

Brad's Sure Guide to SQL Server 2008

The Top Ten New
Features for DBAs

By Brad McGehee



ISBN: 978-1-906434-06-9
Shelving: Database/SQL Server



In association with

redgate®

www.simpletalkpublishing.com

Brad's Sure Guide to SQL Server 2008

The Top Ten New Features for DBAs

by Brad McGehee

First published 2008 by Simple-Talk Publishing

Copyright Brad McGehee 2008

ISBN 978-1-906434-06-9

The right of Brad McGehee to be identified as the author of this work has been asserted by him in accordance with the Copyright, Designs and Patents Act 1988

All rights reserved. No part of this publication may be reproduced, stored or introduced into a retrieval system, or transmitted, in any form, or by any means (electronic, mechanical, photocopying, recording or otherwise) without the prior written consent of the publisher. Any person who does any unauthorised act in relation to this publication may be liable to criminal prosecution and civil claims for damages.

This book is sold subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out, or otherwise circulated without the publisher's prior consent in any form other than which it is published and without a similar condition including this condition being imposed on the subsequent publisher.

Typeset by Andrew Clarke

CONTENTS

| | |
|---|----|
| Contents..... | 2 |
| About the author..... | 6 |
| Introduction..... | 7 |
| Foreword..... | 9 |
| Chapter 1: Management Studio Improvements..... | 10 |
| Activity Monitor..... | 10 |
| Object Explorer Details..... | 12 |
| Object Search..... | 15 |
| Multi-Server Queries..... | 16 |
| IntelliSense Added to the Query Editor..... | 17 |
| T-SQL Debugger Added to the Query Editor..... | 18 |
| Summary..... | 19 |
| Chapter 2: Policy-Based Management..... | 20 |
| What Policy-Based Management Does..... | 20 |
| How You Might Use Policy-Based Management..... | 21 |
| How Policy-Based Management Works..... | 23 |
| How to Implement a Simple Policy..... | 24 |
| Step 1: Selecting Facets..... | 24 |
| Step 2: Setting the required Property Conditions..... | 26 |
| Step 3: Creating the Policy..... | 27 |
| Step 4: Running the Policy..... | 29 |
| Checking Multiple Servers for Compliance..... | 31 |
| Summary..... | 32 |
| Chapter 3: Data Compression..... | 34 |
| Data Compression Overview..... | 34 |
| Row-Level Data Compression..... | 35 |
| Page Level Data Compression..... | 36 |
| Data Compression Demo..... | 36 |
| Backup Compression..... | 41 |

| | |
|---|----|
| Summary | 44 |
| Chapter 4: Resource Governor | 45 |
| Uses for Resource Governor | 45 |
| How Resource Governor Works..... | 46 |
| Configuring Resource Governor..... | 48 |
| Creating Resource Pools..... | 49 |
| Creating Workload Groups | 49 |
| Creating the Classification Function | 50 |
| Enabling Resource Governor..... | 50 |
| Summary | 51 |
| Chapter 5: Performance Data Collector | 52 |
| How to Configure the Performance Data Collector | 53 |
| How the Performance Data Collector Works..... | 54 |
| Reports Available from the Performance Data Collector..... | 57 |
| Summary | 61 |
| Chapter 6: Transparent Data Encryption | 62 |
| Why Use Transparent Data Encryption?..... | 62 |
| Limitations of TDE..... | 63 |
| How Transparent Data Encryption Works | 63 |
| How to Implement Transparent Data Encryption | 65 |
| Creating a Master Key..... | 65 |
| Create or Obtain a Certificate Protected by the Master Key..... | 66 |
| Create a Database Encryption Key..... | 66 |
| Backup the Private Encryption Key and Certificate | 66 |
| Turn TDE On..... | 66 |
| Summary | 67 |
| Chapter 7: SQL Server Audit | 68 |
| Advantages of SQL Server Audit..... | 68 |
| Limitations of SQL Server Audit | 69 |
| How SQL Server Audit Works..... | 69 |
| A Simple Auditing Example | 71 |

| | |
|---|-----|
| Creating the Audit Object..... | 72 |
| Creating a Server or Database Audit Specification | 76 |
| Starting the Audit..... | 83 |
| Reviewing Audit Events | 83 |
| Summary | 85 |
| Chapter 8: New Data Types..... | 86 |
| Date and Time | 86 |
| DATE..... | 87 |
| TIME..... | 87 |
| DATETIME2..... | 87 |
| DATETIMEOFFSET | 87 |
| Spatial | 88 |
| HIERARCHYID..... | 89 |
| FILESTREAM | 89 |
| Benefits of FILESTREAM | 90 |
| Limitations of FILESTREAM..... | 90 |
| How to Implement FILESTREAM Storage..... | 90 |
| Summary | 93 |
| Chapter 9 Extended Events..... | 94 |
| An Overview of Extended Events..... | 94 |
| Benefits and Limitations of Extended Events | 95 |
| Extended Events Sessions | 96 |
| Creating an Extended Events Session..... | 96 |
| Summary | 101 |
| Chapter 10: Change Data Capture..... | 102 |
| Change Data Capture Architecture | 102 |
| Source Tables | 104 |
| Change Data Capture Tables..... | 104 |
| Change Data Capture Query Functions | 104 |
| Capture Instances | 104 |
| Capture and Cleanup Jobs..... | 105 |

| | |
|---|-----|
| How Change Data Capture Works | 105 |
| Implementing Change Data Capture for a Single Table | 105 |
| Summary | 109 |
| Index..... | 110 |

ABOUT THE AUTHOR

Brad McGehee is a MCSE+I, MCSD, and MCT (former) with a Bachelors' degree in Economics and a Masters in Business Administration. Currently the Director of DBA Education for Red Gate Software, Brad is an accomplished Microsoft SQL Server MVP with over 13 years' SQL Server experience, and over 6 years' training experience.

Brad is a frequent speaker at SQL PASS, SQL Connections, SQLTeach, Code Camps, SQL Server user groups, and other industry seminars, where he shares his 13 years' cumulative knowledge.

Brad was the founder of the popular community site SQL-Server-Performance.Com, and operated it from 2000 through 2006, where he wrote over one million words on SQL Server topics.

A well-known name in SQL Server literature, Brad is the author or co-author of more than 12 technical books and over 100 published articles. His most recent book was "How to Become an Exceptional DBA," and later this year he will be releasing a book on how to use the SQL Server 2005/2008 Profiler.

When he is not travelling to spread his knowledge of SQL Server, Brad enjoys spending time with his wife and young daughter in Hawaii.

INTRODUCTION

Brad M. McGehee

7 August 2008

SQL Server 2008 has hundreds of new features and improvements for Production DBAs, Developer DBAs, and Business Intelligence specialists. Some of the changes are significant, while others are minor, but still important.

When I decided to write this book, I had to narrow my focus, as it would take more than a thousand pages to cover every new change in SQL Server 2008. After a lot of consideration, I decided to focus on what I thought were the ten most important new features for Production DBAs. As you can imagine, my choice for the top ten new features in SQL Server 2008 may not match yours, but I think I have covered many of the features that will be of most interest to Production DBAs.

In addition, I decided not to cover the new features in great detail. Instead, the focus of each chapter is to introduce you to the new feature, discuss why you might want to use it, describe how it works, and to provide a simple example. If you find any of these topics of special interest, I suggest you follow up and learn more about them from Books Online, and by experimenting with the new features on a test SQL Server 2008 box.

Here are the features that I picked out for inclusion, with a chapter of the book on each topic:

- **Management Studio Improvements** – SSMS 2008 starts to rebuild its reputation in the DBA community with the introduction of IntelliSense and a debugger for T-SQL, the ability to run multi-server queries as well as improved performance and numerous other enhancements.
- **Policy-Based Management** – every DBA knows the frustration of trying to manage tens of servers, each of which has a subtly different configuration. Policy-based management could ease a lot of this pain.
- **Data Compression** – the first immutable law of database administration is that databases will grow (and grow) over time. SQL 2008 enterprise brings with it data and backup compression, thus reducing physical file sizes as well as disk I/O.
- **Resource Governor** – everyone knows it's not "best practice" but most DBAs, nevertheless, deal with situations whereby a SQL Server that supports OLTP applications is also used extensively for reporting. Often this can cause enough resource contention to affect production activity. Resource Governor promises to be a valuable tool in curtailing this problem
- **Performance Data Collector** – historically, DBAs have used a mishmash of different tools to get performance data out of SQL Server, including Profiler, System Monitor, DMVs, and more. Performance Data Collector is one of the tools that starts to bring together all this information, moving more towards a single-location, single-format model of collecting, storing and analyzing performance data.
- **Transparent Data Encryption** – whether we like it or not, Security is an issue with which DBAs are going to become more and more familiar. 2008 Enterprise adds full database-level encryption to column-level encryption.
- **SQL Server Audit** – again, auditing is an area in which DBAs have tended to use several different methods to track the activities they need to monitor, from DDL triggers, to SQL Trace to third-party tools. The new SQL Server Audit attempts to expand the range of activities that can be easily monitored, as well as being easier for the DBA to use and more granular.

- **New Data Types** – it's a long time since SQL Server has been a simple "relational data store", and SQL 2008 pushes boundaries again, introducing spatial data types, hierarchical data, as well as unstructured data such as videos, Excel files and so on.
- **Extended Events** – These have been introduced in order to help "centralize" performance data collection in SQL. They provide a generic event-handling system aimed at making it much easier to capture, store and act on disparate troubleshooting data from SQL Server, the Windows OS, and applications.
- **Change Data Capture** –DBA always strive to separate OLTP and OLAP activities cleanly onto separate servers. However, moving data between the two, while keeping them synchronized, has proved a difficult problem. Change Data Capture could well be the answer.

SQL Server 2008 is not as difficult to learn as SQL Server 2005 was, but neither is it so trivial that it can be accomplished overnight. I suggest you reserve a few hours every week for the next several months, devoting them to learning the new features SQL Server 2008, along with the benefits they provide.

I hope this book gets you off to a good start.

FOREWORD

The SQL Server platform expanded substantially with the arrival of SQL Server 2005, introducing many features that people now rely on every day (Try-Catch error handling, DDL triggers), as well as a few that captured people's attention but ultimately left many scratching their heads (CLR integration).

SQL Server 2008 is not a radical product in the same way as SQL Server 2005. Of course, there have been polite murmurs of interest around such features as Resource Governor, Transparent Data Encryption, table-valued parameters, policy-based management, the new GIS data types and functions, data and backup compression, and the MERGE statement. However, in terms of added bells and whistles, there seems not to be a standout reason to upgrade to SQL 2008, and maybe this is not a bad thing.

With SQL 2008, the improvements are more of a massive tidy-up. Without a doubt, SQL Server 2008 is a much better, and more usable, product than its predecessor and one that most of us will be happy to move to because there is no demand for any code re-engineering, or radical cultural changes in the users. SS 2008 works much like SS 2005, but with the lumps ironed out.

It's clear that most parts of the system have had been refined. The improvements to Reporting services, SSAS and SSIS are typical; no real fireworks, but everything just works a little better and faster. Some new features that have not been given much publicity are quite astonishing, my personal favorite being the new data mining add-ins for Office 2007, which really do make light work of some complex analysis, and do so with quite a bit of style.

It is easy to joke that SQL Server 2008 is a strange name for the biggest Service Pack ever, but there is a certain truth in that. There isn't any reason to stay with SQL Server 2005, just as it is generally wise to apply the latest service pack. Unencumbered by any of the wild changes in the features that we saw in the development of SQL Server 2005, SQL 2008 has slid quietly into view showing all the hallmarks of a product that has been driven by the requests of the ordinary user rather than the bleatings of the large corporate users.

Tony Davis

Red Gate Software Cambridge

Cambridge 2008

CHAPTER 1: MANAGEMENT STUDIO IMPROVEMENTS

DBAs have never really taken to SQL Server Management Studio (SSMS) with enthusiasm. However, SQL Server 2008 has brought with it a multitude of improvements to SSMS, which are intended to make it a little easier and more convenient to use.

I originally intended to include in this article every new change I could find, but I had to give up. There are so many improvements that it would take a short book to cover them all. No, this is not to say that Microsoft has changed SSMS so much that you will have to relearn it. SSMS is essentially the same tool we learned when it was first introduced with SQL Server 2005. On the other hand, it includes many new features and enhancements that make performing the routine tasks of a DBA a little easier. In addition, SSMS has gotten a speed boost, thanks to faster communications between SSMS and the SQL Server engine.

Here is what I consider to be some of the most important improvements in SSMS 2008:

- Activity Monitor
- Object Explorer Details
- Object Search
- Multi-Server Queries
- Intellisense Added to the Query Editor
- T-SQL Debugger Added to the Query Editor

Besides these key new features, there are other hidden gems that I hope you discover for yourself when you begin using SQL Server 2008 SSMS.

Activity Monitor

In SQL Server 2005, it was easy to find and start the Activity Monitor. You just opened up the Management object in SSMS and double-clicked on Activity Monitor, and you could instantly view process information. In SQL Server 2008, they have made the Activity Monitor a little harder to find, but once you have learned where it is, you will probably be very impressed with all the new features that have been added.

To start Activity Monitor in SQL Server 2008 SSMS, right-click on the SQL Server name you want to monitor, and then click on Activity Monitor. The following screen appears:

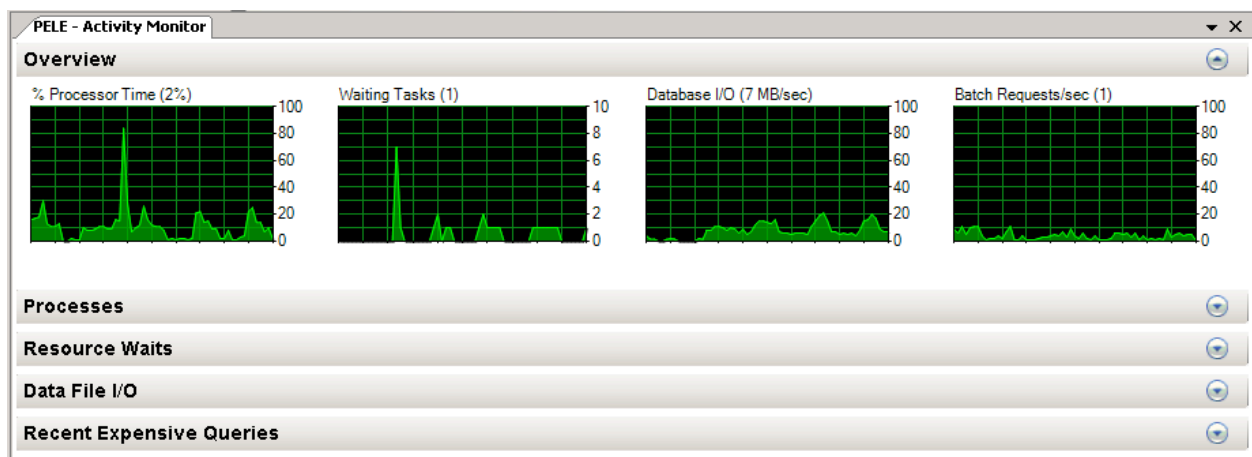
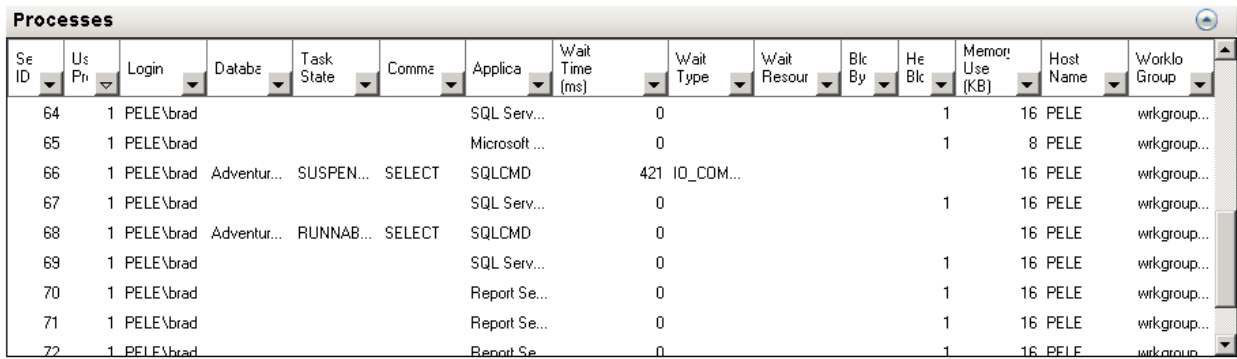


Figure 1: The Activity Monitor has had a radical facelift, and feature boost.

Immediately, you can see that Activity Monitor looks entirely different than in SQL Server 2005. The first things that jump out at you are the four graphs. These display % Processor time (of the SQL Server process “sqlserv” spread over all of the available CPUs, not for the entire server), Waiting tasks, Database I/O, and Batch Requests/sec. In the past when you needed this information, you had to use System Monitor or some other tool. Now, if your server is behaving strangely and you need a quick overview of what is happening, you can get it directly from Activity Monitor.

Below the graphs you will find four additional windows of information, the first of which (Processes) is shown in **Figure 2**:

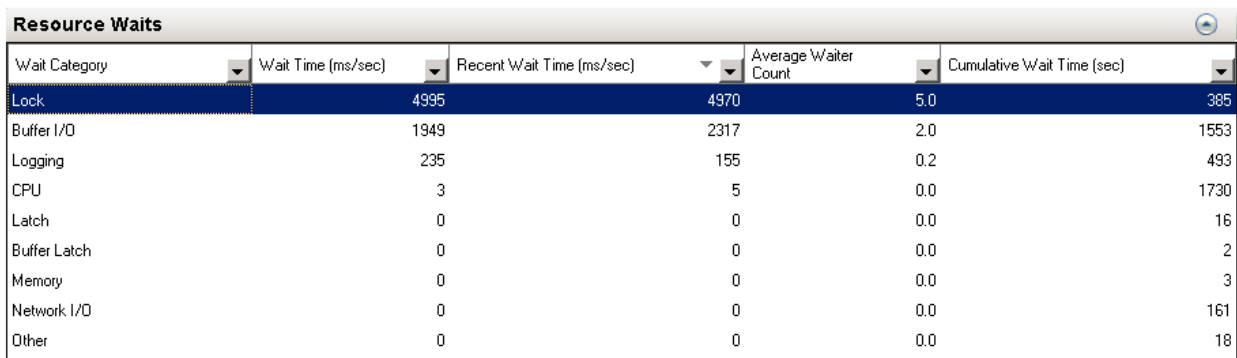


| SPID | User | Login | Database | Task State | Comments | Application | Wait Time (ms) | Wait Type | Wait Resource | Blocked By | Held By | Memory Use (KB) | Host Name | Workload Group |
|------|------|-----------|-------------|------------|----------|---------------|----------------|-----------|---------------|------------|---------|-----------------|-----------|----------------|
| 64 | 1 | PELE\brad | | | | SQL Serv... | 0 | | | | | 1 | 16 PELE | wrkgroup... |
| 65 | 1 | PELE\brad | | | | Microsoft ... | 0 | | | | | 1 | 8 PELE | wrkgroup... |
| 66 | 1 | PELE\brad | Adventur... | SUSPEN... | SELECT | SQLCMD | 421 | IO_COM... | | | | 16 | PELE | wrkgroup... |
| 67 | 1 | PELE\brad | | | | SQL Serv... | 0 | | | | | 1 | 16 PELE | wrkgroup... |
| 68 | 1 | PELE\brad | Adventur... | RUNNAB... | SELECT | SQLCMD | 0 | | | | | 16 | PELE | wrkgroup... |
| 69 | 1 | PELE\brad | | | | SQL Serv... | 0 | | | | | 1 | 16 PELE | wrkgroup... |
| 70 | 1 | PELE\brad | | | | Report Se... | 0 | | | | | 1 | 16 PELE | wrkgroup... |
| 71 | 1 | PELE\brad | | | | Report Se... | 0 | | | | | 1 | 16 PELE | wrkgroup... |
| 72 | 1 | PELE\brad | | | | Report Se... | 0 | | | | | 1 | 16 PELE | wrkgroup... |

Figure 2: This looks a little more familiar. Here, we see all of the active SPIDs.

When you open the Processes window, you see the SPIDs that you are so familiar with from the SQL Server 2005 SSMS Activity Monitor. Now, not only can you see the SPIDs, and sort them, but you can also filter on them using the drop-down boxes at the top of each column. And if you right-click on any of the SPIDs, you can choose to automatically launch Profiler, which will begin a trace on the SPID in question. This makes it very easy to begin a Profiler analysis of any SPID that you want to investigate.

Next, we move on to the Resource Waits window, shown in **Figure 3**:



| Wait Category | Wait Time (ms/sec) | Recent Wait Time (ms/sec) | Average Waiter Count | Cumulative Wait Time (sec) |
|---------------|--------------------|---------------------------|----------------------|----------------------------|
| Lock | 4995 | 4970 | 5.0 | 385 |
| Buffer I/O | 1949 | 2317 | 2.0 | 1553 |
| Logging | 235 | 155 | 0.2 | 493 |
| CPU | 3 | 5 | 0.0 | 1730 |
| Latch | 0 | 0 | 0.0 | 16 |
| Buffer Latch | 0 | 0 | 0.0 | 2 |
| Memory | 0 | 0 | 0.0 | 3 |
| Network I/O | 0 | 0 | 0.0 | 161 |
| Other | 0 | 0 | 0.0 | 18 |

Figure 3: We can see the current wait states of active threads.

The ‘Resource Waits’ screen provides a snapshot of key resource waits occurring on the server, thereby helping you to identify potential trouble with your SQL Server. Resource waits measure the amount of time a worker thread has to wait until it can gain access to the resources on the server that it needs, such as memory or CPU. A high resource wait time might indicate a resource bottleneck. As with Processes, you can sort and filter on any column.

The third window is Data File I/O, as shown in **Figure 4**:

| Database | File Name | MB/sec Read | MB/sec Written | Response Time (ms) |
|----------------|--|-------------|----------------|--------------------|
| tempdb | C:\Program Files\Microsoft SQL Server\MSSQ... | 0.0 | 11.0 | 51 |
| tempdb | C:\Program Files\Microsoft SQL Server\MSSQ... | 0.0 | 0.0 | 35 |
| AdventureWorks | C:\Program Files\Microsoft SQL Server\100\T... | 8.5 | 0.0 | 7 |
| msdb | C:\Program Files\Microsoft SQL Server\MSSQ... | 0.0 | 0.0 | 7 |
| AdventureWorks | C:\Program Files\Microsoft SQL Server\100\T... | 0.0 | 0.0 | 0 |
| master | C:\Program Files\Microsoft SQL Server\MSSQ... | 0.0 | 0.0 | 0 |
| master | C:\Program Files\Microsoft SQL Server\MSSQ... | 0.0 | 0.0 | 0 |
| MDW | C:\Program Files\Microsoft SQL Server\MSSQ... | 0.0 | 0.0 | 0 |
| MDW | C:\Program Files\Microsoft SQL Server\MSSQ... | 0.0 | 0.0 | 0 |
| model | C:\Program Files\Microsoft SQL Server\MSSQ... | 0.0 | 0.0 | 0 |

Figure 4: Use the Data File I/O screen to identify databases with heavy I/O activity.

If you suspect that a particular database is being heavily hit with disk I/O, you can quickly find out by using the Data File I/O screen. You can sort and filter on any column with this or the other related screens.

The final screen is "Recent Expensive Queries":

| Query | Executions/m | CPU (ms/sec) | Physical Reads/sec | Logical Writes/sec | Logical Reads/sec | Average Duration (ms) | Plan Count |
|--|--------------|--------------|--------------------|--------------------|-------------------|-----------------------|------------|
| SELECT COUNT (DISTINCT c2) FROM Table... | 480 | 45796 | 353068 | 7 | 1068582 | 124444 | 1 |
| SELECT 'Total income is', ((OrderQty * UnitPric... | 240 | 5107 | 426 | 0 | 7869 | 39380 | 1 |
| SELECT p.Name AS ProductName, ISNULL(Disc... | 180 | 4275 | 387 | 0 | 6078 | 2645 | 1 |
| SELECT p.[Name], AVG (pch.StandardCost) A... | 49200 | 3204 | 0 | 0 | 23780 | 4 | 1 |
| SELECT ProductID, OrderQty, LineTotal FROM P... | 180 | 2973 | 0 | 0 | 3738 | 1259 | 1 |
| SELECT p1.ProductModelID FROM Productio... | 26640 | 1213 | 0 | 0 | 583865 | 3 | 1 |
| SELECT ProductID, LineTotal FROM Sales.Sa... | 360 | 917 | 0 | 0 | 7476 | 164 | 2 |
| SELECT SalesOrderID, SUM(LineTotal) AS Su... | 180 | 857 | 0 | 0 | 3738 | 295 | 1 |
| SELECT ProductID, OrderQty, UnitPrice, LineT... | 180 | 685 | 0 | 0 | 3738 | 321 | 1 |

Figure 5: Want to know what your most expensive queries are? Find out here.

If you are having performance problems due to resource-intensive queries, then the Recent Expensive Queries window will show you the most recent expensive queries (those currently in cache), allowing you to sort or filter them by any column, making it easy to identify problem queries. If you right-click any of the queries, you have the option of displaying the entire query (not just the small part of the query you see in the window) and you also have the option of displaying a graphical execution plan of the query.

Another feature that you might miss, if you are not careful, is the use of Tool Tips throughout all the screens of the Activity Monitor. If you move the cursor on top of almost any text on the Activity Monitor screen, a Tool Tip will appear, providing you with useful information on what you are seeing. Most of the data displayed in the Activity Monitor are from DMVs. Many of the Tool Tips even tell you the name of the DMV used to return the data you are viewing.

When you first lay hands on a copy of SQL Server 2008, you should start by trying out the new Activity Monitor. I guarantee it will make it much easier for you to quickly get a high-level perspective on what might be ailing your SQL Server. Once you know the big picture, then you can use other tools, such as DMVs, Profiler, or System Monitor, to drill down for more details.

Object Explorer Details

In SQL Server 2005, the Object Explorer's 'Details' screen, the one that appears by default to the right of the Object Explorer, wasn't particularly useful. Essentially, it just displayed the same

information that you saw in the Object Explorer and I generally closed it because it didn't offer me any value.

This has changed in SQL Server 2008. Instead of repeating what you see in Object Explorer, you are generally presented with a lot of useful information about the object selected in Object Explorer. The level of detail depends on which object in the Object Explorer you have selected. For a quick demo, let's take a look at the Object Explorer Details window when "Databases" has been selected in the Object Browser.

Note If the Object Explorer Details window does not appear by default when the Object Explorer is displayed, either press the F7 key, or select "Object Explorer Details" from the "View" menu.

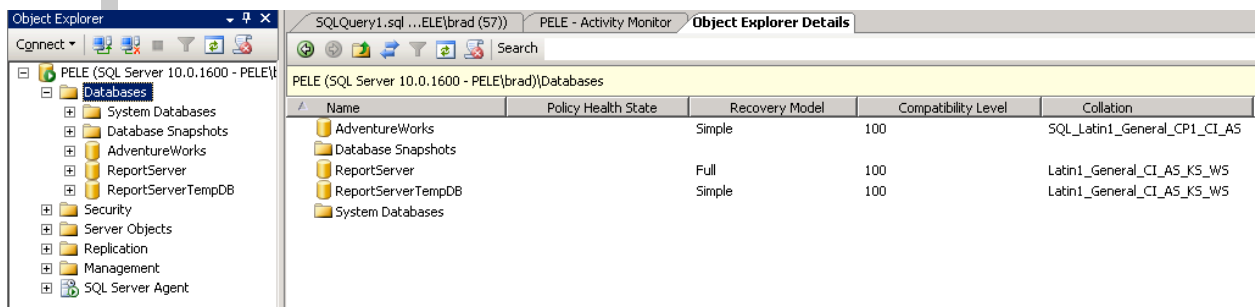


Figure 6: The Object Explorer Details screen can provide detailed information about many of the objects found in the Object Explorer.

When I click on "Databases" in Object Explorer, I can see in the Object Explorer Details screen a variety of information about each of the databases, such as its Policy Health State, Recover Model, Compatibility Level, Collation, and so on. Notice the small folder icons next to System Databases and Database Snapshots in figure 6 above. Since these are represented as folders, you can click on these to drill down for additional information.

By default, five columns of data are displayed for each database, although you can only see four in figure 6 because of the lack of room. But what is really cool is that you are not limited to only five columns of data. If you right click on any of the column headings, you get a drop-down box, see figure 7, which allows you to select up to 36 different columns to be displayed. In addition, you can sort any of the rows, and you can move them in any order you prefer.

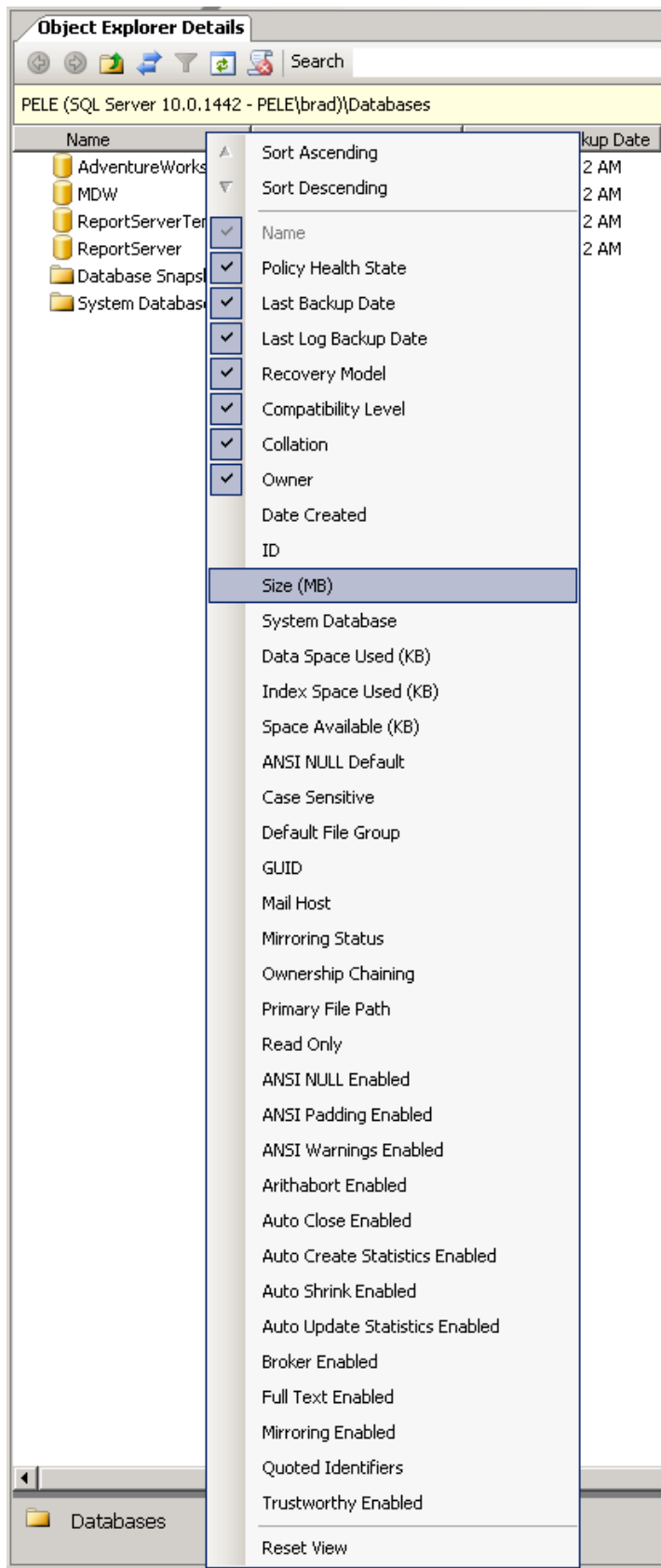
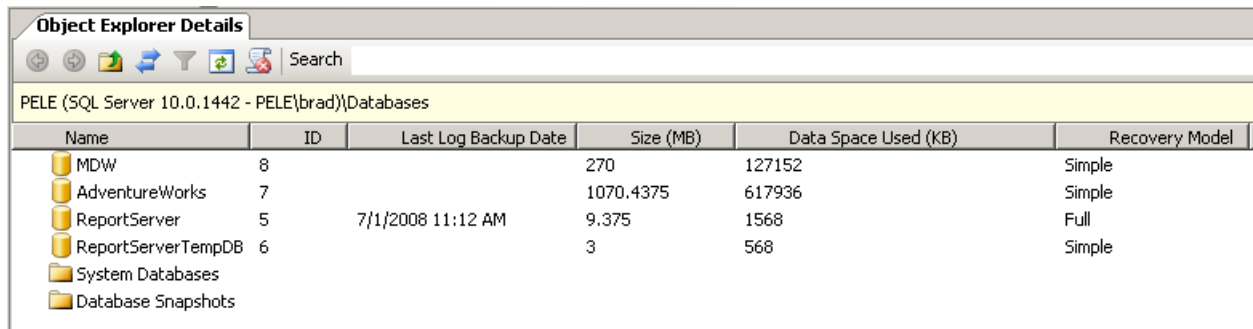


Figure 7: You can display up to 36 different columns of information about each database.

