day	month
2388	15/07/2006	Saturday	July
2485	30/10/2006	Monday	October

Product dimension table: dim_product

product_key	product_id	product_name	description
53076	BGCKQ	Aike IDE HD	case USB 2
92184	KMSCG	Sony BP71	VIAO CG7 battery

Notes:

Look how dimensional keys are looked up in a single step, by joining the fact table to the dimension tables on the natural keys. Because they are LEFT JOINS, the fact table rows without corresponding dimensional rows will result in NULL dimensional keys in the fact table. The ISNULL then convert these NULLs to zeros.

Notice that we don't populate the load_time with getdate(), but with a variable named @current_load_time. This variable is populated with getdate() at the beginning of the loading batch and used by all processes in the batch. This is necessary so that in the event of failure, we know the point in time we have to restart the process from.

Notice that we only load the rows between @last_run and @current_run. This is necessary if we keep the records on the stage table for a few days, i.e. if the stage contains more than 1 day data. If we clear the data from the stage table as soon as we load them into data warehouse, we don't need to specify this where clause.

The example above is using only 2 dimensions but in the real practice we would have to deal with more dimensions. 10 to 15 dimension key columns on the fact tables are common occurrence.

Step 3. Update existing records

```
UPDATE f

SET f.date_key = t.date_key,

f.product_key = t.product_key,

f.order_id = t.order_id,

f.quantity = t.quantity,

f.price = t.price,

f.load_time = t.load_time

FROM dw.fact_sales f

INNER JOIN #fact_sales t ON f.fact_key = t.fact_key

WHERE t.fact_key <> 0 AND

(f.date_key <> t.date_key OR

f.product_key <> t.product_key OR

f.order_id <> t.order_id OR

f.quantity <> t.quantity OR

f.price <> t.price)
```

Source table: order_detail

order_id	order_date	product_id	quantity	price	last_update
352	15/07/2006	BGCKQ	1	12.99	30/10/2006
410	30/10/2006	KMSCG	2	7.99	30/10/2006

fact_sales after the update:

fact_key	date_key	product_key	order_id	quantity	price	load_time
830923	2388	53076	352	1	12.99	30/10/2006

Notes:

Here we update the fact table, based on the data on the temp table. In this case the price was updated from 11.99 to 12.99. We only update the rows where the tmp tables fact_key is not 0, i.e. the rows already exist on the target fact table. For the rows where the fact_key is 0 (not exist on the fact table), we will insert them into the fact table later on.

Notice that when updating rows we update the load time column as well. The last line is used to specify which changes we want to pickup. In most cases, we want to pick up changes on all columns, but sometimes there are legitimate reasons for business rules to specify that changes on certain columns are to be ignored.

Step 4. Insert new records

```
INSERT INTO dw.dbo.fact_sales
(date_key, product_key, order_id, quantity, price)
SELECT date_key, product_key, order_id, quantity, price
FROM #fact_sales
WHERE fact_key = 0
```

Source table: order_detail

order_id	order_date	product_id	quantity	price	last_update
352	15/07/2006	BGCKQ	1	12.99	30/10/2006
410	30/10/2006	KMSCG	2	7.99	30/10/2006

Date dimension table: dim_date

date_key	date	day	month
2388	15/07/2006	Saturday	July
2485	30/10/2006	Monday	October

Product dimension table: dim_product

product_key	product_id	product_name	description
53076	BGCKQ	Aike IDE HD	case USB 2
92184	KMSCG	Sony BP71	VIAO CG7 battery

fact_sales after the insert:

fact_key	date_key	product_key	order_id	quantity	price	load_time
830923	2388	53076	352	1	12.99	30/10/2006
916912	2485	92184	410	2	7.99	30/10/2006

Notes:

It is a good practice to always declare the column names. This is important for flexibility and maintenance. Let RDMBS maintains the fact_key. When setting up the data warehouse, set this column to be IDENTITY (1,1).

Logging and closing

In every step above we need to do error handling and logging. Error handling is important because if the loading fails on any steps, we need to be able to recover from the failure. Logging is important to know what exactly happened on each steps, i.e. how many records are processed, etc.

At the end of the program, we should not forget to clean everything up, i.e. drop the temp table(s), follow control protocol e.g. set the process to complete state, etc.

The above code shows how to do upsert with native SQL codes, which is very fast and efficient. But it is worth to note here, that good dedicated ETL tools such as Informatica and Data Integrator have the facilities to do in-memory lookups which has very good performance too. Disk-base lookup is definitely not they way to go here, as they are slow. Mainly because there are a lot of SQL statements to execute, i.e. one for each lookup, and each of these statements touches the disk, which is a costly operation.

Now that we have understand the basic steps in loading fact tables, lets familiar ourselves with a few practical experiences such as loading partitioned fact tables, slim lookup tables, deduplication, loading snapshot and delta fact tables, and dealing with fact table indexes.

Slim Lookup Tables

When a dimension is very large, sometimes it would be a significant performance improvement if we provide a separate lookup table for key management. For example, if our customer dimension has 100 columns and 10 million rows, we could create a customer key lookup with only 3 columns: customer_id, customer_key, load_time, which would increase the performance of dimensional key lookup process on step 2 above. It is also more suitable for performing in-memory lookup.

Example of Slim Key Lookup table: lookup_customer

customer_id	customer_key	load_time
493238	30012	02/10/2006
493240	30013	03/10/2006
493241	30014	03/10/2006

The load_time column would be useful if we have very long dimension table, e.g. 25 million rows. This is not uncommon when the data warehouse is utilised to drive CRM systems (Customer Relationship Management), especially in the dot com companies dealing with online campaigns, be it subscription based or tactical campaigns. In this case we can specify a where clause to limit the selection on this slim key lookup, for example where load_time is within the last 2 years. This would cover, say, 98% of the lookup. For the remaining 2%, e.g. the one older than 3 years, we then go to the main customer dimension table to find the customer_key.

Although it takes a bit of time to maintain the lookup table, overall we still save time as the time saved by querying a slim key lookup table is a lot greater than the time required to maintain the lookup table itself. This is especially true for a Master Data Management (MDM) dimensions such as customer, product and account, where they are used all over the place in the data warehouse and data marts. Time dimension is not that long - 10 years is only 3651 rows - hence I tend not to have a lookup for time dimension. For those of us who think to set the grain of time dimension to hours or minutes, the general advice is: don't. Either put a time stamp column on the fact table or have a time of day dimension. But this discussion (time of day dimension) is for another article.

Natural Key Changes

Sometimes a natural key which we thought was a good solid natural key could change and therefore can no longer be a natural key. Perhaps the best way to explain it is using an example. In a OLTP source system the customer table has a composite primary keys as combination of branch code and customer code. Customer code is only unique within a branch. Occasionally a branch could be closed and all the customers in that branch are moved or assigned to another branch. For example, see the customer table below.

branch_code	customer_code	customer_name	other_attributes
1	1	Andrew	...
1	2	John	...
2	1	Steve	...
2	2	Paul	...

When branch 1 is closed, and its customers are moved to branch 2, the customer table becomes:

branch_code	customer_code	customer_name	other_attributes
2	1	Steve	...

2	2	Paul	...
2	3	Andrew	...
2	4	John	...

If we use branch_code and customer_code as natural key in the data warehouse, we need to handle this branch closure event with care. In the previous project we utilise another column in the customer table which would help identify a unique record and we create a special table in the data warehouse to translate the change natural key to the correct data warehouse key.

Unknown Records

If a fact table record does not have a corresponding dimension record, basically we have 2 choices: either we don't load that record into the fact table, or we load it but we put 0 as the dimension key, referring to the unknown record in the dimension table. An unknown record is a record in the dimension table, with a dimension key of 0 and all the attributes are populated with blank string, number 0 or low value date, depending on the data type. Name and description columns are usually populated with the word "Unknown". The load_time column is populated with the date of the record was created. This date is normally equal to the date the data warehouse was setup, because the record was created by the data warehouse setup scripts. For example:

Product dimension table: dim_product

product_key	product_id	product_name	description	min_level	valid_until	load_time
0	0	Unknown	Unknown	0	1/1/1900	12/10/2004

Here is an example of a row in the source table with a product_id that does not exist in the dimension table:

order_id	order_date	product_id	quantity	price	last_update
358	19/08/2006	BGCKZ	3	2.99	31/10/2006

This is how fact_sales looks after that record is loaded:

fact_key	date_key	product_key	order_id	quantity	price	load_time	
	830937	2424	0	358	3	2.99	31/10/2006

If we load it into the warehouse, we need to flag it into the data quality system, so that it can be reported and corrected on the subsequent load. If we don't load it it should still be set as a data firewall rule. If we don't load it, the

total of measure on fact table would not be accurate. In the example above, the total sales amount for July 2006 would be 3 x 2.99 less than what it should be. Because of this we tend to load fact table record that does not have a corresponding dimension record and set the dimension key to 0. This way the fact table measure total would be correct, it's just that the sales could not be traced to a valid product record. But all other dimension keys would still be valid, e.g. it can be traced to a valid date dimension record, a valid customer dimension record, etc. And above all, the referential integrity between the fact tables and the dimension tables are still valid.

Deletion

There are 2 main causes why we perform deletion on the fact tables: 1. because the data extraction approach is fixed period extraction, and 2. to accomodate reverse transactions such as cancellations. No 1 is usually physical deletion and no 2 is usually logical deletion. Let's discuss them one by one, using examples to clarify.

No 1, loading approach. In one of the data warehousing projects I've been involved with, we had difficulties extracting a transaction table from the source system incrementally, because the date stamp was not very reliable. We tried with 3 weeks tolerance i.e. where last updated date or created date is within the last 3 weeks but we still find some leakage, i.e. a few records were updated without the date stamp columns were not updated. Please refer to this article for discussion about incremental extraction and leakage. Luckily, source system did not allow the user to update records that were more than 6 months old. When a user tried to update a transaction record that was created more than 6 months ago, the system displayed an error, something like "You can not modify this record." So our approach of extracting this transaction table to the stage was to get 6 months data every time. And consequently, the approach of loading the fact table from stage was to delete the rows on the fact table that exist in stage, then reload all the stage records into the fact table. It was actually the same as updating those records, but we found that it was quicker to delete then reload. We identify the records on the fact table that exist in stage by comparing all the dimension key columns, i.e. the composite natural keys of the source transaction table.

No 2, accomodating reverse transaction. Sometimes, in the source table we had a main transaction table containing normal transactions and a smaller secondary transaction table containing cancellation records. We have 2 options loading this kind of tables into the fact table in the data warehouse: either we load the cancellation as a new record with negative measures, or we load the cancellation as logical deletion. Reverse transaction such as refund and credit notes needs to be implemented as negative measures, but for cancellation we have 2 options. Each approach has its own advantages and disadvantages. Which one is better depends on the situation, for example whether we will be loading from that fact table into OLAP cubes or not, whether we will be creating a summary fact table or not, whether we need the historical dates or not, and whether the secondary source table contains complete data or not. If we decided to go for logical deletion, then for each cancellation record exists on the secondary source table, we mark the logical delete column on the fact table record. All processes further down the stream such as loading into data marts, creating summary tables or loading into OLAP cubes need to be aware of this column and they need to handle it properly.

Deduplication

When loading records from stage to fact tables, sometimes we have duplicate records. For example: we declare that the grain of the fact table is 1 day for every product for each store. But we found this on the stage table of the fact table:

date	product_id	store_id	quantity
19/8/2006	BGCKZ	309	30
19/8/2006	BGCKZ	309	1

Do we add them up, take the maximum, or take the minimum, or take the average? First we need to understand why it happen. In this case I always found it useful to go back to the business users or the source system expert. The second record could be an error, and in this case we take the earliest record. Or it could be a correction and in this

case we sum them up. Or there can only be 1 measurement per day and in the case of 2 or more records found it must be a system error and therefore they want us to take an average. Whatever the business rule is, we need to document it, obtain sign off and then implement it in the code. This process is called deduplication. This normally happens if the source system allow duplication, i.e. it does not have the necessary constraints in the database to make it unique as per the grain criteria.

Deduplication does not only happen in fact tables, but also in dimension tables, especially MDM dimensions such as customer. Deduplication can also occur when the fact table is loaded from 2 or more different source system. Many data quality software such as Trillium, DataFlux, DQ Global, have facilities to deduplicate data.

Fact Table Indexes

Fact tables can be very large. They can have millions of rows. To improve query performance, fact tables are normally indexed. The cluster index of a fact table is normally a composite of the dimensional keys. Or the fact key column, if there is one. For the non clustered indexes, deciding which column to index depends on how the fact table is used. Whether it is used for direct queries by end users, for populating OLAP cubes or by reports, SQL Profiler and Index Tuning Wizard are useful to understand what indexes would help improving the query performance.

If our fact table contains 100,000 rows or less, we just load the fact table with the indexes on. There is no need to worry about dropping indexes when loading. If our fact table contains more than 1 million rows, it may be quicker to drop the indexes before loading the data, and recreate them afterwards. If the load is less than 10% of the fact table length, generally speaking we don't need to worry about dropping indexes. Chances are we could decrease loading performance by doing so. But if we are loading 20% or more (of the fact table length, i.e. number of rows in the fact table) we may want to consider dropping and recreating indexes. It is very difficult to generalise this, as the performance differs depending on what indexes we have and what loading operations we perform. Hence we always need to test it to prove that our load is significantly improved by dropping and recreating indexes, before implementing it in production.

Fact Table Partitioning

Table partitioning is new in SQL Server 2005, but has been in Oracle since 1997 on Oracle 8 and improved in version 8i and 9i. In SQL Server 2000 we only have partitioned view, not partitioned table. This article provides a good overview on the partitioning on SQL Server 2005. Joy Mundy wrote an article about partitioning on SQL Server 2000. For Oracle, it's on this document and this manual. In DB2, table partitioning was introduced in version 8 since 2002, and was greatly improved in version 9 which was release in July 2006. Paul McNerney describes DB2 partitioning features for data warehousing in this article.

Fact table partitioning is basically dividing the fact table into several physical parts. Each part is called a partition. Each partition is ideally located on a different physical data file and ideally each file is located on different disk. For example, we can divide the fact table so that rows for January 2006 are located on partition 1, rows for February are located on partition 2, and so on. Each partition can be loaded separately in parallel. Partitioning can improve query performance and increase availability. We can add new partition, drop existing partition, truncate (empty) a particular partition, move a partition, split a partition, merge several partitions and exchange/switch partitions. All this improves maintenance operations and greatly simplify administration tasks.

If our fact table contains more than 1 million rows, we need to consider partitioning it. It can improve the loading performance significantly. We are talking 5-10 times quicker. This is because we load new data to just 1 partition, which is say 12 times smaller than the size of the whole fact table, hence quicker. There are also techniques to load data into a new empty table with exactly the same structure as the fact table, then switch/exchange partition that new table with the main table. Queries can be a lot quicker too. We can also partition indexes so that each part of the index serves only one table partition. This allows more processes to run in parallel. Partition can really be a life saver for a data warehouse.

Internal Data Warehouse Entities

Some data such as performance target or budget does not exist in any source system but it needs to exist in data warehouse. Hence they are known as internal data warehouse entities. For example, for each store we can calculate out of stock percentage, i.e. how many times a required product is out of stock. There may be a consensus that it needs to be under 5%. This 5% is not written anywhere in any source system but will be required by the reports so they need to exist in the data warehouse somewhere.

We have 2 options on how to get this performance target into the warehouse. 1) we can build an application which will store or persist the data into its database, then from there we ETL into staging and into the warehouse, or 2) put it on a simple spreadsheet and we import it into the warehouse. In any case we should not allow the data to be directly entered into data warehouse, for example using direct SQL statement or through SQL Server enterprise manager. All data stored in the warehouse must go through the data quality / data firewall rules, so that any exceptions to the rules are recorded and reported.

Loading Snapshot and Delta Fact Tables

A snapshot fact table is a fact table that contains a measurement of status at specific point in time. For example:

a fact table that contains actual inventory level for each product at 9 am every day a fact table that contains balances of every saving account in all branches on the first day of every month a fact table that contains the details of all accounts that each customer has every day There are 2 kinds of snapshot fact tables, periodic and accumulating. Periodic snapshot fact tables contain regular statement of status. All 3 examples above are periodic snapshot fact tables. Accumulating snapshot fact tables show the status at any given moment. It is useful to track items with certain life time, for example: status of order lines. Please refer to this Ralph Kimball article for more details about snapshot fact tables. An example of accumulating snapshot can be found [here](#).

How do we load periodic snapshot fact tables? We extract all records that satisfy the criteria from the source table at certain period. For example, take all active rows from the account tables including the balance. Do this automatically once a month on the first day. Loading accumulating snapshot is rather different. We still take all records from the source table that satisfy the criteria, then we update the fact table. For the example of purchasing accumulating snapshot above, everytime there is new piece of information about a particular purchase, we update the fact table record. We only insert a new record in the fact table when there is a new purchase requisition.

Delta fact table is a fact table that we produce as a result of comparing the condition of a source table on 2 different time points. For example: account table. This table in the source system contains all customer accounts. Say on Monday we have 100,000 active accounts and on Tuesday we have 100,001 active accounts, i.e. there were 2 new accounts opened, 1 account closed and 3 accounts changed. Out of the 3 accounts changed, 1 is changed interest rate (from 5.75% to 5.50%), 1 changed the credit limit (from 3000 to 3500), and 1 the interest payment frequency (from daily to monthly). On the delta fact table there will be 6 new rows today (suppose today is 6th Nov 2006). Legend for change_type: NA = new account, CA = closed account, IR = Interest Rate, CL = Credit Limit, IPF = Interest Payment Frequency.

account_key	change_date	change_type	IR_before	IR_after	CL_before	CL_after	IPF_before	IPF_after
49912	6/11/2006	NA						
26077	6/11/2006	CA						

32109	6/11/2006	IR	5.75	5.50
19387	6/11/2006	CL	3000	3500
29462	6/11/2006	IPF	D	M

To populate delta fact table, we download the source account table everyday and compare today's copy with yesterday's copy and entered the differences on the delta fact table.

Purging Or Pruning A Fact Table

Some people call it purging, some call it pruning. Purging or pruning a fact table is an activity to remove certain rows from the fact that satisfy certain criteria. To give us an idea below are some examples of purging criteria:

older than 2 years
older than 2 years and status is not active
keep daily records for the last 4 weeks then Monday only for the last 2 years

Example 1 and 2 is normally for transaction fact table and example 3 is normally applicable for periodic snapshot. Purging is important when our data warehouse is a few years old. Purging can improve query and load performance significantly. If the fact table is partitioned, it is a common exercise to archive the oldest partition (say older than 5 years old, partitioned by month) then drop or truncate the partition. Archiving can be done by exporting the partition (using exp in Oracle) or by backing up the partition.