Any changes you make are automatically remembered, so the next time you come back to this screen, it will appear just like you left it, as you see in figure 8.



The screenshot shows the 'Object Explorer Details' window for a SQL Server instance. The window title is 'PELE (SQL Server 10.0.1442 - PELE\brad)\Databases'. Below the title bar is a search box. The main area displays a table with the following columns: Name, ID, Last Log Backup Date, Size (MB), Data Space Used (KB), and Recovery Model. The data is as follows:

| Name | ID | Last Log Backup Date | Size (MB) | Data Space Used (KB) | Recovery Model |
|--------------------|----|----------------------|-----------|----------------------|----------------|
| MDW | 8 | | 270 | 127152 | Simple |
| AdventureWorks | 7 | | 1070.4375 | 617936 | Simple |
| ReportServer | 5 | 7/1/2008 11:12 AM | 9.375 | 1568 | Full |
| ReportServerTempDB | 6 | | 3 | 568 | Simple |
| System Databases | | | | | |
| Database Snapshots | | | | | |

Figure 8: I have rearranged the screen to better meet my needs and interests. Notice that I have listed the Database ID right after the database name. Now I no longer have to look up the ID whenever I need it.

What if you decide that 36 rows of data are too much to deal with inside of SSMS? One option would be to select all of the rows displayed in the Object Explorer Details screen you want to work with, then press CTRL-C, and all of the data is copied to the clipboard in the tab-delimited format, and then you can paste the results directly into an Excel spreadsheet, and voila, you have an instant report describing the current state of all your databases.

As you can see, the Object Explorer Details window has lots of potential for quickly assessing the status of any of your databases.

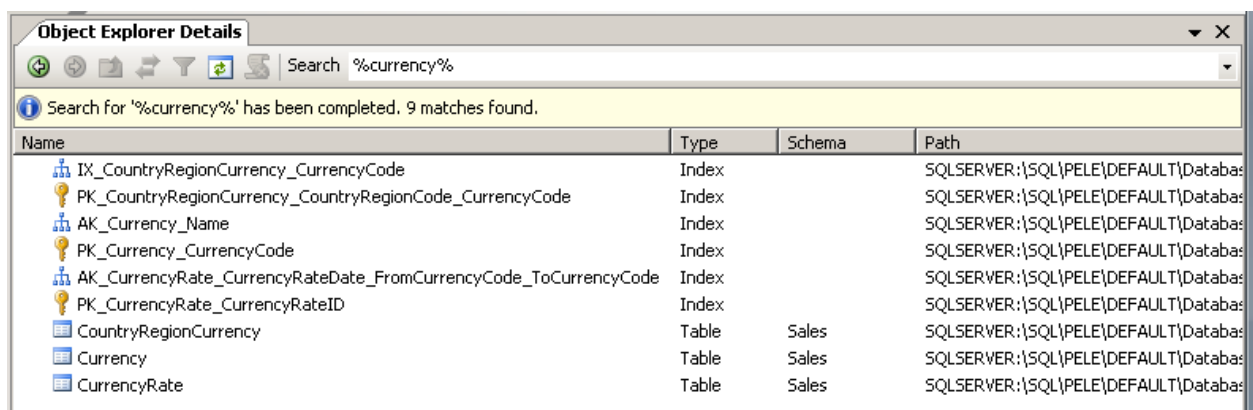
Object Search

If you looked closely at the past several screen shots, you probably noticed the search box shown in **Figure 9**:



Figure 9: I bet you can guess what this search box does.

Technically speaking, the Object Search box is part of the Object Explorer Details window, but it works independently of the window. Let me show you what I mean. Let's say that I want to find all the objects, in all the databases on my SQL Server instance, which include the word "currency". To do this, I would enter into the search box `%currency%`. The percent signs on either side of the word are wildcard symbols, which mean that any object in all my databases that includes the word "currency" will be found, as you can see in **Figure 10**:



The screenshot shows the 'Object Explorer Details' window with the search box containing the text '%currency%'. Below the search box, a message states: 'Search for '%currency%' has been completed. 9 matches found.' The results are displayed in a table with the following columns: Name, Type, Schema, and Path. The data is as follows:

| Name | Type | Schema | Path |
|--|-------|--------|-------------------------------------|
| IX_CountryRegionCurrency_CurrencyCode | Index | | SQLSERVER:\SQL\PELE\DEFAULT\Databas |
| PK_CountryRegionCurrency_CountryRegionCode_CurrencyCode | Index | | SQLSERVER:\SQL\PELE\DEFAULT\Databas |
| AK_Currency_Name | Index | | SQLSERVER:\SQL\PELE\DEFAULT\Databas |
| PK_Currency_CurrencyCode | Index | | SQLSERVER:\SQL\PELE\DEFAULT\Databas |
| AK_CurrencyRate_CurrencyRateDate_FromCurrencyCode_ToCurrencyCode | Index | | SQLSERVER:\SQL\PELE\DEFAULT\Databas |
| PK_CurrencyRate_CurrencyRateID | Index | | SQLSERVER:\SQL\PELE\DEFAULT\Databas |
| CountryRegionCurrency | Table | Sales | SQLSERVER:\SQL\PELE\DEFAULT\Databas |
| Currency | Table | Sales | SQLSERVER:\SQL\PELE\DEFAULT\Databas |
| CurrencyRate | Table | Sales | SQLSERVER:\SQL\PELE\DEFAULT\Databas |

Figure 10: You can search for any object using the Search box.

When the search is complete, all objects that contain the word "currency" are displayed, along with their object type, schema name (if applicable), and the path to the object (most of the path has been cut off from figure 10).

Of course, you don't have to use wildcards if you don't want to, but I wanted to show you the power of this feature. One thing to keep in mind about object searching is that the scope of the search depends on what object has been selected in the Object Explorer. If you want to search all databases, then you need to select the "Databases" object. If you only want to select objects for a specific database, then first select that specific database from Object Explorer, and then perform the search. This way, you only get results from the database you are interested in searching.

In the past, finding objects by name was not intuitive or easy. Now, with Object Search, you can quickly find objects on your SQL Server instances.

Multi-Server Queries

While third-party tools have been offering this feature for years, SSMS now offers the ability to query multiple servers at the same time, returning the results to a single window. This makes it very easy for DBAs to run the same script on multiple SQL Server instances simultaneously.

To accomplish this task, the first step is to create a Server Group from the Registered Servers window, then add SQL Server registrations to the group. Next, right-click on the Server Group and select "New Query." A new Query Editor window appears, where you can enter Transact-SQL code. When you click on "Execute," the query runs on all the registered servers belonging to the group, and returns the results in a single window.

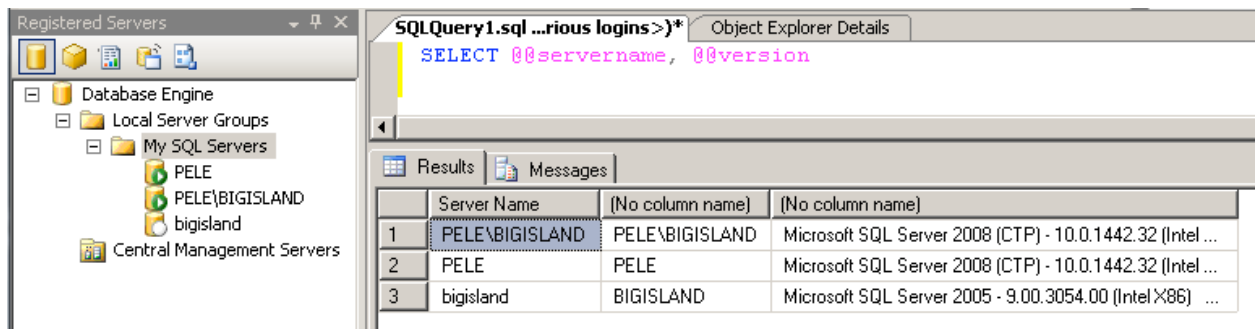


Figure 11: You can now query multiple servers from the same Query Editor window.

In the left-hand pane of figure 11, there is a Server Group named "My SQL Servers" that has three registered servers in it. I right-clicked on the "My SQL Servers" group, and a new Query Editor window opened. I then entered a short query and executed it. As you can see, the names and versions of each of my servers were returned. This feature is backward-compatible with SQL Server 2005.

If you want to query just a selection of SQL Servers, rather than all of the SQL Servers you manage, you will need to create separate Server Groups for each subset, and then add the appropriate servers to the groups. If need be, a SQL Server instance can be a member of more than one Server Group. As a result, you have the facility to create Server Groups, to define sets of servers in any combination that you require, in order to control precisely on which SQL Server instances your queries will execute.

IntelliSense Added to the Query Editor

One of the most requested features for the SSMS Query Editor has been IntelliSense, and it's finally available in SQL Server 2008, although it does not provide all the features you may have become accustomed to when using Visual Studio or third-party add-ins. For example, it only works with SQL Server 2008 and is not backward compatible with SQL Server 2005, nor does it provide advanced formatting capabilities.

While there are some DBAs who have every Transact-SQL statement and parameter memorized, I'm not one of them. This means that I often spend a lot of time in Books Online looking up the syntax of a particular statement. The addition of IntelliSense reduces my need to refer to Books Online, and so helps me to write code faster, and more accurately (the first time).

You don't have to do anything to use IntelliSense in the Query Editor. All you have to do is to begin typing. As you type, IntelliSense can automatically:

- Identify incorrect syntax, underlining it in red so you can immediately identify and correct it.
- Complete a word as you type in a variable, command, or function once you have typed in enough characters so that it is uniquely identified.
- List the available parameters required by a function or stored procedure.
- Open a list that provides available database objects and user-defined variables that you have previously declared.

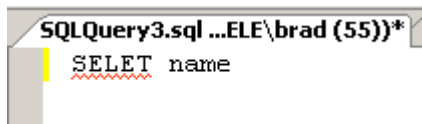


Figure 12: Syntax errors become obvious very quickly when they have a red underline underneath them.

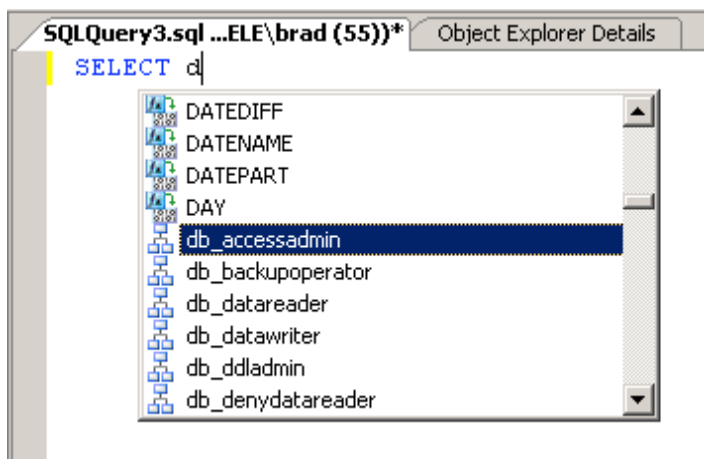


Figure 13: IntelliSense helps you to complete variables, commands, or functions as type them in.

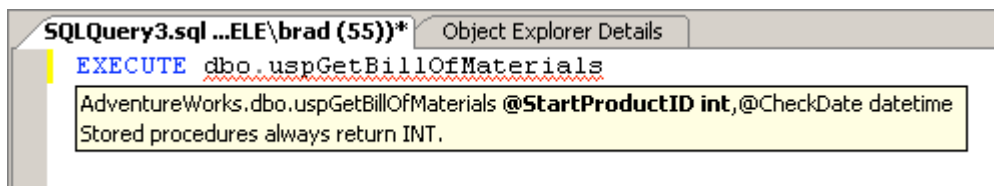


Figure 14: IntelliSense can tell you what parameters are needed for a function or a stored procedure.

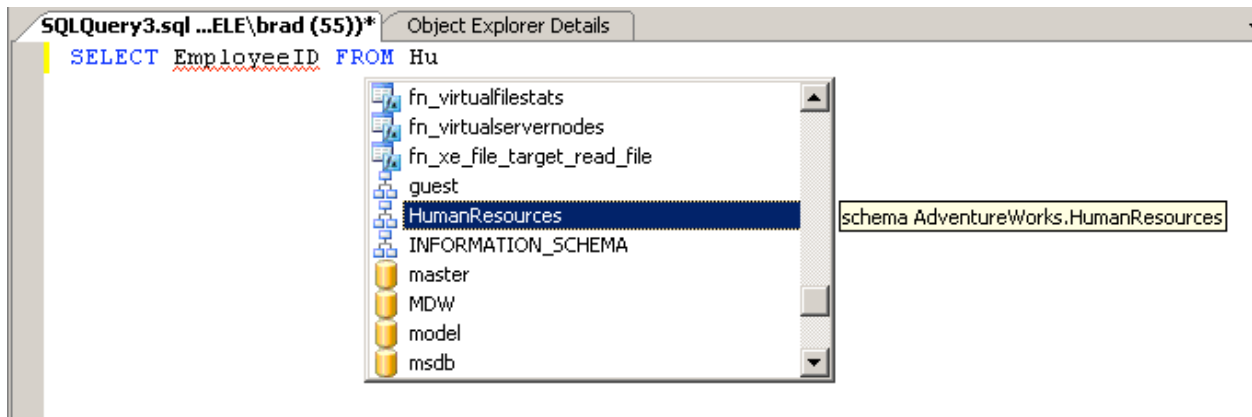


Figure 15: Under some conditions, IntelliSense can provide a list of available objects you can select from.

Using IntelliSense can take a little time to get used to, but once you do, you will find that it can help boost your Transact-SQL coding performance.

T-SQL Debugger Added to the Query Editor

Another feature that has been sorely missing from SQL Server 2005 is a query debugger. In SQL Server 2008, a debugger is now integrated within the Query Editor, making it easy for you to debug your Transact-SQL code.

The debugger includes many features, including the ability to:

- Step through Transact-SQL statements, line by line, or by using breakpoints.
- Step into or over Transact-SQL stored procedures, functions, or triggers that are part of your code.
- Watch values assigned to variables, in addition to viewing system objects, including the call stack and threads.

To use the debugger, all you have to do is to add the code you want to debug to a Query Editor window, then click on the green debug arrow (next to Execute on the Query Editor tool bar), click ALT-F5, or go to the "Debug" menu option and select "Start Debugging." At this point, the Query Editor starts the debugging process, and you can use any of the features previously described to step through your code, looking for potential problems.

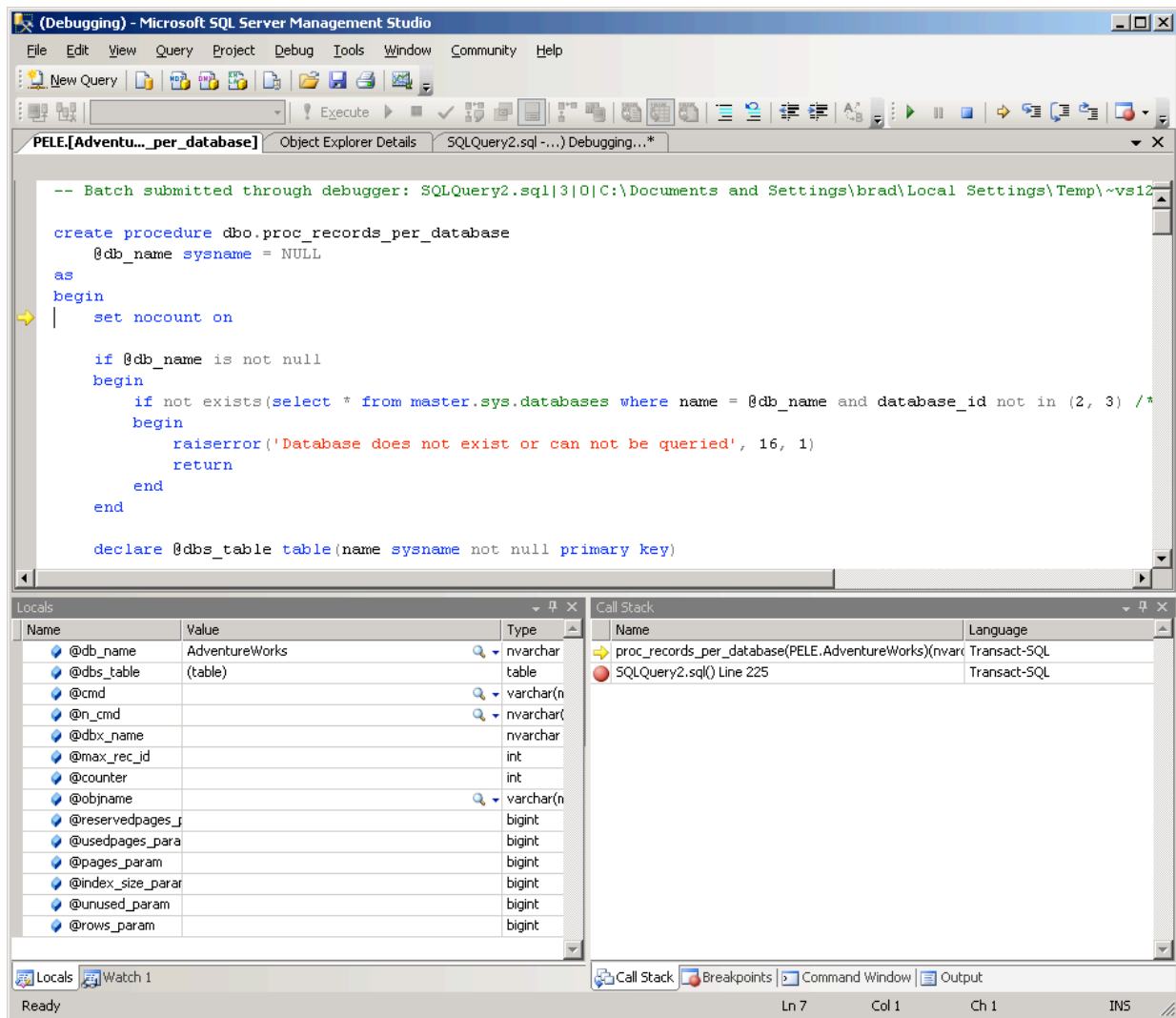


Figure 16: Transact-SQL is currently being debugged.

If you have not used a debugger before, then it might take some time getting used to, as you have the learning curve of understanding exactly how the debugger can be used, besides learning the proper techniques. If you have used a debugger before, you should be able to begin using it immediately, as its features are very straight-forward for the experienced developer.

Keep in mind that the debugger should only be used on test servers, not on production servers. This is because debugging sessions may take some time to complete, and often during debugging sessions, locks are acquired and may be held by the debugging session for long periods of time.

Summary

This ends our brief look at some of the new, cool features added to the SQL Server 2008 SSMS. All of these features are available in all editions of SQL Server 2008 that come with SSMS. As you can imagine, this is just a sampling of the many new improvements made to SSMS. When you first start using SQL Server 2008 SSMS, the first thing you will want to do is to spend extra time learning about all these features, and the many more that I did not have time to discuss. By investing time upfront, you will more quickly be able to master the new features, making you that much more productive right at the start.

CHAPTER 2: POLICY-BASED MANAGEMENT

Traditionally, SQL Server instances have been treated as "islands", each one configured and managed separately. For example, if you have 25 SQL Server instances to manage:

- Ten instances might be using Windows Authentication and the other 15 are using Mixed Authentication.
- Database Auto Shrink might be turned on for five instances and turned off for the rest.
- SQLMail might be enabled for 15 instances, Database Mail might be enabled for five instances, and the rest may not have mail enabled.
- Each database may have its own object naming standard, depending upon who designed the database.

These sorts of inconsistencies (and the list could go on) can be maddening for DBAs who have to "manage" separate instances differently. To make their lives easier, many DBAs try to develop and enforce common, enterprise-wide standards for the SQL Server instances they manage, as they recognize the benefits of standardization. The difficulty has been that even if policies are developed, there has been no easy way to enforce them (unless the DBA is fortunate enough to be the only one ever touching an instance), other than manually checking each instance for compliance on a regular basis, and what DBA has time to do this?

New to SQL Server 2008 is Policy-Based Management, an innovative feature that can help DBAs define and enforce policies (standards) for the SQL Server instances throughout their organization. It is available in both the Standard and Enterprise Editions of SQL Server 2008. While it may not offer DBA nirvana, it is a great first step toward more efficient enterprise-wide SQL Server management.

What Policy-Based Management Does

Essentially, this is what Policy-Based Management offers DBAs:

- **The Ability to Define Standard Policies**
Defining policies allows the DBA to proactively choose how SQL Server instances, databases, and objects are configured. These policies allow the senior DBA (or whoever else is responsible for internal standards) to establish standards that others have to follow when they interact with the organization's SQL Servers, ensuring that consistency is maintained. The DBA can choose to create as few or as many policies as are needed to accomplish the required level of consistency.
- **The Ability to Selectively Enforce Policies**
Once policies have been defined, the DBA can choose on which instances, database, or database objects to enforce the policies. Policy-based management is very granular, allowing the DBA to create policies that meet special circumstances. For example, if the DBA needs to exempt a particular instance, database, or database object from a policy, this is no problem.
- **The Ability to Automate Policy Checking and Enforcement**
Once a policy has been defined and then assigned at the correct level (instance, database, or object), the DBA has several different ways to enforce compliance with the designated policies. For example, he may decide that he only wants to know about out-of-compliance policies. In other cases, he may want to prevent out-of-compliance policies from occurring in the first place. We will learn more about enforcement options a little later in this chapter.
- **The Ability to Fix Out of Policy Conditions with the Click of a Button**

In some cases, if you find that a policy is out of compliance, you can click a button and force the out-of-policy condition to be fixed. For example, if you use Policy-Based Management to determine if all of your databases have Auto Update Statistics on, and you find that some databases have this option turned off, with the click of a button, you can turn Auto Update Statistics on for all of the databases out of compliance with your policy. This has the potential of being a great time saver.

Now that we know a little about Policy-Based Management, let's learn some more about the kinds of policies that you can create.

How You Might Use Policy-Based Management

After you install SQL Server 2008, there are no policies created for you. That's because, as the DBA, it is our job to decide what policies we want to create and enforce for our SQL Servers. Fortunately, Microsoft has not left us completely out in the cold. Although hidden, Microsoft has included a number of sample policies, stored as XML files in our SQL Server installation, which we can review and learn from. If we like them, we can choose to import them into our SQL Server instances and put them to work. In addition, we can use the policies as examples on which to model our own policies. In any event, it is valuable to review the included policies so we can better appreciate what Policy-based Management has to offer.

These policies are hidden away in this folder in your SQL Server installation:

```
Drive_letter:\Program Files\Microsoft SQL Server\100\Tools\Policies\DatabaseEngine\1033
```

You can import these sample policies (you can delete them later if you don't want to use them) from SSMS by navigating to Management, opening up Policy-Based Management, right-clicking on "Policies", then selecting "Import Policy". At this point, select all the policies in the folder described above, and click "OK". Some of the available policies are shown in **Figure 1**:

| Name | Category | State |
|---|--|-------|
| SQL Server Default Trace | Microsoft Best Practices: Audit | False |
| SQL Server System Tables Updatable | Microsoft Best Practices: Configuration | False |
| Database Page Verification | Microsoft Best Practices: Maintenance | False |
| Last Successful Backup Date | Microsoft Best Practices: Maintenance | False |
| Read-only Database Recovery Model | Microsoft Best Practices: Maintenance | False |
| Database Page Status | Microsoft Best Practices: Maintenance | False |
| Backup and Data File Location | Microsoft Best Practices: Maintenance | False |
| SQL Server I/O Affinity Mask For Non-enterprise SQL Servers | Microsoft Best Practices: Performance | False |
| SQL Server Dynamic Locks | Microsoft Best Practices: Performance | False |
| SQL Server Blocked Process Threshold | Microsoft Best Practices: Performance | False |
| SQL Server Max Worker Threads for 64-bit SQL Server 2000 | Microsoft Best Practices: Performance | False |
| SQL Server Max Worker Threads for SQL Server 2005 and above | Microsoft Best Practices: Performance | False |
| SQL Server Network Packet Size | Microsoft Best Practices: Performance | False |
| SQL Server Lightweight Pooling | Microsoft Best Practices: Performance | False |
| SQL Server Open Objects for SQL Server 2000 | Microsoft Best Practices: Performance | False |
| SQL Server Max Degree of Parallelism | Microsoft Best Practices: Performance | False |
| SQL Server Affinity Mask | Microsoft Best Practices: Performance | False |
| Database Auto Shrink | Microsoft Best Practices: Performance | False |
| Database Collation | Microsoft Best Practices: Performance | False |
| Database Auto Close | Microsoft Best Practices: Performance | False |
| SQL Server Max Worker Threads for 32-bit SQL Server 2000 | Microsoft Best Practices: Performance | False |
| Data and Log File Location | Microsoft Best Practices: Performance | False |
| SQL Server 32-bit Affinity Mask Overlap | Microsoft Best Practices: Performance | False |
| File Growth for SQL Server 2000 | Microsoft Best Practices: Performance | False |
| SQL Server 64-bit Affinity Mask Overlap | Microsoft Best Practices: Performance | False |
| SQL Server Password Policy | Microsoft Best Practices: Security | False |
| Trustworthy Database | Microsoft Best Practices: Security | False |
| Symmetric Key Encryption for User Databases | Microsoft Best Practices: Security | False |
| Symmetric Key for System Databases | Microsoft Best Practices: Security | False |
| Symmetric Key for master Database | Microsoft Best Practices: Security | False |
| Public Not Granted Server Permissions | Microsoft Best Practices: Security | False |
| SQL Server Login Mode | Microsoft Best Practices: Security | False |
| CmdExec Rights Secured | Microsoft Best Practices: Security | False |
| Guest Permissions | Microsoft Best Practices: Security | False |
| SQL Server Password Expiration | Microsoft Best Practices: Security | False |
| Asymmetric Key Encryption Algorithm | Microsoft Best Practices: Security | False |
| Windows Event Log Disk Defragmentation | Microsoft Best Practices: Windows Log File | False |

Figure 1: This is a partial list of the sample policies that can have been included with the SQL Server 2008 installation.

The first thing you should notice when you look at this list is that you can create categories of policies to make them easier to manage. Within each category are one or more policies that the DBA can choose to enforce on their SQL Server instances. For example, you can create policies to enforce:

- "Database Page Verification" on databases
- "Database Auto Close" is turned on or off for your database
- "SQL Server Password Policy" for your SQL Server instances.

Of course, this list is not exhaustive. DBAs can also include in their policies checks of collation, or for use of full recovery model without transaction log backups, and many other "safety catches" that could save them a lot of trouble further down the line.

The aim of Policy-Based Management is to make it easy for the DBA to formulate most any policy required for their SQL Servers, then to enforce this standard as deemed necessary.

How Policy-Based Management Works

Up until this point, I have used the term "policy" interchangeably with "Policy-based Management" to keep things simple. In this section, we will take a more detailed look at how Policy-Based Management works. In fact, Policy-Based Management includes four major steps, each of which you need to understand before you implement it on your SQL Servers:

1. **Selecting Facets:** Before you create a policy, the first step is to select a Policy-Based Management **facet** and configure its properties. A facet is a collection of pre-defined properties that describe some functionality of SQL Server. For example, some common facets are Audit, Database Maintenance, Database Options, Database Performance, Server, and Stored Procedure. There are a total of 74 facets available in SQL Server 2008. Each of these facets has one or more properties. For example, the Database Options facet has 45 different properties. Some of them include AutoClose, AutoShrink, AutoUpdateStatisticsEnabled, and PageVerify. When it really comes down to it, think of a facet (and its properties) as something inside of SQL Server that is configurable. Facets and properties are all predefined by SQL Server 2008.
2. **Setting Conditions.** Once you have selected a property of a facet that you want to create a policy for, the next step is to create a property **condition** that specifies what *state* you want the property of the facet to have. In other words, SQL Server has no idea what state you want a particular property of a facet to have, so you must specify this state. For example, if you want the AutoClose property of the Database Options facet set to "false," you must create a condition that specifies this.
3. **Creating and evaluating Policies:** Now that you have created a condition that specifies the state you want a facet's property to have, you now create an actual **policy**. In this context, a policy is used to specify the condition you just created, the **targets** that are to be evaluated with the condition, and its **evaluation mode**. A target can be a SQL Server instance, database, or database object. Evaluation mode refers to how you want the policy to be evaluated. By evaluated, this means that the condition you specify is compared to the actual setting of the target you specify. If your condition matches the actual setting of the target, the policy evaluates to be true, which is your goal. If your condition does not match the actual setting of the target, the policy evaluates to false, which means that your policy is not in compliance with the condition you have established. There are four options for evaluation mode:
 - a) **On Demand:** This means that you will evaluate your policies whenever you want.
 - b) **On Schedule:** This means that your policies will be evaluated on a predefined schedule you create.
 - c) **On Change Prevent:** This means that if someone makes a change that will cause a policy to evaluate to false, that you want to prevent this action from being taken. This option is only available on activities that can be rolled back.
 - d) **On Change Log Only:** This is like On Change Prevent, but instead of preventing the action, it allows the action, but logs the fact that the out-of-policy action occurred.

To summarize, when you create a policy, you are specifying a combination of a condition, target, and evaluation mode, all in the same step.

4. **Executing the Policy:** The last step is to actually execute a policy and see the results. If the evaluation mode of the policy was On Demand, then when you run the policy, you get a report back on what targets met or failed the policy. If the evaluation mode was On Schedule, then this evaluation occurs at a predetermined schedule. If the evaluation mode was On Change

Prevent, then whenever someone does some action that does not match the policy's condition, then the action is prevented. If the evaluation mode is On Change Log Only, then if someone does some action that doesn't match policy, then the action is permitted, but the fact that it was out of compliance is logged in the event log.

This is a lot of material to absorb. To make it easier, let's look at how we implement all four of these steps in a simple example.

How to Implement a Simple Policy

Policy-Based Management is generally implemented using SQL Server Management Studio (SSMS), although, if you prefer a more automated or programmatic approach, it can also be implemented using PowerShell scripts. We will use SSMS to implement our example policy.

For our example, consider that you, as the DBA, want to create a simple policy that specifies that all the SQL Server instances in your organization should have their server authentication method set to *Windows Authentication mode*. On the other hand, while you want this policy enforced, you also realize that there might need to be some exceptions. For example, not all SQL Server-based applications use Windows Authentication and some require SQL Server authentication instead.

Creating and implementing a policy is a four step process, as described previously, and each step is outlined in the following sections.

Step 1: Selecting Facets

As I mentioned earlier, Policy-Based Management allows the DBA to create policies on 74 different facets of SQL Server, and each facet has one or more properties. In fact, there are hundreds of properties, and when you first begin creating your own policies, one of the difficulties you will have is figuring out which property(s) of which facet(s) describes the feature of SQL Server on which you want to create a policy. Perhaps the easiest way to get started is to open up the Facets folder under Policy Management, and scroll through them, as you see in Figure 2 below.