Reloading Fact Table History

Soon after the data warehouse is in production, user may want us to reload say last 2 years history from the OLTP source system. Say they want 2 years history of order tables. It's not difficult to reload the history. Normally we just need to modify the time window on the ETL package. For example, if you use SSIS or DTS, we could set the time window on a table in the metadata database, where the Last Successful Extraction Time (LSET) and Current Extraction Time (CET) are kept. See this article for the concept of LSET and CET.

The problem with reloading fact table history is: we need to reload all related dimensions too. And most OLTP do not keep this history. All orders in the last 2 years, yes no problem they have it. But all customer and products and store details in the last 2 years? Often the customer and products history are not kept, their details are overwritten with new ones. SCD concept is not known and implemented in OLTP systems. This way they lost the history. The best way is probably to load the last condition of product, customer and store tables from the source system, and for all order table records which we can't find a match in the product, customer and store tables, we reference them to the unknown records. This way the sum of measure in the fact table will still be valid and the referential integrity will still be intact.

Reporting Services 2005 101 with a Smart Client

By Asif Sayed

Introduction

I still remember it was a neatly done report that got me my first pay raise. Ever since, I am very passionate about report writing (every one likes pay raise right?). In this article, I will guide you through step by step how to create a simple report using MS Reporting Services 2005; and host it with a Smart Client application.

So, are you ready to get your pay raise? Why not! Who knows, your neatly done report can just do that.

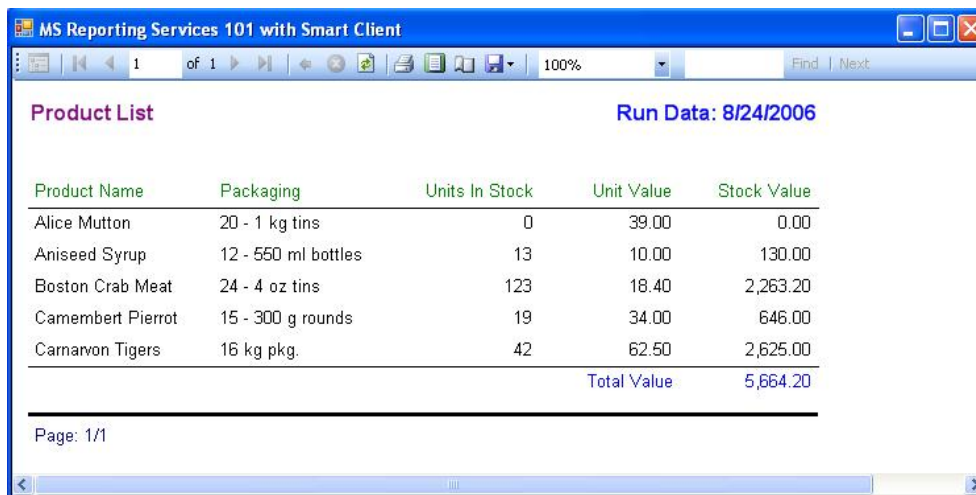
Prior to this article, I wrote three others, which were addressing different issues related to the reporting services. However, all of them were targeted towards the intermediate-advance level audience. From all the feedback I received, one was common: quite a few of you asked for an article which will be specially geared towards the novice-beginner level.

I assume the reader has the basic understanding of the Visual Studio 2005 IDE and comfortable with writing code using C#. You dont have to know the MS Reporting Services to understand this article; although, any pervious experience with the report writing would help to fast track yourself.

Although, I am calling this article 101, my intention is to adopt the applied approach rather then discussing each and every topic associated with reporting services. I am touching on most common aspect of report designing with most commonly used controls. I would strongly encourage you to please go through MSDN documentation for more detailed information.

Lets roll up our sleeves, its reporting time

Please take a look at **Image 1**. How complex is that report? How much time do you think it will take to create such a report? Well, as for complexity, it is a simple report extracted out of source *NorthWind->Products* (SQL Server 2000) and lists all the products information with summary totals.



The screenshot shows a web browser window titled "MS Reporting Services 101 with Smart Client". The report is titled "Product List" and shows "Run Data: 8/24/2006". The report contains a table with the following data:

Product Name	Packaging	Units In Stock	Unit Value	Stock Value
Alice Mutton	20 - 1 kg tins	0	39.00	0.00
Aniseed Syrup	12 - 550 ml bottles	13	10.00	130.00
Boston Crab Meat	24 - 4 oz tins	123	18.40	2,263.20
Camembert Pierrot	15 - 300 g rounds	19	34.00	646.00
Carnarvon Tigers	16 kg pkg.	42	62.50	2,625.00
Total Value				5,664.20

Page: 1/1

Image: 1

About time, obviously, it should not take you hours to do it. About R&D and trial & error time, I leave that to you; dig down deep; the deeper you will explore, the better the treasure you will find.

Here it is, the million \$ question: How to start? What is going to be the first step?

Often, it is very easy to find out what should be the first step. Have you seen a house built before the foundation? No! So, have I given you a hint here? Sure, we must first develop the Smart Client to host our report.

Step 1: Create Windows Application Project

Please do the following to create a Windows Application (Smart Client) project:

- Select **File menu -> New -> Project**.
- Choose **C#** from **Project Types pane**.
- In the **Templates pane**, choose *Windows Application* for Visual C# projects.

In the Name box, name the project something unique (I named the attached project code **rsWin101**) to indicate the application's purpose. In the Location box, enter the directory in which you want to save your project, or click the Browse button to navigate to it. Once you are done, you will find Form1 added to the project and you can start working on it using *Forms Designer*.

Please update following properties of Form1:

```
Form1.Text = "MS Reporting Services 101 with Smart Client"
Form1.Size = 750, 300
```

Feel free to change any other property of Form1 as per your requirement.

Step 2: Add Report Viewer to the Form

So, what is report viewer? As we need the DVD player to play a DVD; same goes with the reports, we need a report viewer to have the report preview done.

For all those who are brand new to report writing, I would say, report viewer gives life to your reports. It not only previews you the output, further, it also facilitates you to generate the information in the various popular formats (pdf, excel etc.). You can also take a hard copy print of the report while you are viewing the output.

Please perform following actions to setup Report Viewer Control on Form1:

- Drag **ToolBox -> Data -> ReportViewer** and drop it on Form1. This step will create a new instance of **ReportViewer** with name *reportViewer1*. I always wanted to name *reportViewer1* to *rpvAbraKaDabra*, hence, wont let this chance pass by now. As I picked *rpvAbraKaDabra*, feel free to pick yours, let those imagination horses run wild!
- By setting *reportViewer1.Dock = Fill*, report viewer will fill the entire surface of form for report display purpose.

After step 1 and step 2, your project should look as per **Image 2**.

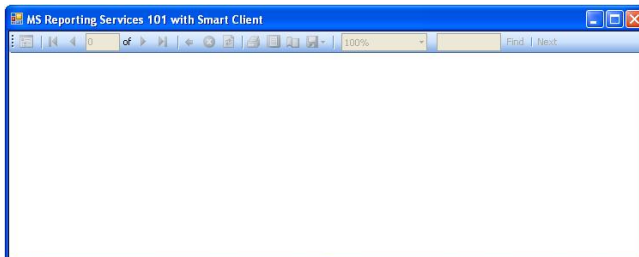


Image: 2

Step 3: Add DataSet to the Project

Hurray! We are done with the foundation. Its time to put walls around the foundation; eventually these walls will hold the doors and windows of your home. *DataSet* is just that for Report Viewer, it holds and provides the raw data from data source to be processed and ready to be outputted on the Smart Client interface.

Following step is required to have DataSet added to project:

- Select **Add -> New Item -> DataSet** from Solution Explorer. Change name from *DataSet1* to *dsProduct* and click on Add button to complete the action.

Lets add a *DataTable* to our newly created *DataSet*. *DataTable* is essential to load the reporting data; we will use the information from *DataSet/DataTable* while designing the report.

Following step are required to have *DataTable* added to *DataSet(dsProduct)*:

- Double click on *dsProduct* from Solution Explorer; it will open the designer view. Right-click on the designer surface and **Add -> DataTable**. Please click on header and change the name to *dtProductList*. Please see **Image 3**.

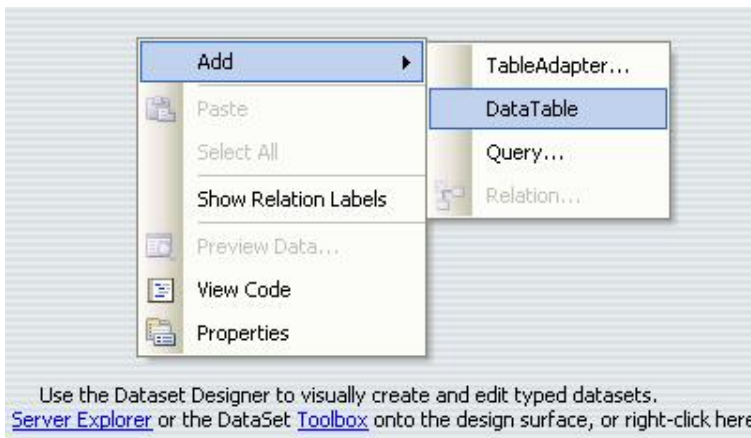


Image: 3

Lets start adding columns to *DataTable(dtProductList)*. Your designer screen should look like **Image 4**. Right-click on *dtProductList* and select **Add -> Column** to start adding columns to *DataTable*.

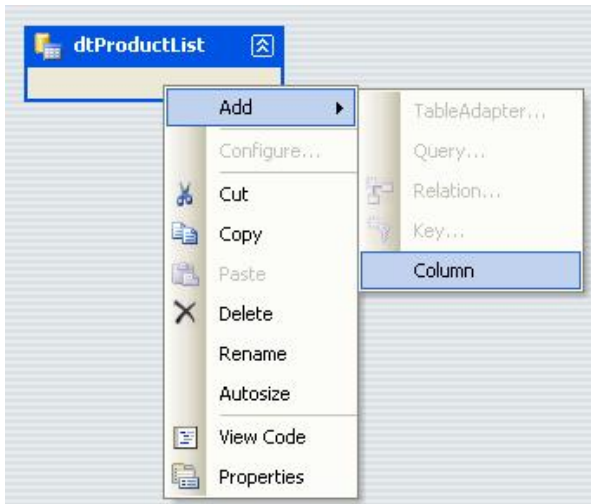


Image: 4

Please repeat the action for following columns:

- *ProductName* (String)
- *QuantityPerUnit* (String)
- *UnitPrice* (Double)
- *UnitsInStock* (Double)
- *UnitValue* (Double) A calculated field based on *UnitsInStock* * *UnitPrice*

As you are adding columns, by default it is string data type. Please go to properties windows after selecting column to change it from String to Integer or Double.

Please see **image 5**. Your *DataTable* should look the same. Also, you can see the properties window to change the data type.

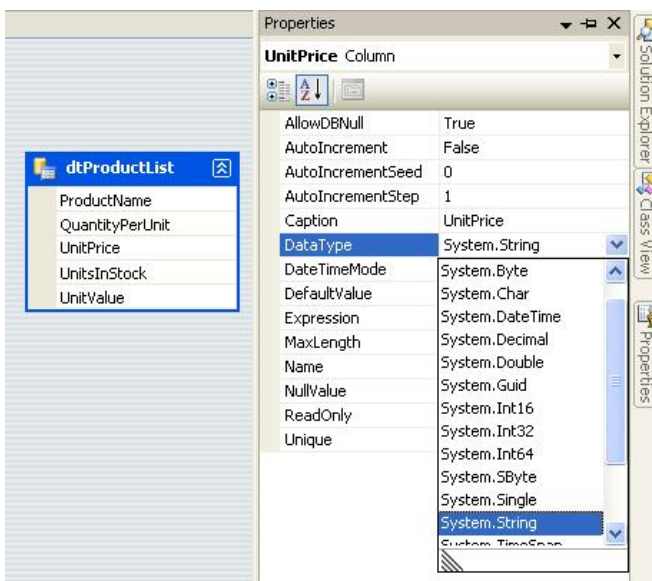


Image: 5

Have you heard of **Typed DataSet**? If not, then we have just created a *Typed DataSet* here. Please consult online help to know more about *Typed DataSet*.

Step 4: Add Report to the Project

All right, so far we created the project; added Report Viewer and DataSet. Now, it is the time to deal with star of the show! Lets create that neat report. The following steps is required to have *Report (rptProductList.rdlc)*:

- Select **Add -> New Item -> Report** from Solution Explorer. Change name from *Report1.rdlc* to *rptProductList.rdlc* and click on Add button to complete the action.

Typically, after add action is finished your screen should be similar to **Image 6**. When a report is added to project, it is ready to use the *DataSet* for designing.

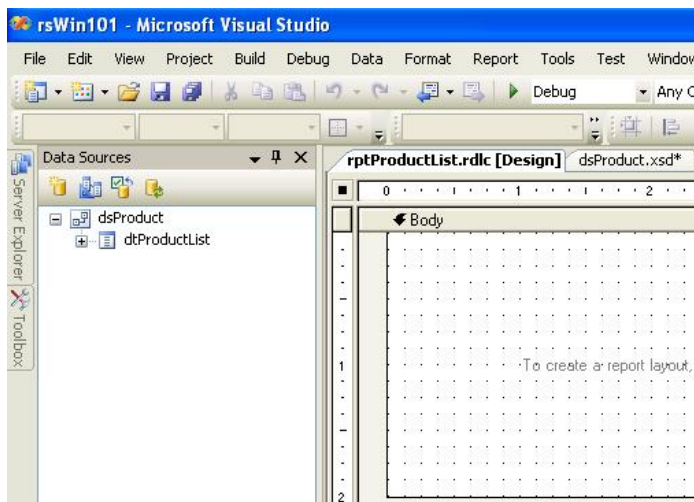


Image: 6

Whether this is your very first report or you are a reporting junkie like me; we have to deal with the most basic building blocks of report writing, which is: Header, Body and Footer.

Typically, reports are designed with specific page size and layout in mind. Our report is Letter size and Portrait layout. You can explore various properties attached to report layout by right clicking anywhere on open designer surface and select properties.

It is always advisable to draw a prototype of your report on paper, before you start the design attempt. As you can see in **Image 1**, we have Report Name and Report Date in header section. The body section has the product list information together with summary totals; and footer carries the Page Numbers.

Let's start working on Page Header: When new report is added to project, by default, all you will see in report designer is the body section. Right click on report designer surface anywhere other than body and select Page Header. This will add header to report. Feel free to adjust the height of header and body section. See Image 7, I have reduced the height of body and increased the height of the header.

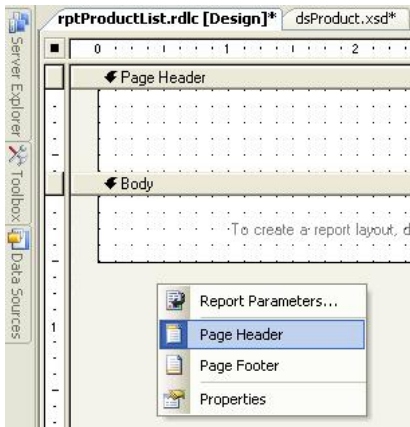


Image: 7

While inside the report designer, if you explore the Toolbox, you will see variety of controls which can be used to design report. For our example, we will use, *TextBox*, *Line* and *Table* control. I would encourage you to go through the online documents if you need detailed information for all available controls.

Header Section

Lets start designing the header. We will start by dragging two *TextBox* and dropping on header section. *Textbox* can show both static and dynamic data. *Line* control is used to separate header from body section.

After dropping controls over report designer surface, you can control the look and feel by changing associated properties. We will designate one *TextBox* to report title and another one to show current date. We can directly type static text into *TextBox* control by selecting it and start typing inside.

Please change following properties of Title *TextBox*:

```
Value = "Product List"
Color = Purple (you like purple for title, right?)
```

Please change following properties of Date *TextBox*:

```
Value = ="Run Date: " & Today
Color = Purple (you like purple for title, right?)
```

Please note *Value* property for Date *TextBox* starts with a = sign. This is not a simple static text, instead it is an expression. This expression is a result of string Run Date and **VB.NET script** keyword *Today* (to get current system date). You can specify desired names to all objects in report; I choose to stay with default name for most of the controls, however, for demo purpose I did specified *txtTitle* to Title *TextBox*.

Please refer to **Image 8**; your finished design for header should look relatively same.

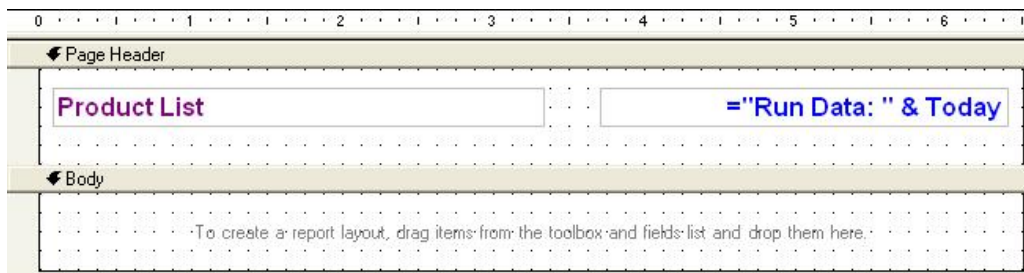


Image: 8

Body Section

The Body section, also referred as details section, is by far the most important part of the report. As you can see when we added the report to the project; body section was added for us automatically. All we have to do is start putting controls on it.

Traditionally the Body section is used to display details (in our example it is product information) usually more than one row of information. The Body section can expand as per the growth of reported data. Often report is designed with intention to have one physical page (Letter/A4 etc.) output; in this case Body section still can be used to display information. Out of *Table*, *Matrix* and *List*, the three most commonly used control on Body section; we will use *Table* control for our example. All three can repeat information; *Matrix* goes a step further and even produces Pivot output.

Lets drag and drop *Table* control on body section of report designer surface. If you notice, this action will produce a table with three rows and three columns. You may have also noticed that center column also has been labeled: Header, Detail and Footer. Now, dont be surprise if I tell you that *Table* control is nothing but bunch of *TextBox* attached together! Yes, each and every Cell in *Table* is like *TextBox*, which means you can either type static text on it or specify a dynamic expression.

Before we start designing the Body section, lets add two more columns (remember we have total of five columns in the report). Adding columns is easy; please do the following to get new columns added to report:

- Select Table Control inside Body section
- Click on right most column header (I assume we are adding new columns to right side)
- Right click on header and select -> **Insert Column to the Right**

Make sure your report resemble to **Image 9**. Feel free to adjust the width of column based on length of data it will hold.

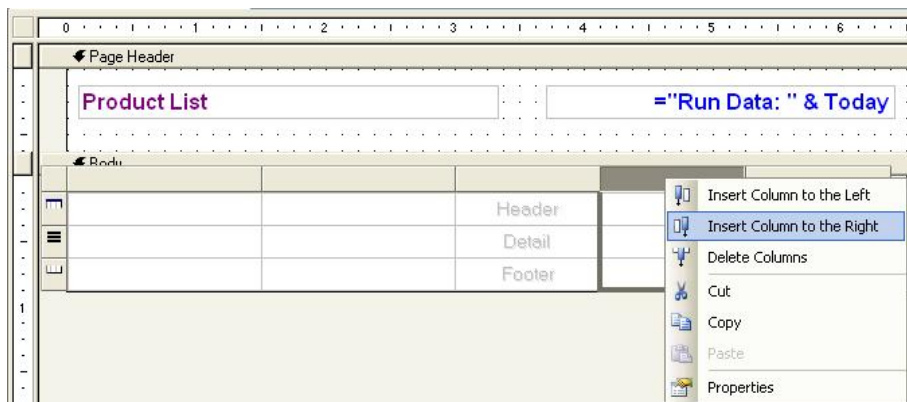


Image: 9

I am sure majority of us have used Excel or something similar; think of same for *Table* control as mini worksheet. We can apply borders, change font of individual cell etc. etc. So, all you have to do is to think of desired formatting theme and start applying it.

Starting with first column to the last one, please **click on individual column header cell** and type the following text:

```
Header 1: "Product Name"
Header 2: "Packaging"
Header 3: "Unit Price"
Header 4: "Units in Stock"
Header 5: "Stock Value"
```

Lets continue to do so the same for Detail section, here one thing to know is, instead of text we have to type the expression which is columns from *dsProduct.dtProductInfo*. You can either type the expression or simply drag and drop the column from Data Sources Toolbar (see **Image 7** on left side).

In case if you decide to type it out, starting with first column to the last one, please **click on individual column detail cell** and type the following text:

```
Detail 1: "=Fields!ProductName.Value"
Detail 2: "=Fields!QuantityPerUnit.Value"
Detail 3: "=Fields!UnitsInStock.Value"
Detail 4: "=Fields!UnitPrice.Value"
Detail 5: "=Fields!UnitsInStock.Value * Fields!UnitPrice.Value"
```

Please take notice of Detail 5: it is the calculated output by performing multiplication of Units in Stock and Unit Value.

Tip: If you drag and drop the column to detail section of Table control, it will try to add column header automatically, if column header is empty.

Finally, lets add summary total in footer section of Table control. Please make sure to select footer cell on column 4 and 5 inside Body section and type following text:

```
Cell 4: "Total Value:"
Cell 5: "=SUM(Fields!UnitsInStock.Value * Fields!UnitPrice.Value)"
```

Please check the expression in Cell 5; I am using a built-in function **SUM()** to find out total stock value of all the products listed in report.

Footer Section

Before we start writing some cool C# code to bring our report alive, lets finish the report footer section. As we have added report header earlier, similarly we have to right click on open report designer surface and select **Page Footer**

(see **Image 7**). Drag and drop a Line and TextBox control on Footer section. Please type the following expression inside TextBox:

```
Value: ="Page: " & Globals!PageNumber & "/" & Globals!TotalPages
```

As you can see I have used *PageNumber* and *TotalPages*, both are Global variables maintained by the reporting engine.

Tip: Make sure all expression you type must start with = in front of it.

Please make sure your report looks like **Image 10**. As you can see I have introduced some color and right alignment to numeric data etc. Feel free to try out all the different formatting options, just think of Table control as mini spreadsheet with columns and rows and now you know all the formatting you can try on them.

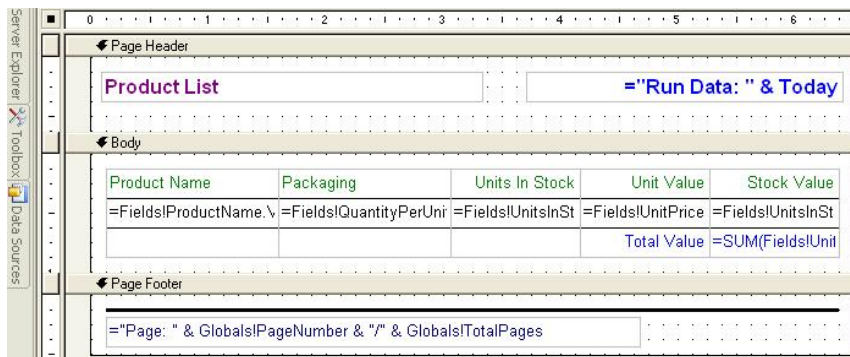


Image: 10

Expression Builder

Expression builder is a very powerful feature of Reporting Services. As you can see in **Image 11**, Stock Value is calculated with the help of SUM function. All fields in *DataSet* can be access with *Fields!* keyword.

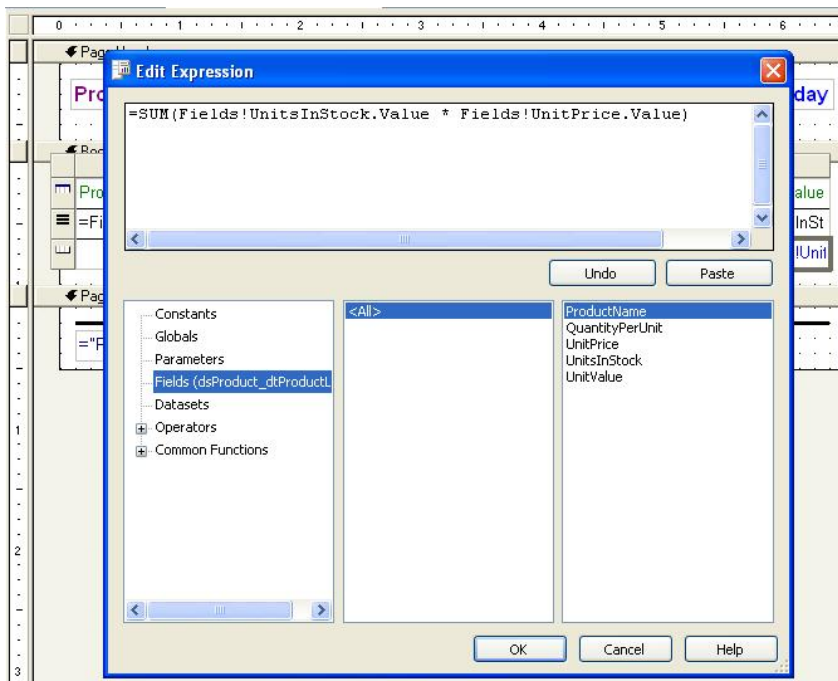


Image: 11

Step 5: Lets write some C# code to bring life to our report

Phew I hope you guys are not exhausted already. Hang in there; we are on last step now. Its like we have waited for that long nine months and the time has come to witness the miracle of birth.

From solution explorer, select Form1. Right click on surface of form and select View Code.

```
using System.Data.SqlClient;  
  
using Microsoft.Reporting.WinForms;
```

Make sure the Form1_Load event has following code: (code at www.sqlservercentral.com)

You might be wondering why I have used TOP 5 for select query; the reason is, I wanted to limit the output so that I can show you summary total in **Image 1**.

Tip: Name property of *ReportDataSource* object should be always DataSet_DataTable.

Conclusion

Although, I tried to keep the language of this article as simple as possible; however, please feel free to get back to me if you need any further clarification. I consider myself a budding author; I have to learn a lot; it is the reader like you, who has always helped me to improve my writing.

I am looking forward to receive any comments/suggestion you have for me. Thank you for reading; I sincerely hope this article will help you a bit or two to know reporting services better through my applied approach.

Data Driven Subscriptions for Reporting Services (2000 and 2005)

By Jason Selburg

Introduction

As I covered in my previous article, there are many limitations of the subscription feature that comes with SQL Reporting Services 2000 and 2005 Standard Editions. I dug into the RS database and came across a way to push my information into the subscription, call it and then leave as if I was never there. After my initial article, I came across several ways to improve its' stability and performance. The following method/procedure is much more stable. Although I received many requests to handle multiple parameters, I quickly realized that there are countless ways to address this and it really depends on your personal preference and each report's possible requirements.

- Are some or all parameters set based upon other parameters?
- Are some or all parameters static for all or some instances of the particular subscription run?
- What parameters are defined in a set table, or come from other sources in your database?

Another request was to handle the "File Share" delivery method. To be quite honest, the username and password encryption has me stumped. So if any of you have a suggestion or solution, let us all know. However, I did make a few key improvements to the original procedure. These Include:

- Comprehensive error handling
- A more efficient/stable method of updating the subscription settings and returning them to the original state.
- A method that allows the naming of your procedures, therefore reducing the headaches of administration and maintenance.
- Better commented code for your benefit.
- The ability to specify the rendering format of the report.
 - These may be different depending on the installation and configuration of your server, but these are listed in the "reportServer.config" file.
 - This file is located in a folder similar to "C:\Program Files\Microsoft SQL Server\MSSQL.2\Reporting Services\ReportServer\"
 - The standard formats are:
 - XML
 - IMAGE
 - PDF
 - EXCEL
 - CSV

Assumptions

1. You know how to create a standard subscription.
2. You are familiar with executing stored procedures.

Instructions

1. Create or Replace the stored procedure and History table.
2. Create a standard subscription, placing the name of your subscription in the subject line of the email.

Report Delivery Options

Specify options for report delivery.

Delivered by: Report Server E-Mail

To:	<input type="text" value="any valid email"/>
Cc:	<input type="text"/>
Bcc:	<input type="text"/>
(Use ";" to separate multiple e-mail addresses.)	
Reply-To:	<input type="text"/>
Subject:	<input type="text" value="your report name here (rpt_monthlySales_01)"/>
<input checked="" type="checkbox"/> Include Report	Render Format: <input type="text" value="Web archive"/>
<input checked="" type="checkbox"/> Include Link	
Priority:	<input type="text" value="Normal"/>
Comment:	<input type="text"/>

NOTE: It is strongly suggested that you define a naming convention that will not allow duplicate subscription names.

1. Set the beginning and ending date to dates prior to "today". This will keep your subscription from running unless you explicitly call it.
2. Execute the procedure with the (procedure's) parameters you require.

Conclusion

As there is no way in telling what Microsoft will change in upcoming releases this method may become obsolete if the Reporting Services database changes, so be careful when implementing this as a full scale solution to your subscription needs. However, this procedure works under all versions of SQL RS 2000 and all current versions of RS 2005, so the chances are good that it will continue to work. Again, any comments, suggestions or improvements are welcome, email me.

Acknowledgements

Thanks to Hugh Thomas for discovering the method to "wait" for a report's execution.

Stored Procedure code is available at www.sqlservercentral.com.

Service Broker

This year we have added a new section that we expect to grow more and more over the next few years. Service Oriented Architecture (SOA) applications are catching on as a way to develop more flexible applications and reuse more code throughout your organization. With Service Broker, SQL Server 2005 brings us a platform that allows you to easily develop SOA applications.

The articles we bring you this year are basics that look at this new technology from a basic point of view. We hope you enjoy these and contribute one of your own next year.

Adventures With Service Broker.....	286
Building a Distributed Service Broker Application	292

Adventures With Service Broker

By Johan Bijmens

Introduction

As a part of my SQLServer2005 learning curve, the focus for now is Service Broker. I first heard about SQL Server Service Broker during introduction sessions for Yukon.(codename for what later has been named SQL Server 2005). Many developers were looking forward on the integration of CLR in SQL Server, meaning they wouldn't have to learn and use SQL. We all know by now that's a myth. The second thing developers liked was message queuing also had been integrated. Let's just say they were blinded by the light. But indeed, it is message queuing. But it isn't the MSMQ they have known.

The reason why I was considering service broker was that I introduced a system of raising SQL Server alerts to launch SQLAgent jobs. ([help to tighten use of cmdshell or sp_start_job](#)). Maybe I could ship those jobs of to another (existing) application server, in stead of the actual db-server, so the dbserver could reclaim the full dedicated server's capabilities.

So what is Service Broker?

Simply said: "SQL Server to SQL Server message queuing"

How hard can it be?

How to use this article

Let me start with mentioning I use SQL2005 Std (and EE) with SP1.

With this article, you'll find the [solutions](#) I used with SQL Server Management Studio. Download them and keep them at hand, you'll find a solution per subject mentioned later on in this adventure.

Each solution contains a sequence of scripts per phase and per broker-party. You'll notice that script-sequence 004, the execution, doesn't change sequence number. This is on purpose so everyone can refer to it without being confused over a sequence number. Also, this way, you can experiment with the kind of error messages SQL Server raises, when and where they are raised.

Building the whole sequence per party is, in my opinion, very welcome when finding scripts regarding new stuff. So I'd like you to run the scripts on a test server, one step a time, because the scripts may result in messages for you to move or copy files from one server to another.

Just keep in mind always to start with the cleanup steps, because each solution is supposed to stand by itself. Take good care to connect to the correct instance when you test the multi-instance solutions.

When you get to use a "routing" script (sequence 007) check the create route statement so it points to your correct test server. If you're using firewalls, you may need to open the ports you've picked when creating the endpoints (listener_port).

I hope you'll enjoy my adventure and get a good reference in the scripts I've provided.

First things I did for service broker exploration

- Search the forums to find out if something is too buggy to start with in Service Broker. Like always, [Google](#) or [MSN Search](#) are your friends.
- Search the forums for some more examples. Since I went to the [SQLPass2006](#) convention in Seattle, finding an example has been the easy part. (session AD405 "Message in a bottle")
- Read BOL (yes, *I do*) in a diagonal fashion.
- *Ouch!* ...XML...well, not really needed, but it makes life so much easier. So have a glimpse at it!

I wanted to drive the car, I didn't want to become a car technician. Casual Ferrari-drivers just drive the car and especially don't touch the nuts and bolts under the hood, right? That was my aim!

Phase one: a simple LOCAL service broker scenario.

This is a piece of cake, because most examples you'll find are implemented in the local SQL Server instance. First of all the elementary concepts need to be known. Also elementary, but you have to know some terminology, what's their function and why they are used.

- How do we call the participants
- message types
- Contracts
- Queues
- Services
- BEGIN DIALOG CONVERSATION
- Send message
- Receive message
- End conversation
- and find out that there is no easy GUI-setup path for service broker !

I tested my first SSB application with sender and receiver in a single database. The example I used was the one provided by Bob Beauchemin in his SODA presentation. (scripts not included with my solutions but you can find them at <http://www.sqlskills.com/resources/conferences/200510SODA.zip>)

Copy/Paste of the example just did the trick. It was also nice to see that you don't need to stop / start SQL Server to enable you first SSB-service. (Unlike with SQLAgent alerts for sql7.0 and sql2000)

Now it was worth to investigate more since I had seen the light. At that time, surfing the web, I stumbled onto Klaus Aschenbrenner who asked for a SSB-newbie to co-read the book he was writing about SSB. (<http://www.csharp.at/blog/>) I contacted him and he did send me his first couple of chapters. This was actually the first time I co-read chapters of a book, and I enjoyed doing the co-reading and send him my comments. I learned a bunch of things that I normally wouldn't even consider digging into when throwing a first glimpse to a product, because it also handles nuts and bolts.

There are also a couple of nice articles on the web regarding "when to use the various asynchronous technologies". It makes sense to have some selection criteria about which technology to use, so have a look.

Phase two: sender and receiver in separate databases.

Secondly I tested a SSB application with sender and receiver in separate databases. For this example I started with a scenario I found with a presentation I attended at Flexcom-Azlan at the end of 2005. <http://www.azlan.be>. My thanks to them for allowing me to elaborate on their example.

This scenario has a concept of parties called *Airline* and *TravelOffice*. Because these names generate typical scenarios with most people, regarding "asynchronous processing", I thought it would make life easy to elaborate on that.

Basic scenario: An airline-company provides an SSB-service "TravelOfficeBooking" composed of an "OpenBooking", "SeatBooking", "CloseBooking" and a "BookingConfirmation" method.

You can find this test in my solution in folder [01_1_ALZDBA_ServiceBrokerSequence_Local](#). When I run the scripts from the solution, I just run them in sequence, actually building the situation one step at a time, each time the same step for each database. Working this way clears out the parallels and the differences at each side of the service broker services.

The most important thing I had to do is to alter the database that contains the initiator because it has to have the "trustworthy ON" property set for local usage.

So within the same instance of SQL Server, things still work fine by default, provided you performed the previous alter database.

Phase three: sender and receiver in separate instances

My third test was now to detach the initiating database and move it to a second instance of SQL Server. I installed that second instance on the same physical box. (Or should I say operating system, because nowadays virtualisation is hot) For the record, all my SQL Server instances are being served using a windows domain account and have builtin\administrators revoked.

This time there is no need for the database to be trustworthy because that setting is only used within the SQL Server instance. Guess what. It didn't work anymore! This urged the need for DMV's and queries to find out what's going on and where things got stuck. Btw: I published my [Troubleshooting Service Broker \(SSB\)](#) script at www.SQLServercentral.com

Why didn't it work?

Transport security

You can find this test in my solution in folder [02_1_ALZDBA_ServiceBrokerSequence_Transport](#). If you go outside of your SQL Server instance, you need to setup transport security and routing. This thing is secured by default, so time needed to be spent to check out the options.

You might imagine transport security to be the choice of your package-delivery-server like DHL, UPS, ABX,.... Without *dialog security* you would be giving them your love letters in an open envelope or in transparent foil. With *dialog security* on the other hand, you'd hand your package-delivery-server an envelope or a vault containing your encrypted message.

My choice was to always use SSL-certificates when setting up transport security. Don't be frightened by the word SSL, you get it from SQL Server itself! Why SSL-certificates? Because with SQL Server 2005 you can create your own SSL certificates, so you don't have to hassle outside of your beloved and known software. I won't go into strategies or politics regarding this security-choice. That's something you'll have to determine yourself, but it makes it easy for me during this experiment. Like the word says, Transport security is all about transport. There is only transport when you go from one SQL Server instance to another.

Because we have a standard for composing a sa-password, I thought it might be opportune to also use a standard to compose the password used for the master database master key. This key is then used to generate the certificate to be exchanged between the instances for transport security. With SQL Server 2005 this part is handled in the master databases of the involved instances.

It is mandatory to give SEND authority on the service to PUBLIC if you do not implement dialog-security!!