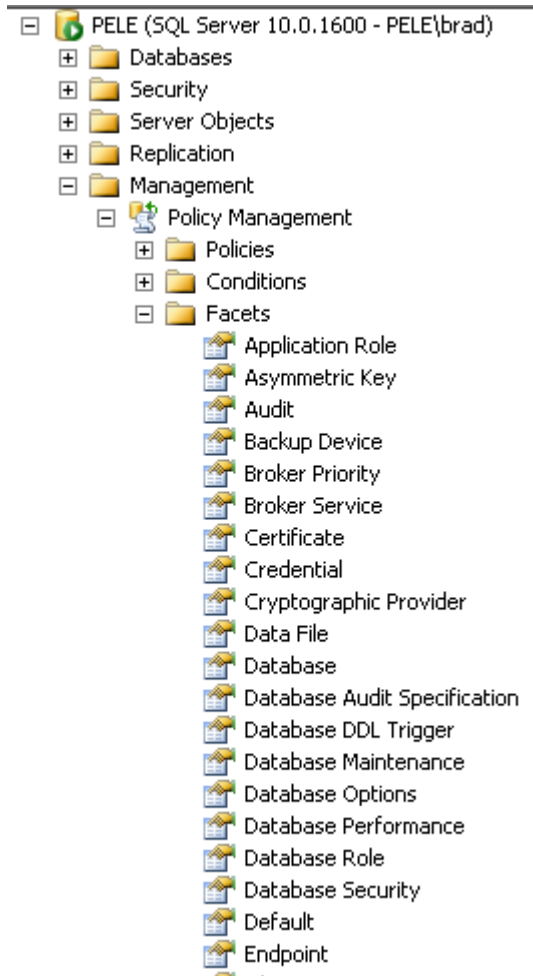


Figure 2: Above are some of the 74 facets available in SQL Server 2008.

Once you find a facet that seems to describe what you are looking for, right-click on it and select "Properties". This will list all of the properties, and hopefully you will find what you are looking for right away. If not, you may have to keep on trying until you find the correct facet and property that meets your needs.

For our example, we will be using the **Server Security** facet, which has nine different properties (see Figure 3 below), one of which is **LoginMode**, which is used to describe the authentication mode used by SQL Server.

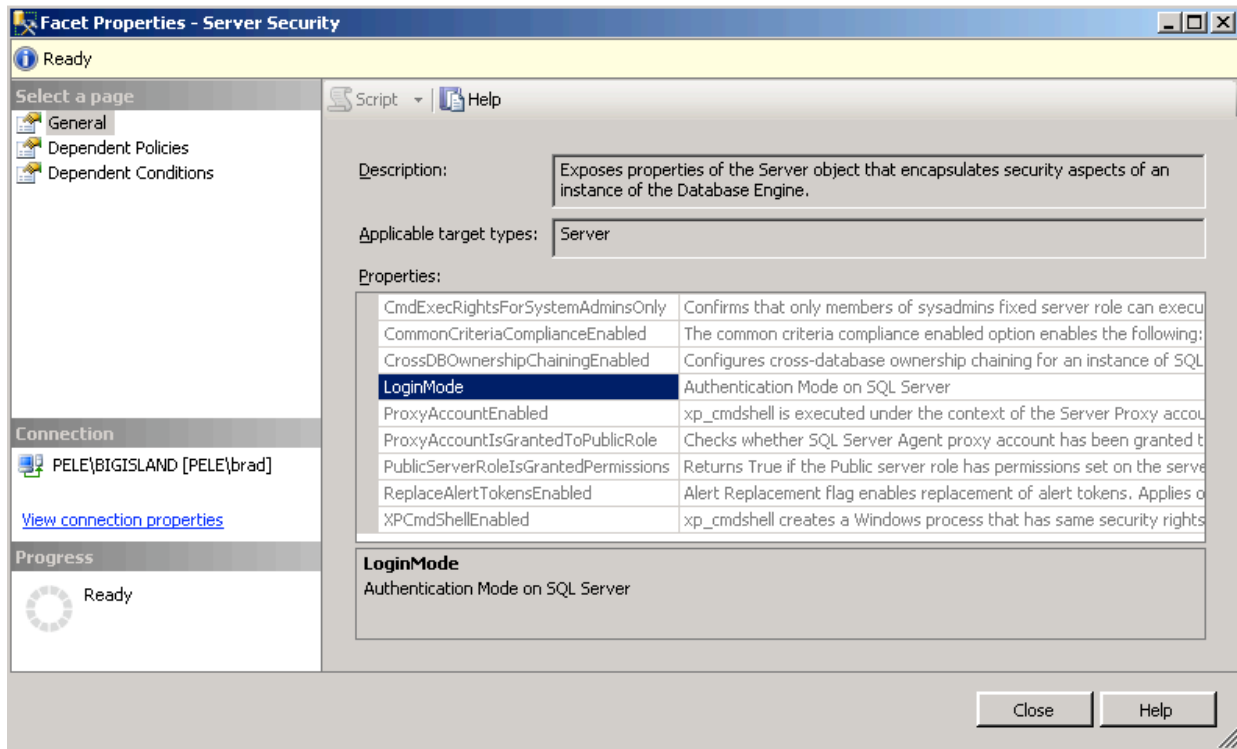


Figure 3: Each of the 74 built-in SQL Server facets has multiple properties you can create policies on.

Once you have identified the facet and property you want to create a policy on, the next step is to create the property condition.

Step 2: Setting the required Property Conditions

Once a facet (and its property) has been selected as the basis for a policy, the next step is to define a logical condition that specifies the desired state of the selected property. The LoginMode property has four different states: **Normal**, **Integrated**, **Mixed**, and **Unknown** (all of the available states are listed for us in the "Create New Condition" dialog box, so all we have to do is to select the one we want to use). These states of the LoginMode property determine which authentication mode a particular SQL Server instance might have. In our case, we want our policy to enforce Windows authentication, so the LoginMode property should have a state of **Integrated**.

To create this condition, right-click on the "Server Security" facet and select "New Condition". The "Create New Condition" dialog box appears, as shown in **Figure 4**:

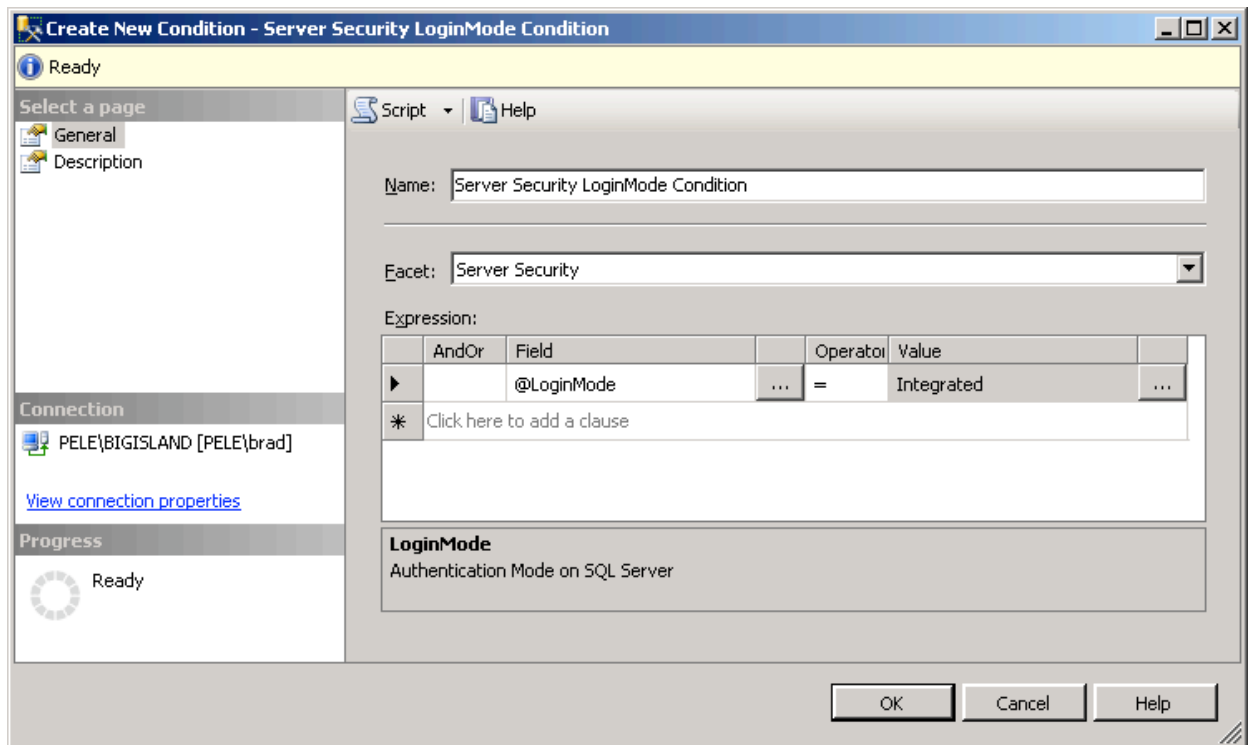


Figure 4: After selecting the correct facet, the next step is to create a logical condition defining what we want to test for.

Creating a condition involves several steps. First, we have to give the condition a name. In this case, I have used the name "Server Security LoginMode Condition".

Next, from the "Field" column for our expression, we need to select **@LoginMode**, which is the property of the Server Security facet we want to test for. This property is available from a drop-down box under "Field," as are all of the properties for this facet.

Now we must select an operator, which is used to test our condition. In this case, we select the "=" operator because we want the **@LoginMode** Field to equal a value that represents integrated security. The drop-down box, that includes the "=" sign, includes seven additional operators you can choose when creating a condition.

The last step is to select the state we want to test for. Here, we select the **Integrated** state of the **@LoginMode** because this is what we are testing for. What, in effect, we are doing here is creating an expression like this:

```
@LoginMode = Integrated
```

By clicking on the OK button, this condition is saved. Next, when we create the actual policy, it will use this condition to evaluate if the **@LoginMode** state of a particular SQL Server instance is Integrated or not. A return value of "true" indicates that the server is configured to use Windows Authentication. If the condition returns a "false," then we know that the SQL Server instance is not using Windows Authentication.

Step 3: Creating the Policy

Once the condition for the facet has been created, our next step is to create the policy. To do this, we again right-click on the "Server Security" facet, but this time we select "New Policy". The dialog box shown in Figure 5 appears.

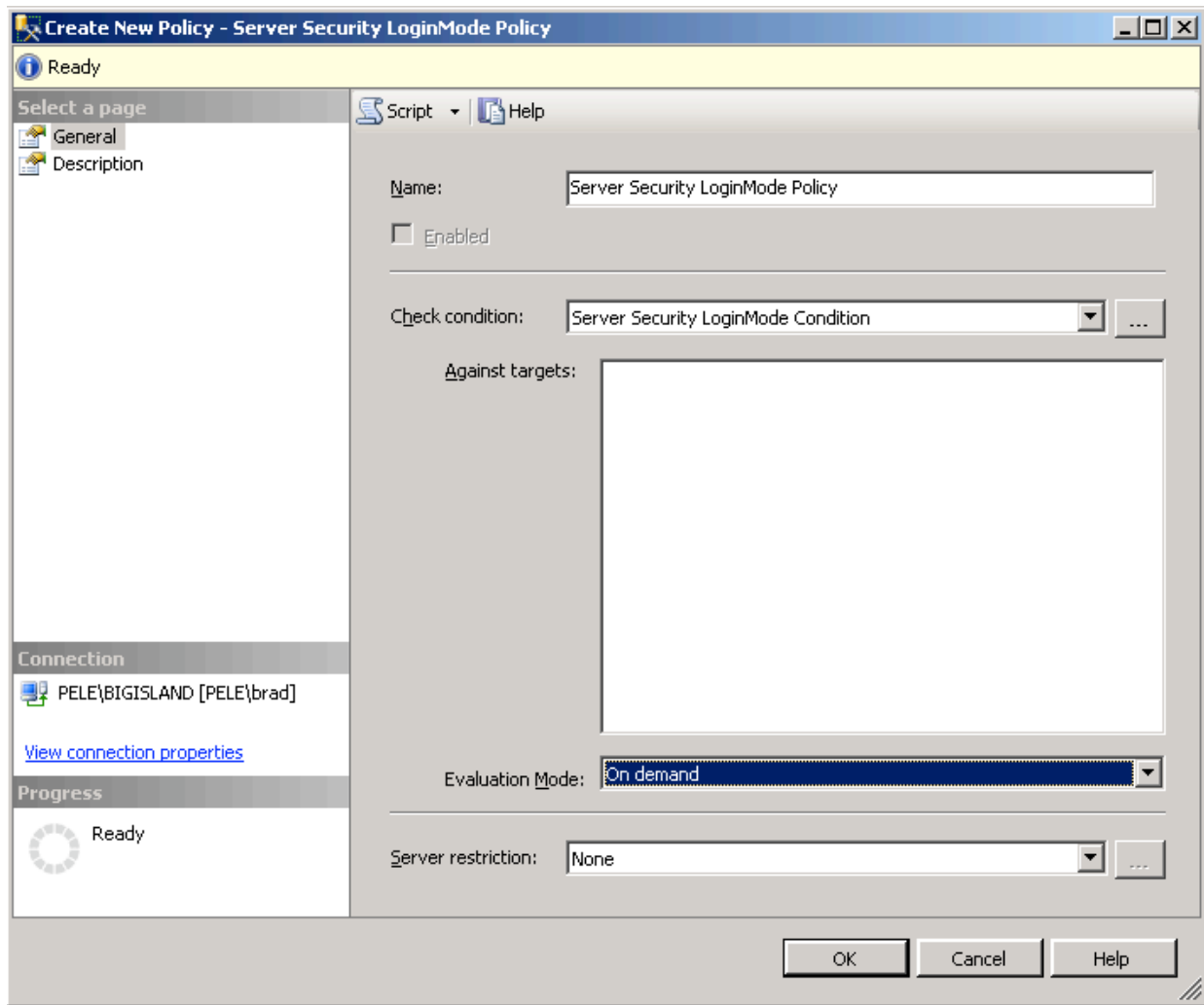


Figure 5: Creating a new policy is simply a matter of selecting the appropriate options.

At first, this screen might seem like it involves a lot of work, but it is actually very straight-forward. The first step is to give the policy a name. In this example, the name is "Server Security LoginMode Policy".

The next step is the most important. Next to "Check Condition" we select the condition we created in the previous step. It will be available from the drop-down box, so you don't have to remember how to type it in.

The **Against Targets** option on this screen is not active because the facet and property we selected is at the instance level. If the facet and property were at the database or object level, then we would be able to select from the appropriate database or objects here.

The next step is to select which type of Evaluation Mode we want to use for this policy. The available options for this particular facet include "On Demand" or "On Schedule." While Policy-Based Management offers four different ways to evaluate a policy (mentioned earlier), only two are available for this particular facet and property. The available Evaluation Modes depend on the particular facet and property you are working with. For this example, we are going to choose "On Demand."

We have one last step, and that is to set the "Server Restriction" option. The default choice is "None", which means that the policy can be run against any SQL Server instance. Alternatively, it can be set such that the policy is only run on specific SQL Server instances. For this particular policy,

it makes sense to allow it to be run on all servers, so we accept the default option. Once we are done, we click OK, and the policy is saved.

Step 4: Running the Policy

We are now done creating our policy, and the only thing left to do is to run it on demand to see if a particular SQL Server instance is in or out of compliance with the policy. There are several different ways to run a policy on demand. In our example, we will run this policy on demand by going to the specific policy, right-clicking on it, and then choosing "Evaluate", as shown in Figure 6. This runs the policy on demand on the local SQL Server instance and returns the results, letting us know whether or not this particular instance is in compliance with our policy.

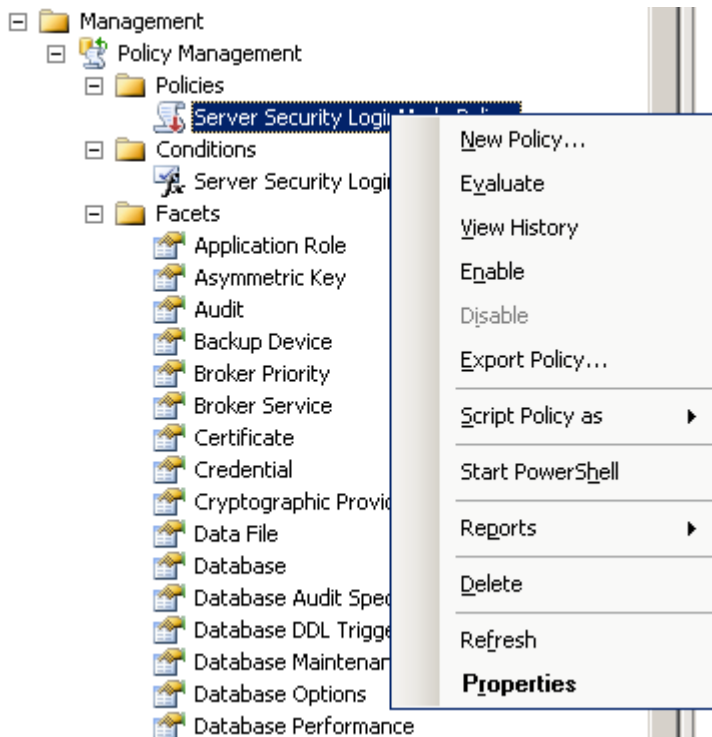


Figure 6: One way to run a policy on demand is to "Evaluate" it.

After clicking "Evaluate", the "Evaluate Policies" dialog box appears, as shown in **Figure 7:**

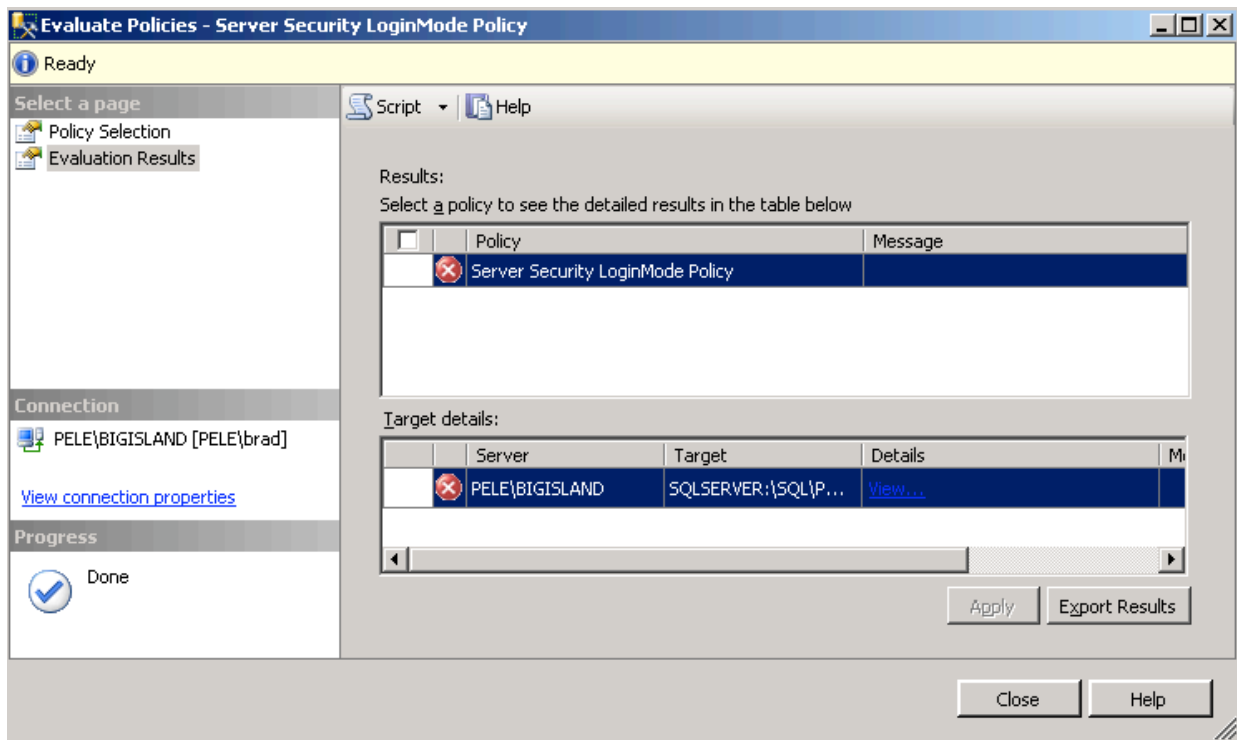


Figure 7: The policy has been run on demand, evaluated, and now we see a red icon, which indicates that the server we just evaluated for this policy failed the evaluation.

The top part of the box shows us what policy was evaluated. In this case, only one policy was evaluated, and it was the "Server Security LoginMode Policy" we previously created. Note that there is a red icon next to it. This indicates that one or more targets (in our case, a target is a SQL Server instance) is out of compliance with the policy (compliance is indicated by a green icon).

To find out why, we can click on the "View" link under "Details." Note, the color scheme used makes this difficult to see in figure 7. When you click on "View," the screen shown Figure 8 appears:

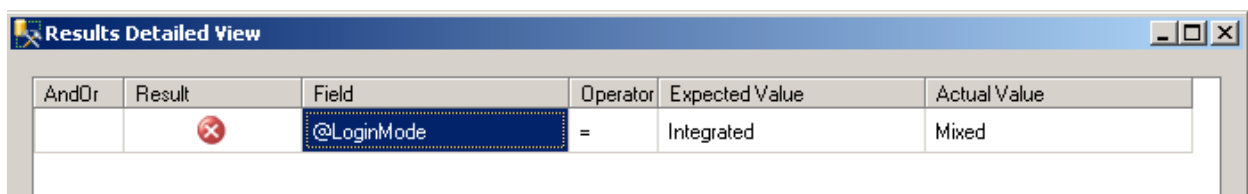


Figure 8: When we view the details of the policy's results, we can find out why the policy shows that the server that was evaluated was not in compliance with the policy.

The "Results Detailed View" screen shows us what was tested. The @LoginMode property of the Server Security facet is supposed to be equal to "Integrated," but that it wasn't, as the actual value was "Mixed."

Now that we know that our SQL Server instance is out of compliance with our policy, what do we do? As the DBA, we have to decide if it is critical that this instance be in compliance or not. If it should be in compliance, then we can make the necessary change and bring it into compliance, or we can decide that this instance is an exception and should remain out of compliance.

You might be asking, can't I press a button and have the non-compliant server made compliant with the policy? Not in this case. Only some facets and properties can be automatically brought into compliance with a policy. Our example is one of the cases where this option is not available.

That's it, we are now done. We have created a policy, we ran it against a SQL Server instance, and now we know that our server is out of compliance. At this point, we have only evaluated a single server to see if it is in compliance with our policy. If we have more than one server, how do we check for compliance?

Checking Multiple Servers for Compliance

Checking multiple servers for compliance with a policy involves several additional steps, which we will describe briefly here.

1. Create, deploy, and test a policy on a designated instance of SQL Server, just as we did above.
2. Export the policy from the instance as an XML file by right-clicking on the Policy and selecting "Export Policy".
3. Using SSMS, create a local server group, or a central management server group, that includes all the SQL Server instances on which you want to apply the policy. For this example, there are only two SQL Server instances in our server group.
4. Right-click on the server group, and then select "Evaluate Policies." This causes the "Evaluate Policies" dialog box to appear, as seen in figure 9. The next step is to import the XML policy file that you exported in step 2 above. You do this by clicking on the "Browse" button next to "Source", pointing to the XML file, and then opening it.

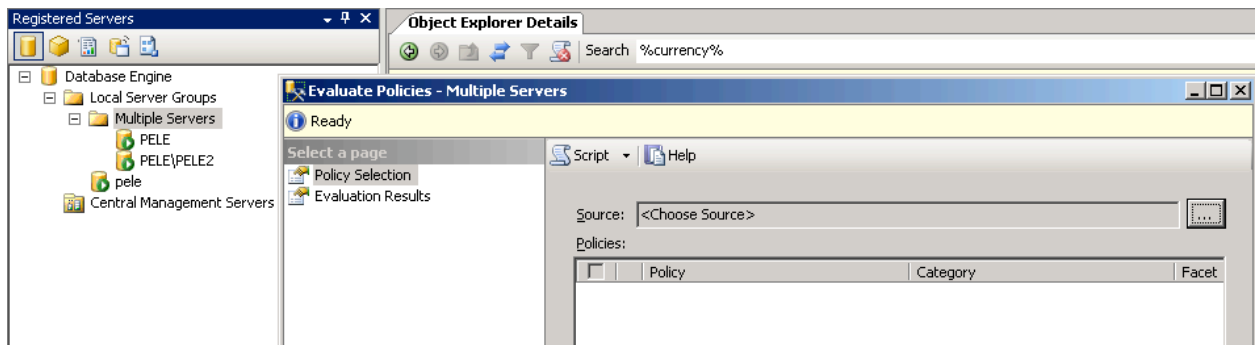


Figure 9: Use the above dialog box to select the XML file of the policy you want to evaluate.

Once the XML policy file has been loaded, click on the "Evaluate" button, and then the results are immediately displayed in the "Evaluate Policies" dialog box, as show in **Figure 10:**

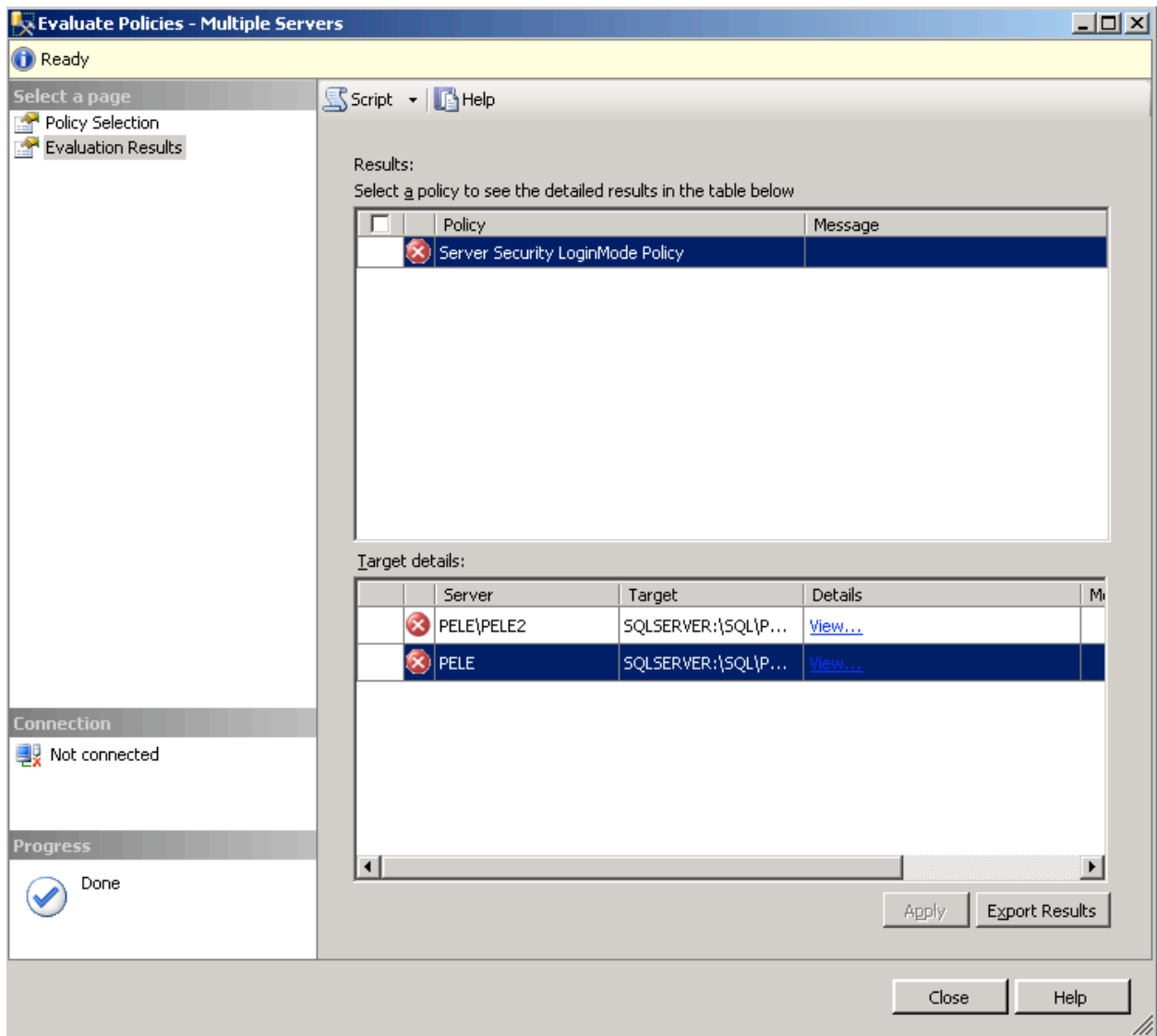


Figure 10: The "Evaluate Policies" dialog box looks like the one we saw in the previous example, except that now it shows the results of two servers.

In this example, we evaluated only one policy, and both of the servers in our server group are out of compliance with that policy. As you can quite well imagine, if our server group had 100 servers in it, this would indeed be quite a powerful demonstration of how Policy-Based Management can be used by a DBA to quickly determine if the servers he controls are in or out of compliance with the organization's policies.

Summary

In this chapter, you have learned that Policy-Based Management is designed to accomplish the following goals:

- Help DBAs to define standard policies for an organization's SQL Server instances.
- Help DBAs to granularly select what instances, databases, and objects are to be affected by policies.
- Helps DBA to enforce established policies.
- Helps DBAs to correct out of compliance policies.

At this point, you should have a basic understanding of what Policy-Based Management can do for you. Just bear in mind that Policy-Based Management is a huge topic, with lots of nuances and implications that have not been covered here. Because of this, I highly recommend that it only be implemented by experienced DBAs.

CHAPTER 3: DATA COMPRESSION

There is one thing every DBA knows with certainty, and that is that databases grow with time. MDFs grow, backups grow, and it never stops. The more data we have, the more work SQL Server has to perform in order to deal with it all; whether it's executing a query on a table with 10 million rows, or backing up a 5 TB database. Whether we like it or not, we are fighting a losing battle, and DBA's can't reverse the information explosion. Or can we?

While we can't stop growth, SQL Server 2008 (Enterprise Edition only), gives us some new tools to help us better deal with all this data, and that is the promise of compression. Given the right circumstances, DBAs can use data compression to reduce the size of our MDFs, and backup compression can help us reduce the amount of space our backups take. Not only does compression reduce physical file sizes, it reduces disk I/O, which can greatly enhance the performance of many database applications, along with database backups.

When we discuss SQL Server compression, we need to think of it two different ways. First, there is *data compression*, which includes row-level and page-level compression that occurs within the MDF files of our databases. Second, there is *backup compression*, which occurs only when data is backed up. While both of these are forms of compression, they are architected differently. Because of this, it is important to treat them separately.

Data Compression comes in two different forms:

- **Row-level Data Compression:** Row-level data compression is essentially turning fixed length data types into variable length data types, freeing up empty space. It also has the ability to ignore zero and null values, saving additional space. In turn, more rows can fit into a single data page.
- **Page-level Data Compression:** Page-level data compression starts with row-level data compression, then adds two additional compression features: prefix and dictionary compression. We will take a look at what this means a little later in this chapter. As you can imagine, page-level compression offers increased data compression over row-level compression alone.

Backup Compression comes in a single form:

- **Backup Compression:** Backup compression does not use row-level or page-level data compression. Instead, backup compression occurs only at the time of a backup, and it uses its own proprietary compression technique. Backup compression can be used when using, or not using, data compression, although using backup compression on a database that is already compressed using data compression may not offer additional benefits.

In the next section, we will take a high-level overview of data compression, and then we will drill down into the detail of the different types of compression available with SQL Server 2008.

Data Compression Overview

Data compression has been around for years. For example, who hasn't zipped a file at some point in their career? While compression isn't a new technology, it's new to SQL Server. Unlike zip compression, SQL Server's data compression does not automatically compress an entire database; instead, data compression can only be used for these database objects:

- A table stored as a heap
- A table stored as a clustered index
- A non-clustered index
- An indexed view
- Partitioned tables and indexes

In other words, as the DBA, you must evaluate each of the above objects in your database, decide if you want to compress it, then decide whether you want to compress it using either row-level or page-level compression. Once you have completed this evaluation, then you must turn on compression for that object. There is no single switch you can flip that will turn compression on or off for all the objects listed above, although you could write a Transact-SQL script to accomplish this task.