You can only have one Service Broker endpoint per SQL Server instance, so before you rollout an SSB application, play with it so you learn what things to keep in mind and which decisions need to be taken.

Since you only have one Service Broker endpoint, and the transport security concern inter-instance communication through the endpoint, you need to tell the endpoint it has to use your certificate. In my experiment's case resulting in:

```
CREATE ENDPOINT BrokerService

    STATE = STARTED

    AS TCP (LISTENER_PORT=55552)

    FOR SERVICE_BROKER (

        AUTHENTICATION= CERTIFICATE
[AirlineServiceBROKERCommunicationCertificate]

        , ENCRYPTION=SUPPORTED

    )
```

I also decided to create a login per connecting instance because this gives me an excellent opportunity for auditing.

So after creating the SSB-endpoint that uses the communication-certificate, certificates need to be exchanged. Need it to be mentioned "what's in a name" ...

There was still a route to be created at each side of the transport. This route resides at database level, meaning it is no other databases business to know who you route your messages to. Because I was only setting up transport security at this time, I needed to grant send authority to public for my local SSB-service in the userdatabases.

I need to emphasize that I did look over the impact of this last line a number of times because as a DBA, I never grant anything to public. So never say never ...!

```
GRANT SEND ON SERVICE::[AirlineBookingService] TO PUBLIC ;-- airline db

GRANT SEND ON SERVICE::[TravelOfficeBookingService] TO PUBLIC --TtravelOffice db
```

Things started to work again.

Dialog security: You can find this test in my solution in folder [02_2_ALZDBA_ServiceBrokerSequence_AnonymousDialog](#)

So how about dialog security? As you can imagine by now, as long as your message stays within the same instance of SQL Server, it is considered to be secure. Dialog security is about encrypting the content of your message. By itself this has nothing to do with transport, but you'll need transport security to get to your counterpart instance.

So dialog security actually is an additional security measure which lives its own life, but needs transport. Like when you send an encrypted letter using a package-delivery-server. A dialog resides in the userdatabases of the instances, so dialog security is a user databases procedure.

Just like with the setup of transport security, my choice was to always use SSL-certificates when setting up dialog security. Just keep in mind there are other options to be explored. So you also need a user database master key to encrypt the SSL-certificate you create to support dialog security. Then also telling the service to encrypt dialogs using a specific certificate:

```
CREATE USER [TravelOfficeServiceUser] WITHOUT LOGIN;

-- This defers from Transport security! This way the dialog gets encrypted
-- by the certificate bound to the service USER !

Alter AUTHORIZATION ON SERVICE::[//wonderland.world/Broker/TravelOfficeBookingService]
    to [TravelOfficeServiceUser] ;

go

CREATE CERTIFICATE [TravelOfficeDialogCertificate]
    AUTHORIZATION [TravelOfficeServiceUser]
    WITH SUBJECT = 'Certificate for the TravelofficeService service',
    START_DATE = '11/30/2006',
    EXPIRY_DATE = '12/31/2050';

go
```

Exchanging these dialog certificates and implementing in both userdatabases and off course telling the sprocs to begin the dialog using encryption and off we go ... **NOT**. *(Kind of slang my teenage daughter uses these days)*

```
CREATE BEGIN DIALOG @dialog
    FROM SERVICE ..
    TO SERVICE
    ON CONTRACT
    WITH ENCRYPTION = ON;
```

One little bit I overlooked was that you also have to create a remote service binding at initiator side when using dialog security!

```
CREATE REMOTE SERVICE BINDING [AirlineServiceBinding]
    TO SERVICE '//wonderland.world/Broker/AirlineBookingService'
    WITH USER = [ProxyAirlineServiceUser] , ANONYMOUS = ON ;
```

With my scripts, you'll also find a solution using full dialog security (anonymous=off). You can find this test in my solution in folder [02_3_ALZDBA_ServiceBrokerSequence_FullDialog](#). In that case there are dialog certificates to be exchanged at both sides of the conversation.

Phase four: Forwarding

Ok, so now I had tested a simple scenario with two participators. Hey, I have heard you could move your db to another server without informing your counterpart. So I focused on forwarding.

Once again I started off with the example where only transport security was implemented. Easy does it, so off we go.

The scenario is simple: all you do is detach the database, move it to another SQLServer2005, attach it like you would when moving any user databases (users & stuff). All you have to do is to setup transport security from the original server to the new server, setup the forwarding and off course also altering the existing route to your forwarding server.

Setting up forwarding is actually quit easy, because it is handled in the original master and msdb. You have to tell the endpoint for service broker (master db) that message forwarding is enabled. In msdb you provide the forwarding details by providing the routes (forward and backward).

You can find this test in my solution in folder [03_1_ALZDBA_ServiceBrokerSequence_Transport_Forwarding](#)

Guess what..... It worked!

It got a bit tricky when I tried it with the scenario where I implemented dialog security. Well it was actually the same as with the example with only the transport security implemented, but this time it no longer worked. You can find these tests in my solution in folders [03_2_ALZDBA_ServiceBrokerSequence_AnonymousDialog_Forwarding](#) and [03_3_ALZDBA_ServiceBrokerSequence_FullDialog_Forwarding](#)

And I had to actually use the SQL Server-errorlog to figure out why it didn't work. It stated it couldn't decrypt the message because of lack of a key to open the database. Which certificate or key? The database's master key!

I had to provide the new server the knowledge how to open the old database's master key to decrypt the dialog. Because I encrypted the database originally using a password, I used that same password to tell the new server how to open the database master key.

```
EXEC sp_control_dbmasterkey_password @db_name = N'Airline',  
                                     @password = N'P@ssword', @action = N'add';
```

My conclusion

I learned a lot and wanted to share, just to give everyone the chance to experiment the copy/paste way with this powerful out of the box solution of our beloved SQL Server 2005. Combined with the obvious advantages for messages using XML, SSB can be very powerful.

The things that need to be figured out before implementing it into production:

1. Elaborate on security.
2. Naming conventions for type, queues, services,...
3. Guidelines for transport security
4. Guidelines for dialog security

5. Is using SSL certificates the way to go?
6. Selection criteria for SSB to be used.
7. XML. We have to learn it anyway because it makes our performance related life more meaningful figuring out bottlenecks, execution plans,...
8. How to handle poison messages. Detection, notification and solution.

There are user efforts to build a GUI for SSB administration: e.g. [SSB Admin - a GUI for SQL Server 2005 Service Broker](#). For more info regarding the nuts and bolts of service broker I gladly redirect to Klaus Aschenbrenner's book, which is still in the publishing process at this time.

Building a Distributed Service Broker Application

By Santhi Indukuri

In this article, we will discuss about the advanced service broker objects used for building a Distributed Service Broker Application. We will see how messages will be transferred between two databases existing in two different servers.

Pre-requisites: Should have basic knowledge of Service Broker Basic Concepts like Message Types, Contracts, Services, Queues, Conversation, Sending messages on Conversation and Receiving messages.

Advanced Service Broker Objects

For Service Broker Applications which use the same database in the same server, we don't need to use the Advanced Service Broker Objects.

The following are the advanced Service Broker Objects used by Distributed Service Broker Application.

- **End Point:** End Points will accept incoming and outgoing TCP/IP connections on a Specific port. We can have only one End Point for instance which can be shared between all services in the instance.
- **Routes:** Route is used to locate a service that it is sending message to. When no route is explicitly associated with a service then Service Broker will deliver the message within the current instance.
- **Remote Service Binding:** Remote Service Binding (RSB) is used to establish security credentials which will be used by Initiating service to authenticate itself with the Remote Service. RSB uses a Certificate associated with the specified database user account to connect to Remote Instance

For more details on Service Broker Objects, refer to SQL Server Books Online.

Security

Before proceeding further, we should know how Service Broker Security allows services to communicate securely, even if they are located on different computers.

Service Broker security relies on certificates that are shared among remote databases, however no other information is shared. Service Broker allows two types of security.

- **Dialog Security:** It provides remote authorization for conversations to specific services and encrypts individual messages when the message leaves the sending instance until the messages reaches the destination instance (end-to-end encryption).
- **Transport Security:** It prevents unauthorized network connections from sending Service Broker messages to databases in the local instance. It controls which instances can communicate and

provides encryption between the two instances, but it doesn't secure the contents of individual messages

Steps to create a Distributed Service Broker Application

1. Create the basic Service Broker Objects (i.e. message types, contracts, services, queues etc)
2. Set up Transport Security:
 1. Create a master key for master database.
 2. Create certificate and End Point that support certificate based authentication. (i.e. creating a Private Key for the Server)
 3. Take a backup of the certificate created and install it into the remote instance.
 4. Create certificate from the certificate backup file copied from the other server. (i.e. creating a Public Key of the Remote Server in current server)
 5. Create login from the certificate created in Step 4.
 6. Grant the login, connect permissions on the end point.

Note: Steps 1-6 should be performed in both the servers

3. Set up Dialog Security:
 1. Create a master key in the local database i.e. the database we are going to use for our application.
 2. Create a user certificate. (i.e. creating a Private Key for the Server)
 3. Take a backup of the user certificate created and install it into the remote instance.
 4. Create a user with the same name as the user who has access rights on the other Database.
 5. Create a user certificate from the user certificate backup file copied from the other server, allowing authorization to the user created in Step 4. (i.e. creating a Public Key of the Remote Server in current server)
 6. Grant connect permissions to the user.
 7. Grant send permissions to the user on the local service
 8. Create a Remote Service Binding with the user created.

Note: Steps 1-8 should be performed in both the servers

4. Send Messages & Receive Messages

Example

In this example we'll first send message from one Server to another Server and the server that received the message processes the message and sends a message back to sender server. We'll be using two servers, Server A and Server B. And the databases used are DatabaseA (in Server A) and DatabaseB (in Server B).

The following are the tasks performed by our Sample Application.

- a) Server A sends message to Server B
- b) Server B receives the message and sends a message to Server A.

Steps:

I. Create Basic Service Broker Objects:

In DatabaseA in Server A, Let's perform the following Operations

- 1) Create Message Types

```
Create Message Type SenderMessageType validation=NONE  
Create Message Type ReceiverMessageType validation=NONE
```

2) Create Contract on the above message types

```
Create Contract SampleContract  
(  
    SenderMessageType SENT BY INITIATOR,  
    ReceiverMessageType SENT BY TARGET  
)
```

3) Create an Initiator queue

```
Create Queue InitiatorQueue  
WITH status = ON
```

4) Create a Service on the queue and the contract

```
Create Service SenderService ON QUEUE InitiatorQueue (SampleContract)
```

In DatabaseB in Server B, Lets perform the following Operations

1) Create Message Types:

```
Create Message Type SenderMessageType validation=NONE  
Create Message Type ReceiverMessageType validation=NONE
```

2) Create Contract on the above message types

```
Create Contract SampleContract  
(  
    SenderMessageType SENT BY INITIATOR,  
    ReceiverMessageType SENT BY TARGET  
)
```

3) Create an Target queue

```
Create Queue TargetQueue WITH status= ON
```

4) Create a Service on the queue and the contract

```
Create Service ReceiverService ON QUEUE TargetQueue (SampleContract)
```

Note: In the above code snippets we have created identical Message types and Contracts in both the servers. We need to create identical Message Types and Contracts in each database that participates in the conversation.

II. Create a Route:

Once the Services are created on both the servers we need to create routes in each database and associate it with a remote service to which it is sending message to.

In DatabaseA in Server A,

```
Create Route RouteA  
  
WITH  
  
    SERVICE_NAME = 'ReceiverService',  
  
    BROKER_INSTANCE = '1B9C40BC-7FCF-41F7-9813-61C11A49D0DE',  
  
    ADDRESS = 'TCP://157.57.100.9:4022'  
  
GO
```

In the above Route, ReceiverService is the service in DatabaseB of Server B. If we don't specify the broker_instance then the service with a similar name will be randomly picked by the server B from any database. But if you want to specifically mention that we need to map to the ReceiverService of DatabaseB, then we need to get the Service_broker_guid from sys.databases for DatabaseB using the following query.

```
select service_broker_guid  
  
from sys.databases  
  
where name = 'DatabaseB'
```

The address field tells us that we need to connect to 4022 port of Server B and IPAddress of ServerB in 157.57.100.9.

In DatabaseB in Server B,

Create a Route in the same manner. (We need to create this route in our example, as we are sending a message back to Server A, once we process the message sent by Server A in Server B)


```
Create Route RouteB

WITH

    SERVICE_NAME = 'SenderService',

    BROKER_INSTANCE='D164787D-590A-47AC-83AB-987F880E3F2A',

    ADDRESS = 'TCP://172.22.26.216:4022'

GO
```

III. Set up Transport Security:

Note: All actions related to Transport Security will be performed in the master database of the Servers.

- 1) Create a master key for master database.
- 2) Create certificate and End Point that support certificate based authentication.

Server A:

```
Use master

Go

--1. Create a master key for master database.

Create Master Key Encryption BY Password = 'gs53&"f"!385'

Go

/*2.Create certificate and End Point that support

    certificate based authentication

*/

Create Certificate EndPointCertificateA

WITH Subject = 'A.Server.Local',

    START_DATE = '01/01/2006',

    EXPIRY_DATE = '01/01/2008'

ACTIVE FOR BEGIN_DIALOG = ON;

GO

CREATE ENDPOINT ServiceBrokerEndPoint

    STATE=STARTED

    AS TCP (LISTENER_PORT = 4022)

    FOR SERVICE_BROKER
```

```
(  
  
    AUTHENTICATION = CERTIFICATE EndPointCertificateA,  
  
    ENCRYPTION = SUPPORTED  
  
);
```

Server B:

```
Use master  
  
Go  
  
--1. Create a master key for master database.  
  
Create Master Key Encryption BY Password = '45Gme*3^&fwu';  
  
Go  
  
--2.Create certificate and End Point that support certificate based authentication.  
  
Create Certificate EndPointCertificateB  
  
WITH Subject = 'B.Server.Local',  
  
    START_DATE = '01/01/2006',  
  
    EXPIRY_DATE = '01/01/2008'  
  
ACTIVE FOR BEGIN_DIALOG = ON;  
  
GO  
  
CREATE ENDPOINT ServiceBrokerEndPoint  
  
    STATE=STARTED  
  
    AS TCP (LISTENER_PORT = 4022)  
  
    FOR SERVICE_BROKER  
  
    (  
  
        AUTHENTICATION = CERTIFICATE EndPointCertificateB,  
  
        ENCRYPTION = SUPPORTED  
  
    );
```

3) Take a backup of the certificate created and install it into the remote instance by physically copying this certificate to Other Server.

Server A:

```
BACKUP CERTIFICATE EndPointCertificateA

TO FILE =

'C:\Documents and Settings\Santhi\Desktop\Service Broker\Session\EndPointCertificateA.cer';

GO
```

Copy the certificate from the above location to the following location: Destination: Server B - Path: C:\Documents and Settings\Santhi\Desktop\ServiceBroker\

Server B:

```
BACKUP CERTIFICATE EndPointCertificateB TO FILE=

'C:\Documents and Settings\Santhi\Desktop\ServiceBroker\EndPointCertificateB.cer';

GO
```

Copy the certificate from the above location to the following location: Destination: Server A , Path: C:\Documents and Settings\Santhi\Desktop\ServiceBroker\Session\

4) Create certificate from the certificate backup file copied from the other server.

Server A:

```
Create Certificate EndPointCertificateB

From FILE =

'C:\Documents and Settings\Santhi\Desktop\Service Broker\Session\EndPointCertificateB.cer';

GO
```

Server B:

```
Create Certificate EndPointCertificateA

From FILE =

'C:\Documents and Settings\Santhi\Desktop\ServiceBroker\EndPointCertificateA.cer';

GO
```

5) Create login from the certificate in remote server in the current server.

Server A:

```
CREATE LOGIN sbLogin
FROM CERTIFICATE EndPointCertificateB;

GO
```

Server B:

```
CREATE LOGIN sbLogin
FROM CERTIFICATE EndPointCertificateA;

GO
```

6) Grant the login, connect permissions on the end point.

Server A:

```
GRANT CONNECT ON ENDPOINT::ServiceBrokerEndPoint To sbLogin

GO
```

Server B:

```
GRANT CONNECT ON ENDPOINT::ServiceBrokerEndPoint To sbLogin

GO
```

IV. Set up Dialog Security:

Note: All actions related to Dialog Security will be performed in DatabaseA of Server A and DatabaseB of Server B, not in master databases.

1) Create a master key in the local database i.e. the database we are going to use for our application.

Server A:

```
Use DatabaseA

GO

Create Master Key Encryption BY

Password = 'gs53&"f"!385'

Go
```

Server B:

```
Use DatabaseB

GO

Create Master Key Encryption BY

Password = '45Gme*3^&fwu';

Go
```

2) Create a user certificate.

Server A:

```
Create Certificate UserCertificateA

WITH Subject = 'A.Server.Local',

    START_DATE = '01/01/2006',

    EXPIRY_DATE = '01/01/2008'

ACTIVE FOR BEGIN_DIALOG = ON;

GO
```

Server B:

```
Create Certificate UserCertificateB

WITH Subject = 'B.Server.Local',

    START_DATE = '01/01/2006',

    EXPIRY_DATE = '01/01/2008'

ACTIVE FOR BEGIN_DIALOG = ON;

GO
```

3) Take a backup of the user certificate created and install it into the remote instance.

Server A:

```
BACKUP CERTIFICATE UserCertificateA TO FILE=

'C:\Documents and Settings\Santhi\Desktop\Service Broker\Session\UserCertificateA.cer';

GO
```

Copy the certificate from the above location to the following location: Destination: Server B , Path: C:\Documents and Settings\Santhi\Desktop\ServiceBroker\

Server B:

```
BACKUP CERTIFICATE UserCertificateB TO  
FILE='C:\Documents and Settings\Santhi\Desktop\ServiceBroker\UserCertificateB.cer';  
  
GO
```

Copy the certificate from the above location to the following location: Destination: Server A , Path: C:\Documents and Settings\Santhi\Desktop\ServiceBroker\Session\

4) Create a user with the same name as the user who has access rights on the other Database.

Server A:

```
Create User UserB WITHOUT LOGIN  
  
GO
```

Server B:

```
Create User UserA WITHOUT LOGIN  
  
GO
```

5) Create a user certificate from the user certificate backup file copied from the other server, with authorization to the user created in Step 4.

Server A:

```
CREATE CERTIFICATE UserCertificateB  
  
AUTHORIZATION UserB  
  
FROM FILE = 'C:\Documents and Settings\Santhi\Desktop\Service  
Broker\Session\UserCertificateB.cer';  
  
GO
```

Server B:

```
CREATE CERTIFICATE UserCertificateA  
  
AUTHORIZATION UserA  
  
FROM FILE = 'C:\Documents and Settings\Santhi\Desktop\ServiceBroker\UserCertificateA.cer';  
  
GO
```

6) Grant connect permissions to the user.

Server A:

```
GRANT CONNECT TO UserB;
```

Server B:

```
GRANT CONNECT TO UserA;
```

7) Grant send permissions to the user on the local service.

Server A:

```
GRANT SEND ON SERVICE::SenderService To UserB;  
  
GO
```

Server B:

```
GRANT SEND ON SERVICE::ReceiverService To UserA;  
  
GO
```

8) Create a Remote Service Binding with the user created.

Server A:

```
CREATE REMOTE SERVICE BINDING ServiceBindingB  
  
    TO SERVICE 'ReceiverService'  
  
    WITH USER = UserB
```

Server B:

```
CREATE REMOTE SERVICE BINDING ServiceBindingA  
  
    TO SERVICE 'SenderService'  
  
    WITH USER = UserA
```

V. Send Messages from Server A:

In Database A:

```
/******Begin a Dialog and Send a Message*****/
```

```
Declare @ConversationHandle uniqueidentifier

Begin Transaction

Begin Dialog @ConversationHandle

    From Service SenderService

    To Service 'ReceiverService'

    On Contract SampleContract

    WITH Encryption=off;

SEND

    ON CONVERSATION @ConversationHandle

    Message Type SenderMessageType

    ('<test>test</test>')

Commit
```

The above snippet opens a Transaction and begins a Dialog on the specified contract with no encryption. It then sends a message on the conversation using the ConversationHandle created and then commits the Transaction. While beginning a Dialog we also specify the services we are going to use to send and receive the messages.

Now check for this record in TargetQueue of Database B (Server B)

```
select cast(message_body as xml) from TargetQueue
```

VI. Receive Messages from Server A:

In Database B:

```
/*****Receive the Message and send a message to the ender*****/

Declare @ConversationHandle as uniqueidentifier

Declare @MessageBody as nvarchar(max)

Declare @MessageType as sysname

Begin Transaction

Print 'Started Receiving ';
```



```
RECEIVE top (1)

    @MessageType = message_type_name,

    @ConversationHandle = conversation_handle,

    @MessageBody = message_body

FROM TargetQueue;

if @MessageType = 'SenderMessageType'

    Begin

        SEND

            ON CONVERSATION @ConversationHandle

            Message Type ReceiverMessageType

            ('Message is received')

        END Conversation @ConversationHandle

    END

Commit
```

The above snippet opens a Transaction and Receives the first message from the TargetQueue. After receiving the message, We can perform some extra logic but in our example to make it simple we are just sending a message back to the sender that we have received a message.

Now check for records in TargetQueue of DatabaseB (Server B). The record will be removed as it has been processed successfully. Now check the records in InitiatorQueue of DatabaseA (Server A). Two new records will be inserted one related to conversation and the other related to end Dialog as we have used End Conversation.

```
select cast(message_body as xml) from InitiatorQueue
```

Conclusions

This article does not talk about the basic concepts of Service Broker. It deals with building a distributed service broker application.

XML

XML is becoming more and more embedded in all SQL Server technologies, as well as other Microsoft products. Used for everything from configurations to query plans, XML is fast becoming the way in which all applications settings will be stored.

SQL Server 2005 brings many new capabilities for working with XML as well as having much of the data in its tools stored in XML. So, we have gathered together some of our best articles on XML from the past year and republished them for you to work more closely with this technology.

The SQL Server 2005 XML Temptress	306
I've Got the XML - Now What?	312
XML Argument Protocols for SQL 2005 Stored Procedures	317

The SQL Server 2005 XML Temptress

By Simon Munro

I am neither a big fan of XML nor an expert. Over the years I have seen XML used and abused yet another silver bullet to solve all of our problems. I have seen XML fail on many projects because it is too verbose - consuming too much bandwidth and processing power where good ol' fixed-length records would have been fine. I have also seen XML as a good solution to interfacing problems and I (almost) unknowingly use it every day interfacing with web services and configuring systems.

Although XML is interesting, I never found the motivation or need to go out and buy the biggest book on XML that I can find in order to master what should really be a simple markup language. XML gets complicated fast and before you know it you are either confused or an XML bigot (where your confusion is hidden behind a veneer of expertise).

I wouldn't call myself a DBA, but I have done enough late night babysitting of production databases to have an idea how my design and architectural decisions impact the database the least scalable of any part of the architecture. So in my opinion XML - as a bloated, misunderstood, misinterpreted, over-hyped and often badly implemented technology should be nowhere near the database. End of discussion no budging. Holding such a view won't get me fired and will win the welcome support of the production DBAs who could always point fingers at my bad SQL syntax and data structures as a reason for poor performance.

I confess that I have used **XML in the database** in my latest project.

There, I have said it and in the company of expert DBAs who have had enough of people like me putting all sorts of crap into the database. Amongst such company I had better come up with an explanation and fast. Allow me to do so.

There are two situations where XML in SQL 2005 comes in quite handy. They are when implementing classifications and temporal data designs.

SQL 2005 XML for Classifications

One thing that is missing in SQL databases is a construct to handle classifications. It is a problem that designers always encounter and the mechanisms for the physical implementations vary extensively.

What do I mean by a classification? Consider a system that has an order and a customer table, where the key from the customer is put into the order table simple normalization. Now what if your customer can be a person, with a first name, surname and date of birth or it can be an organization, with a name and company registration number? The problem is that it doesn't feel right to create a single table with all of the attributes (breaking some relational model rules in the process) and it is difficult to implement separate [Person] and [Organization] tables.

There are two basic approaches, either a roll-up or a roll-down approach.

In the roll-up approach a single table is created with all of the fields and discriminator or classification attributes to distinguish between the classes. The individual classes can then be implemented as views as follows: (code at www.sqlservercentral.com)

In the roll-down approach multiple tables are created with their correct attributes and the 'leaf' tables are unioned together for an index table as follows:

```
CREATE TABLE Person(  
  
StakeholderId int,  
  
Firstname varchar(50),  
  
Surname varchar(100),
```

```
DateOfBirth datetime
```

```
)
```

```
CREATE TABLE Organization(
```

```
StakeholderId int,
```

```
Name varchar(100),
```

```
RegistrationNo varchar(20),
```

```
)
```

```
CREATE VIEW Stakeholder
```

```
AS
```

```
SELECT StakeholderId, Firstname+' '+Surname AS Name, CAST(1 AS bit) AS isPerson, CAST(0 AS bit)  
AS isOrganization
```

```
FROM Person
```

```
UNION
```

```
SELECT StakeholderId, Name, CAST(0 AS bit) AS isPerson, CAST(1 AS bit) AS isOrganization
```

```
FROM Organization
```

Combinations of the two approaches also exist where some fields are created on an index table and other fields exist only on the leaf tables. Things get a bit complicated when implementing complex classification structures. What if some persons are employees (add an employee number) and what if some employees are contractors (add contract period)? Not only do complex structures become difficult to implement but CRUD across index tables and views becomes a pain.

I have implemented such mechanisms on large databases quite successfully - such as a bank that had 14 million customers. But it can become quite complex and I was looking for a simple classification mechanism that would be reasonably easy to implement, not contain excessive tables or attributes and would be able to be extended or replaced. Enter the XML temptress

I create a simple index table with a column to store all the class-specific attributes as follows:

```
CREATE TABLE Stakeholder(
```

```
StakeholderId int IDENTITY(1,1),
```

```
Name varchar(100),
```

```
ClassAttributes xml,
```

```
isPerson bit,
```

```
isOrganization bit
```

```
)
```

Since my application data access layer uses only sprocs to access the database, some insert and update sprocs for the person and the organization need to be written. For example with person:

```
CREATE PROCEDURE InsertPerson

@Firstname varchar(50),

@Surname varchar(100),

@DateOfBirth datetime

AS

DECLARE @Extended xml

SET

@Extended='<person><firstName>'+@FirstName+'</firstName><surname>'+@Surname+'</surname></person>'

IF (@DateOfBirth IS NOT NULL)

BEGIN

    DECLARE @Dob nvarchar(10)

    SET @Dob=CONVERT(nvarchar(10),@DateOfBirth,20)

    SET @Extended.modify('insert <dateOfBirth>{sql:variable("@Dob")}</dateOfBirth> as last into (/person)[1]')

END

INSERT INTO Stakeholder(Name,ClassAttributes,isPerson)

VALUES(@FirstName+' '+@Surname,@Extended,1)
```

Executing the above sproc like this:

```
EXEC InsertPerson 'Joe', 'Soap', '1 Jan 1980'
```

Results in a record with the basic information and an XML structure that neatly contains all of the other bits of information and would store the following XML:

```
<person>

    <firstName>Joe</firstName>

    <surname>Soap</surname>

    <dateOfBirth>1980-01-01</dateOfBirth>

</person>
```

Notice the use of the XQuery insert that only adds the attribute if it is not null, resulting in neater looking XML data.

A similar sproc for organization would store XML something like this:

```
<organization>

  <name>SQLServerCentral</name>

  <registrationNo>ABC1234</registrationNo>

</organization>
```

My individual [Person] and [Organization] tables are implemented as views like this:

```
CREATE VIEW Person

AS

SELECT StakeholderId, ClassAttributes.value('(/person/firstName)[1]', 'varchar(50)') AS
FirstName,

    ClassAttributes.value('(/person/surname)[1]', 'varchar(100)') AS Surname,

    ClassAttributes.value('(/person/dateOfBirth)[1]', 'datetime') AS DateOfBirth,

    ClassAttributes.value('(/person/title)[1]', 'varchar(10)') AS Title

FROM Stakeholder

WHERE      (isPerson = 1)


CREATE VIEW Organization

AS

SELECT      StakeholderId, Name,

    ClassAttributes.value('(/organization/organizationTypeId)[1]', 'int') AS OrganizationTypeId,

    ClassAttributes.value('(/organization/registrationNo)[1]', 'varchar(20)') AS RegistrationNo

FROM Stakeholder

WHERE (isOrganization = 1)
```

The views are an interesting implementation in that from a relational model point of view they are valid relations and the syntax to use them will be standard SQL. Consider the query where we want to search on the name of a stakeholder, but with people we need to query the surname and on organizations we need to query the name. The following query, even though it has XML innards is a perfectly valid and understandable query.

```
SELECT StakeholderId, Name

FROM Organization

WHERE Name LIKE 'S%'

UNION

SELECT StakeholderId, Surname

FROM Person

WHERE Surname LIKE 'S%'
```

There are other ways to query the XML directly using XQuery but I want to stay as close to ANSI 92 syntax as possible.

Even though we are storing non-relational data in our SQL database we don't really break that many relational rules. Technically the relational model states that the storage of data is independent of the model so, the argument that the use of views is non-relational is invalid (sort of) - if [Person] and [Organization] are implemented as views, which are valid relations, then we are not breaking any rules.

By now, any real DBA would be thinking This guy is frikkin insane, it will perform like a dog! This is both a correct and incorrect thought no, I am not frikkin insane and yes, performance can be an issue. I would not recommend this approach if you have a structure with millions of records, which would be the case with the stakeholder structure in large enterprises. But what about structures with fewer rows, even in the tens or hundreds of thousands? Maybe a product classification or retail outlet classification would perform adequately? You may also notice in this example that Name is redundant, since it is contained in the XML anyway this has been done on purpose for performance reasons since most of the queries only want the name which is a common attribute, so there is no point in mucking about with the XML.

Another key aspect with regard to performance is understanding the interfaces. In my particular implementation, if I wanted to create proper fields for the attributes there would be no far-reaching impact. The interfaces to the sprocs wouldn't change and the fact that I may have replaced the Person view with a table would make no difference to existing SQL.

Denormalization of Temporal Data Using XML

Consider a requirement stating "For ordered items, display the part name and the stock on hand". This may be turned into a query something like this:

```
SELECT o.OrderId, o.PartId, o.Quantity, o.Cost, p.Name, p.StockAvailable
FROM OrderItem o INNER JOIN Part p ON p.PartId=o.PartId
WHERE OrderId=1
```

As with most requirements it is not specific enough and should read "For ordered items, display the part name and the stock on hand **when the order was placed**".

The problem with the above query is that part information, particularly the available stock, changes continuously and the query doesn't take this into account. Many similar problems exist with data that is date dependant (temporal data). Again, there are many ways to resolve this problem you could have [Part] movement records and join based on the order date to the movement tables, or you could denormalize your structures and create an [OrderItem] table with the [Part].[Name] and [Part].[StockAvailable] values in fields on [OrderItem]. The most common approach by far is to do neither and land up with all sorts of issues relating to the temporality of the data which puts the integrity of the entire database in question by the users.

Generally, unless I need to create a specific structure to handle temporality where it may be important I tend to take the easy route and denormalize my structures. The problem is figuring out how many of the fields from [Part] should be stored on [OrderItem] to handle all the various combinations of queries that the users may think up in future. Also, it looks ugly when [Part] fields are reproduced on the [OrderItem] table apart from breaking a few relational model rules along the way.

In a recent system there was a need to store some 'snapshot' temporal data, but since other parts of the system were not specified, never mind developed, we were unsure which fields to store the solution was to store most of them in an XML field and worry about it later.

So in the above example I would create a table something like this:

```
CREATE TABLE OrderItem(
OrderId int,
```

```
PartId int,  
Quantity int,  
Cost money,  
PartStore xml)
```

With sprocs to handle the inserts and updates,

```
CREATE PROCEDURE InsertOrderItem  
  
@OrderId int,  
  
@PartId int,  
  
@Quantity int,  
  
@Cost money  
  
AS  
  
DECLARE @PartStore xml  
  
SELECT @PartStore='<part><name>'+Name+'</name><stockAvailable>'+CAST(StockAvailable AS  
varchar(10))+ '</stockAvailable></part>'  
  
FROM Part  
  
WHERE PartId=@PartId  
  
  
  
INSERT INTO OrderItem(OrderId,PartId,Quantity,Cost,PartStore)  
  
VALUES(@OrderId,@PartId,@Quantity,@Cost,@PartStore)
```

This would create a store of temporal data something like this

```
<part>  
  <name>Part 1</name>  
  <stockAvailable>10</stockAvailable>  
</part>
```

When querying data I simply look in the XML for the part data that I need,

```
SELECT OrderId, PartId, Quantity, Cost,  
  
PartStore.value('(/part/name)[1]', 'varchar(50)') AS Name,  
  
PartStore.value('(/part/stockAvailable)[1]', 'int') AS StockAvailable  
  
FROM OrderItem  
  
WHERE OrderId=1
```


And as per the classification examples above, I can wrap the queries into nicely named views if I prefer.

The plan is that over time, provided my interfaces remain the same, I can add Part attributes directly to the OrderItem table if needed. This can be done on a production database and I would just need to alter the table,

```
ALTER TABLE OrderItem ADD PartName varchar(50)
```

```
UPDATE OrderItem
```

```
SET PartName=PartStore.value('(/part/name)[1]', 'varchar(50)')
```

and change any sprocs or views that reference the table all very backwards compatible.

Summary

A year ago I would have recoiled at the mere suggestion of using XML directly in the database, thinking that it was the domain of junior asp developers who can't tell a [relation from a relationship](#). Over the last few months the XML Temptress in SQL 2005 has lured me into her parlour providing a mechanism to approach old problems in new ways.

I would still be in line violently opposing persisting objects as XML in the database as object oriented bigots would be tempted to do, after all, they say that the database is just a persistence mechanism for their objects. I can picture object orientation bigots advocating a database of one table with two attributes, ObjectId (GUID) and ObjectData (XML) such an image is concerning.

However, I hope that I have demonstrated that if carefully thought through that XML in the database can be useful and elegant provided that it is done within the context of the overall architecture. The biggest issue with XML in the database is understanding when performance becomes the overriding factor as to where and how data is stored after all it is mostly the performance of databases that your DBA has to deal with anyway.

[Simon Munro](#)

Simon Munro is currently pursuing the Microsoft Certified Architect (MCA) certification and maintains a blog at <http://www.delphi.co.za/> that covers many interesting ideas on using Microsoft technologies.

I've Got the XML - Now What?

By David McKinney

OK, I've got the XML - Now what can I do with it?!!

In this article, I'm going to show you how can start by extracting XML from your database and finish up with an html file which documents your database tables. Along the way we'll also be touching upon System Views, XSL, XPATH and SSIS.

The goal is to give you a taste of what you can do with your XML, and hopefully inspire you to put XML to use, and pick up where I've left off. The stages are listed below

1. Write the SQL to get the XML.
2. Write the XSL to transform the XML to HTML.
3. Write the SSIS package to make the transformation happen.

To keep things as simple as possible, the example we'll be working through will just create a list of table and field names, but I'll also include source code to extract information about Primary keys, indexes, foreign keys etc..

1. Writing the SQL to get the XML

Figure 1 shows the SQL to extract the table and field information. Figure 2 shows example XML that this generates. The query that extracts the field names is a subquery of the query which extracts the table names. This means the fields appear as child nodes of the parent table.

```
select xTable.name as table_name
, (
select col.name from sys.columns col
where col.object_id=xTable.object_id
ORDER BY col.column_id
for XML auto, type
)
from sys.tables xTable for
XML auto, type
```

Figure 1

```
<ROOT>

<xTable table_name="Customer">

<col name="CustomerID" />

<col name="SalesPersonID" />

<col name="TerritoryID" />

<col name="AccountNumber" />

<col name="CustomerType" />

<col name="ModifiedDate" />

<col name="rowguid" />

</xTable>

<xTable table_name="CustomerAddress">

<col name="CustomerID" />

<col name="AddressID" />

<col name="AddressTypeID" />

<col name="rowguid" />
```

```
</xTable>  
  
</ROOT>
```

Figure 2

2. XSL which transforms the XML into HTML.

The XSL is shown in figure 3. Basically we start with the ROOT node, and navigate down through the document. Inside the root node we put the HTML that we only want to display once in the document -- as we know we will only have one ROOT node in the XML document. The `<XSL:apply-templates select="xTable">` will write the text inside the `<XSL:template match="xTable">` node, for each xTable node in the XML document. The xTable template in turn calls the col template for each child col node.