Fortunately, other than turning compression on or off for the above objects, you don't have to do anything else to enable compression. You don't have to re-architect your database or your application, as data compression is entirely handled under the covers by the SQL Server Storage Engine. When data is passed to the Storage Engine, it is compressed and stored in the designated compressed format (on disk and in the Buffer Cache). When the Storage Engine passes the information to another component of SQL Server, then the Storage Engine has to uncompress it. In other words, every time data has to be passed to or from the Storage Engine, it has to be compressed or uncompressed. While this does take extra CPU overhead to accomplish, in many cases, the amount of disk I/O saved by compression more than makes up for the CPU costs, boosting the overall performance of SQL Server.

Here's a simplified example. Let's say that we want to update a row in a table, and that the row we want to update is currently stored on disk in a table that is using row-level data compression. When we execute the UPDATE statement, the Relational Engine (Query Processor) parses, compiles, and optimizes the UPDATE statement, ready to execute it. Before the statement can be executed, the Relational Engine needs the row of data that is currently stored on disk in the compressed format, so the Relational Engine requests the data by asking the Storage Engine to go get it. The Storage Engine (with the help of the SQLOS) goes and gets the compressed data from disk and brings it into the Buffer Cache, where the data continues to remain in its compressed format.

Once the data is in the Buffer Cache, the row is handed off to the Relational Engine from the Storage Engine. During this pass off, the compressed row is uncompressed and given to the Relational Engine to UPDATE. Once the row has been updated, it is then passed back to the Storage Engine, where it is again compressed and stored in the Buffer Cache. At some point, the row will be flushed to disk, where it is stored on disk in its compressed format.

Data compression offers many benefits. Besides the obvious one of reducing the amount of physical disk space required to store data—and the disk I/O needed to write and read it—it also reduces the amount of Buffer Cache memory needed to store data in the Buffer Cache. This in turn allows more data to be stored in the Buffer Cache, reducing the need for SQL Server to access the disk to get data, as the data is now more likely to be in memory than disk, further reducing disk I/O.

Just as data compression offers benefits, so it has some disadvantages. Using compression uses up additional CPU cycles. If your server has plenty to spare, then you have no problem. But if your server is already experiencing a CPU bottleneck, then perhaps compression is better left turned off.

Row-Level Data Compression

The simplest method of data compression, row-level compression, works by:

- Reducing the amount of metadata used to store a row.

- Storing fixed length numeric data types as if they were variable-length data types. For example, if you store the value 1 in a bigint data type, storage will only take 1 byte, not 8 bytes, which the bigint data types normally takes.
- Storing CHAR data types as variable-length data types. For example, if you have a CHAR (100) data type, and only store 10 characters in it, blank characters are not stored, thus reducing the space needed to store data.
- Not storing NULL or 0 values

Row-level data compression offers less compression than page-level data compression, but it also incurs less overhead, reducing the amount of CPU resources required to implement it.

Page Level Data Compression

Page-level data compression offers greater compression, but at the expense of greater CPU utilization. It works using these techniques:

- It starts out by using row-level data compression to get as many rows as it can on a single page.
- Next, prefix compression is run. Essentially, repeating patterns of data at the beginning of the values of a given column are removed and substituted with an abbreviated reference that is stored in the compression information (CI) structure that immediately follows the page header of a data page.
- And last, dictionary compression is used. Dictionary compression searches for repeated values anywhere on a page and stores them in the CI. One of the major differences between prefix and dictionary compression is that prefix compression is restricted to one column, while dictionary compression works anywhere on a data page.

The amount of compression provided by page-level data compression is highly dependent on the data stored in a table or index. If a lot of the data repeats itself, then compression is more efficient. If the data is more random, then little benefits can be gained using page-level compression.

Data Compression Demo

Data compression can be performed using either SQL Server Management Studio (SSMS) or by using Transact-SQL. For this demo, we will take a look at how you can compress a table that uses a clustered index, using SSMS.

Let's say that we want to compress the Sales.Customer table (which has a clustered index) in the AdventureWorks database. The first step is to right-click on the table in SSMS, select "Storage," and then select "Manage Compression."

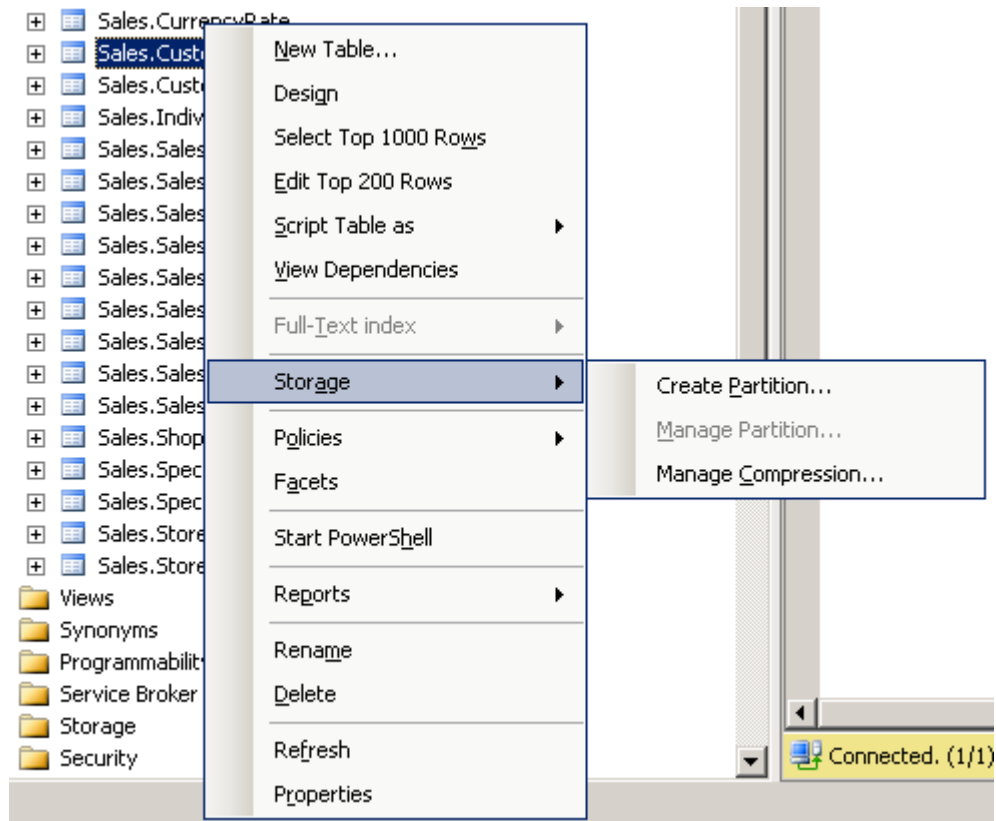


Figure 1: SSMS can be used to manage compression.

This brings up the Data Compression Wizard, displayed below.

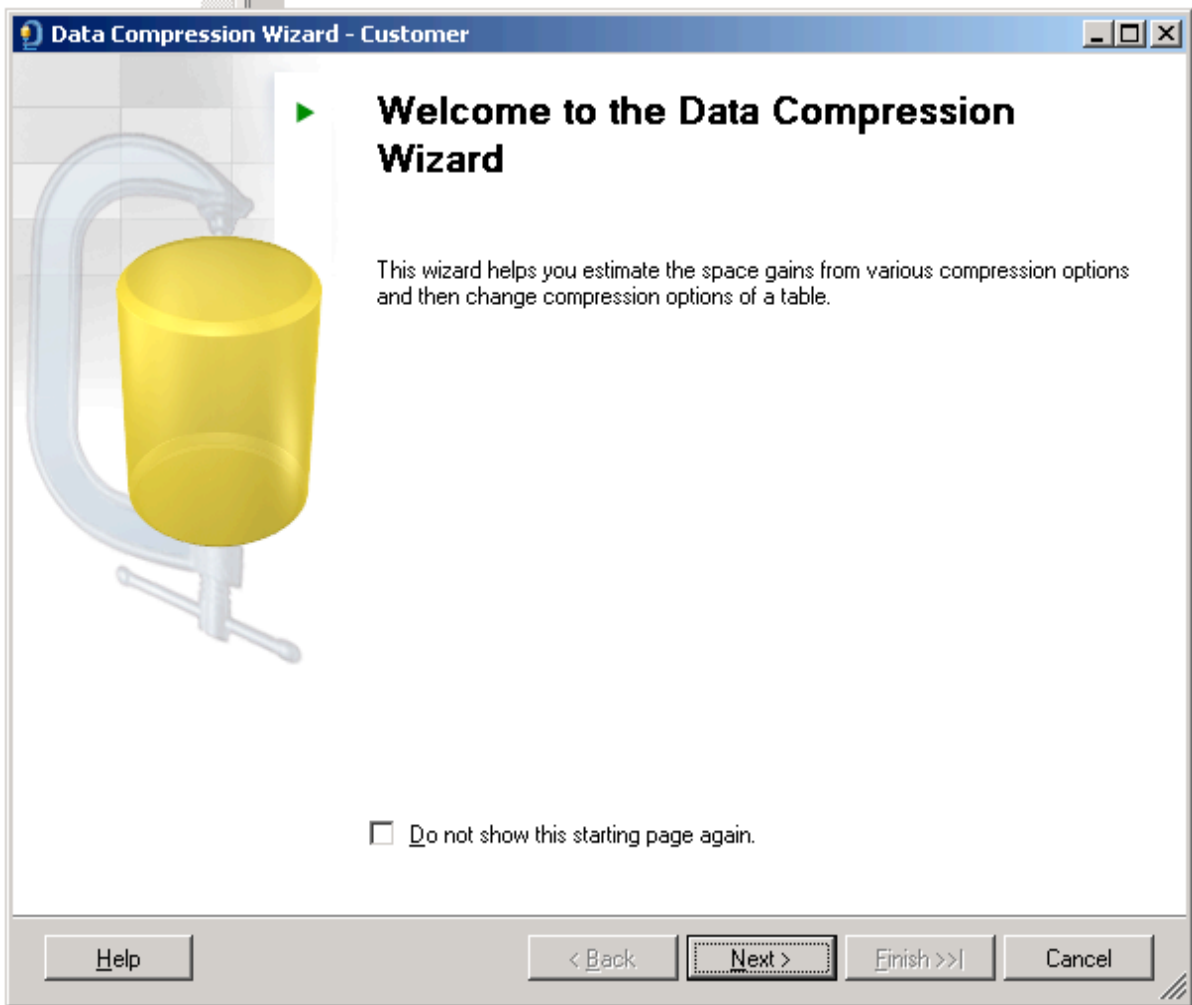


Figure 2: The Data Compression Wizard, or Transact-SQL commands, can be used to manage data compression.

After clicking “Next,” the wizard displays the following screen, which allows you not only to select the compression type, but it also allows you to calculate how much space you will save once compression has been turned on.

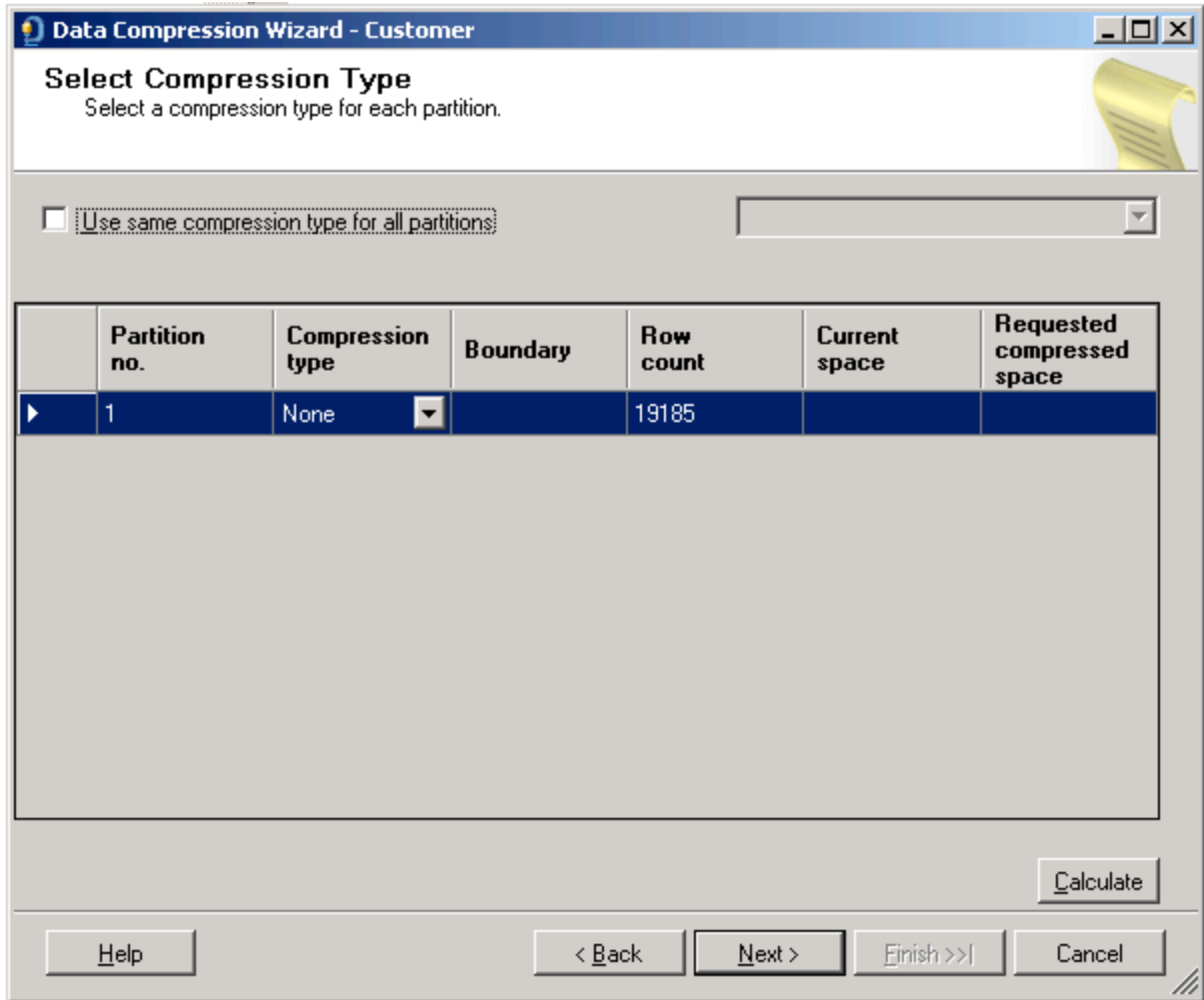


Figure 3: Use this screen to select the compression type, and to calculate how much space will be saved.

First, let’s see how much space we will save if we use row-level compression on this table. To find out, click on the drop-down box below “Compression Type,” select “Row,” and then click “Calculate.”

| | Partition no. | Compression type | Boundary | Row count | Current space | Requested compressed space |
|---|---------------|------------------|----------|-----------|---------------|----------------------------|
| ▶ | 1 | Row | | 19185 | 1.953 MB | 1.922 MB |

Figure 4: For this table, row-level compression doesn’t offer much compression.

After clicking “Calculate,” the wizard runs and calculates how much space is currently being used, and how much space would be used after row-level compression. As we can see, very little space will be saved, about 1.6%.

Now, let’s see how much compression savings page-level compression offers us for this particular table. Again, I go to the drop-down menu under “Compression Type,” select “Page,” then press “Calculate.”

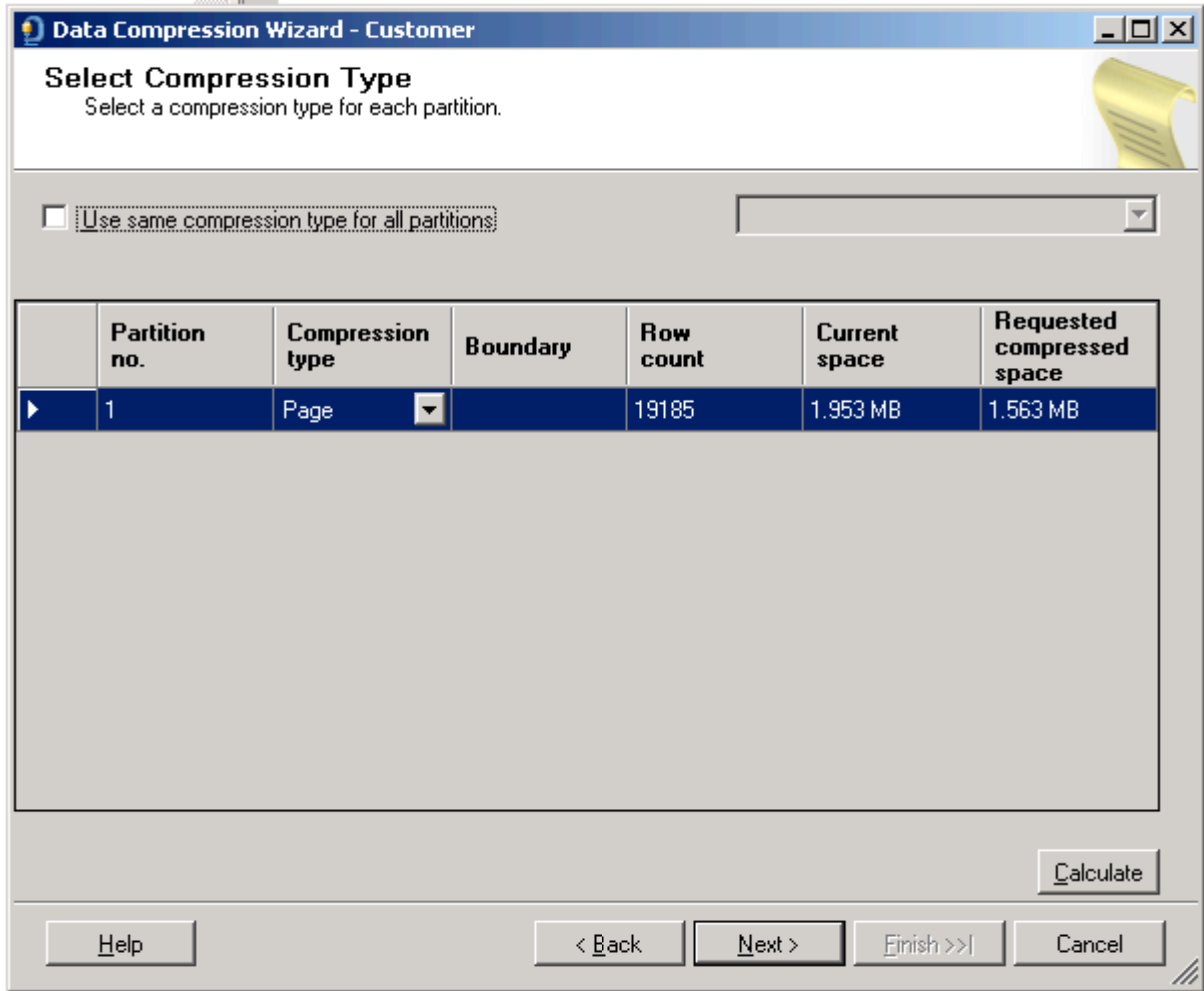


Figure 5: Page-level compression is higher than row-level compression.

After pressing “Calculate,” we see that compression has improved greatly, now saving about 20% space. At this point, if you should decide to turn on page-level compression for this table, click on the “Next” button.

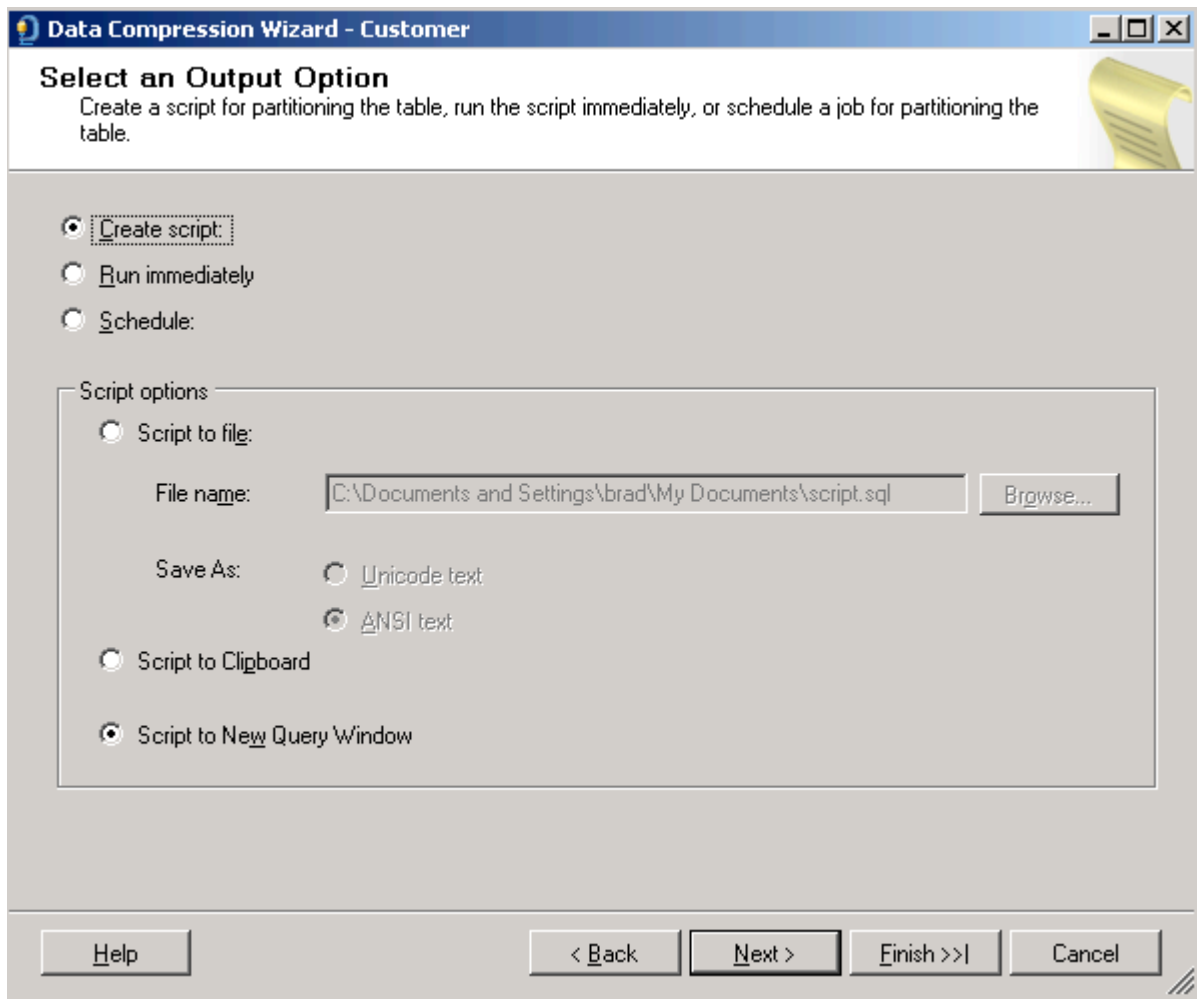


Figure 6: The wizard allows you several options in which to turn on compression.

At the above screen, you can choose to perform the compression now (not a good idea during busy production times because the initial compression process can be very CPU and disk I/O intensive), schedule it to run later, or just to script the Transact-SQL code so you can run it at your convenience.

Once you have compressed this table (a clustered index), keep in mind that any non-clustered indexes that this table may have are not automatically compressed for you. Remember, compression is based on a per object basis. If you want to compress the non-clustered indexes for this table, you will have to compress each one, one at a time.

While this wizard helps you to see how much compression either method offers, it does not suggest which compression method should be used, nor does it recommend whether compression should be used at all for this object. As the DBA, it will be your job to evaluate each compressible object to determine if it should have compression enabled or not. In other words, you must decide if the benefits of compression outweigh the negatives.

Backup Compression

For years, there have been third-party programs that allow you to compress and speed up SQL Server backups. In most regards, the backup compression included with the Enterprise Edition of SQL Server is very plain vanilla. In fact, if you already are using a third-party backup program, I would suggest you continue using it, because SQL Server 2008 backup compression offers fewer features. In fact, the only option SQL Server 2008 backup compression offers you is to turn it off or on. That's it.

SQL Server 2008 backup compression, like the third-party add-ons, compresses backups, which not only saves backup space, but it can substantially reduce backup times. Unlike data compression, there is very little downside to using backup compression, other than the additional CPU resources required to perform the compression (or decompression during a restore). Assuming that you perform backups during slower times of the day, the additional CPU resources used will not be noticeable.

The time and space savings offered by backup compression depends on the data in your database. If you are heavily using data compression in your databases, or are using Transparent Data Encryption, then using backup compression probably won't offer you many benefits, as already compressed data, or encrypted data, is not very compressible.

Let's take a brief look at how you turn on SQL Server 2008 backup compression. While our example will use SSMS, you can use Transact-SQL to perform the same task.

To backup AdventureWorks, right-click on the database in SSMS, select "Tasks," and then select "Back Up," and the backup dialog box appears.

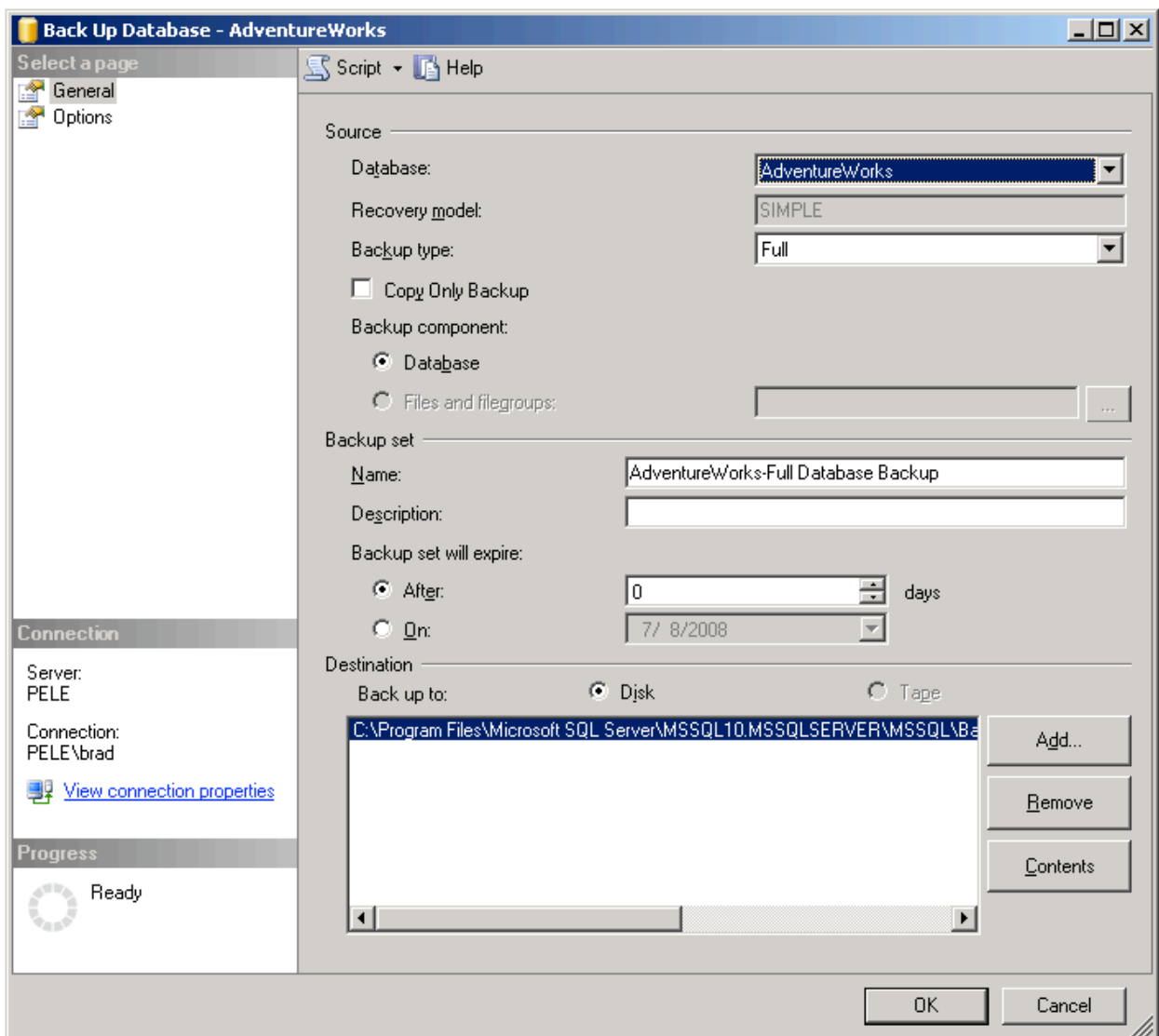


Figure 7: As with any backup, you can use the backup dialog box to make your selections.

Once you have selected your backup options, next click on “Options,” and the following screen appears.

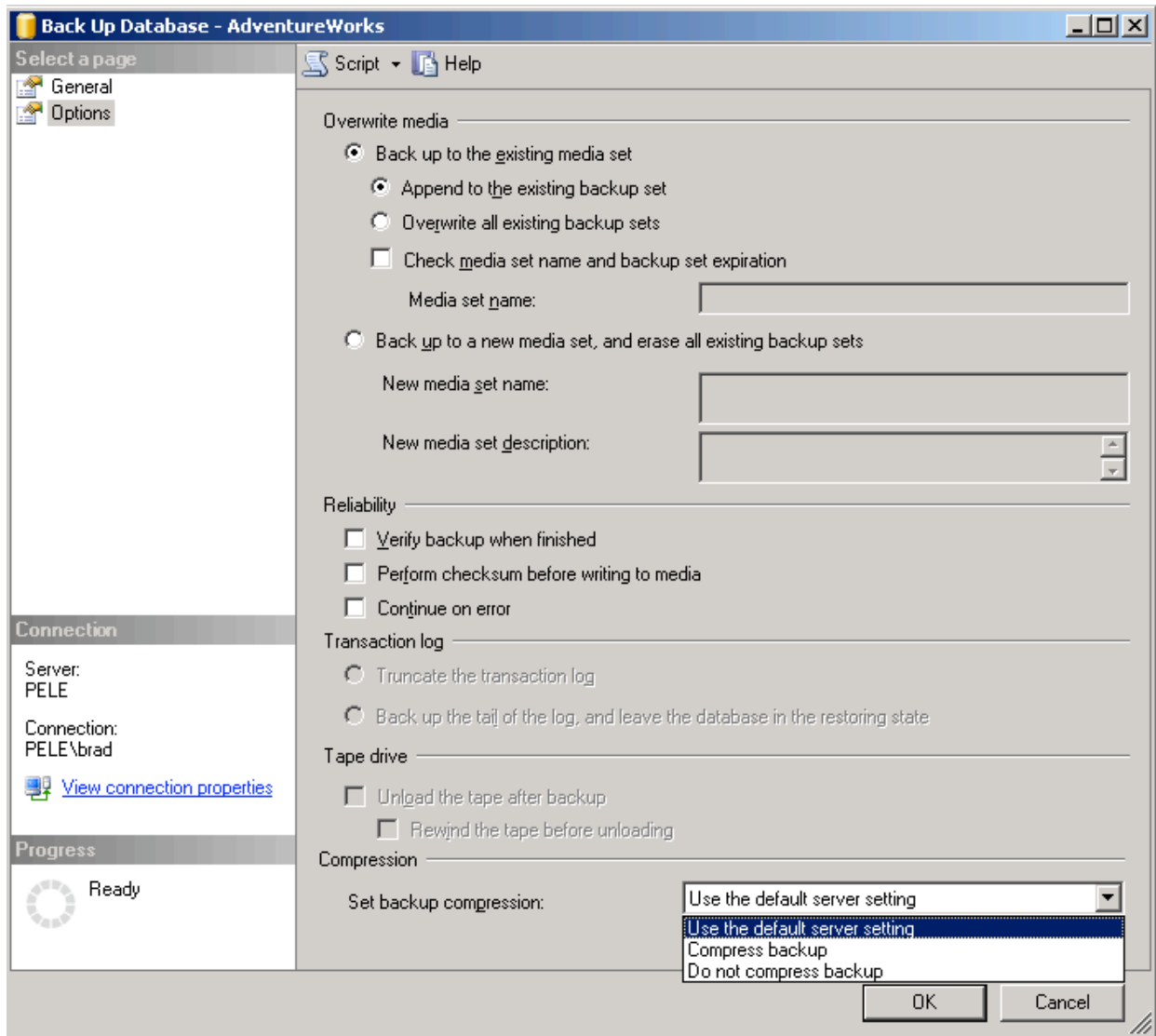


Figure 8: Backup compression options are limited.

At the top of figure 8 are the standard backup options, while at the bottom of the screen you see the options for backup compression. Notice that you only have three choices.

The first option, “Use the default server settings” tells the backup to use the server’s default backup compression setting. In SQL Server 2008, there is a new `sp_configure` option called “backup compression default.” By default, it is set to have backup compression off. If you want, you can set this option so that backup compression is turned on by default. So if you choose the “Use the default server settings” option above, then whatever option is set for the “backup compression default” will be used for the backup.

The “Compress Backup” option turns backup compression on, and the “Do not compress backup” option turns it off. Both of these options override the “backup compress default” server setting, whatever it happens to be.

Once you have chosen your backup compression method, you proceed with the backup just like any other SQL Server backup.

If you need to restore a compressed backup, you don't have to do anything special, it will uncompress itself automatically. Although you can only compress backups using the Enterprise Edition of SQL Server 2008, you can restore a compressed backup to any edition of SQL Server 2008. On the other hand, you cannot restore a compressed SQL Server 2008 backup to any previous version of SQL Server.

Summary

In this chapter, we have learned about the two forms of data compression, and about backup compression. While data compression might seem like a seductive new feature of SQL Server, I highly recommend that it is only used by experienced DBAs. While it offers lots of promise for increased performance, it can just as easily cause performance problems if misused. Backup compression, on the other hand, can be used by DBAs of all skill levels.

CHAPTER 4: RESOURCE GOVERNOR

I think most of us are familiar with this situation: a SQL Server database is the backend of an OLTP (Online Transaction Processing) application, but you are required to allow users to run any reports they want on the production data, any time they want. This often results in long-running reports that negatively affect OLTP performance. Haven't you ever wished you could limit the amount of hardware resources allocated to reporting, so that normal production activity is not affected?

In SQL Server 2008, you can. The Resource Governor, available only in the Enterprise Edition of SQL Server 2008, is a new technology that allows you to selectively control how much CPU and memory resources are assigned to a particular workload. For example, the Resource Governor allows you to specify that no more than 20% (or any figure you designate) of CPU and/or memory resources can be allocated to running reports. When this option is turned on, then no matter how many reports are run, they can never exceed their designated resource allocation. Of course, this will reduce the performance of reporting, but at least now production OLTP activity won't be negatively affected by reports.

Currently, Resource Governor only works within the database engine. It does not work with Integration Services, Reporting Services, or Analysis Services.

In this chapter you will learn how you can use the Resource Governor to better prioritize and manage a SQL Server's workload, how Resource Governor works under the hood, and how to configure it for use.

Uses for Resource Governor

Resource Governor is designed to help DBAs four different ways:

- **Managing Unpredictable Workloads**

As in my reporting example above, it is often difficult to know how your SQL Server resources are being used. While OLTP activity might follow a predictable pattern, there is no knowing when a junior marketing executive will start running reports against your server. Resource Governor allows you to classify different types of loads on your server, which in turn allows you to control how server resources are assigned to a given activity. In SQL Server 2005 and earlier, queries fought amongst themselves to decide which one would grab the necessary resources first, and it was hard to predict who would win out. By using Resource Governor, I can instruct SQL Server to limit the resources a particular activity can access. In this way, I can ensure that my predictable OLTP performance is not hurt by unpredictable activity.

- **Putting Limits on Run-Away Queries**

Closely related to the problem of managing unpredictable workloads is the difficulty of dealing with "run-away queries". If you have been a DBA for very long at all, you will have seen queries that seem to take forever to run, tying up valuable and limited physical resources, and hurting the performance of other activity on your server. By limiting user or applications that have the potential for executing run-away queries, then run-away queries become much less painful because they can never use more resources than have been assigned to them.

- **Establishing Workload Priority**

Most SQL Server instances have different demands placed on them throughout the day or week. For example, an OLTP server may need to be up 24/7 and maintain a specific service level. On the other hand, this same server may need to run SSIS jobs and maintenance jobs in order to properly function. When these jobs kick in, there is the potential that they can interfere

with production work. Resource Governor allows you to allocate resources in such a way as to ensure that priority OLTP activity runs without interference from necessary maintenance jobs.

- **Prioritizing Preferred Users**

For better or worse, Resource Governor allows you to assign specific users, or groups, higher priority than others, on an arbitrary basis. One of the inevitable consequences of this is that, as soon as managers hear about it, you may be asked to give priority to certain users over other users. Hopefully you can avoid this political battle, but you may not be able to. As George Orwell puts it in his novel, *Animal Farm*: "all animals are equal, but some animals are more equal than others".

How Resource Governor Works

The best way to explain how the Resource Governor works is to walk through an example. Let's assume we have a SQL Server 2008 instance, with a single database, that is designed to store data from an ERP (Enterprise Resource Planning) application. This is essentially a 100% pure OLTP application that must provide a specified service level 24/7. Let's also assume that the reporting for this ERP application is not done through the application, but via a third-party reporting tool, directly against the database. In addition to running standard reports, the reporting tool permits users to run ad-hoc reports anytime they want. Essentially, this reporting can be considered an OLAP (Online Analytical Processing) activity.

This is a classic example of OLTP and OLAP activities running on the same server, against the same database, at the same time. As most of you know, this is a terrible architecture because it can result in non-time sensitive OLAP reporting blocking time-sensitive OLTP activity, potentially preventing you from attaining your specified service level for the ERP application. Unfortunately, as most of you will also know, this architecture is all too common.

As the DBA, you want to use Resource Governor to ensure that your ERP application attains its specified service level, while at the same time, allowing reports to be run, but at a lower priority so they don't interfere with OLTP activity. To accomplish this goal, you have decided that you want to limit both the amount of SQL Server CPU and memory resources used by the Reporting application to a maximum of 20% of what the server has available. Based on your research, you have determined that, with this restriction in place, your ERP application can consistently attain its service levels.

Figure 1 presents a somewhat-oversimplified picture of this scenario. In this figure we can see two applications connecting to SQL Server. The first is our ERP application, called "ERP", and the second is our reporting application, called "Reporting":

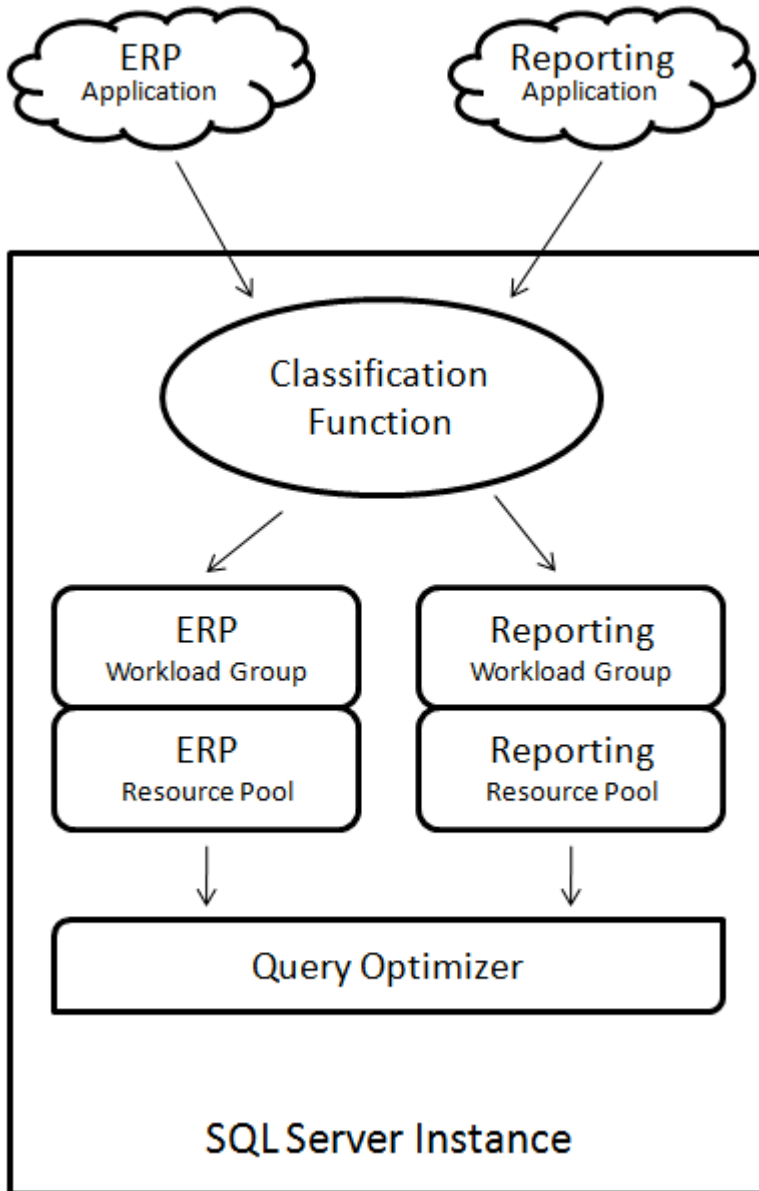


Figure 1: An oversimplified illustration of how the Resource Governor classifies incoming application connections and then assigns the designated resources.

In order for the Resource Governor to be able to allocate CPU and memory resources as you require, it must be able to differentiate between different workloads. It does this using a user-defined **classification function**, created using Transact-SQL. When a connection is made to SQL Server, Resource Governor will use the classification function to evaluate each connection to determine which application it came from: either the ERP or the Reporting application. This is why it is important to be able to identify each application as it connects to SQL Server, because the classification function uses the `APP_NAME()`¹ function to tell the Resource Governor which application is connecting to the database.

¹ Besides using the `APP_NAME()` function to differentiate one workload from another, Resource Governor can use other system functions, such as `HOST_NAME()`, `SUSER_NAME()`, `SUSER_SNAME()`, `IS_SRVROLEMEMBER()`, and `IS_MEMBER()`.

Once an incoming connection has been classified, the connection is then routed to a particular **workload group**. Put simply, a workload group is a container that holds connection requests that match a set of SQL Server connections, as defined by the classification function. For example, the ERP application connections go to a workload group named "ERP", and the Reporting application connections go to a workload group named "Reporting."

At this point, all we have done is to separate out the connections from each application. As yet, we still have not allocated any resources to these connections, and this is the next step.

Associated with every workload group is a **resource pool**. A resource pool represents the physical resources of a server. It is used to allocate the minimum and maximum amount of CPU and/or memory that is to be allocated to a specific type of SQL Server connection. For example, if you want to limit the connections from the Reporting application to no more than 20% of the available CPU and memory resources of the server, then you will configure a resource pool with this information.²

In our example, there are two resource pools, one associated with the ERP workload group and the other with the Reporting workload group. The ERP resource pool permits connections in this group to use up to 100% of the available CPU and memory resources on the server. The Reporting resource pool limits connections in the Reporting workload group to no more than 20% of the server's CPU and memory resources.

Once a connection has been classified and put into the correct workload group, then the connection is allocated the CPU and memory resources that have been assigned to it, and then the query is passed on to the query optimizer for execution. In our example, Reporting queries would be allocated resources in such a way that they would not use in excess of 20% of the CPU and memory resources when passed to the query optimizer for execution.

Given that we specified that the ERP application has the ability to use up to 100% of the available CPU and memory resources, you're probably wondering what would happen if a request from the Reporting Application came through while the ERP application is using 100% of the resources. Under this condition, the Reporting connection would simply have to wait until resources become available for it. In this way, the ERP application get all the resources it needs to function properly, while allowing the Reporting application to continue to run, but at a lower priority.

Configuring Resource Governor

Unlike many other features in SQL Server, you cannot use SSMS to entirely configure and manage Resource Governor. You can either use a combination of Transact-SQL and SSMS, or you can use Transact-SQL only. To keep things as simple as possible, all of the following examples will be demonstrated using Transact-SQL only. Check out Books Online to see how SSMS can be used for some of the following steps.

² Resource Governor only "kicks in" if there are *resource contentions*. For example, if the ERP workload happens to be small, and there are unused resources available, then the Reporting workload can use more than 20% of its allocated resources. But if the ERP workload is large, causing the ERP and Reporting workloads to fight for server resource, only then would the Resource Governor kick in and prevent the Reporting workload from using more than 20% of the available resources. For the rest of this chapter, we will be assuming that there is continuous resource contention and that the Reporting workload will be constrained by the Resource Governor.

Now, let's go through an example of how to configure the Resource Governor, based on the Reporting example we've used throughout the chapter.

Creating Resource Pools

The first step is to create a resource pool, each for the ERP and the Reporting applications.

```
--The master database must be used to configure the Resource Governor
USE MASTER
GO
--Create Resource Pool to Limit Resources for ERP application
CREATE RESOURCE POOL poolERP
WITH
(
  MAX_CPU_PERCENT=100,
  MAX_MEMORY_PERCENT=100,
  MIN_CPU_PERCENT=80,
  MIN_MEMORY_PERCENT=80
);
GO

--Create Resource Pool to Limit Resources for Reporting application
CREATE RESOURCE POOL poolReporting
WITH
(
  MAX_CPU_PERCENT=20,
  MAX_MEMORY_PERCENT=20,
  MIN_CPU_PERCENT=0,
  MIN_MEMORY_PERCENT=0
);
GO
```

As you can see above, the CREATE RESOURCE POOL statement has a number of parameters. They include:

- **MIN_CPU_PERCENT**: Allows you to specify the guaranteed average CPU bandwidth for all requests to the resource pool *when there is CPU contention*.
- **MAX_CPU_PERCENT**: Allows you to specify the maximum average CPU bandwidth that all requests for the resource pool *when there is CPU contention*.
- **MIN_MEMORY_PERCENT**: Allows you to specify the minimum amount of memory that is reserved for the resource pool that *cannot be shared with other resource pools*.
- **MAX_MEMORY_PERCENT**: Allows you to specify the total server memory that can be used by requests to the resource pool.

In the code above, I have created a resource pool named "poolERP" that will allow the ERP application to use up to 100% of the available server CPU and memory resources. In addition, I have specified that the "poolERP" be allocated a minimum of 80% of the total server CPU and memory resources available. This guarantees that "poolERP" will always have access to a minimum of 80% of the resources.

For "poolReporting", I have specified that the maximum amount of resources that go to the Reporting application is 20%. Assuming there are resource contentions, the Reporting application's resources will be constrained to this amount.

Creating Workload Groups

Next, I need to create the workload groups that will act as containers for the user connections that are defined in the classification user-defined function. During this same step, I also associate to each workload group one of the resource pools I just created.

```
--Create Workload Group to Hold Connections From ERP Application and Assign Resource Pool
CREATE WORKLOAD GROUP wrkgroupERP
USING poolERP
GO
--Create Workload Group to Hold Connections From ERP Application and Assign Resource Pool
CREATE WORKLOAD GROUP wrkgroupReporting
USING poolReporting;
GO
```

I created a workload group called "wrkgroupERP" using the CREATE WORKLOAD GROUP statement, and I have associated it to the resource pool "poolERP". In addition, I created a workload group called wrkgroupReporting that is associated to the resource pool "poolReporting".

Creating the Classification Function

Next, I create the classification user-defined function that specifies which connections will be put inside each of the workload groups created in the previous step. Only one user-defined classification function can be created for each SQL Server instance, so when creating your own user-defined classification function, it is important to consider all the various connections that can be made to a SQL Server instance, and deal with them appropriately by assigning them to an appropriate workload group (and its associated resource pool). If a connection is not assigned to a specific workload group (and pool) by the classification function, then the connection goes to a default workload group (and pool). See Books Online to learn how to properly configure a default workload group and pool.

```
--Create Classification User-Defined Function
CREATE FUNCTION rgclassifier1() RETURNS SYSNAME
WITH SCHEMABINDING
AS
BEGIN
DECLARE @grp_name AS SYSNAME
IF (APP_NAME() = 'Reporting')
SET @grp_Name = 'wrkgroupReporting'
ELSE
SET @grp_Name = 'wrkgroupERP'
RETURN @grp_name
END;
GO
```

The user-defined classification function is created using the CREATE FUNCTION statement and is called "rgclassifier1()". Inside this function, I specify the criteria on which SQL Server user connections are evaluated. For example, I have specified that if the **APP_NAME()** of a user connection equals "Reporting," which is the application name of the Reporting application, then it should be added to the "wrkgroupReports" workload group. If the **APP_NAME()** of a user connection does not equal "Reporting," then it should be added to the "wrkgroupERP" workload group.

Enabling Resource Governor

We are almost done, but we have a couple of maintenance tasks to perform before our Resource Governor will work. First, we need to run the following code, which assigns the "rgclassifier1" function to the Resource Governor.

```
-- Register the classifier function with Resource Governor
ALTER RESOURCE GOVERNOR WITH (CLASSIFIER_FUNCTION= dbo.rgclassifier1);
GO
Second, we need to run the following code, which enables the Resource Governor.
-- Start Resource Governor
ALTER RESOURCE GOVERNOR RECONFIGURE;
```

GO

At this point, the Resource Governor is turned on, and any time a report is run from the Reporting application, it will be limited to a maximum of 20% CPU and memory resources, while the ERP application will be able to use up to 100% of the available CPU and memory resources.

Summary

As you can imagine, there is a lot more to the Resource Governor than what I have discussed in this chapter. I have left out a lot of details, including additional configuration options, how to make changes after the fact, how to monitor if the Resource Governor is running properly, and troubleshooting information. If you are interested in learning more about the Resource Governor, I suggest you dive into Books Online and begin experimenting with it on a test server.

While Resource Governor offers many potential benefits, it also offers some potential pitfalls. For example, a misconfigured Resource Governor can not only hurt a server's overall performance, it can potentially lock your server up, requiring you to use the Dedicated Administrator Connection (DAC) to attach to the locked up SQL Server in order to troubleshoot and fix the problem. Because of this, I highly recommend that you only use this option if you are an experienced DBA, and that you first test your configuration on a test server before rolling it out into production.

CHAPTER 5: PERFORMANCE DATA COLLECTOR

Traditionally, SQL Server has included a lot of tools to help DBAs identify and troubleshoot performance problems, such as Profiler, System Monitor, the Database Engine Tuning Engine, SSMS (SQL Server Management Studio), DMVs, and T-SQL commands. Some of these tools provide the ability to log historical activity, storing the data in a variety of formats. Other tools, such as DMVs, don't even permit any kind of logging, unless you write your own logging solution. This mishmash of different tools and data collection methods makes it difficult for DBAs to easily identify and fix a wide variety of performance problems.

In a perfect world, SQL Server would include a single tool that would not only automatically collect all the important performance data DBAs need to identify and fix performance issues; it would store the data in a single format, in a single location, and include advanced reporting that allowed DBAs to easily interpret the data so they could identify relevant solutions.

While SQL Server 2008 doesn't include the perfect performance collection and analysis tool, it does provide a new feature called the Performance Data Collector, and it is a good first step toward this ideal tool.

In a nutshell, the Performance Data Collector is designed to help DBAs these ways:

- **Act as a Central Data Repository:** Part of the Performance Data Collector is what is called the Management Data Warehouse (MDW). It is a database used to store all of the data collected in one central location. It can store the data from a single SQL Server instance, or from multiple instances. While the focus of the MDW in SQL Server 2008 is for storing performance-related data, in future versions of SQL Server, expect it to be used to store virtually any data you want to collect from SQL Server, such as Extended Events, audit data, and more. The MDW is extensible, so if you are up to the task, you can store your own data in the MDW.
- **Collect Selected SQL Server Performance Data:** While the MDW is used to store performance data, the actual collection of data is performed by what are called Data Collection Sets. SQL Server 2008 comes with three built-in Data Collection Sets: one is to collect Disk Usage information; another is used to collect Query Statistics, and the third is used to collect a wide variety of Server Activities. According to Microsoft, these three Data Collection Sets collect the essential data needed to identify and troubleshoot most common SQL Server performance problems. If you don't think the data that is collected is enough, you can create your own custom Data Collection Sets.
- **Display Performance Reports:** Data stored inside a data warehouse is not very useful unless you can get at it. SQL Server 2008 includes three built-in reports, including Disk Usage Summary, Query Statistics History, and Server Activity History. Each of these reports allow you to examine the history of a single SQL Server instance, one at a time. In addition, each of the reports allow you to drill down into sub-reports, giving you an even more detailed view of what is going on in your server. If you think the built-in reports are incomplete, or if you want to report on custom-collected data, or produce reports that combine information from multiple SQL Server instances, then you can create your own custom Reporting reports using the SQL Server Business Intelligence Development Studio, or with any of the Reporting Services tools.

The Performance Data Collector is fully supported by the Enterprise and Standard editions of SQL Server 2008. Out of the box, it only works only on SQL Server 2008 instances, and is not backwards compatible with previous versions of SQL Server.

How to Configure the Performance Data Collector

Once you complete the initial installation of SQL Server 2008, the Performance Data Collector is not configured by default. To configure it and turn it on, you have to go through two steps.

The first step is to use the “Configuration Management Data Warehouse” wizard to create the MDW database using the “Create or Upgrade a Data Management Warehouse” option. While this database can be stored on any SQL Server, ideally you should create it on a SQL Server instance designated specifically for this purpose. By doing this, you will help to reduce the overhead the Performance Data Collector puts on your production servers. One central MDW can hold the data for many SQL Server instances.

The second step is to enable the Performance Data Collector. To do this, you again use the “Configuration Management Data Warehouse” wizard, but this time you select the “Setup Data Collection” option, which allows you to select the server and MDW database where you want the performance data to be stored. Once the wizard completes, the Performance Data Collector is enabled and data begins collecting almost immediately.

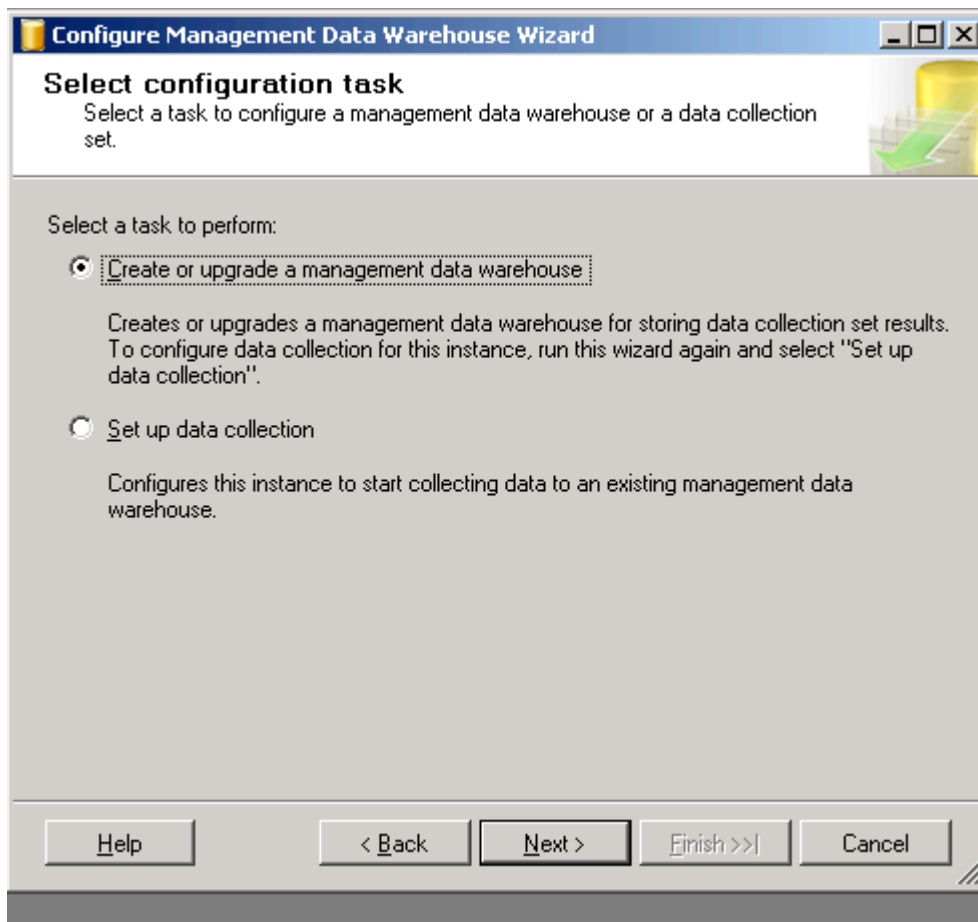


Figure 1: The Configure Management Data Warehouse Wizard is used to create the MDW, and to configure, setup, and enable the data collection components of the Performance Data Collector.

If you want to enable the Performance Data Collector on multiple instances of SQL Server 2008, you will have to run the wizard once for each 2008 instance you want to monitor, each time pointing to the server where the central MDW database is located.

How the Performance Data Collector Works

When the Performance Data Collector is first set up, it does a lot of behind the scenes work. For example, it creates SSIS package that will be used to collect data and then to move it to the MDW. It will also create a series of scheduled jobs that will be used to execute jobs on a regular basis. And it will add new tables to the MSDB database to store logging and other configuration information.

The easiest way to find out how the Performance Data Collector works is to follow the flow of data from the source, all the way to the MDW, where it is stored. To keep this example short and simple, we will focus only on the Server Activity Data Collector. We won't be covering the Disk Usage and the Query Statistics Data Collectors in the following example because all three Data Collectors work similarly, although the data they collect is obviously different.

The Server Activity Collection Set is designed to collect two different types of data: DMV snapshots and performance counters that are useful for monitoring SQL Server's overall performance. The best way to see this is to check out the Server Activity Properties screen.

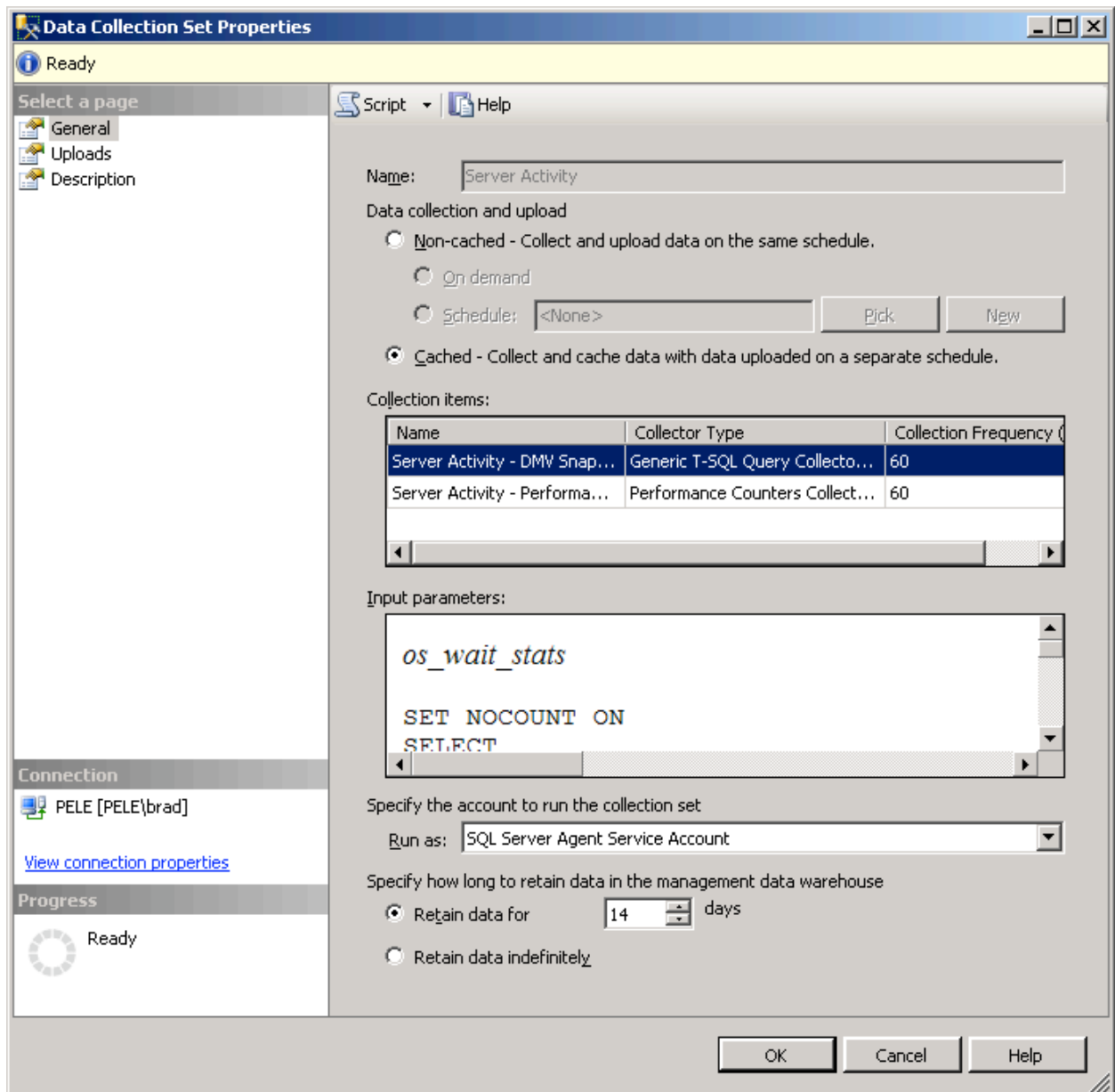


Figure 2: Each Data Collection Set has its own properties screen. The Server Activity – DMV Snapshot collection item is highlighted.

In Figure 2, under “Collection Items,” there are two types of data the Server Activity Data Collection Set gathers. The first one, “Server Activity – DMV Snapshots,” is designed to literally take snapshots of specific DMVs every 60 seconds (see this under Collection Frequency). The Transact-SQL code it uses to collect this data is shown under “Input Parameters.” Obviously, you can’t see much of the code on this screen, but if you were to scroll down the window, you will see exactly what information is being selected from which DMVs.

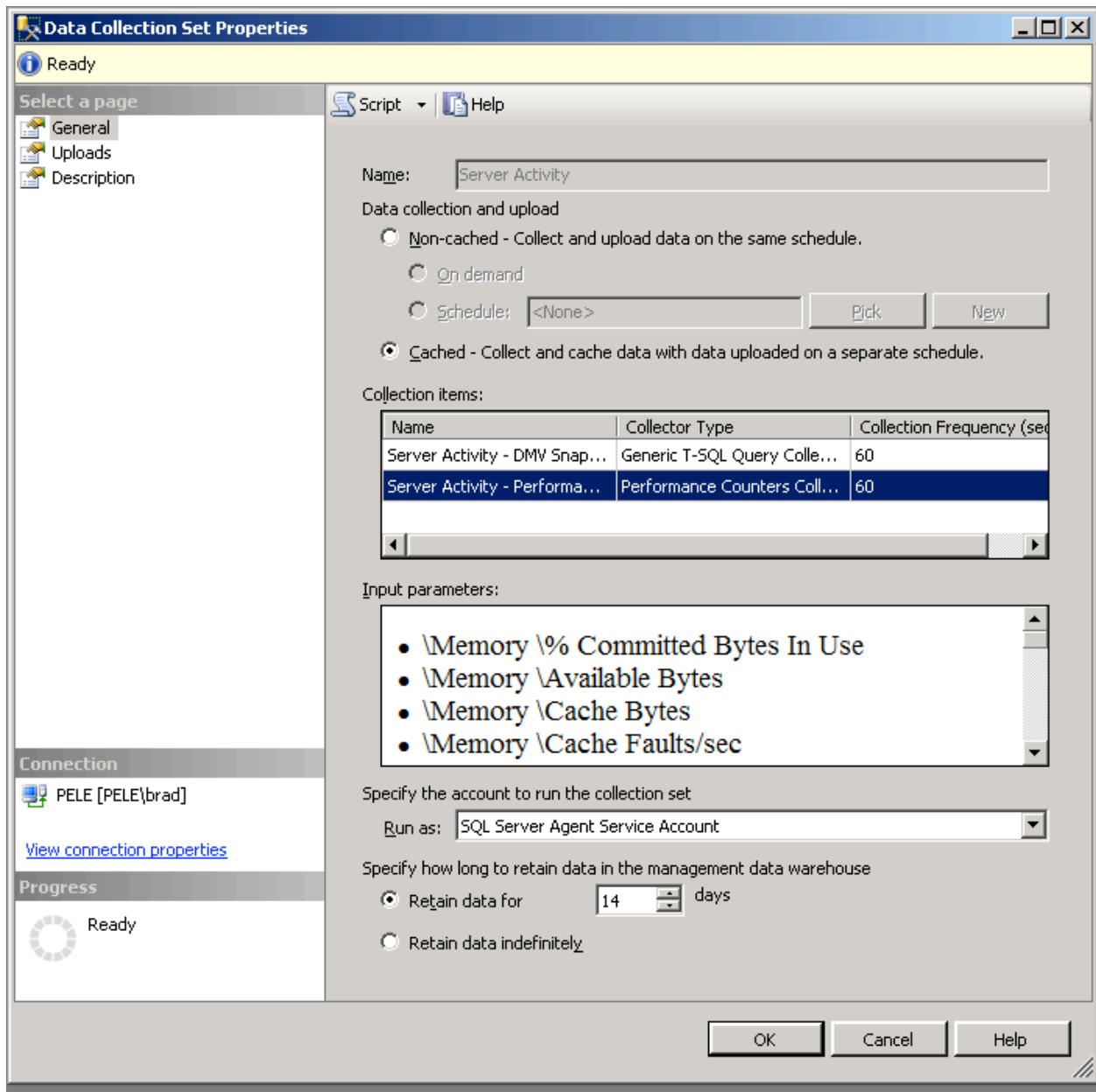


Figure 3: The Server Activity – Performance Monitor Counters collection item is highlighted.