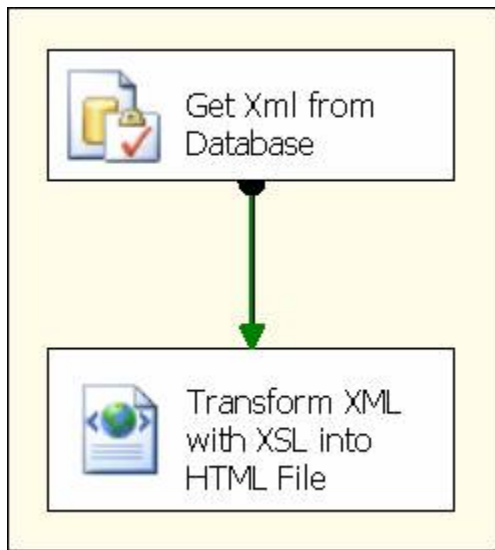
You could parallel this to pseudo code like

```
write header  
  
Foreach xTable in ROOT  
  write something  
    Foreach col in xTable  
      write something  
    Next  
  Next  
write footer
```

I'd recommend you get an XML editor such as XML SPY to help you author XSL sheets. (There is a free home edition, with sufficient functionality.) This will give you access to intellisense, will syntax check your documents, and will execute transformations. (code at www.sqlservercentral.com)

3. Write the SSIS package to make it happen.

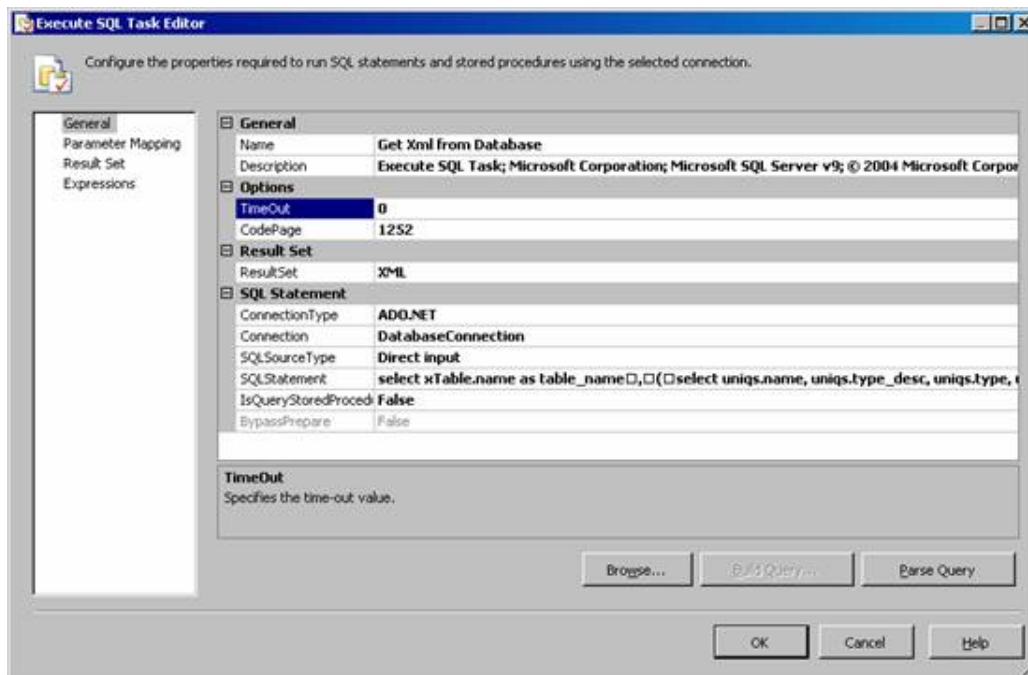
I mentioned earlier that a XML authoring tool can be used to execute a transformation. Sometimes, though, you need a more automated solution. For example, if you want to carry out multiple transformations on the same document or on multiple documents. (e.g. if you wanted to create a separate HTML file for each table.) There are several ways to achieve this. If you don't want to write code, then perhaps SSIS is the easiest method available.



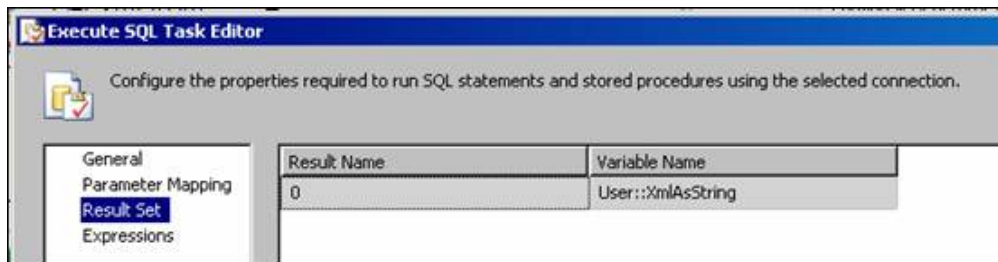
Picture 1 - The SSIS package

1. Execute SQL Task

You need an Execute SQL task with a connection to your database. Make sure the ResultSet is set to XML, SQLSourceType to DirectInput, and paste the SQL statement we constructed earlier into the SQLStatement.



We need to catch the output in a global variable, so create a string variable called XMLAsString, and set the Result Set Variable Name to this. Note that Result Name must be set to zero. (Don't ask me why. It just does!)



2. XML Task

You need an XML Task, with OperationType XSLT (to denote that you're doing a transformation). You need two file connections, one for the XSL file (which you save to the file system) and another for the output HTML file.

Running this package generates an HTML file like the one below. I agree, it's hardly spectacular. I've included a more sophisticated SQL query and XSL file which will give you the following style of output.

CountryCurrency
CountryRegionCode
CurrencyCode
ModifiedDate
rowguid
CurrencyRate
CurrencyRateID
CurrencyRateDate
FromCurrencyCode
ToCurrencyCode
AverageRate
EndOfDayRate
ModifiedDate
rowguid

CurrencyRate	
CurrencyRateID	(PK, int, NULL)
CurrencyRateDate	(datetime, NULL)
FromCurrencyCode	(nchar(6), NULL)
ToCurrencyCode	(nchar(6), NULL)
AverageRate	(float, NULL)
EndOfDayRate	(float, NULL)
ModifiedDate	(datetime, NULL)
rowguid	(uniqueidentifier, NULL)
Customer	
CustomerID	(PK, int, NULL)
SalesPersonID	(int, NULL)
TerritoryID	(tinyint, NULL)
AccountNumber	(int, NULL)
CustomerType	(nchar(2), NULL)
ModifiedDate	(datetime, NULL)
rowguid	(uniqueidentifier, NULL)
CustomerAddress	
CustomerID	(PK, int, NULL)
AddressID	(PK, int, NULL)
AddressTypeID	(tinyint, NULL)
rowguid	(uniqueidentifier, NULL)

This is still nothing spectacular, but if you can follow the code and are good at HTML, you can quite easily expand on it to generate much more comprehensive output. I should mention that you don't have to generate HTML. It could be a csv file, or a generate table script. There is an infinite number of possible applications. If you're interested, you can download some [XSL code](#) for more sophisticated output. So go on - what are you waiting for? Give it a try!

XML Argument Protocols for SQL 2005 Stored Procedures

By Jack Hummer

This article describes using some of the new XML functionality provided by MS SQL Server 2005 as applied to argument passing to T-SQL stored procedures. One of the more tedious areas of application support is managing arguments passed to and from T-SQL stored procedures. Unlike other RDBMS engines such as Oracle, T-SQL does not support the notion of passing in record structures as arguments from a client directly to a T-SQL stored procedure. Realizing that the new XML functionality might offer some relief in this area, I did some searching to see what anyone else had come up with. I found some articles related to XML, but the articles I did find used SQL 2000 and had the primary focus of using XML for bulk Inserts with no preliminary data validations or other processing steps (see References below). As I had some different objectives in mind, and I found no articles highlighting the new SQL 2005 XML features, I decided to present my ideas and results here.

Background

The use of MS SQL Server as a web back-end storage system is well understood. Maintaining the stored procedures that interact with web clients can be tedious, with the lack of record structure support to and from stored procedures a contributing factor. There are several consequences to this, including:

- Commonly used procedures have multiple copies created with slightly different arguments
- There are many instances where passing numerous arguments and/or varying number of arguments to stored procedures can be awkward or nearly impossible
- Altering the arguments to a commonly used procedure can be a high-maintenance task any dependent code snippets must be changed in parallel

- T-SQL does not include a formal mechanism to pass in record structures from a client application, much less support passing in a varying number of records

Transaction support is another labor intensive effort: In a web back end, the database context, or session, only exists as long as that one procedure call is active. Thus, a transaction that requires multiple procedure calls requires some way to wrap the transaction/rollback around the multiple invocations. For example, storing an e-commerce transaction can involve a dozen tables and multiple, varying number of rows per table (e.g. sales line items) which is really impossible to pack into standard procedure call arguments. Some people use this difficulty as a reason to embed SQL in their application code instead of placing it into a stored procedures.

Another interesting case arises when you want to send a stored procedure the minimal number of columns in a row to update. First, we have the issue of variable number of arguments (which columns are getting updated?); next, we are faced with future maintenance issues as columns are added to (or deleted from) the table; then the update code has to decipher a tri-state assertion for each column in the table: Do we leave the column alone, update it to a value, or update it to Null? Traditional argument calling sequences might involve two arguments per column, one to pass the new value, and an indicator variable to say if we are really updating or not - not pretty. The open structure of XML lets us define such things easily.

Finally, there is the reverse interface issue of sending back to the calling program multiple values, possibly structured; for this discussion I will use error messages. Suppose we have a row insert procedure, and we do a number of validations on the column values before executing the Insert command. What we do not want to do is bail out after detecting the first error, go back to the user with one error, they fix it and resubmit, we bail out on the next error, etc. until the user gets dizzy. We would like to do all our data validations and pass back all the conditions and messages in one pass, so the user can fix them in one try. Here again we need to pass multiple results (independent of a dataset), which the basic T-SQL Procedure argument list makes difficult if not impossible.

Thus the issues at hand are:

- Number of arguments (complexity), and maintenance issues (add/delete arguments)
- Tendency to create multiple copies of the same procedure for slightly varying input conditions
- Passing a record structure
- Passing a varying number of multiple records
- Adding processing criteria to arguments (e.g. no update; update with value; update to Null)
- Pass back a varying number of (potentially structured) values, e.g. error messages.

XML As A Solution

T-SQL has native support for manipulating XML in SQL 2000, but with the release of SQL 2005, XML support is even better. T-SQL arrives with additional functions and an actual XML native datatype [essentially an nvarchar(max) with storage in "optimised UTF-16 characters" per BOL]. There are some curious comments in the BOL about the overhead of XML processing, but I have not yet tried to measure it. My general rule is, if a certain syntax or process makes for increases in programming efficiency and reliability as well as reducing long term maintenance, I care less about a little additional overhead on the machine.

Client Code

On the application (e.g. web page) side, instantiating the MSXML object to create an XML document certainly has overhead, especially as it supports a lot of esoteric XML syntax that may be seldom used. With the new SQL Server 2005 enhancements, some of the client-side processing can be entirely relieved: XML arguments can be treated as a string and passed to a stored procedure without any need for complicated objects to help us. You should certainly use the MSXML object if it fills other requirements the client-side application needs. Note: If you do not use the MSXML object, then you need to handle the translation of reserved characters into their alternate representations, a.k.a. 'entities'. For example the XML reserved character "<" [less than sign/open angle bracket] in user data must be replaced with the XML Entity "<". The same holds for ">", "'", '"', and "&". You could write a small function to handle those [hint: process "&" first].

Using XML makes the calling convention to stored procedures very simple: As the XML is by its nature structured, when we extract values out of the XML from inside the procedure, the code need make no assumptions about ordering of individual data items; handling data element existence or non-existence is also handled gracefully. The presence of possibly obsolete data items is no longer a bother to the stored procedure - it simply ignores them. All of this helps reduce maintenance.

Web Server Application Code

We utilize the fact that an XML document can be represented by a character string, and simply append and build it. Note, if you really wanted, you could instantiate the MSXML object and build your XML with that, but this seems far too much for the job at hand. My presumption is that all of the user input has been collected and validated to some extent by the application program, and is then to be passed to the database as a single transaction.

Example: Insert Customer Row [vbscript]

```
xcmd = "<arg><first_name>" & firstName & "</first_name><last_name>" & lastName &_
"</last_name></arg>"
```

where arg is the root tag for the whole document, and firstName, lastName etc. are the variables holding the user input. If a data item is missing, the stored procedure can fill in a default or raise an error, depending on the business rules involved.

Example: Update Customer Row [vbscript]

```
xcmd = "<arg><customerid>74285</customerid><first_name>Robert</first_name>" &_
"<company_name null='yes' /></arg>"
```

This could be sent to an Update procedure. It provides a primary key, a new first name, and an explicit instruction to set the company name to Null; all other columns are unmodified.

Example: Create Sales Transaction [vbscript]

```
xcmd = "<arg><customer><first_name>Mary</first_name>. . . </customer>" &_
"<order_header><order_date>08/15/2006</order_date>. . . </order_header>" &_
"<order_lines count='6'><line productid='7294' . . .>" &_
"    <line productid='8241' . . .>. . . </order_lines>" &_
"<payment><type>Visa</type>. . . </payment></arg>"
```

Here we consolidate multiple record types, and multiple records for sales line items, all into one procedure call. This can then easily be wrapped in a single Transaction in case of unexpected database errors. The line count is optional, but if you know it then why not send it along, the procedure can make use of it. Otherwise the procedure will have to loop on searches for order_lines until it runs out of them. Remember that in most cases the non-significant white space (blanks, new lines, etc. in between elements) are ignored and often disappear when the XML document is stored in a canonical form, such as an XML variable.

I have written web page scripts that do just this kind of processing. First you call a procedure to add the customer. Then call another procedure to add the order header. Then a big loop call to insert each sale line item. Yet another

procedure to post the payment. And if one of these burps along the way? Go write another procedure to try to clean up a half-baked transaction! That kind of processing is where this interface really shines.

Example: Error Message Return Results

```
<err>

  <error>

    <err_code>APP001</err_code><err_msg>First Name is a required field.</err_msg>

  </error>

  <error>

    <err_code>APP002</err_code><err_msg>Last Name is a required field.</err_msg>

  </error>

</err>
```

This might be returned by a procedure call. Multiple messages can easily be handled, as well as a record structure (shown here) composed of both an error code as well as the error text message. Successful execution is indicated by simply returning Null.

Stored Procedure Interface and Code

The calling program (a server-side web page script, for our examples) makes a standard procedure call, passing the string that has been built as an input XML argument. plus an output XML argument for the error status return. The variable types could typically be double-wide character, e.g. Unicode.

Here is an excerpt of a stored procedure to validate values then Insert a customer row.

Editor's Note: This code is available from www.sqlservercentral.com.

For convenience as well as modularity, I created a small T-SQL function to insert one error record into the XML error document. This will get enhanced in the future, for example returning the error message in the users language, or optionally writing it to a log file.

Editor's Note: This code is available at www.sqlservercentral.com.

A few notes on this code and related programming issues:

- The XML data type acts very much like an object, a new twist for T-SQL. It is also very flexible, as it is valid as a variable, argument, or function return. However, from ASP, ADO did not like it as an output variable type.
- Many of the new methods/arguments require lower case only, an abrupt change from most vendors SQL syntax which has usually been case insensitive. Not my preference but, hey, they forgot to ask me.
- The .value method utilizes XQuery syntax. If you have not seen this before, it is a complex language used to find things inside an XML document. Don't try to learn it from BOL, look for an introductory tutorial on the web (but, most of the introductory material I found progressed very quickly into advanced language features without completely explaining the fundamentals).
- You will need to experiment a bit to see what happens when you search for element values or attribute values and they do or do not exist.
- The .modify method uses both standard XQuery plus new extensions that Microsoft had to invent. The standards people have not yet addressed XML update syntax. Re-read the BOL articles a few times until it starts to make

sense. In particular, the first argument must literally be a string literal, which seems rather limiting at first, unless you manage to stumble across the "sql:variable()" work-around.

- Contrary to popular belief, apostrophes (a.k.a. single quote) may optionally be used around XML attribute values.
- The .value method returns the reserved entities back into their original characters.
- Within T-SQL, CAST or CONVERT an XML variable to a character string type leaves the special entities intact.
- The .value method will right-trim the return string according to the data type specification, with no error.

Special Note: when putting together these ideas into some real production code, I found the procedure return argument as an XML data type did not work coming back through ADO to vbScript, so I changed it to an NVARCHAR. This worked just fine when loading it into an MSXML object. The .Net infrastructure may provide enhanced support for passing XML.

Stored Procedure - XML Code Loop

So you may ask, How do I exactly process an unknown number of records in the input XML structure? Given the restrictive syntax on the built-in XML methods, the only solution I have come up with is sp_executesql. You will customize this for your situation, but here is a small example:

```
SET @n = 1

SET @params = N'@x XML, @value INT OUTPUT'

WHILE 1 = 1

BEGIN

    SET @cmd = N'SET @value = @x.value(''(/arg/cat)[ ' + CAST(@n AS NVARCHAR(2)) + ''],'',
    'INT')'

    EXECUTE sp_executesql @cmd, @params, @x = @xarg, @value = @id_cat OUTPUT

    IF @id_cat IS NULL

        BREAK

    -- do some processing

    SET @n = @n + 1

END -- while
```

Example: ASP Calls the Stored Procedure [vbscript]

Editor's Note: Code at www.sqlservercentral.com

The above is a very simple example, but demonstrates everything we have discussed. The error display to the web page could be wrapped in a small function, and as an alternative could be handled nicely with some XSLT code as well.

What About Schemas?

Yes, for added data validation you could add schema definitions. But for something as transient as argument passing, it seems like too much extra overhead. But for some very critical processing it may be appropriate.

Speaking of Overhead

The extra overhead of converting native data types into XML and back, as well as the overhead of instantiating the MSXML object seems to be the only noticable downside to this approach. For me, the added capability of passing variable amounts of data plus any reduction in long term maintenance makes it a winner. But probably no worse than the creation of a recordset to return structured data. Certainly many if not most procedure calls will not need complex variable input arguments, and the XML output processing is only invoked if there is an error, presumably only a small per centage of the time.

In Conclusion

MS SQL Server T-SQL does not provide a robust interface for passing in varying numbers of arguments or any kind of record structure. On output we may also desire a logical multiple record structure, separate from a data recordset. And the more arguments a procedure takes, the bigger the maintenance chore.

An XML input argument provides a simple way to send from application code a variable number of arguments, including arbitrarily complex logical record structures. An XML output argument likewise is a good container for multiple return values, again including optional record structures. Since the stored procedure just ignores any input elements it does not need, and can often provide defaults for elements not passed, you have a greater chance of being able to make changes to the stored procedure and/or underlying database structure and have relatively little or no impact on existing programs that use it.