In Figure 3, the “Server Activity – Performance Counters” collection item is selected. Below it, in the “Input Parameters” screen, you see some of the Performance Monitor counters that are collected every 60 seconds. Like the DMV Snapshots, this is a snapshot of specific counters at a point in time.

Now that we have a basic idea of what type of data is collected by the Server Activity Collection Set, exactly how does the Performance Data Collector gather this information and store it in the MDW?

Data collection is scheduled as part of a SQL Server Agent job. When the job is run, it starts what is called the Data Collector Run-Time Component (dexec.exe) that is used to load and execute SSIS packages. In this particular example, what happens is that the DMV and Performance Monitor counter snapshots are collected every 60 seconds by an SSIS package, then this information is cached (stored) in a local folder on the SQL Server instance being monitored. Notice that the option, “Cached – Collect and Update Data on the Same Schedule option,” is selected on the properties page above.

Then, every 15 minutes, a different SQL Server Agent job and SSIS package executes, collecting the data stored in the local cache and moves it to the MDW. Below, you can see another part of the properties page that allows you to control how often uploads occur.

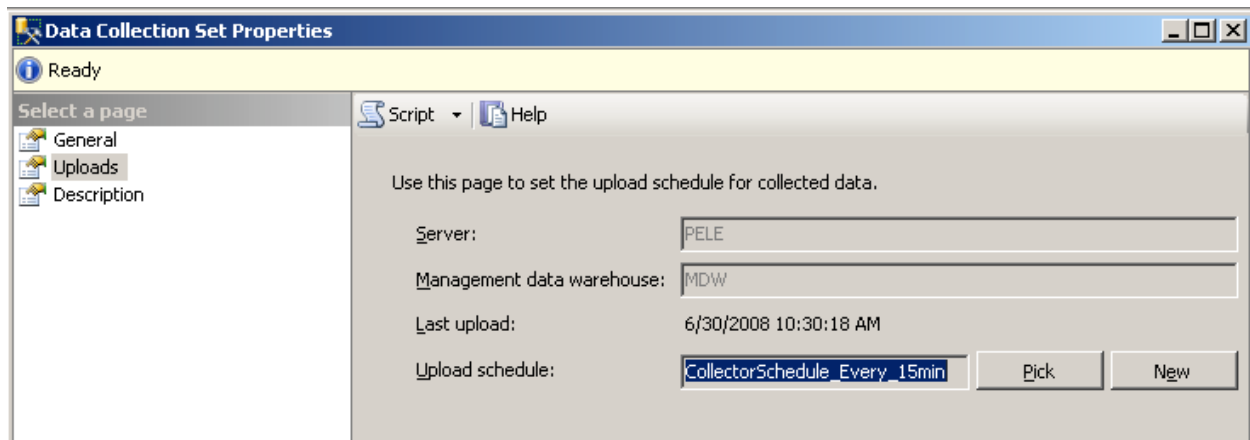


Figure 4: The Uploads option of the Server Activity Properties Page is used to designate how often data is moved from the local cache to the MDW.

Once the data is moved to the MDW, then it can be reported on using one of the three built-in reports, or any custom reports you create.

The Data Collector offers two different ways data can be collected and moved to the MDW: Cached and Non-Cached. The Cached option is what I described above, where one job and SSIS package is used to gather the data and store it in a local cache, and then a second job and SSIS package runs and moves the data from the local cache to the MDW. This option helps to reduce the overhead of the Data Collector by reducing how often data is moved between the monitored instance and the MDW.

The Non-Cached option, which is used by the Disk usage Data Collection Set, works a little differently. Instead of using two steps to move data to the MDW, it collects and uploads the data in one step. This option incurs slightly more overhead than the Cached method, but if not used too often, using it won't make much impact on SQL Server's performance.

Another feature of the Data Collector is that older data is automatically purged from the MDW based on a default schedule, or one you set. In figures 2 and 3, under "Specify how long to retain data in the management data warehouse," you can see that data is only retained for 14 days for Server Activity data.

One question that you may be asking yourself is how much overhead the Performance Data Collector will take. While this will depend on the load of your server, and your server's hardware, using this option will add about 4% to CPU utilization, in addition to collecting about 250-300 MB of data each day, and this is for the default Data Collection Sets only. If you create your own Data Collection Sets, then the overhead will be greater.

Reports Available from the Performance Data Collector

The Performance Data Collector includes three built-in reports, one report for each of the default Data Collection Sets. While we don't have enough time to take a detailed look at each, let's take a quick look at each report from a high-level perspective. Even though I won't show it here, you can drill into each report for more detail.

The first report we will look at is the Disk Usage report.

Disk Usage Collection Set

on PELE at 6/30/2008 11:29:19 AM



This report provides an overview of the disk space used for all databases on the server and growth trends for the data file and the log file for each database for the last 4 collection points between 6/27/2008 2:24:38 PM and 6/30/2008 11:28:55 AM.

| Database Name | Database | | | | Log | | | |
|------------------------------------|-----------------|-------|-------------------|-------------------------|-----------------|-------|-------------------|-------------------------|
| | Start Size (MB) | Trend | Current Size (MB) | Average Growth (MB/Day) | Start Size (MB) | Trend | Current Size (MB) | Average Growth (MB/Day) |
| AdventureWorks | 172.44 | | 172.44 | 0 | 2.00 | | 2.00 | 0 |
| master | 4.00 | | 4.00 | 0 | 1.00 | | 1.00 | 0 |
| MDW | 100.00 | | 100.00 | 0 | 10.00 | | 20.00 | 5 |
| model | 1.25 | | 1.25 | 0 | 0.50 | | 0.50 | 0 |
| msdb | 12.75 | | 15.50 | 1.375 | 2.25 | | 2.25 | 0 |
| ReportServer | 3.25 | | 3.25 | 0 | 6.13 | | 6.13 | 0 |
| ReportServerTempDB | 2.25 | | 2.25 | 0 | 0.75 | | 0.75 | 0 |
| tempdb | 8.00 | | 19.00 | 5.5 | 1.00 | | 1.25 | 0.125 |

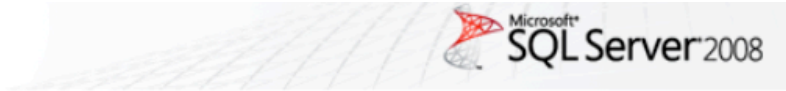
Figure 5: The Disk Usage Collection Set report tracks disk space over time.

This reports tracks disk space usage for your MDB and LDF files, providing both actual figures, along with displaying simple trend lines. This information can help you be more proactive as a DBA, as you can use this data to prevent yourself from running out of disk space at inopportune times. If you click on any of the databases, you will get a sub-report showing more details on how data is allocated in each database, among other information.

The next report is the Query Statistics History report.

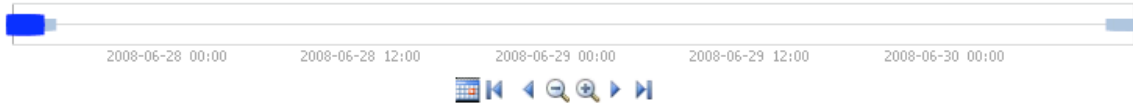
Query Statistics History

on PELE at 6/30/2008 11:33:40 AM

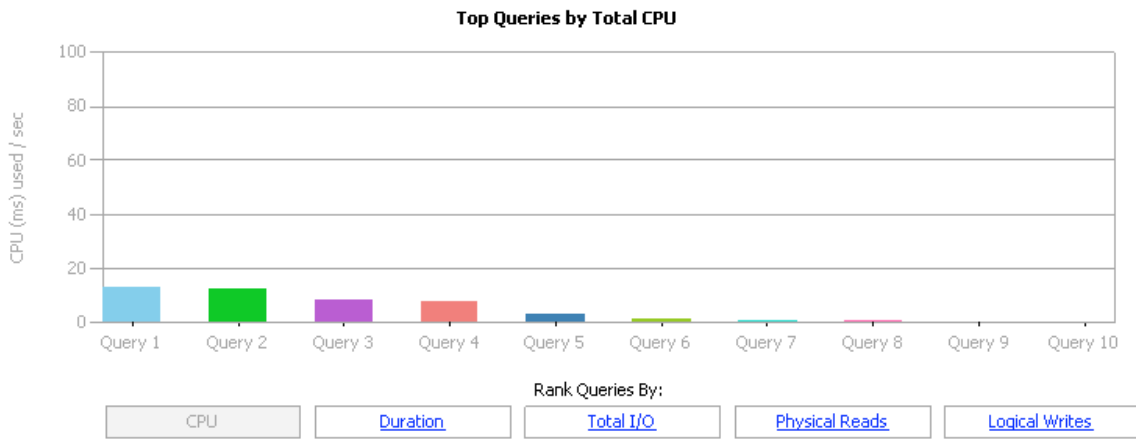


This report provides an overview of query execution statistics for the SQL Server instance.

Navigate through the historical snapshots of data using the time line below.



Selected time range: 6/27/2008 12:15:04 PM to 6/27/2008 4:15:04 PM



| Query # | Query | Executions / min | CPU ms / sec | Total Duration (sec) | Physical Reads / sec | Logical Writes / sec |
|---------|--|------------------|--------------|----------------------|----------------------|----------------------|
| 1 | SELECT 'Total income is', ((OrderQty * UnitPr... | 0 | 13 | 320 | 1 | 0 |
| 2 | SELECT p.Name AS ProductName, NonDiscountSale... | 0 | 12 | 301 | 1 | 0 |
| 3 | SELECT ProductID, OrderQty, LineTotal FROM Sa... | 0 | 8 | 156 | 0 | 0 |
| 4 | SELECT p.[Name], AVG(pch.StandardCost) AS Av... | 122 | 7 | 131 | 0 | 0 |
| 5 | SELECT p1.ProductModelID FROM Production.Prod... | 81 | 3 | 60 | 0 | 0 |
| 6 | SELECT Name, AVG(ListPrice) AS 'Average List ... | 206 | 1 | 28 | 0 | 0 |
| 7 | SELECT ProductID, AVG(OrderQty) AS AverageQua... | 0 | 1 | 19 | 0 | 0 |
| 8 | SELECT ProductModelID, GROUPING(ProductModelI... | 1 | 0 | 14 | 0 | 0 |
| 9 | SELECT ProductID, LineTotal FROM Sales.SalesO... | 0 | 0 | 12 | 0 | 0 |
| 10 | SELECT ProductID, SUM(LineTotal) AS Total FRO... | 0 | 0 | 10 | 0 | 0 |

Figure 6: Use the Query Statistics History report to help you identify queries that use a lot of SQL Server resources.

While there is not a lot of activity shown in the above example, what you see are the top 10 queries that use the most CPU resources over the time range that has been selected. (Note: you can go back onto history as much as 14 days—by default—to view this data). You have the option to sort this data by CPU usage, Duration, Total I/O, Physical Reads, and Logical Writes. Once you have identified the most resource-consuming queries, you can drill down into each one, which provides you additional detailed information about the query, including its entire code and a graphical execution plan.

The third report is the Server Activity History report.

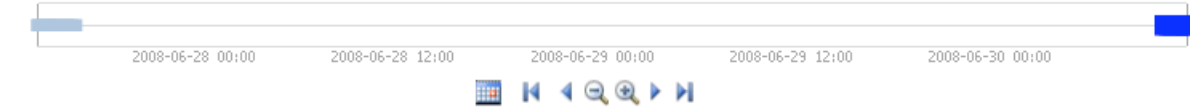
Server Activity History

on PELE at 6/30/2008 11:47:55 AM

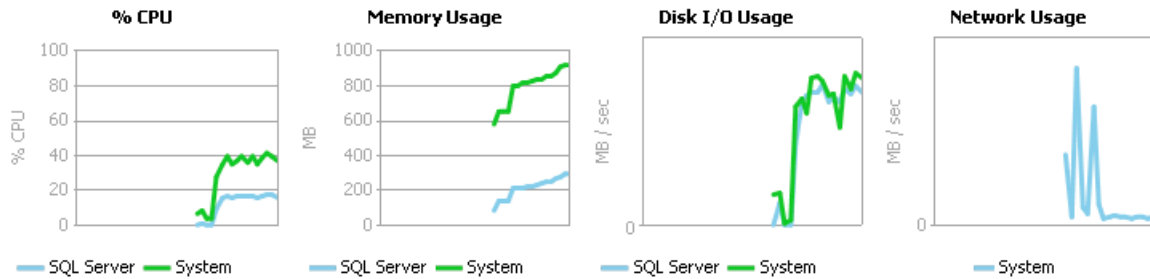


This report provides an overview of resource consumption and server activity for the SQL Server instance and for the host operating system.

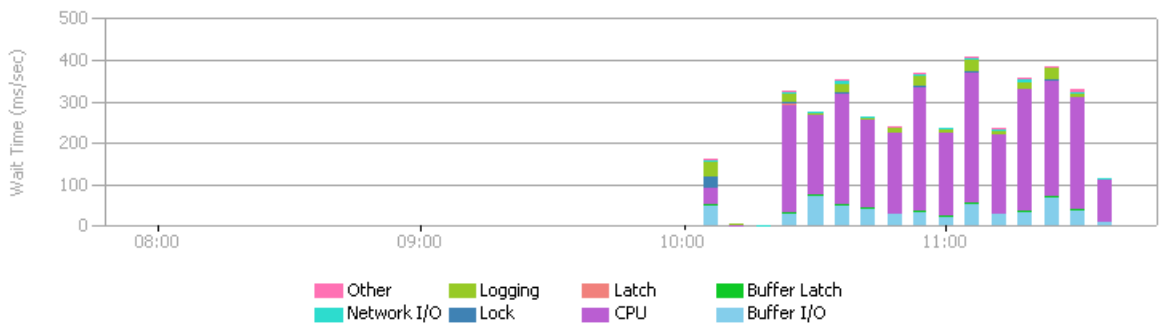
Navigate through the historical snapshots of data using the time line below.



Selected time range: 6/30/2008 7:47:55 AM to 6/30/2008 11:47:55 AM



SQL Server Waits



SQL Server Activity

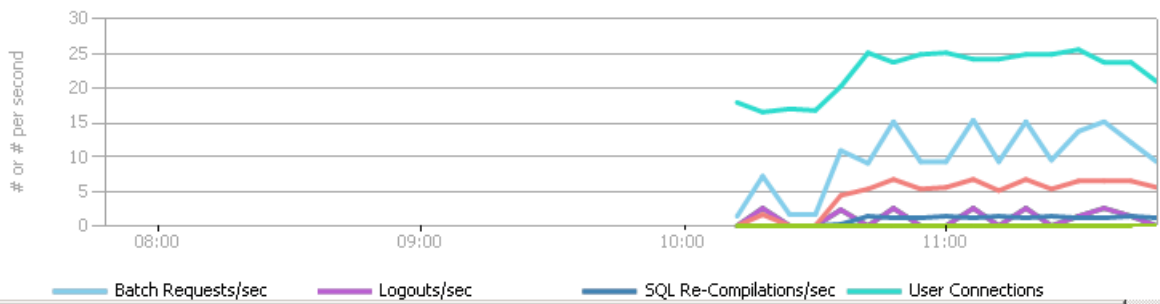


Figure 7: The Server Activity History report includes both Performance Monitor Counters and wait state information from DMVs.

Of all the reports, this one provides the greatest range of data. You can not only track basic hardware resources, such as CPU usage, memory usage, Disk I/O usage, and network usage, you also can view the most active SQL Server wait states; plus SQL Server activity, such as Logins/Second, Transactions, User Connections, and so on. You can drill down into almost any information on the screen, providing even more detail.

Summary

In this chapter, we have learned that the SQL Server 2008 Performance Data Collector allows us to create a central repository to store performance data; that it includes three built-in Data Collection sets that can be used to collect and store performance data; and that we can use any of the three built-in reports to access the stored data in order to help us to identify and troubleshoot SQL Server performance-related problems.

One thing I want to emphasize is that you will want to test this option before you decide to turn it in for your production servers. The reason for this is that you must first evaluate whether this tool provides you with the information you want, and second, if you are willing to accept the overhead that goes along with using it. If you say yes to both, then go ahead and install this on your new SQL Server 2008 production servers.

CHAPTER 6: TRANSPARENT DATA ENCRYPTION

Whether we like it or not, DBAs are becoming security experts. It's not a job we want, but it's been thrust upon us as we are the protectors of the organization's data. Whether required by law, or just for self-protection, more and more of the data in our databases need to be encrypted.

In SQL Server 2000 and earlier, if we wanted to encrypt data in our databases, this usually meant client-side encryption, where all the encryption and decryption occurred in the application, not in the database. This required custom-written applications.

In SQL Server 2005, column-level (sometimes called cell-level) encryption became available. Now, encryption could occur within the database, but it was not easy to use, offered poor performance, and it required a re-architecture of the application, along with changes to the database schema.

In SQL Server 2008 (Enterprise Edition **only**), a new form of database encryption has been introduced: Transparent Data Encryption (TDE), which includes several major features that overcome many of the drawbacks associated with the 2005 version, as well as considerably improving performance, and ease of use and administration.

This chapter considers why you might consider using TDE, its limitations, and a first look at its architecture, and how to implement it.

Why Use Transparent Data Encryption?

Some interest was shown in SQL 2005 encryption, due to the advantages that column-level encryption offered, such as:

- Granular security
- Data is encrypted in memory and disk
- Explicit key management, which allows different users to protect their own data using their own keys, even preventing the DBA from seeing a user's data.

Even so, the disadvantages were so great that only the most sensitive columns of a table were generally encrypted, which meant that much of the data in a database was still left unencrypted. Column-level encryption is still available in SQL 2008 and has been improved in that it can now, via a new Extensible Key Management (EKM) feature, allow use of encryption keys stored on third party security systems.

However, the major advance is the introduction, with Transparent Data Encryption, of full database-level encryption. TDE offers:

- **Encryption of the Entire Database:** With essentially a flip of a switch, the entire contents of MDF files, LDF files, snapshots, tempdb, and backups are encrypted. Encryption occurs in real-time as data is written from memory to disk, and decryption occurs when data is read from disk and moved into memory. Encryption is done at the database level, so you can choose to encrypt as few or as many databases as you want.
- **Ease of Implementation and Administration:** As its name implies, Transparent Data Encryption is transparent to applications. This means that your applications, and database schema, don't have to be modified to take advantage of TDE. In addition, initial setup and key management is simple and requires little ongoing maintenance.

- **Use of Minimal Server Resources to Encrypt Data:** While additional CPU resources are required to implement TDE, overall, it offers much better performance than column-level encryption. The performance hit, according to Microsoft, averages only about 3-5%.

The major benefit of encrypting a database with TDE is that if a database or backup is stolen, it can't be attached or restored to another server without the original encryption certificate and master key. This prevents those nasty situations you hear about in the news where a backup of a database has been shipped from one location to another and is "lost", which potentially exposes a company to liability issues.

Limitations of TDE

While TDE offers many benefits over other types of encryption, it has some of its own limitations, which are important to consider. These include the following:

- TDE does not protect data in memory, so sensitive data can be seen by anyone who has DBO rights to a database, or SA rights to the SQL Server instance. In other words, TDE cannot prevent DBAs from viewing any data they want to see.
- TDE is not granular. The entire database is encrypted.
- TDE does not protect communications between client applications and SQL Server, so other encryption methods must be used to protect data flowing over the network.
- FILESTREAM data is not encrypted.
- When any one database on a SQL Server instance has TDE turned on, then the tempdb database is automatically encrypted, which can contribute to poor performance for both encrypted and non-encrypted databases running on the same instance.
- Although fewer resources are required to implement TDE than column-level encryption, it still incurs some overhead, which may prevent it from being used on SQL Servers that are experiencing CPU bottlenecks.
- Databases encrypted with TDE can't take advantage of SQL Server 2008's new backup compression. If you want to take advantage of both backup compression and encryption, you will have to use a third-party application, such as SQL Backup, which allows you to perform both of these tasks without penalty.

Some organizations might want to consider implementing both column-level encryption along with TDE for a database. While more complex to set up and administer, this combination offers greater security and encryption granularity than does either method used alone.

How Transparent Data Encryption Works

TDE is able to minimize resource utilization and hide its activities from user applications, and the Relational Database Engine, because all encryption/decryption occurs when data pages are moved between the buffer pool and disk. Figure 1 illustrates how TDE works.

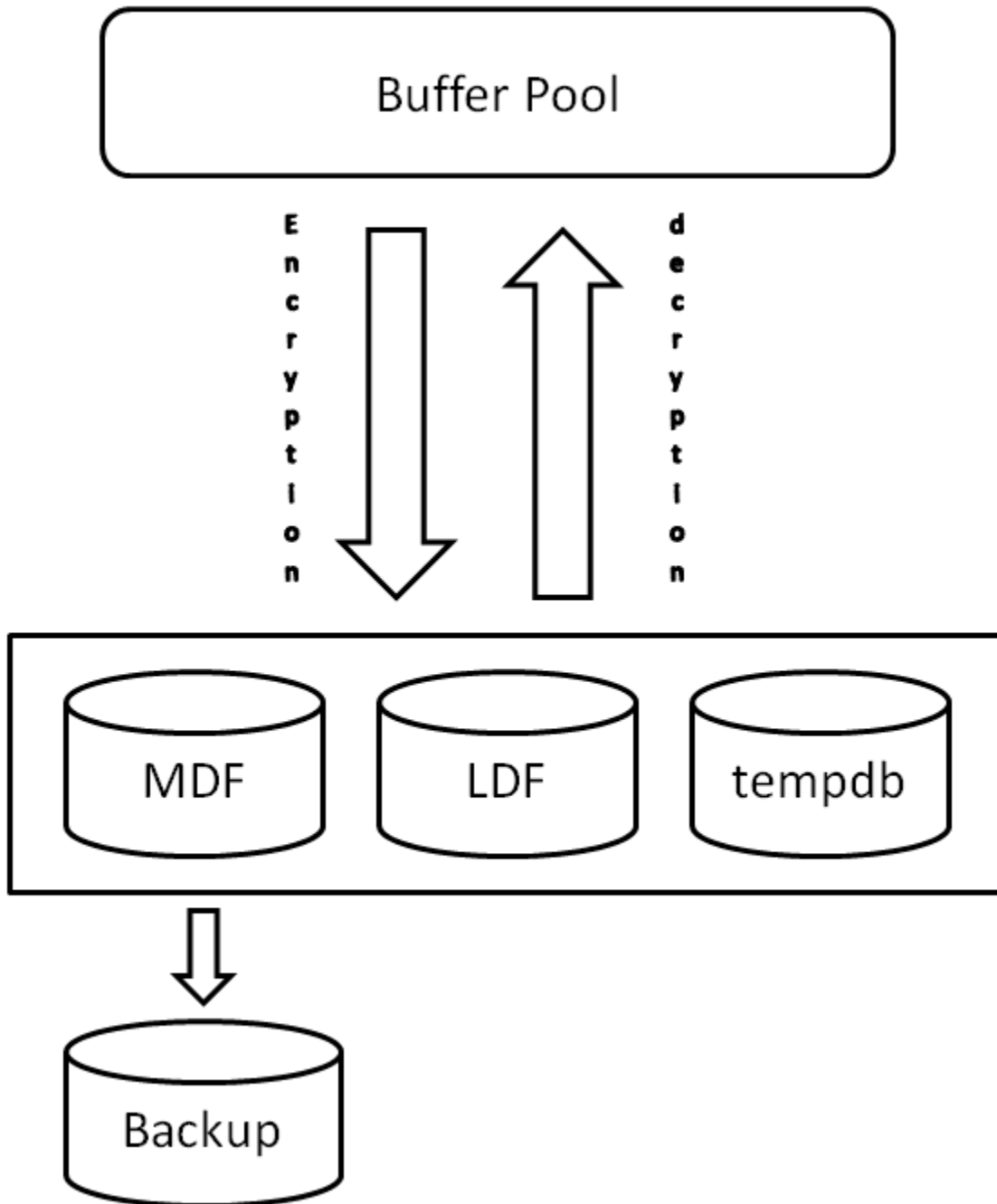


Figure 1: With TDE, encryption/decryption occurs when data pages are moved between disk and the buffer pool.

Let's say that TDE has been turned on for a database that includes a single MDF file, a single LDF file, and tempdb. As I mentioned earlier, whenever any database on a SQL Server instance is encrypted using TDE, then the tempdb database for that instance is also encrypted.

As SQL Server moves data pages from the buffer pool to the MDF file, the LDF file, or tempdb, the data is encrypted in real-time before it is written to disk. On the flip side, as data pages are moved from the MDF file or tempdb to the buffer pool, they are decrypted. In other words, when data is on disk, it is encrypted, but when data in memory, it is not encrypted.

When a backup is made of an encrypted database, it cannot be restored unless the DBA has access to the certificate and master key that was used to encrypt the database. This prevents anyone from

stealing a backup and restoring it on a different SQL Server. The same goes for when you detach and reattach a database to a different SQL Server.

TDE supports several different encryption options, such as AES with 128-bit, 192-bit, or 256-bit keys or 3 Key Triple DES. You make your choice when implementing TDE.

How to Implement Transparent Data Encryption

In this section, we'll take a brief look at how to turn on TDE for a database. This can only be done using Transact-SQL code, as SSMS has no option to perform this task. Before we drill down into the details, let's look at the five key steps required to turn on TDE:

- **Create a Master Key:** A master key is first created. This key, which is accessible with a password, is used to protect a certificate, which we will create in the next step. This key is stored in the master database in an encrypted format.
- **Create or Obtain a Certificate Protected by the Master Key:** This certificate is used to protect the database encryption key we will create in the next step. In addition, this certificate is protected by the master key we created in the previous step. The certificate is stored in the master database in an encrypted format.
- **Create a Database Encryption Key:** This is the key that will be used by SQL Server to actually encrypt the data. It is protected by the certificate created in the previous step. This key is stored in the database that is encrypted, and is stored in an encrypted format.
- **Backup the Encryption Key and Certificate:** One of the most important steps, once the key and certificate have been created, they must be backed up and stored securely.
- **Turn TDE On:** Once all the above has been created, a command is run to tell SQL Server to begin encrypting all of the data using the database encryption key created in the previous step. This process may take some time, depending on the size of the database. Ideally, the database should not be used in production until the database has completed the initial encryption process.

Creating a Master Key

Assuming one has not already been created for another reason, the first step is to create a master key. The master key is a symmetric key used to protect the private keys of certificates and asymmetric keys. In this particular case, the master key is used to protect the certificate which will be created in the next step. When a master key is created, it is encrypted using the Triple DES encryption method and protected by a user-provided password.

To create a master key, run the following code:

```
--Create a master key
--The master key must be created within the master database
USE master ;
CREATE MASTER KEY ENCRYPTION BY PASSWORD =
    'User-Provided Password' ;
GO
```

Obviously, the password you provide should be very obscure, and you will want to document in a secure location.

Create or Obtain a Certificate Protected by the Master Key

For this example, we are going to create a new certificate, although you can use a pre-existing certificate if available. The certificate is used to protect the database encryption key that we will create next. In addition, this certificate is protected by the master key created in the previous step.

```
--Create a certificate by the name of TDECert
USE master ;
CREATE CERTIFICATE TDECert WITH SUBJECT = 'TDE Certificate' ;
GO
```

Create a Database Encryption Key

Now that the certificate has been created, the next step is to create a database encryption key and protect it with the certificate we created in the last step. This is the encryption key that is used by the database to encrypt all of the data. It is during this step that you choose which encryption method is used to encrypt your database.

```
--Create Database Encryption Key Inside Database to Be Encrypted,
--and Protect It with the Certificate
USE AdventureWorks ;
GO
CREATE DATABASE ENCRYPTION KEY WITH
    ALGORITHM = AES_256 ENCRYPTION BY
    SERVER CERTIFICATE TDECert ;
GO
```

Backup the Private Encryption Key and Certificate

Once you have created the master key and certificate, they should be backed up immediately. If you lose these, you can't move or restore the database.

```
--Backup the private key and certificate to two separate disk files
USE master ;
GO
BACKUP CERTIFICATE TDECert TO FILE =
    'c:\certificate_backups\AdventureWorks_Certificate.cer'
WITH PRIVATE KEY ( FILE =
    'c:\certificate_backups\NorthwindCert_Key.pvk',
    ENCRYPTION BY PASSWORD =
    'User-Provided Password' ) ;
GO
```

When this command is run, the master key and the certificate are taken from the master database and written to separate files (both in an encrypted format).

Turn TDE On

The last step is to turn TDE on. Once you run the following command, the database will begin to encrypt itself. Depending on the size of the database, and the hardware running it, this process could be lengthy. While it is possible to keep the database in production during this process, it will cause some user blocking and performance will suffer. Because of this, ideally you should only turn TDE on when the database is not being used.

```
--Turn TDE on
USE AdventureWorks
ALTER DATABASE AdventureWorks SET
```

```
    ENCRYPTION ON ;  
GO  
If you want to watch the progress of the encryption, run this statement:  
SELECT  DB_NAME(database_id),  
        encryption_state  
FROM    sys.dm_database_encryption_keys ;  
GO
```

When the statement above is run, a state is returned. A database encryption state of "2" means that encryption has begun, and an encryption state of "3" indicates that encryption has completed. Once the tempdb database and the user database you are encrypting reach a state of "3," you are ready to put them back into production. From this point on, the entire user database, and tempdb database will be encrypted, although your applications will never know the difference.

Summary

If you got lost with all the keys and certificates required to implement TDE, you are not alone. It is a complex topic and beyond the scope of this chapter. The focus of this chapter was to provide you an overview of what TDE is, how it works, and how to implement it. Because of the complexity involved in using TDE, you should only implement this technology when you full understand its many complexities and after thorough testing in a test environment.

CHAPTER 7: SQL SERVER AUDIT

Previous versions of SQL Server have included a variety of built-in ways to audit activity inside SQL Server. These included:

- **Login Auditing:** Only tracks user login successes and/ or failures. Results are sent to the OS Application Log.
- **SQL Server Profiler (SQL Trace) Audit and Related Events:** Profiler includes over 40 specific audit events, and over 130 additional events that can be traced. Results are stored in trace files
- **DDL Triggers:** DDL triggers can be added to a database to identify when any DDL event occurs. These must be coded, including writing code to store the results for later retrieval.
- **C2 Audit Mode:** This former auditing standard (now superseded by Common Criteria Compliance) uses SQL Trace to capture audit events, which are stored in trace files.
- **Common Criteria Compliance:** A new international auditing standard which also uses SQL Trace to capture data, which are stored in trace files.

None of the above methods offer much granularity, most are not easy to administer, and with the exception of Login Auditing, they can add a large amount of overhead to running SQL Server, hurting its performance.

To help overcome these problems, SQL Server 2008 includes a new feature called SQL Server Audit. SQL Server 2008 Enterprise Edition includes all of the features described in this chapter, which includes both the SQL Auditing Foundation and Fine Grained Auditing. SQL Server 2008 Standard Edition only provides the SQL Auditing Foundation.

Advantages of SQL Server Audit

SQL Server Audit includes these features and benefits:

- **The Ability to Audit at the Instance and Database Levels:** When you configure an audit, you are given the ability to capture audit events at the instance-level or the database-level. In most cases, you will probably want to audit events at both levels.
- **The Ability to Audit Many Different Activities:** SQL Server Audit includes many pre-defined activities that you can audit, including DML and DDL activity. In addition, you can even audit "audit activities". In other words, the activities of DBAs, whose job it is to manage SQL Server Audit, can be monitored by outside parties, if so desired.
- **The Ability to Capture and View Audit Results:** Audit results can be captured and stored in disk files, or in the operating system's Event Logs. Data on disk can be imported into a database for further analysis and reporting. Data stored in Event Logs can be viewed using the Event Log Viewer.
- **High Granularity:** SELECT, INSERT, UPDATE, DELETE, REFERENCES and EXECUTE statements can be captured for individual users at the object level, if desired.
- **Fast and Lightweight:** SQL Server Audit uses SQL Server 2008's new **Extended Events** engine to capture audit data. This results in fast performance and minimal overhead as compared to using SQL Trace to capture activity.
- **Easy to Setup and Administer:** SQL Server Audit can be setup and managed using either SSMS (SQL Server Management Studio) or Transact-SQL.

Limitations of SQL Server Audit

While SQL Server Audit includes many nice features, it also has some drawbacks you need to consider:

- While SQL Server Audit takes up less physical resources than SQL Trace-based auditing options, it still uses SQL Server resources, and the more detailed your auditing, the more resources that are used. Because of this, it may not be practical to use SQL Server Audit on very busy OLTP servers, especially if they are already experiencing hardware bottlenecks.
- SQL Server Audit is instance-based. In other words, there is no easy way to manage SQL Server Audit on all the SQL Server instances in your organization from a centralized location, unless you create your own method using Transact-SQL scripts.
- Audit data is stored either in a file, or as part of the operating system's event logs. If you want to be able to analyze and report on this data, you will have to manually import it into your own database. In addition, DBAs will have to manually archive or delete old audit data.
- There is no built-in reporting, other than looking at the events in the Log Viewer, assuming that is where you store the audit data. For effective reporting, you will have to create your own reports, most likely using SQL Server Reporting Services.

How SQL Server Audit Works

When you first begin using SQL Server Audit, you may find it somewhat unintuitive and a little confusing. In this section, I want to start with a high-level overview of how it works. In the following section, I will provide a short demonstration of it in action, so you can better see how it all fits together. The flow chart shown in Figure 1 should provide a broad overview of what's involved in setting up auditing:

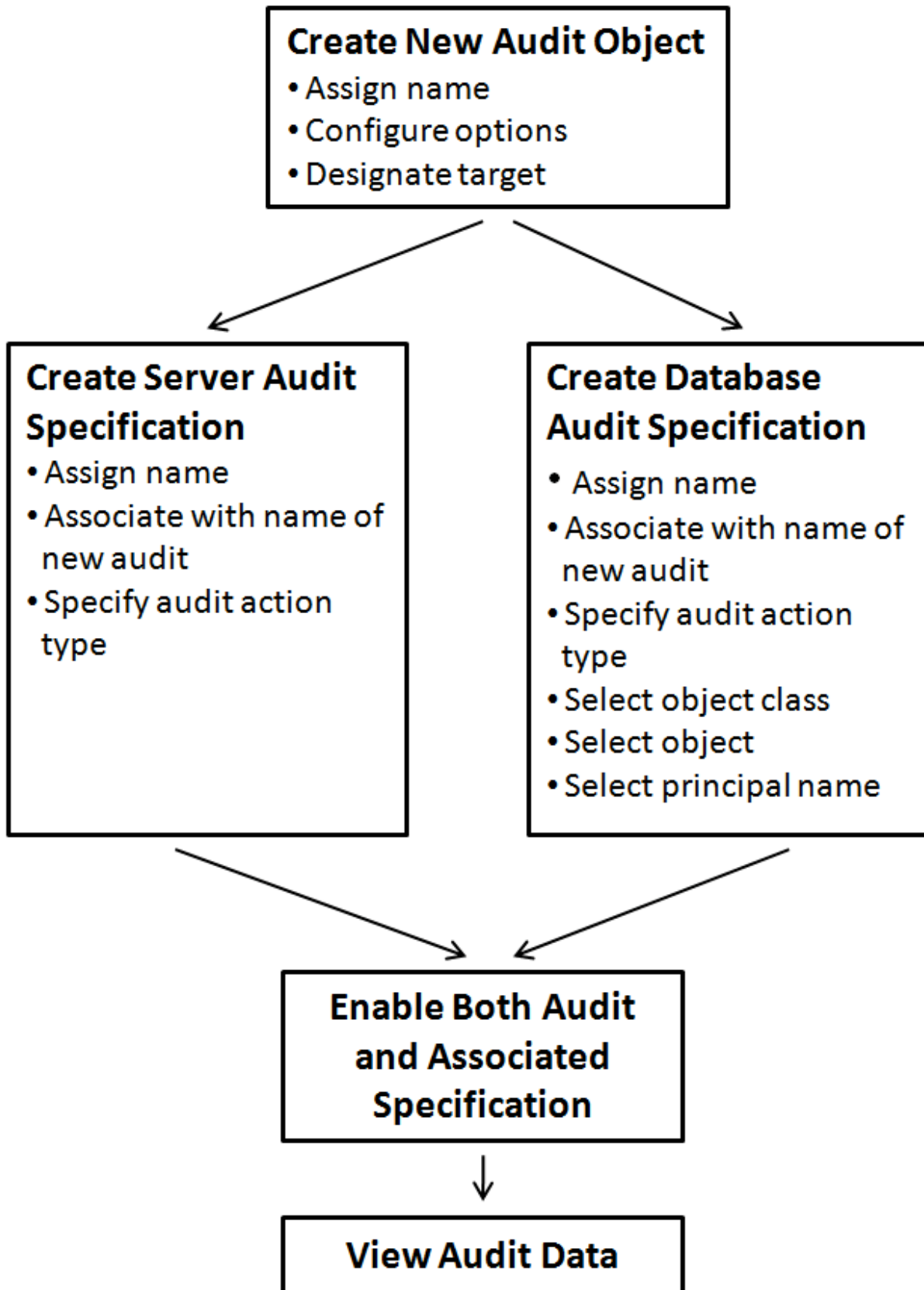


Figure 1: A flow chart showing how to create a new SQL Server Audit

SQL Server Audit allows you to create many different audits, covering most every activity that occurs inside SQL Server.

The first step when creating a new audit is to create a **SQL Server Audit object**. When you create a new SQL Server Audit object, you assign it a name, select from several configuration options, and you designate a target. A target is the location where the audit data will be stored. A target can be a file on disk, the Applications Event Log, or the Security Event Log. Once you have completed these steps, the new SQL Server Audit object is saved.

The second step is to create what is called an **Audit Specification**. SQL Server Audit offers two different types of Audit Specifications:

1. **Server Audit Specifications** - used when you want to audit an activity that occurs at the SQL Server instance level, such as auditing login and logout activity.
2. **Database Audit Specifications** - used when you want to audit an activity within a database, such as who is SELECTing data from a particular table.

Server and Database Audit Specifications are created differently so you need to decide which type you need up-front. When creating either type of Audit Specification, you first assign it a name, and then you must associate the Audit Specification with the SQL Server Audit object created in the first step. The rule is that a SQL Server Audit object can only be associated with one Audit Specification. In other words, you can't reuse SQL Server Audit objects when you create Audit Specifications. And the last step to creating a Server Audit Specification is to assign it an Audit Action Type. An Audit Action Type is a predefined activity that occurs in SQL Server that can be audited.

When creating a Database Audit Specification, you assign it a name; then you associate the Audit Specification with a SQL Server Audit object; and specify an Audit Action Type, just as you do with a Server Audit Specification. However, in addition, you must also specify an Object Class (database, object, or schema), the name of an object to audit, and the security principal (the user account) that you want to audit.

Once the Audit Specification is completed, you must then enable both the SQL Server Audit Object and its associated Audit Specification. At this point, audit events will begin to be collected in the designated target.

And last, to view the audit data, you have several choices. If you store audit data in a file, you can import it into a database for viewing and reporting. If you store it in one of the two Event Logs, you can view it using the Event Log Reader.

I have left out a lot of details, but this picture and explanation should give you a fair understanding of how the overall process works.

A Simple Auditing Example

SQL Server Audit can be configured and managed using either SQL Server Management Studio (SSMS) or Transact-SQL commands. In this simple demonstration, we will use SSMS to create a simple audit because it is easier to understand for DBAs new to SQL Server Audit.

Creating an audit, and reviewing audit results using SSMS, is a four-step process, as outlined in the previous section:

1. Creating the Audit object
2. Creating a Server or Database Audit Specification
3. Starting the Audit
4. Reviewing Audit Events

In the following example, we want to find out who is looking at the `HumanResources.EmployeePayHistory` table in the `AdventureWorks` database. In other words, we want an audit trail of everybody who runs a `SELECT` statement against this table. Obviously, in the real world, your audit would be more comprehensive, but my goal here is only to provide a simple yet illustrative demonstration of how auditing works.

Creating the Audit Object

The first step is to create a new audit object. To create a new audit object using SSMS, go to the SQL Server instance you want to audit, open up "Security," and you will see the "Audits" folder, as shown in **Figure 2**:

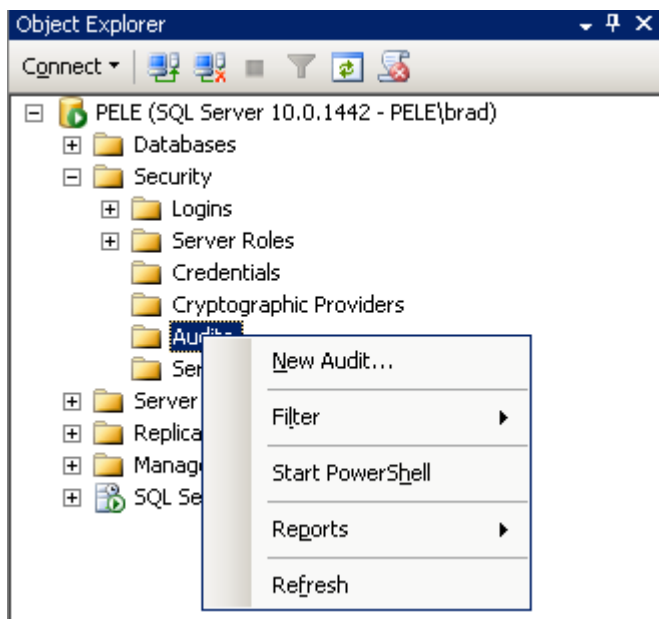


Figure 2: Choose "New Audit" to create an audit from within SSMS.

Right-click on the "Audits" folder and select "New Audit," and the "Create Audit" dialog box appears, as shown in **Figure 3**:

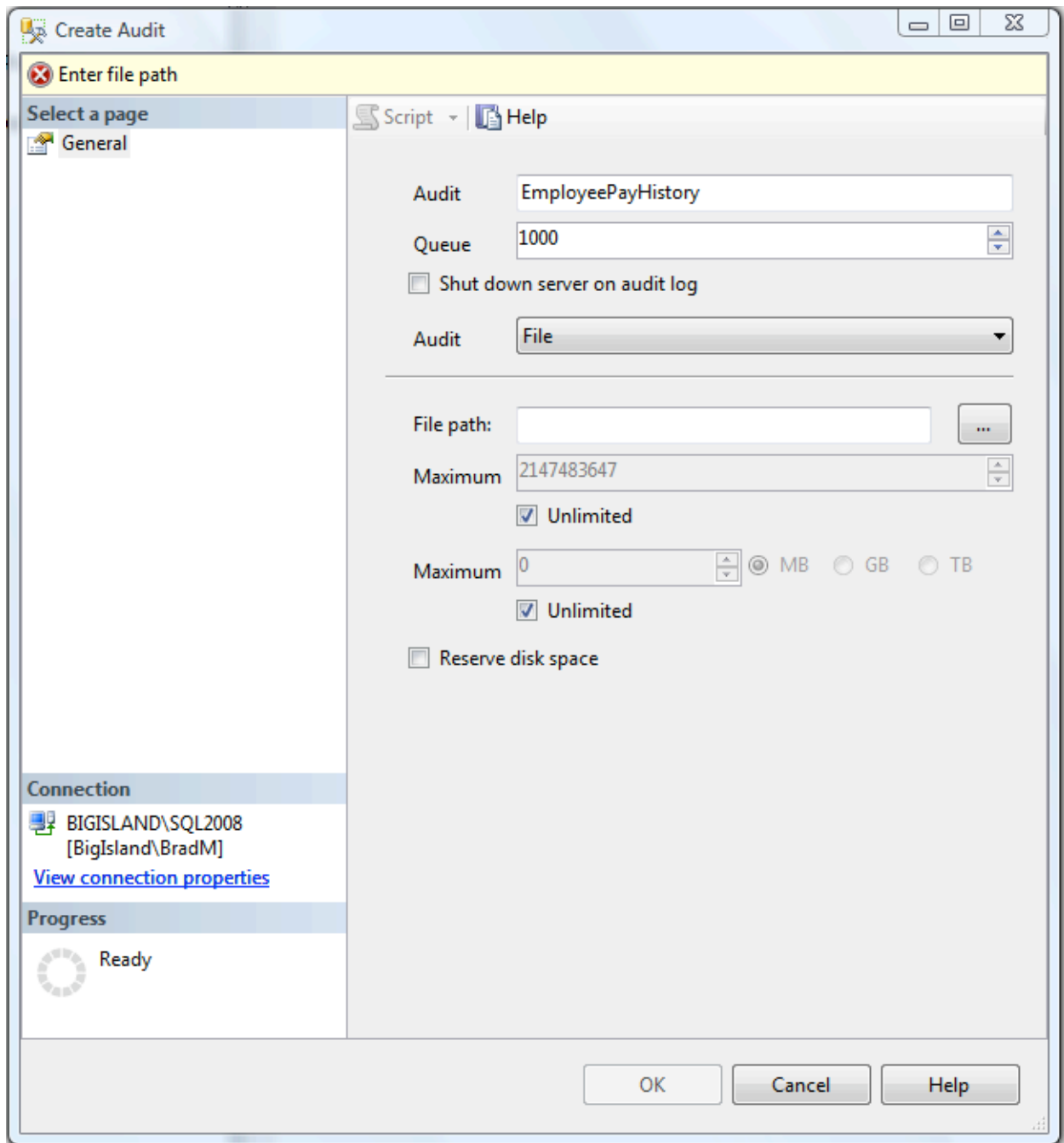


Figure 3: To create an audit, you have to assign it a name and specify where the audit data will reside.

The first thing you need to do is to decide if you want to use the name that is automatically generated for you as the audit object name, or to assign your own name. Since numbers don't mean much to me, I assigned it my own name.

Next, you have to provide a "Queue Delay" number. This refers to the amount of time after an audit event has occurred before it is forced to be processed and written to the log. The default value is 1000 milliseconds, or 1 second. While I am going to accept the default for this demo, you might want to consider increasing this value if you have a very busy server.

The next option on the screen is called "Shut down server on audit log failure". If you select this option, and later SQL Server is restarted, and for whatever reason the audit data can't be logged, then SQL Server will not start, unless you manually start it at the command line using a special

parameter. This option should only be used in environments where very tight auditing standards are followed and you have 24/7 staff available to deal with the problem, should it occur.

Next, beside "Audit," in the dialog box, there is a drop-down box with "File" selected by default. This option is used to tell SQL Server where you want the audit logs to be stored.

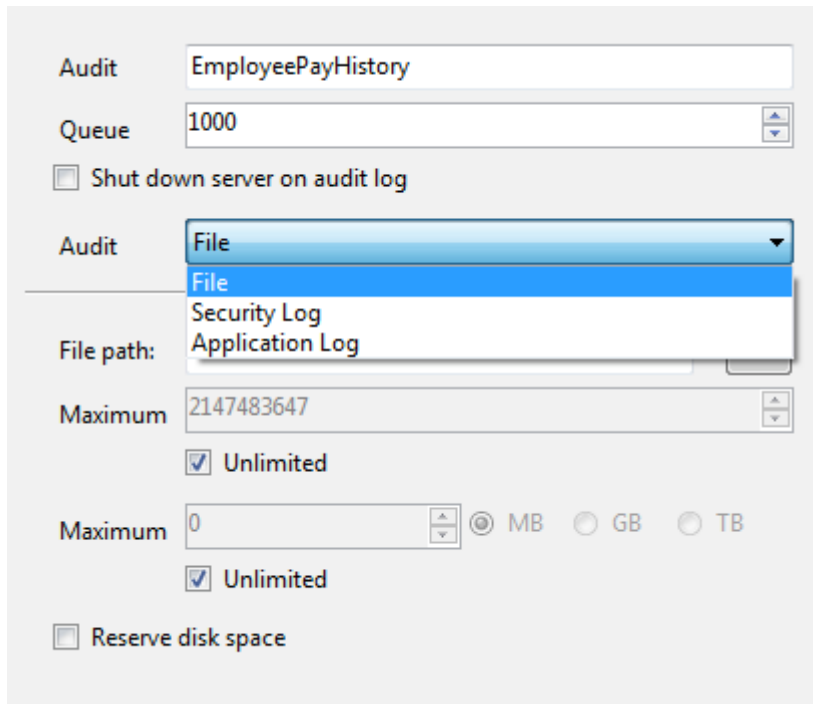


Figure 4: Three are three options where you can store audit data.

SQL Server Audit allows you to store audit data in a file, in the Security Log, or the Application Log. If you choose "File", then you must also specify the location of the file, along with additional information, such as how much data it can collect, and so on. If you choose Security Log or Application Log, then the audit results are stored in these Windows Operating System Event Logs. I am going to choose "Application Log". Once this is done, the dialog box should look as shown in **Figure 5:**

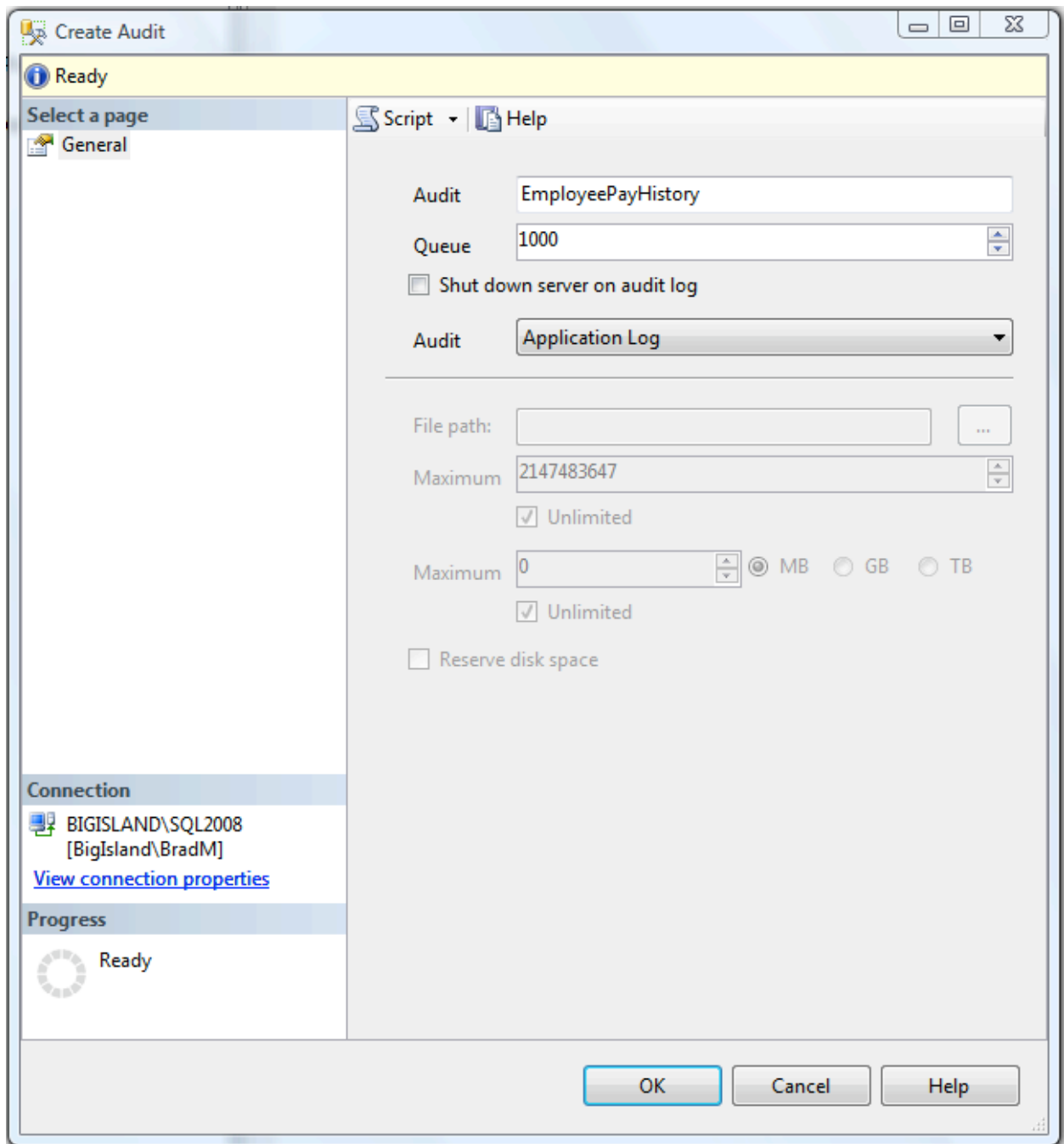


Figure 5: Once all the data has been provided, click "OK" to create the audit.

Now that the audit has been configured, click on "OK" to save it. It should then appear in the SSMS Object Browser, as shown in **Figure 6:**

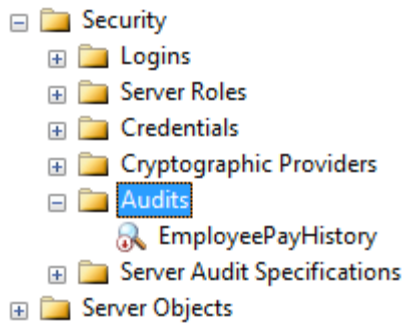


Figure 6: Notice the red arrow next to the newly created audit.

The red arrow next to the audit object means that it is not currently enabled. That's OK for now, we can enable it later.

Creating a Server or Database Audit Specification

Now that we have created the audit, we need to create the matching audit specification. If we wanted to do an instance-wide audit, we would create a server audit specification. But for this example, where the goal is to audit the SELECT activity on a single table in a single database, a database audit specification is created.

To create a database audit specification using SSMS, open up the database to be audited, then open up the security folder under it. Next, right-click on "Database Audit Specifications" and select "New Database Audit Specification", as shown in **Figure 7:**

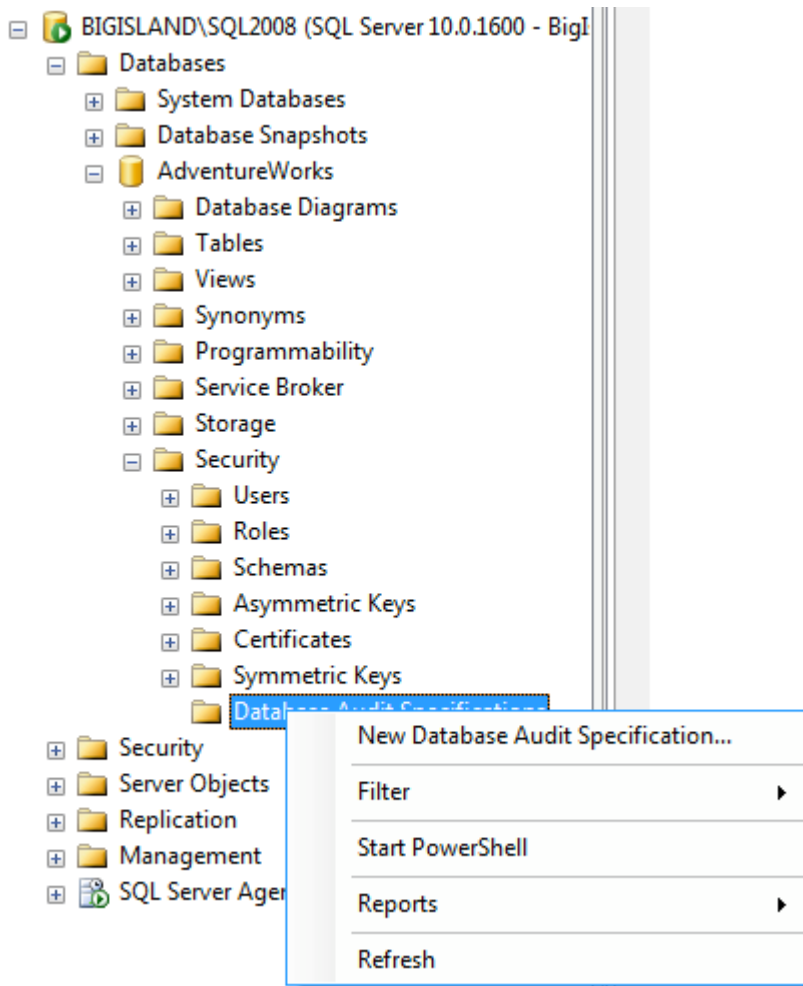


Figure 7: To create a database audit specification, you must do so from within the database you want to audit.

The "Create Database Audit Specification" dialog box appears, as shown in **Figure 8**:

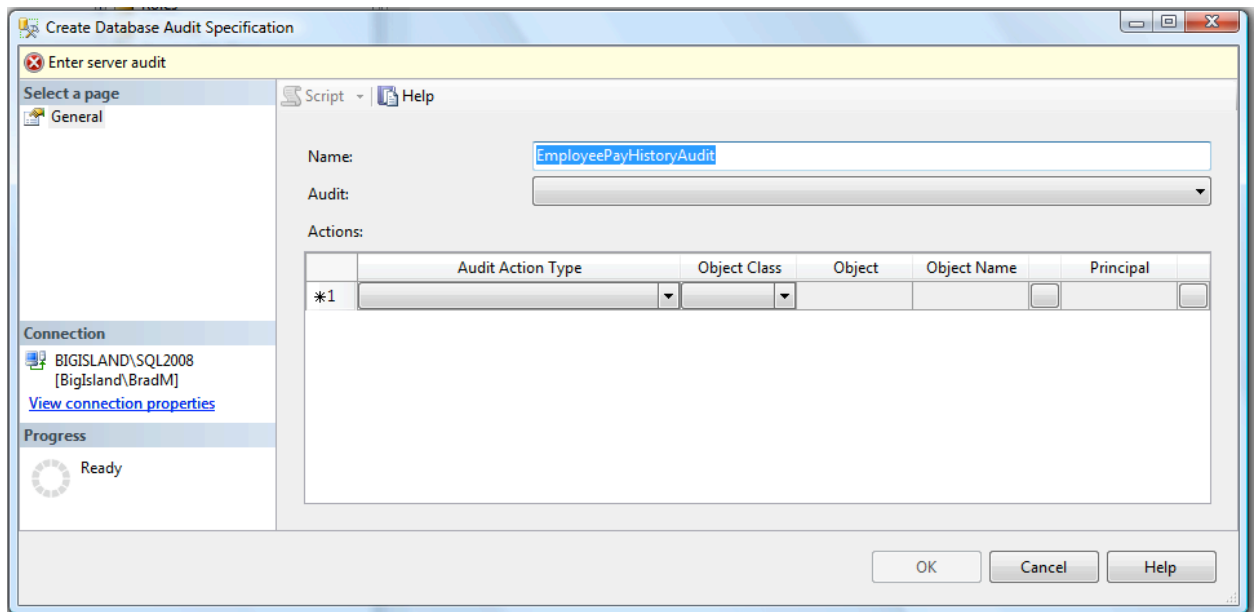


Figure 8: The "Create Database Audit Specification" dialog box has many options to complete.

You can either choose to accept the default name assigned to this database specification, or you can enter your own. Next, select the appropriate audit object from the Audit dropdown box, as shown in **Figure 9**:

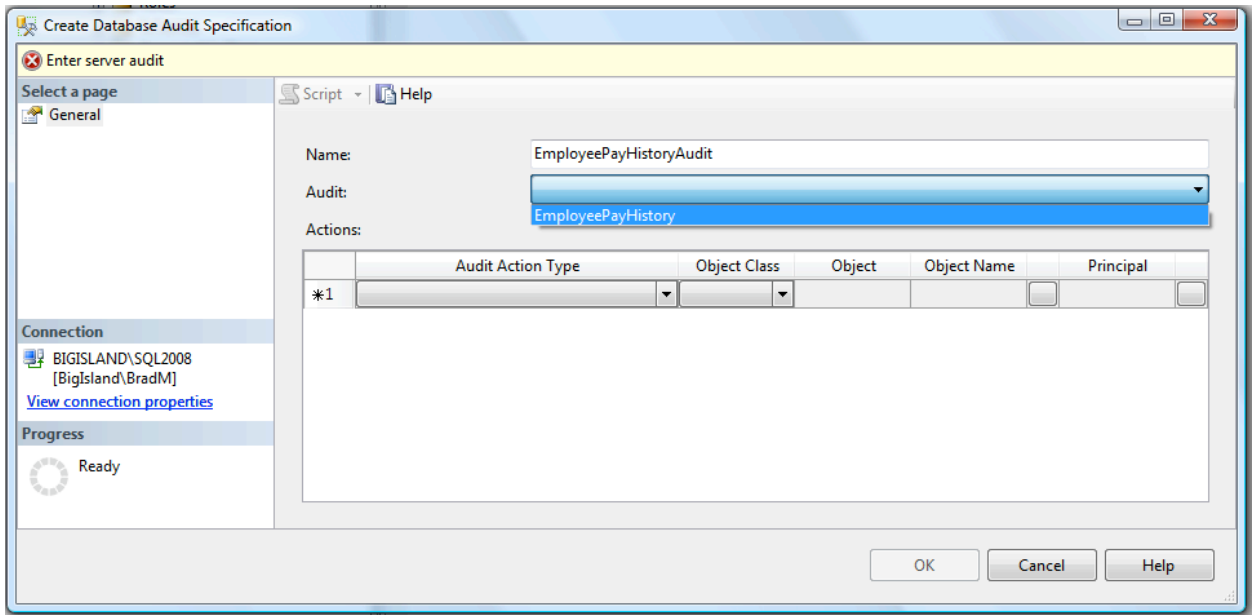


Figure 9: Every database specification has to be associated with an audit object.

In this case there is only one audit object, the "EmployeePayHistory", as this is a newly installed SQL Server and doesn't have any other audit objects on it.

Next, you must specify the kind of audit activity you want to capture by selecting from the "Audit Action Type" drop-down box, as shown in **Figure 10**:

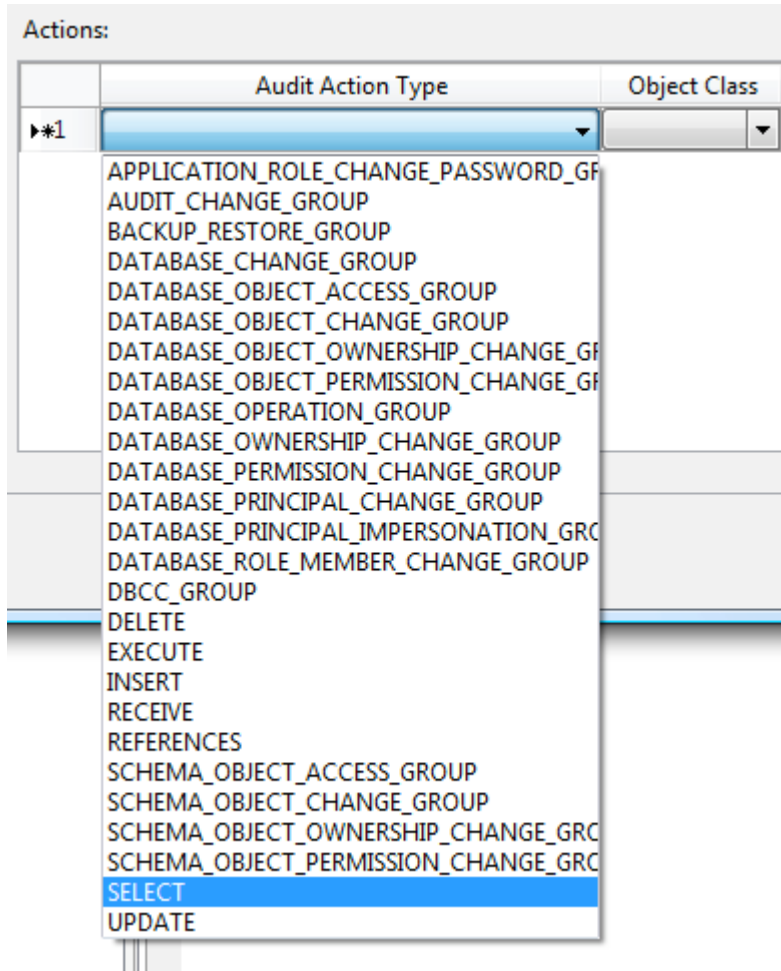


Figure 10: You can select from many pre-defined audit actions.