References:

These all reference SQL 2000, and mostly use the XML input to just do a direct bulk Insert without preliminary data validations.

<http://www.devx.com/dotnet/Article/16155/0/page/2>
http://www.sql-server-performance.com/jb_openxml.asp
<http://msdn.microsoft.com/msdnmag/issues/05/06/DataPoints/>
<http://www.devx.com/dotnet/Article/16032/1954?pf=true>

A Microsoft article on XML functionality overview in SQL Server, emphasis on SQL 2005.

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsq190/html/sql2k5xmloptions.asp>

Introduction to XQuery in SQL Server 2005

http://msdn.microsoft.com/SQL/default.aspx?pull=/library/en-us/dnsq190/html/sql2k5_xqueryintro.asp

Misc.

Every year we publish a number of articles on topics that aren't directly related to SQL Server technologies. Career advice, certification, tools, and various other topics are game here.

This year we've brought you the best of the "other" articles that were very popular with our readers.

Windows Utilities for the SQL Server DBA	324
From DBA to DBAA	326
What's a Good Manager.....	328
Mind Your Manners	332
And now this..the only section devoted to (n)etiquette for those who respond!!!	336

Windows Utilities for the SQL Server DBA

By Chad Miller

This article is a short summary of Windows utilities both GUI and command line intended for new SQL Server DBAs or those interested in learning some useful techniques. All examples use native Windows utilities/command or are available through Resource Kits, Administration Tools or Support Tools. Let's look at some examples.

FOR and FORFILES Commands

This simple command is used to delete old backups with .trn, sqb and bak extensions older than two days for files in the current directory and all subdirectories.

```
for %I in (TRN sqb bak) do FORFILES /S /D -2 /M *.*%I /C "cmd /c del @file"
```

The **for %I in (TRN sqb bak)** command executes **Forfiles** three times once for trn, sqb and bak. The **forfiles** command includes several switches:

- **/S** -- include all subdirectories
- **/D -2** -- select files with date older than two days
- **/M** -- match .trn, sqb or bak extension

I've seen complicated T-SQL xp_cmdshells, Perl, VBScripts and bat files to do what turns out to be included in the OS! The **FORFILES** command is included in Windows 2003 and the Windows Resource Kit. It is not included in Windows XP, however you can simply copy the *forfiles.exe* file from any Windows 2003 Server and it will run on Windows XP also. The **for** command is included in Windows 2000, 2003 and XP.

Need to execute a SQL script against a bunch of SQL Servers, try this command:

```
for /f %i in (servers.txt) do SQLCMD -S%i -i C:\myscript.sql
```

Resource Kit Utilities

There are four Windows tools I'll install on every workstation I use:

- Windows Server 2003 Administration Tools Pack available as a download from Microsoft
- Windows Server 2003 Resource Kit Tools also available as a download from Microsoft
- Windows Server 2003 Support Tools which available from the Windows 2003 CD
- Windows Server 2000 Resource Kit Tools which unfortunately isn't available as a free download, however it can be downloaded from MSDN, if you have a current subscription

There are only four primary tools I'll regularly use as a DBA from the Windows 2003 Administration Tools Tools pack:

- Cluster Administrator for well, administering a cluster
- Terminal Services Manager for viewing who is using the Administrative Terminal Services connections and disconnecting someone who has forgotten to log off one of the connections
- Active Directory Users and Computers for viewing accounts, lockout status, group membership. Although I prefer to do most of this via the command line through other tools
- Computer Manager for connecting to remote systems to view the Event log, manage services and change local accounts

The Windows Server 2003 Resource Kit Tools and Windows Server Support Tools include many utilities, however there isn't one that I use on regular basis. The Windows Server 2000 Resource Kit Tools, however has three utilities which I find invaluable as a DBA:

- SHOWMBRS -- displays NT/AD group membership from the command line. Used primarily to assist in troubleshooting or assigning AD groups to SQL Server
- SHOWGRPS -- displays an NT/AD login groups from the command line. Used like SHOWMBRS only from the login perspective
- SRVINFO -- displays a wealth of information on the OS, hotfixes, drives, services, and uptime of remote systems from the command prompt

WMIC

WMIC is probably one of the most powerful, yet least used series of commands by administrators. It is included with Windows XP and Windows 2003 and can be used for shutting down services, monitoring process, or just getting information.

Need to shutdown the SQL Server services on all SQL Servers remotely? Try this single one-line command:

```
wmic /node:@C:\servers.txt service WHERE "Name Like 'MSSQL%'  
And Name<> 'MSSQLServerADHelper'" CALL StopService
```

Need to know all of the SQL services running on a server?

```
wmic /node:MyServer service WHERE "(Name Like 'MSSQL%' OR Name Like 'SQLAgent%'  
OR Name Like 'SQLServer%' OR Name = 'MSDTC') And Name<> 'MSSQLServerADHelper'" get  
DisplayName, Name, State
```

Here's one of my favorites, a disk command that displays output similar the UNIX **df -kH** command, it's a little more complex than the previous examples in that it uses a custom XSL transform. One of the interesting thing about WMIC is that it can output XML which in turn can be transformed using XSL, so I created an XSL file called [logicaldisk.xsl](#). Copy the xsl file to the C:\Windows\System32\Wbem directory to use this command.

```
wmic /node:MyServer logicaldisk get Caption, VolumeName, FreeSpace, Size, SystemName  
/format:logicaldisk.xsl
```

And the output of the command will look something like this:

Caption	FreeSpace	Size	SystemName	VolumeName	Free%
A:			MYSERVER		
C:	0.8G	8.4G	MYSERVER		10%
E:			MYSERVER		
F:	6.6G	11.2G	MYSERVER	EMC_12	59%
I:	57.8G	67.4G	MYSERVER	EMC_67	86%
J:	62.0G	67.4G	MYSERVER	EMC_67	92%

Windows Command-Line Scripts

Yes, Windows command-line script aka bat files are still relevant in today's world of VBScript and WMI, a few lines in a bat file can be quickest way to write a useful tool. And WMIC is a great way to do a lot of things quickly, however remembering all of the commands and having to type a long command string doesn't seem too appealing. Here's where command line scripts really shine, abstracting away a complex command to simple one word command accepting parameters. For example the command listing disk information can be wrapped in a command line script as follows: (code at www.sqlservercentral.com)

We're still basically executing the same one-line command as before, we've just added some icing to our original solution, so that we can enter a single command **disk** to display disk information. By the default the local machine disk information is displayed. To display disk information for a remote server enter **disk MyServer1** or to display disk information from multiple machines by entering each server separated by a space, **disk MyServer1 MyServer2**

Conclusion

There are many more Windows utilities and commands than can be covered here, this article has demonstrated only a few commands and Windows utilities which are relevant to a SQL Server DBA. The [CLIforDBA.zip](#) file contains several examples of Windows script files.

From DBA to DBAA

By Jeffrey Yang

Introduction

With more business relying on data, we DBAs seem to have a more promising future. But a question naturally comes out, "Where is the next career stop for a DBA in terms of challenges and self-value realization?" In this article, I'd explore one possible career transition path for DBAs, i.e. from a database administrator (DBA) to a database administration architect (DBAA), and I will try to address the following questions: what is a DBAA, the major work and responsibility of a DBAA, the values of a DBAA. and how to be a DBAA.

What is a DBAA?

A DBAA is a professional who is responsible for designing a solution framework that maximizes the efficiency of the resources dedicated to the data system administration to meet the business challenges, such as cost, performance, security and regulatory compliance requirements etc.

The main responsibility of a DBAA is to achieve the highest possible ROI with the available resource in the context of the various business requirements. The details of this responsibility may include

- Define the administration scope in terms of targets and risks / costs
- Build up an optimized processes model which can maximize the ROI for the current resources
- Pioneer in evaluating / choosing the right mix of technology
- Explore / create innovative methodology to adapt to business environment.
- Act as a facilitator / advisor for the stakeholders to best use the data system / asset.

The values of a DBAA

A DBAA's values lie in three fronts:

First to the business: a DBAA focuses more on business instead of servers, which means a DBAA is to take care of

the business needs from database administration perspective. This can range from designing processes to meet special business needs (e.g. auditing purpose), ensuring database system performance / security quality, to facilitating other system architects for better business projects etc.

Second to the team: a mentor for valuable advice; a resource for in-depth technical discussion (have you ever had the feeling that it is tough to find someone knowledgeable enough to discuss your "exciting" technical ideas?); a hero who may help the team out of the hot water from time to time.

Third to the management: an assistant to the management success, a manager succeeds only when his/her team members succeed. With a DBAA providing robust and innovative solutions to manage the business core asset, i.e. data, it will be easier for the manager to demonstrate the value of his/her team to the company.

Professional traits of a DBAA

A DBAA should basically have the following three fundamental traits:

1. An imagination dreamer: a DBAA's capability is only limited by his imagination. Most of the time, it is not difficult to solve a known problem, but it is hard to foresee a potential problem, which, if solved, may bring huge value to the business.
2. An innovation explorer: with all "dreams" at hand, a DBAA needs to explore all the possibilities to realize the dreams. For example, to do a full backup, you can use T-SQL, but you can also use DMO / SMO with VBScript or PowerShell, and you may even use SQL Writer with VB.Net. Which method to use? The famous answer is "It depends".
3. An implementation practitioner: Once a solution is chosen, the capability to implement the solution is critical. A good example here is you may have exact ideas of how to decorate your house, but to implement the decoration is totally different from the pure decoration ideas. To do decoration yourself, you may need to know how to do hardwood flooring, how to do drywalls, how to do painting, where to purchase the best affordable materials etc, etc.

The path to be a DBAA

There is no existing way to be a DBAA, but a road is always created by pioneers' footprints.
A DBAA should first establish his/her working philosophy and then proceed his/her work with this philosophy.
To me, a DBAA's work domain can be summarized as dealing with a simple formula.

Expected Goals = Methodology (Resources, Constraints)

Assuming Methodology is the function, Resources and Constraints are the inputs for the function, and Expected Goals is the result of the function. ("Resources" and "constraints" can be considered as the same depending which side you are on.)

Expected Goals = the sum of known business requirements + Visions of future

Resources (Constraints) = Current human capital + Time + Budget + Current system environment + Corporate policy
Both "Expected Goals" and "Resources" are constrained by boundary factors, such as availability, deadline and policy.

So in essence, a DBAA needs to work out a customized formula to maximize the returns (i.e. "Expected Goals") out of the input.

However, it is important to realize that in real life, any solution is a compromise among the various stakeholders with different interests.

Some potential research fields for a DBAA

I think the following areas may be very worthwhile for a DBAA to explore because the researches are significant to build an applicable methodology, which is the primary work for a DBAA.

Database administration quality model: How to evaluate an existing administration practices? How is the current practice benchmarked against various best practices?

Database administration patterns: Can you find out the proven and repeatable way to solve some common issue? For example, trouble-shooting performance, database system deployment or auditing configuration change?

Database administration operation management: research on the highest value-adding activities / resources distribution in database administration work

Database administration maturity model: a model against which we use to evaluate the general quality of the database administration?

Difference between DBA and DBAA

The difference between a DBA and a DBAA lies in the pattern of thought each role may have and the way each will adopt to approach an issue.

A DBA tackles on day-to-day problems while a DBAA strives for long-lasting methodologies.

A DBA is a solution provider while a DBAA is a process inventor.

A DBA is more of a practitioner with theory while a DBAA is more of a theorist with practices.

In essence, a DBA creates values by providing solutions to tactical issue while a DBAA creates values by architecting a healthy and high quality database administration eco-system,

Summary

A DBAA is the governor of the data administration kingdom. To better manage the kingdom, s/he needs to define "laws" for his/her jurisdiction. With these "laws", it is expected to build a healthy and efficient database management environment together with a culture that will last beyond the scope of the DBAA's regular responsibility and may be merged with the corporate culture. After these "laws" have been tested and proved to be fair and efficient in the database administration world, as a "law-maker",

the person can be considered as successfully transitioning from a DBA to a DBAA role

Post-Note

In the last PASS conference (Nov, 2006 in Seattle, WA), I was fortunate to be chosen to attend a SIG meeting organized by Microsoft to discuss about future high-level SQL Server professional certification, unfortunately I was unable to attend due to my company's urgent business. I promised the organizer from MS, to submit an article on SSC detailing my thoughts about the next level of certification for SQL Server professionals.

So in this article, I discussed what we as DBAs can aim for in our next career step, details about DBAA (what it is, its value and how to achieve it) are discussed.

In short words, a DBAA is a leader that can inspire a team and cultivate a administration culture suitable for the corresponding business environment; a DBAA is a CFO who ensures the best ROI out of his/her resources in database administration, a DBAA is a theorist who pursues the practical methodology that can be applied to real-life world, and finally a DBAA is a practitioner who is capable to see his/her agenda to decision and to final implementation.

What's a Good Manager

By Janet Wong

For the Long Term...

Recently I read Steve's editorial topic, ["For the long term"](#). It really got me thinking. In the editorial, Steve talked about Andy's presentation on "Successful Team Management" and how important it is for a manager to manage his people first, and attend to the technical stuff later. As Steve said,

"[T]he technical stuff is for today, for this project, for now. Managing people is helping them in their careers, which will hopefully last decades. Managing people is a challenge, one that I'm not sure I'd like to do again, but if you find yourself in that job, be sure you accept the responsibility for your people. Think about them, take care of them, and be sure you spend the time on them they deserve."

Wow! How true and important that statement is! Unfortunately, my experience is that, at least as far as developers are concerned, fewer and fewer managers are paying any attention to the needs and careers of the people they manage.

I started working as a developer many, many years ago and in that time I have worked for many managers. Some were very good, some were not so good, and quite a few were awful. I don't think being a good manager has much to do with your technical skills or background. If you have technical skills and you are a good manager you are a blessing to all. If you have no technical skills and are a good manager you are still a blessing to all. Bad managers are a curse regardless of their technical skills or background.

I had some good managers in my earlier years and they were a blessing. I also had a few bad managers in my career and they gave me nightmares. But what really bothers me is that I think I see that the number of bad managers has been increasing in recent years. Is this my imagination or has there been a change for the worse? Certainly, if I use Steve's definition of a good manager as one who helps their people in their careers, there has been a change. Here is a summary of my experience with managers over the years.

The Good

I worked in a very small company for my first job. I was the 21st employee of that company. My department consisted of my manager, a computer operator, and me. I wrote COBOL programs and used an HP 3000 workstation and an IBM mainframe. My manager taught me a lot of programming. But he was also very tough and did not allow people to make mistakes. I got yelled at every time I made a mistake, no matter if it was big or small.

That was fine so long as I had made a mistake; it made me check my work carefully. But one time one of the production jobs went down and he blamed me. He said that I had done something to the job. Two days later the senior programmer found out it was not my fault and told my manager. My manager apologized to me, but I was too upset and decided to leave the company. I was young and could not get this incident out of my mind. The day I left he gave me a "Precious Moment" doll. It is still on my book shelf. He was a good man and I think he was a good manager even though his management style was tough on a young programmer. He certainly taught me to check my work carefully.

My next job was in the IT department of a big company. My group was responsible for the finance department. I failed accounting in college so this was a challenge. My manager was an accountant, not a developer. He had no programming skills and he wanted it to stay that way. Whenever I had any technical questions, I had to ask the senior programmers. However, my manager tested all my programs and was the best tester I ever had.

My job was writing CICS online programs. When he tested my first program he pressed the escape key and my program blew up. He told me when I wrote any online program that I had to think like a user and not a programmer. A user might do anything. A programmer cannot imagine all the things a user might do with their program but you have to try. I never forgot some of the things he did in his tests. I learned how to test my programs from him. Even though he had no technical skills, he was a good manager; he cared about the company and made sure his people delivered quality products.

My next job was with a software development company that developed software for colleges, such as, college registration and student admission programs. There were a lot of talented programmers in that company. I loved to

work with them. I was amazed at how they solved difficult problems. Those people were the ones I called "techies". They touched and tried every new technology. But most of important of all, they did not mind helping others, including me, whenever we had a problem or a question. We could talk, laugh, and learn from each other. Although we did not become very close friends, I trusted them. Some had the opportunities to become managers and some did; but most remained as programmers because they loved the excitement of working with leading-edge technology.

I remember one of my projects. It involved writing COBOL programs and sending information to a Kiosk machine. The company had an Oracle database system. Another co-worker was assigned to write the Oracle queries needed to send the data to the Kiosk machine. But he had a big production problem and could not work on the project. So the company tried to hire an Oracle contractor. There were lots of COBOL programmers available for hiring, but they could not find any Oracle programmers. On the day before my manager took off on vacation, she came over to tell me they would hire a COBOL programmer to take over my work, and that I was now assigned to work on the Oracle part of the project. She gave me a couple of Oracle programs as examples, taught me how to compile and run the program on a UNIX machine and went on vacation for two weeks.

I was stunned! I ran to the bookstore and found any book I could on Oracle and UNIX. By the time my manager came back from vacation I had finished two of the Oracle programs. She told me she had confidence in me and knew I would be able to do it. I thanked her for giving me an opportunity to work in Oracle. From then on I became a database developer writing Oracle Pro C and PL/SQL programs. I also started writing web pages using Oracle PL/SQL when Internet explorer was version 2. And I started building data warehouses using Oracle and Java when the concept was still very new. Our manager encouraged all of us to take on new challenges and to grow in every way we could.

I loved all these challenges. I especially enjoyed working in my group where we all learned these new things together. Even though we did not agree all the time, we shared our knowledge and produced a very fine product. It was a good time, but of course the good times never last long enough.

The Bad

A new VP who used to be the manager of another department in the company replaced our old VP. Shortly thereafter my entire department was eliminated. My manager became a programmer. The new VP moved each of us to different departments. I do not know why he did this because that department was the best I was ever in. Maybe he did not like our department, or, maybe he was jealous that we did such a good job. It made no sense to me or anyone I talked with. None of us liked the changes and we were angry that our department was eliminated. He also put in some new policies that programmers in other departments were not happy with. Immediately thereafter, resignation letters were flying all over the company, my own among them. Although the end was not pleasant, I learned a lot in this company, both from my manager and the developers I worked with. I was sorry to leave it. The ironic thing was the company got rid of that VP a year later.

I left that company in 2000. The level of management seemed to change rapidly after that. I think I saw more and more bad managers. Maybe the dot-com-bust was taking its toll.

My next job was as a Data Warehouse developer in a Fortune 500 company. I understood when I was hired that the company had gone through some serious changes; they had eliminated some departments that did not make money, and many employees had been laid off. But when I was hired into my department, I was told it was a solid department, and was making good profits for the company. I did not think it was likely to be eliminated.

It was an eye opener for me to work in that department. Few people worked very hard and few cared about the company. People would have meeting after meeting to discuss the same things over and over again and they never came up with a solution. I later learned that the manager of the department was hired because he had a friend working in upper management. He had no experience working as an IT manager and did not have any technical or management skills that I could see. The programmers did not respect or listen to him. Some people paid more attention to fantasy football than working on their job. The company spent lots of money hiring a lot of contractors to do what we could and should have done.

Within a couple of months after I started working there, the company announced that it was going to sell off my entire department as a unit. Everyone in the department was shocked although I don't know why. It ended up that no one wanted to buy it and, within six months, the company closed down the whole department.

How did it get like this? Did the people stop working because they knew management was going to dump them, or, did management decide to dump them because they were not working? I don't know where the managers were during all this. I think it was caused by bad management from top to bottom. Fortunately, I left the company and got a new job before the department closed down. I don't know whether to call my experience with this company the result of bad management, or, no management. Certainly, no one cared about my career.

My next job was with another large Fortune 500 company. Management did not get better. There were serious conflicts between departments. Instead of working together for the company, each department tried to put down the other departments. For example, one department was building a data warehouse. My manager knew that I had worked on data warehouses before so he asked me to build a data warehouse and told me not to tell anyone. I built the data warehouse using SQL Server and DTS. The other department built the data warehouse using SQL server and Ab Initio. My manager wanted to prove to upper management that his department was better than the other department. He showed my work to the vice president, told the vice president that his department could do a much better job than the other department, and could save a lot of money. The manager of the other department found out, was furious, and complained to the vice president. So I had to stop working on the data warehouse project. The project was a duplication of effort, a waste of time, and should not have been started in the first place. I call this bad management at all levels.

There were many other examples in this company. Upper management decided to outsource development work to India in order to save money. They promised there would be no layoffs. However the good employees knew better and started to leave. You could not say anything about the quality of the work of the offshore programmers, even to your manager. If you said something that suggested they did not do things right, the offshore company people said you were a racist and prejudiced against Indians. So you just kept quiet about it and corrected their mistakes. If the project did not go well, the upper management would blame the managers and the US employees even the development work was done by the offshore programmers. The managers in this company only cared about their own careers. They did not care about the company and the people working for them. They did whatever the upper management wanted, and upper management only cared about a nice quarterly earning report.

I simply could not work in that environment and left as soon as I could. I think about the only thing I learned at this company was to keep my mouth shut.

And The Ugly

I went from the frying pan into the fire. My next job was in a medium size company. Unfortunately, professionalism was missing in the department I worked in.

My manager was an SQL Server DBA. His style was to yell at people in front of others. When he made a mistake, he hid it under the carpet. When other people made a mistake, he emailed the whole department.

For example, he wrote a defective procedure to update some production data. When it failed, he did not tell the user he wrote a bad procedure. Instead, he told the user he did not have time to write the procedure and an update with the procedure would be issued in the following week. But it was different when others made mistakes. Once I wrote a script to put my work into production and forgot to 'grant permissions' to the users in the script. After it went in production, the users could naturally not access the database. My manager yelled at me in public and emailed the whole department that my implementation failed because I did not check my work carefully.

I think he liked to try to intimidate people in his department. One of my co-workers and friends got a very bad appraisal. In his appraisal my manager stated that my friend did not do a good job and he could not get along with his fellow co-workers. We all thought he did a very decent job and he got along well with us. We told him so, and we had no problems working with him. He thought my manager was biased against him because he disagreed with the manager a couple times, so he complained to HR about the bad appraisal. Unknown to him, the HR manager was a good friend of my manager. As soon as my friend made the complaint, the two managers came to my friend's office and told him that he was fired because of his bad appraisal.

This manager was probably the worst I ever had. He only thought of himself. He liked to intimidate people on his team. I think I reached managerial bottom in that company.

Needless to say, I did not last long at this company.

What has happened?

I don't think I just ran into a bunch of bad-apple managers in recent years. It is clear to me that fewer managers are "managing for the long term" today than were doing so a few years ago. I know I am not the only one to notice this deterioration in management. I think most developers today have observed the same deterioration in the quality of their management. What has happened here? Are we developers facing a permanent deterioration of our professional environment? I surely hope not.

A lot of people would put the blame on the dot-com-bust and the increasing pressure on IT departments to cut costs. Managers in IT departments have tough jobs, especially now when most companies seem only to worry about how to put out a good earnings report every quarter. If the companies fire people, outsource to other countries, or cut benefits, they can easily cut costs and boost earnings. Is this incompatible with managing for the long term, with as Steve observed, being concerned and responsible for the careers of their employees? Are the duties of the manager to his company at odds with the duty of the manager to his or her employees? I doubt it. Why then is this deterioration taking place?

Maybe there is just too much pressure on the managers today. Maybe they don't have the time or energy left to be concerned with our careers. I knew two managers in my old companies who stepped down to become developers. They said they did not make much more money than the senior developer, but the job was too stressful. At another company, one manager I knew took six months off on short term disability. When he came back, he resigned. At another company I knew a manager who had to take Prozac just to survive his daily work. I think all these people were trying to be good managers, working hard for the company and trying to take care of their employee developers. But the pressure was too great; the corporate culture just did not allow them to do so. This may cause the loss of some good managers, and prevent others who might have been good managers from trying. But at best, that is only part of an answer.

When I think about this question, I keep coming back to the one thing that was different at all the companies I worked for that had good managers. There was trust and respect between the manager and the employees in each of those cases. Where there were bad managers there was no trust and respect. Maybe trust and respect are simply the effects and not the cause of good management, but I don't think so. Wherever the managers trusted and respected me, I trusted and respected them. And when they trusted and respected me they were expressing concern for me and my career in the most positive way. In my view, they were clearly "managing for the long term" in every sense of Andy and Steve's description.

So I don't think that there is any incompatibility between a manager's duty to the company and his or her duty to the employees. I don't think the deterioration in the quality of management that I see is due to the pressures of cost-cutting and meeting earnings projections. I think the deterioration is caused by a deterioration of trust and respect between the managers and the employees they manage. These days, with company downsizing and corporate scandals, people may view trust and respect as impractical values. But they are not. I think they are the foundation of good management and the foundation of "managing for the long term."

Mind Your Manners

By Sushila Iyer

I often compare logging on to this site to visiting a pub during happy hour. When I am on the site and I see a familiar name light up in green, it takes all I have to keep myself from shouting out a "hey there - long time no see" or some such. Sometimes you get to interact with some of your faceless friends while you're responding to the same post - sometimes you raise a silent toast to a brilliant solution. At other times you just hang out - listening to the heated

argument at the next table about whose solution performs better - eavesdropping on some friendly bickering going on at that far forum in the corner and sometimes even being privy to personal tidbits and details unfold. What draws us all to this watering hole is gaining and/or sharing knowledge related to SQL Server.

For each rookie with a question or problem, there're any number of eager beavers (ranging from the merely enthusiastic to those with advanced savoir-faire to acknowledged savants) competing in an unpublicized race - for the coveted prize of being "first responder". For the most part things go along fairly well at this SQL-Xanadu - till you come across those notes of discord - some annoying, some just a tad unpleasant and some downright striking.

After years of silently seething over these deviant notes, it struck me that it was possible that some people may not be aware of what constitutes "acceptable norms". Even though it's almost no different from when you're sitting vis-a-vis someone - you may be separated by a table or miles of ether - but the rules for interaction and social skills are mostly similar. I compiled a list of dos and don'ts for all such deviants in order to provide them with a comprehensive catalog. To illustrate by examples, I have used only contents in actual posts on SSC so I can't be accused of making stuff up. Everything in double quotes from here on is therefore, an excerpt from one of the many forums.

Know your subject:

Please post in the relevant forum - make a point of reading the title before posting - they are all divided very comprehensibly between 2000 and 2005 - within these you have Administration, Programming, General, Replication etc. There is nothing more annoying than trotting off to lend a helping hand in a 2000 related t-sql forum only to find that it's all about Yukon integration services or vice-versa. The **Suggestions** forum is the one where I find the most mis-postings. This one is for the members to make suggestions to the site-owners about a wishlist, changes they'd like to see, ideas on improving the site etc. More often than not, people seem to think this is to be used for **asking** for suggestions - I have seen "**coalesce is not working**"; "**IF statement in a function**"; "**How we can restrict a single ROW in SQL server 2005?**" posted here and it's really a measure of how dedicated and thorough the SSC members are, that these get answered at all.

The truth, the whole truth and nothing but...:

The more detailed your query, the quicker someone will post a solution. Vague and cryptic posts end up being frustrating and time-consuming for both the poser as well as the people trying to help. Here're some quotable responses from people who want to help but cannot because they don't have enough information:

- 1) **"Please post DDL, so that people do not have to guess what the keys, constraints, Declarative Referential Integrity, datatypes, etc. in your schema are. Sample data is also a good idea, along with clear specifications."**
- 2) **"Post : - Table definitions - sample data - wrong resultset - expected resultset. It's like asking a mechanic to fix your flat tire without bringing the car in. That's just not gonna work."**

Then the question from someone who wanted to know why his procedure was taking **so** long to execute without really giving much more information on run time etc. In trying to extract more information, one member said that time is really very relative and went on to exemplify - **"Einstein's thoughts on the relativity of time: If you hold your hand on a hot stove for a second that is a long time. If you spend the night with a beautiful woman that is a very short time."**

Help us help you:

This request is joined at the hip with the previous one. Please know that when you post ddl and sample data etc. it saves the troubleshooter valuable time and you get a double action response in record time. The link that is referenced here - ["SQL ETIQUETTE SITE"](#) is obviously not visited enough. Newbies, in particular, should make it a point of stopping here first and making it step 1 in the process of posting their query. I've often toyed with the idea of suggesting to the site-owners that they have a pop-up window that reminds people to visit this site first **before** posting. Maybe they could filter it by forum title and number of post counts so (eg.) anyone posting in the **newbie**

forum that has a post count of less than 100 (or thereabouts) will get this link - like an annoying close cousin of the intrusive Office paperclip!!! Here's a query from someone who not only needs help but really knows how to ask for it :

"I need this for my monthly report. Any help is greatly appreciated.

/* Schema */

DECLARE @MyCalls TABLE

(

CallDate DATETIME,

Calls INT

)

DECLARE @MyCars TABLE

(

DataDate DATETIME,

Qty INT

)

/* Data */

INSERT @MyCalls

SELECT '06/01/2006', 1 UNION ALL

SELECT '06/02/2006', 12 UNION ALL

SELECT '06/03/2006', 5 UNION ALL

SELECT '06/04/2006', 4 UNION ALL

SELECT '06/05/2006', 2

INSERT @MyCars

SELECT '06/04/2006', 1256 UNION ALL

SELECT '06/11/2006', 1267 UNION ALL

SELECT '06/18/2006', 1282 UNION ALL


```
SELECT '06/25/2006', 1298.....
```

```
/* Requirement */
```

Calls are recorded daily and cars are counted weekly

I have to create a report how many calls were received per car
daily.

If the parameter is passed as 'PAST'

```
SELECT A.CallDate, A.Calls / B.Qty (Qty as of previous recorded date from call date)
```

```
FROM
```

```
@MyCalls A
```

```
JOIN
```

```
@MyCars B
```

If the parameter....."

Eggs in one basket:

DO NOT POST IN MULTIPLE FORUMS. I cannot emphasize this one enough. It really does send many of the problem solvers into a downright hissy fit (yes - even the men) if they've just spent some measurable time answering a question to discover that it's been posted in four different forums and was already addressed in one of the others. It's one thing to have enough ambiguity in the question that you feel that it may be appropriate under multiple forums and quite another to think that your post would have more visibility if you have an "in-your-face" approach and thrust it under everyone's collective nose. If you're not sure where to post it, mention that in your post, and if it's too much off the topic it will be pointed out.

Mind your Ps and Qs:

You ask a question and someone takes the trouble to answer it - sometimes immediately and sometimes after going back and forth for a long time to help you resolve it. It really does behoove you to thank the person as well as post the working solution so others can benefit from your post. I know one member who spends an inordinate amount of time running scripts and doing comprehensive analyses before posting his solution. He's (almost) always unfailingly polite but even he reached the end of his tether when he got neither a feedback nor a thank you - **"Ok, then... you ask a question, I bust a hump giving you a pretty good answer... did it work for you or what?"** At the very least, let the person know that you're working on it and will get back as soon as you have everything sorted out.

Do your homework:

If you have the leeway and can spend some time using the various search features on the site you'll find that many questions have been answered in part or fully at some time in the past. If those answers don't satisfy your requirements then you can post your own query. Another good source is Books Online - abbreviated to **BOL** - the help documentation that ships with SQL Server and provides invaluable reference. You will be looked upon much more kindly if you make known the fact that you've tried your best to solve something on your own before seeking help. Otherwise you may come across harsh but justifiable comments like these:

- 1) **"Dude, get a book and learn, this isn't a school for newbies. We don't mind helping but we actually paid and worked to get our information."**
- 2) **"I think it's time you start TRYING to learn on your own. I'm not paid to show you every little details of sql programmings and I surely won't do it."**
- 3) **"Would you mind to open BOL? Or you prefer somebody else to do it for you and copy-paste the topic here?"**
- 4) **"Sorry for being blunt but you need a wake up call."**

Temper that tone:

You're the one seeking help. Set your **attitude** aside and realize that arrogance on your part will only antagonize those who are trying to help. There have been some downright rude and appalling posts that on any other site would not even have merited a response.

- 1) **"..before replying to the question think and if dont know the answer wait for the answer."**
- 2) **"I request to those who suggest me to read books online pls dont reply."**
- 3) **"Ur answer makes me laugh."**

Given the diverse backgrounds of members as well as the fact that the English language is not everyones' strong suit it is possible to make allowances, but only to an extent. Having adopted (what I thought) a particularly clever signature line, it was a rude awakening to realize that some actually misread it as a personal attack.

- 4) **"I m no doubt polite, but I may be rude bcoz of following quotation ASCII stupid question, get a stupid ANSI !!!"**

And now this..the only section devoted to (n)etiquette for those who respond!!!

Just because you're a SQL God...:

is no reason to not be patient and courteous with those who are struggling to understand. Anyone who chooses to help does so because he/she wants to - no one is compelled to help and for the most part most of the members are extremely helpful without being condescending or unkind. There are those, however, who are inarguably far superior in their SQL skills and often their chastisement of the uninitiated is so harsh and severe that these people flee in terror never to come back. Does this not defeat the very purpose of a site like this ?! Here're some classic excerpts from posts where the newbies have been put through the shredder..

Q)"magic tables will created automatically when trigger fires, i want to know that naming convention means tablename_insert like that, i want correct formation"

A) "We do not work and live in FairyLand where elephants drop out of the sky by magic. Please stop programming before you hurt someone and take the time to read at least one book on RDBMS and data modeling."

"Stop progamming and start reading. Then start re-writting all of your code. Have you noticed that most of the posting are not for help, but for kludges? Scenario: The poster has coded himself into a corner because he did not bother with even a basic book. He then goes to a newsgroup and posts what Chris Date calls "do my homework for me" requests."

Sure - these gurus are justifiably angry with all those poorly designed databases out there that're limping along defying with every lame stumble the very foundations of RDMS - when they take the trouble to explain themselves, you can actually feel their angst:

"If this were a woodworking newsgroup and someone posted a request for the best kinds of rocks to smash screws into fine furniture, would you tell them "Granite! Big chunks of it?" Or would you tell them to look up screwdrivers, pre-drilled holes and specialty screws? And try to get that rock out of their hands ..."

Here's another SQL heavyweight with a sarcastic observation:

"It's funny, but I never have read from anybody designed those databases, everybody just inherited and does not have a chance to change anything."

and the impassioned comeback from a clearly angered poster who aptly sums up the frustrations of the real world scenarios that most of us have to face:

"..he also apparently lives in a fantasyland where everyone is hired by a company that is just beginning to use SQL Server, and therefore has full and utter control over the design, where no one is a short term consultant that has to work within the existing framework no matter how badly designed, and where there aren't a million applications already in existence that are running against a database, thus making design changes extremely tricky. He also seems to forget that there is a state in the learning process that is between "I don't have a freaking clue about SQL Server" and "I'm a a SQL Server god". In reality, the vast majority of SQL folks are at some point between the two, and helping them towards the latter goal is why this board is here. While it's always a good idea to attempt to fix problems with design rather than to work around them, some of us in the real world are perfectly willing to let you know what would help if you do have the option of fixing things, while at the same time helping you solve your immediate problem if you don't. Most of us have been in both pairs of shoes."

As this previous posting points out there are people at such an introductory level, that they really don't have a "freaking clue about SQL Server" - I had once compiled a list of my favourite top 10 newbie questions that, in accordance with Murphy's laws, vanished mysteriously a couple of days before I sat down to write this. I did, however, find two others -

1) "Stupid question, but what is books on line?"

2) "...select 1

from

```
dbo.Page_History pgh2

where

pgh1.startdate = max(pgh2.startdate)

and pgh1.starttime = max(pgh2.starttime)

and pgh1.pageid = pgh2.pageid...
```

What are the pgh1 and pgh2? Am I supposed to substitute anything here?"

Are these questions enough cause for launching a diatribe?! Maybe..but it takes courage to ask a question and even more to admit ignorance. These laudable reasons alone should earn them some kindness but if you're not feeling charitable enough, try and recall these quotes to aid you: **Kindness is a language that the deaf can hear and the blind can see** - attributed to Mark Twain and another that's less preachy - **Courtesy is a small act but it packs a mighty wallop** (source unknown). For those who want to get downright colloquial, here's one from one of the SSC members - **"Manners, dude. Learn them. Might be helpful one day."** Tact and diplomacy are really not that hard to master and here's a post by Phil Factor that's an exemplary illustration of how you can be one of the SQL greats and yet not look down on the hoi polloi:

"Forgive me for presuming to re-write your routine, but I don't see why you need the table at all. If all you are doing is getting the datetime value of the previous monday then something like this would do the trick:"

Conclusive evidence:

Ultimately, it's all about communication skills as much as it is about manners and etiquette. It's only fitting, therefore, to conclude with this great bit on communications posted by a member, that is presented here in its entirety:

"No matter what language you use; precision of communication is important. Let me give you an example of imprecise communication:

"Mommy where did I come from?" Her six year old asks. She thinks that it is way too early for that particular lecture and is about to tell him to wait when he continues. "Bobby came from Ireland, June came from France, where did I come from?"

Just as it is important to know what the questioner is asking, it is very important to listen and get all the data before you act. If you can not communicate precisely it is impossible to answer the question. No matter what the language."