For this example, I want to choose the "SELECT" "Audit Action Type," as the goal is to record all SELECT activity for the payroll table. Of course, you can choose any audit action type you want, but you can only choose from those that are listed. You can't create your own.

Now that the audit action type has been chosen, the "Object Class" must be chosen – see **Figure 11:**

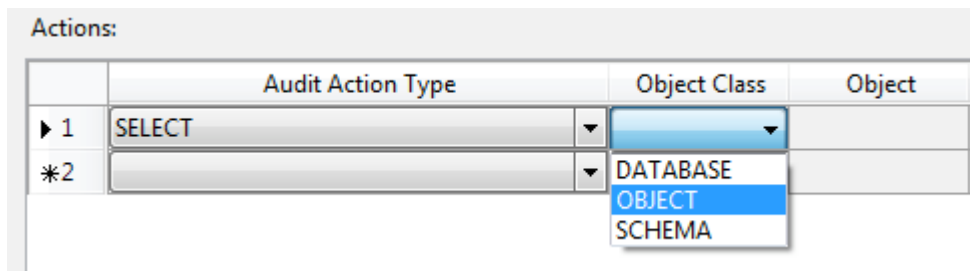


Figure 11: In this case, you can choose from three object classes.

The object class allows us to narrow down the scope of what we want to audit. For this example, because we want to monitor activity on a table, "Object" is selected.

The next step is to specify the object, or the table name, that is to be audited. To do this, click on the browse button under "Object Name," and the "Select Objects" dialog box appears, as shown in **Figure 12:**

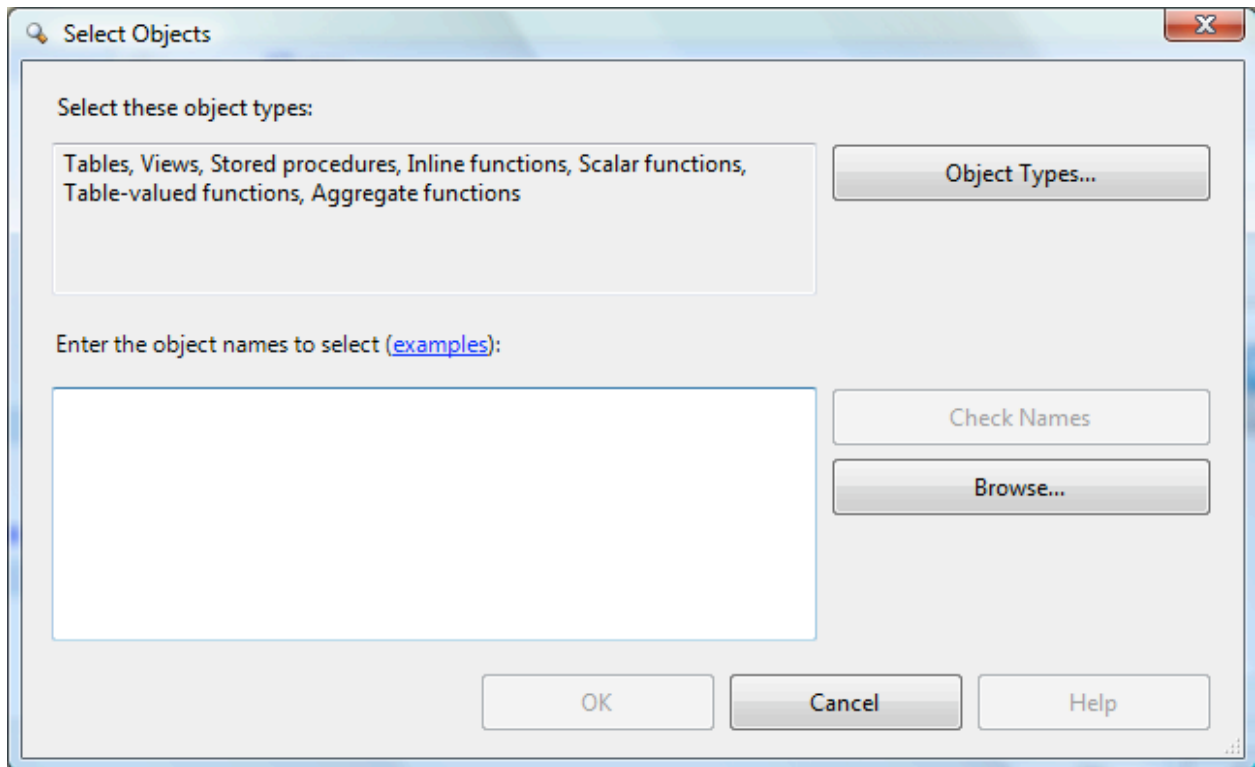


Figure 12: The "Select Objects" dialog box allows you to select which object to audit.

Having clicked on the "Browse" button, the list of available objects will appear, as shown in **Figure 13:**

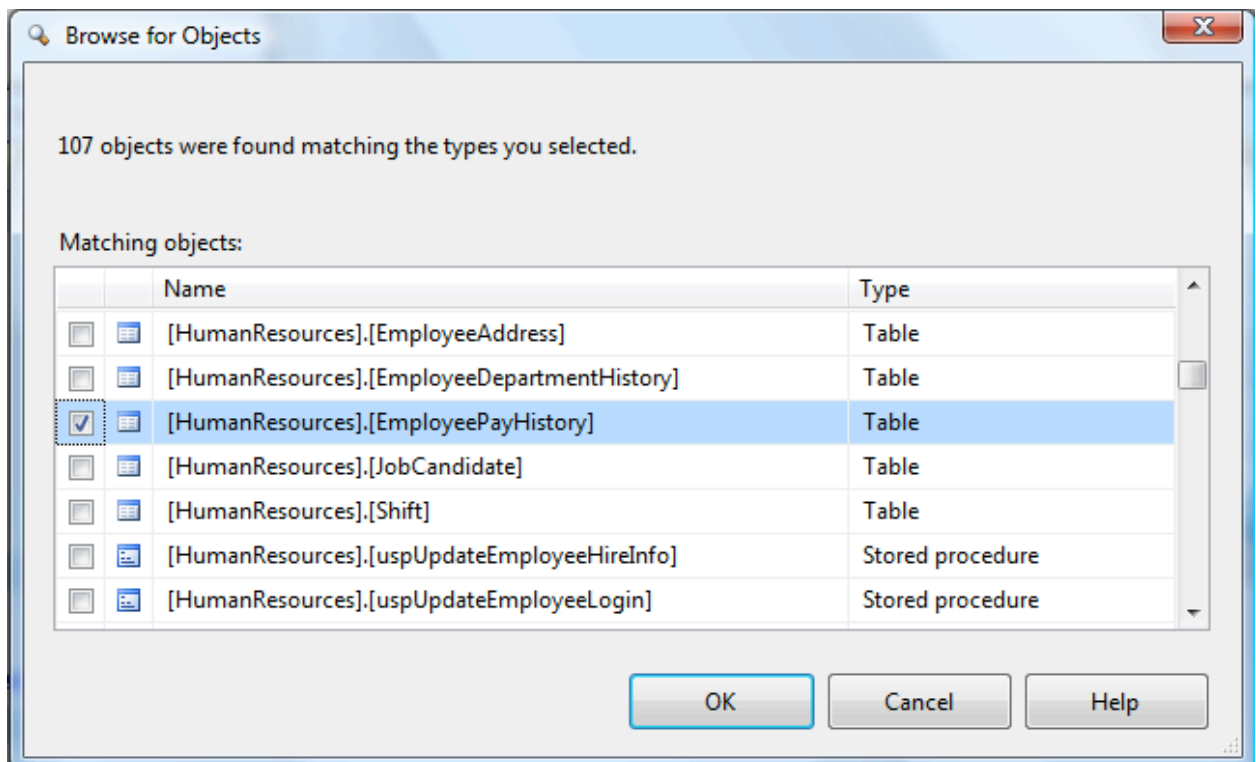


Figure 13: Select the object to be audited from this list.

Browse through the "Browse for Object" dialog box until you find the object or objects you want to audit, then select them. Above, I have selected a single table: HumanResources.EmployeePayHistory.

Once the objects have been selected, click "OK," and the "Select Object" dialog box reappears, as shown in **Figure 14**:

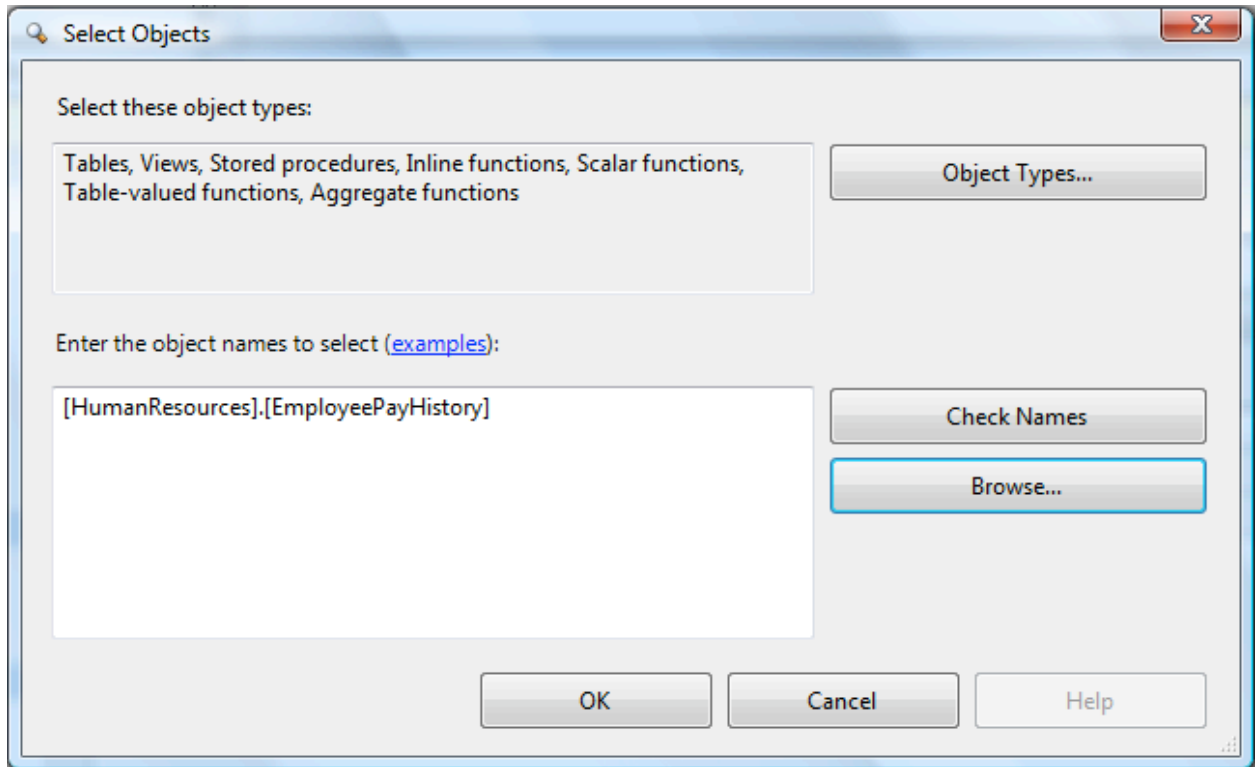


Figure 14: The audited object has been selected.

Now that the object to be audited has been selected, click "OK," and you are returned to the original "Create Database Audit Specification" dialog box, as shown in **Figure 15**:

| Audit Action Type | Object Class | Object | Object Name | Principal |
|-------------------|--------------|---------------|----------------|-----------|
| SELECT | OBJECT | HumanResou... | EmployeePay... | ... |
| | | | | |

Figure 15: We now see all of our actions up to this point.

There is one last step, and that is to specify what security principals (user accounts) that we want to monitor. To do this, click on the browse button under "Principal Name," and another "Select Object" dialog box appears.

³I am going to spare you seeing this screen again, and skip immediately to the "Browse for Object" dialog box, where you can see what principals you can choose from, as shown in **Figure 16**:

³ The number of dialog boxes you have to use in SSMS is enough to force any DBA to learn how to use Transact-SQL to perform this same task!

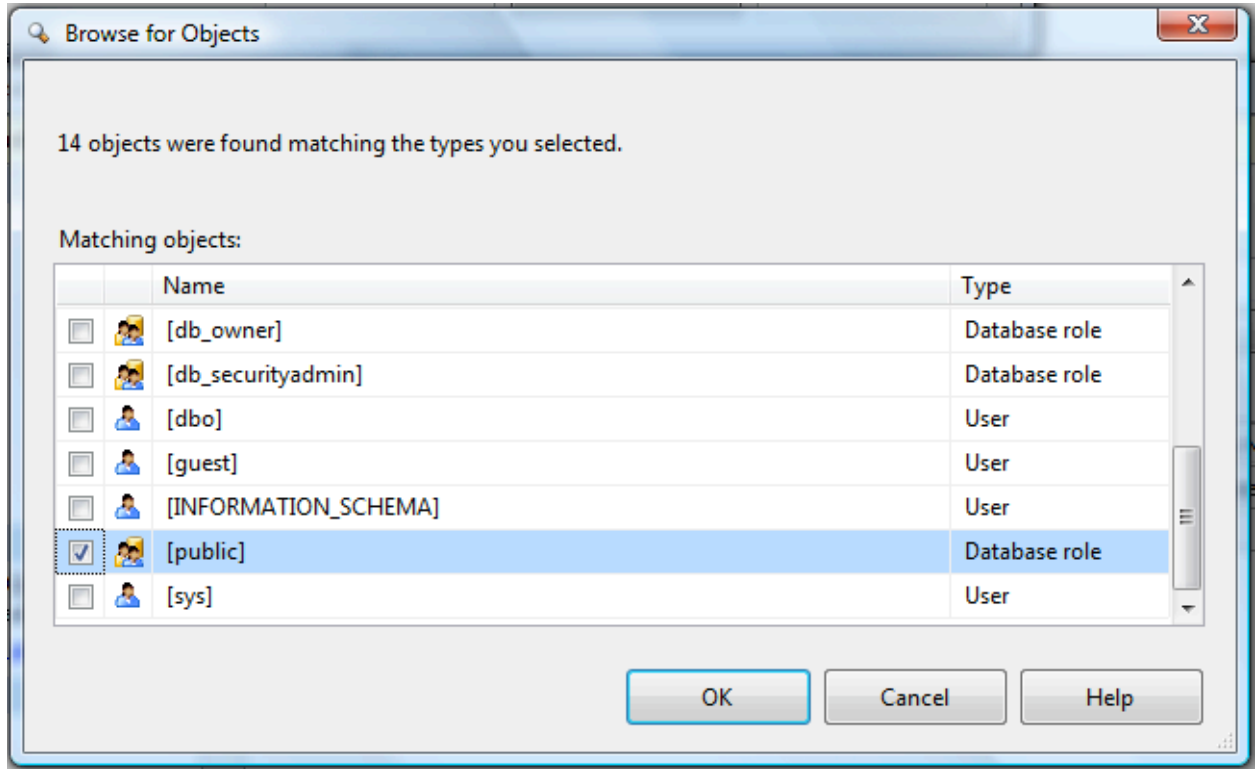


Figure 16: Select the principal you want to audit.

In this case, public is chosen, because the goal of this audit is to identify *anyone* who runs a SELECT against the payroll table. Optionally, you can select on specific users or roles. Click on "OK" for this dialog box, then click on "OK" for the "Select Objects" dialog box, and we reach the final screen, seen on **Figure 17**:

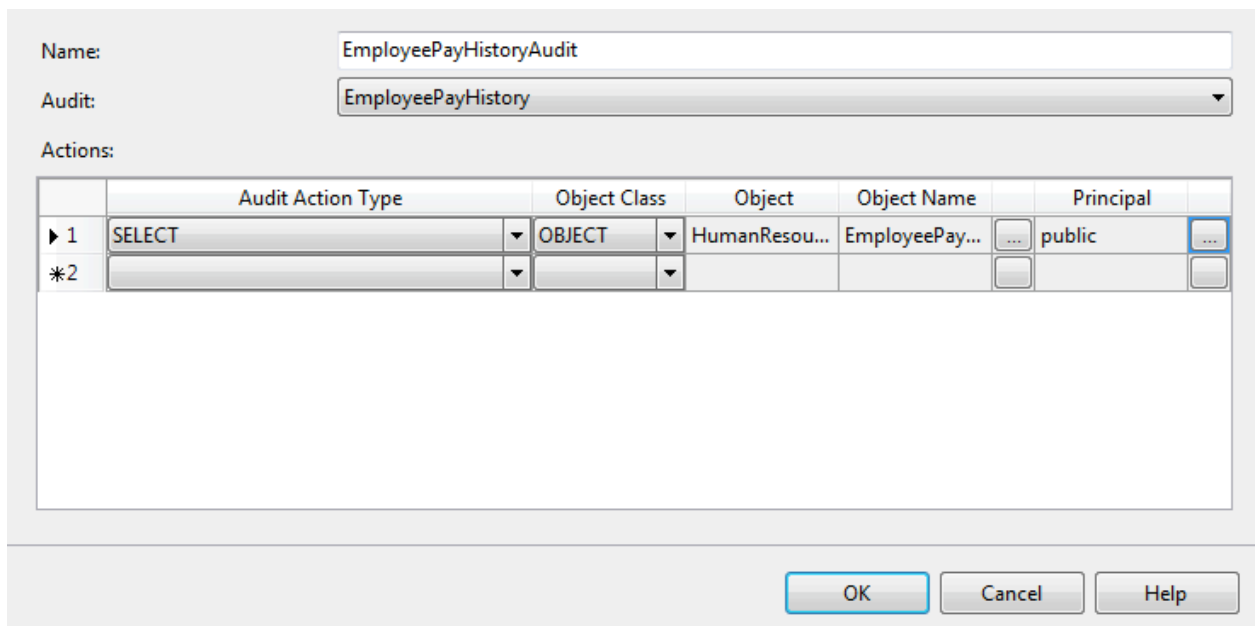


Figure 17: We are finally done creating the database audit specification.

Since we are only interested in auditing this one table for a single action, we will stop now. If you wanted to, you could continue to add addition actions and objects to this audit specification. Click on

"OK," and the database Audit Specification will be saved, and you can view it in object explorer, as shown in **Figure 18**:

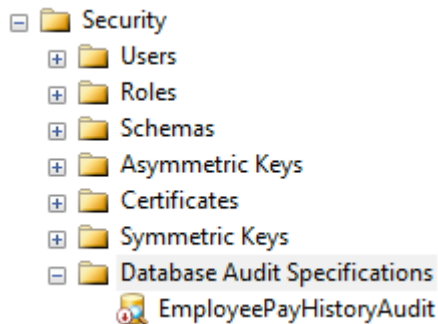


Figure 18: Notice the red arrow next to the specification, which tells us that it is turned off.

Once the new database audit specification has been created, it has a red arrow next to it, indicating that it is turned off. We will turn it on in the next step.

Starting the Audit

I have saved the steps of starting the audit till last because I wanted to show you that when you create an audit object, and a database audit specification, that they are turned off by default, and both must be turned on before audit data is collected.

First, turn on the audit object, which in our case is called "**EmployeePayHistory**," then turn on the database audit specification, which in our case is called "**EmployeePayHistoryAudit**." To turn these on, right-click on them, one at a time, and select "**Enable Audit**." Auditing has now begun. At this point, any **SELECT** statements run against the **HumanResources.EmployeePayHistory** table are recorded and stored in the Application Events log file.

Reviewing Audit Events

Now, let's see what all this work has done for us. To find out, open up the "Application Log File Viewer" and take a look. In this example, you'll see something similar to **Figure 19**:

Application 47,064 Events

| Level | Date and Time | Source | Event ID | Task Category |
|-------------|-----------------------|---------------|----------|---------------|
| Information | 8/15/2008 11:24:58 AM | MSSQL\$SQL... | 33205 | None |
| Information | 8/15/2008 11:24:58 AM | MSSQL\$SQL... | 18453 | Logon |

Event 33205, MSSQL\$SQL2008

General Details

```
Audit event: event_time:2008-08-15 21:24:58.3435098
sequence_number:1
action_id:SL
succeeded:true
permission_bitmask:1
is_column_permission:true
session_id:56
server_principal_id:261
database_principal_id:1
target_server_principal_id:0
target_database_principal_id:0
object_id:1205579333
class_type:U
session_server_principal_name:BigIsland\BradM
server_principal_name:BigIsland\BradM
server_principal_sid:010500000000005150000005c3e313390e12f73fdfa82b4fc030000
database_principal_name:dbo
target_server_principal_name:
target_server_principal_sid:
target_database_principal_name:
server_instance_name:BIGISLAND\SQL2008
database_name:AdventureWorks
schema_name:HumanResources
object_name:EmployeePayHistory
statement:SELECT TOP (200) EmployeeID, RateChangeDate, Rate, PayFrequency, ModifiedDate
FROM HumanResources.EmployeePayHistory
additional_information:
.
```

| | | | |
|-------------------|---------------------------------------|----------------|-----------------------|
| Log Name: | Application | Logged: | 8/15/2008 11:24:58 AM |
| Source: | MSSQL\$SQL2008 | Task Category: | None |
| Event ID: | 33205 | Keywords: | Classic,Audit Success |
| Level: | Information | Computer: | BigIsland |
| User: | N/A | | |
| OpCode: | | | |
| More Information: | Event Log Online Help | | |

Figure 19: When you click on an audit event, this is the detail information you see.

Because the log is large and hard to easily fit on a screen, I have scrolled to one of the many events in the Application Log (there a lot more you can't see) and clicked on it. As you can see in figure 19, the details of the event provide a lot of information about a SELECT statement that ran against the audited table. Because of the potentially large quantity of information you can get back when using SQL Server Audit, I would suggest you store audit data to file, import this data into a SQL Server

database, and then use Transact-SQL or Reporting Services to analyze the data. Using the Event Viewer to review audit data, as I have done here, is not very efficient.

Summary

While this seemed like a lot to cover, it is just small sample of how the SQL Server Audit feature of SQL Server 2008 works. SQL Server Audit is a powerful new tool that allows DBAs to collect almost any activity that occurs within our servers.

Overall, I would suggest that if you have been looking for a SQL Server auditing system, and have been considering purchasing a third-party auditing application, or creating your own, you will want to first carefully evaluate SQL Server Audit to see if it can meet your SQL Server auditing needs.

CHAPTER 8: NEW DATA TYPES

While the focus of this book has been on new features of SQL Server 2008 that are of the most benefit to Production DBAs, this chapter is on new data types and should be of interest to Production DBAs and Development DBAs, as well as Database Developers.

We will take a look at the following new data types, each of which is available in all editions of SQL Server 2008:

- **Date and Time:** Four new date and time data types have been added, making working with time much easier than it ever has in the past. They include: DATE, TIME, DATETIME2, and DATETIMEOFFSET.
- **Spatial:** Two new spatial data types have been added--GEOMETRY and GEOGRAPHY--which you can use to natively store and manipulate location-based information, such as Global Positioning System (GPS) data.
- **HIERARCHYID:** The HIERARCHYID data type is used to enable database applications to model hierarchical tree structures, such as the organization chart of a business.
- **FILESTREAM:** FILESTREAM is not a data type as such, but is a variation of the VARBINARY(MAX) data type that allows unstructured data to be stored in the file system instead of inside the SQL Server database. Because this option requires a lot of involvement from both the DBA administration and development side, I will spend more time on this topic than the rest.

We'll look at each of these, one at a time, with our main focus being on the new FILESTREAM data type.

Date and Time

In SQL Server 2005 and earlier, SQL Server only offered two date and time data types: DATETIME and SMALLDATETIME. While they were useful in many cases, they had a lot of limitations, including:

- Both the date value and the time value are part of both of these data types, and you can't choose to store one or the other. This can cause several problems:
 - It often causes a lot of wasted storage because you store data you don't need or want.
 - It adds unwanted complexity to many queries because the data types often have to be converted to a different form to be useful.
 - It often reduces performance because WHERE clauses with these date and time data types often have to include functions to convert them to a more useful form, preventing these queries from using indexes.
- They are not time-zone aware, which requires extra coding for time-aware applications.
- Precision is only .333 seconds, which is not granular enough for some applications.
- The range of supported dates is not adequate for some applications, and the range does not match the range of the .NET CLR DATETIME data type, which requires additional conversion code.

To overcome these problems, SQL Server 2008 introduces four new date and time data types, described in the following sections. All of these new date and time data types work with SQL Server 2008 date and time functions, which have been enhanced in order to properly understand the new

formats. In addition, some new date and time functions have been added to take advantage of the new capabilities of these new data types. The new functions include **SYSDATETIME**, **SYSDATETIMEOFFSET**, **TODATETIMEOFFSET**, **SYSUTCDATETIME**, and **SWITCHOFFSET**.

DATE

As you can imagine, the DATE data type only stores a date in the format of YYYY-MM-DD. It has a range of 0001-01-01 through 9999-12-32, which should be adequate for most business and scientific applications. The accuracy is 1 day, and it only takes 3 bytes to store the date.

```
--Sample DATE output
DECLARE @datevariable as DATE
SET @datevariable = getdate()
PRINT @datevariable
Result: 2008-08-15
```

TIME

TIME is stored in the format: hh:mm:ss.nnnnnnn, with a range of 00:00:00.0000000 through 23:59:59.9999999 and is accurate to 100 nanoseconds. Storage depends on the precision and scale selected, and runs from 3 to 5 bytes.

```
--Sample TIME output
DECLARE @timevariable as TIME
SET @timevariable = getdate()
PRINT @timevariable
Result: 14:26:52.3100000
```

DATETIME2

DATETIME2 is very similar to the older DATETIME data type, but has a greater range and precision. The format is YYYY-MM-DD hh:mm:ss.nnnnnnnm with a range of 0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999, with an accuracy of 100 nanoseconds. Storage depends on the precision and scale selected, and runs from 6 to 8 bytes.

```
--Sample DATETIME2 output with a precision of 7
DECLARE @datetime2variable datetime2(7)
SET @datetime2variable = Getdate()
PRINT @datetime2variable
Result: 2008-08-15 14:27:51.5300000
```

DATETIMEOFFSET

DATETIMEOFFSET is similar to DATETIME2, but includes additional information to track the time zone. The format is YYYY-MM-DD hh:mm:ss[.nnnnnnn] [+|-]hh:mm with a range of 0001-01-01 00:00:00.0000000 through 0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999 Coordinated Universal Time (UTC), with an accuracy of 100 nanoseconds. Storage depends on the precision and scale selected, and runs from 8 to 10 bytes.

Time zone aware means a time zone identifier is stored as a part of DATETIMEOFFSET column. The time zone identification is represented by a [-|+] hh:mm designation. A valid time zone falls in

the range of -14:00 to +14:00, and this value is added or subtracted from UTC to obtain the local time.

```
--Sample DATETIMEOFFSET output with a precision of 0
--Specify a date, time, and time zone
DECLARE @datetimeoffsetvariable DATETIMEOFFSET(0)
SET @datetimeoffsetvariable = '2008-10-03 09:00:00 -10:00'

--Specify a different date, time and time zone
DECLARE @datetimeoffsetvariable1 DATETIMEOFFSET(0)
SET @datetimeoffsetvariable1 = '2008-10-04 18:00:00 +0:00'

--Find the difference in hours between the above dates, times,
--and timezones
SELECT DATEDIFF(hh,@datetimeoffsetvariable,@datetimeoffsetvariable1)
Result: 23
```

Spatial

While spatial data has been stored in many SQL Server databases for many years (using conventional data types), SQL Server 2008 introduces two specific spatial data types that can make it easier for developers to integrate spatial data in their SQL Server-based applications. In addition, by storing spatial data in relational tables, it becomes much easier to combine spatial data with other kinds of business data. For example, by combining spatial data (such as longitude and latitude) with the physical address of a business, applications can be created to map business locations on a map.

The two new spatial data types in SQL 2008 are:

- **GEOMETRY:** Used to store planar (flat-earth) data. It is generally used to store XY coordinates that represent points, lines, and polygons in a two-dimensional space. For example storing XY coordinates in the GEOMETRY data type can be used to map the exterior of a building.
- **GEOGRAPHY:** Used to store ellipsoidal (round-earth) data. It is used to store latitude and longitude coordinates that represent points, lines, and polygons on the earth's surface. For example, GPS data that represents the lay of the land is one example of data that can be stored in the GEOGRAPHY data type.

GEOMETRY and GEOGRAPHY data types are implemented as .NET CLR data types. This means that they can support various properties and methods specific to the data. For example, a method can be used to calculate the distance between two GEOMETRY XY coordinates, or the distance between two GEOGRAPHY latitude and longitude coordinates. Another example is a method to see if two spatial objects intersect or not. Methods defined by the Open Geospatial Consortium standard, and Microsoft extensions to that standard, can be used. To take full advantage of these methods, you will have to be an expert in spatial data, a topic that is well beyond the scope of this chapter.

Another feature of spatial data types is that they support special spatial indexes. Unlike conventional indexes, spatial indexes consist of a grid-based hierarchy in which each level of the index subdivides the grid sector that is defined in the level above. But like conventional indexes, the SQL Server query optimizer can use spatial indexes to speed up the performance of queries that return spatial data.

Spatial data is an area unfamiliar to many DBAs. If this is a topic you want to learn more about, you will need a good math background, otherwise you will get lost very quickly.

HIERARCHYID

While hierarchical tree structures are commonly used in many applications, SQL Server has, up to now, not made it easy to represent and store them in relational tables. In SQL Server 2008, the HIERARCHYID data type has been added to help resolve this problem. It is designed to store values that represent the position of nodes in a hierarchical tree structure.

For example, the HIERARCHYID data type makes it easier to express the following types of relationships without requiring multiple parent/child tables and complex joins:

- Organizational structures
- A set of tasks that make up a larger projects (like a GANTT chart)
- File systems (folders and their sub-folders)
- A classification of language terms
- A bill of materials to assemble or build a product
- A graphical representation of links between web pages

Unlike standard data types, the HIERARCHYID data type is a CLR user-defined type, and it exposes many methods that allow you to manipulate the data stored within it. For example, there are methods to get the current hierarchy level, get the previous level, get the next level, and many more. In fact, the HIERARCHYID data type is only used to store hierarchical data; it does not automatically represent a hierarchical structure. It is the responsibility of the application to create and assign HIERARCHYID values in a way that represents the desired relationship. Think of a HIERARCHYID data type as a place to store positional nodes of a tree structure, not as a way to create the tree structure.

FILESTREAM

SQL Server is great for storing relational data in a highly structured format, but it has never been particularly good at storing unstructured data, such as videos, graphic files, Word documents, Excel spreadsheets, and so on. In the past, when developers wanted to use SQL Server to manage such unstructured data, they essentially had two choices:

1. Store it in VARBINARY(MAX) columns inside the database
2. Store the data outside of the database as part of the file system, and include pointers inside a column that pointed to the file's location. This allowed an application that needed access to the file to find it by looking up the file's location from inside a SQL Server table.

Neither of these options was perfect. Storing unstructured data in VARBINARY(MAX) columns offers less than ideal performance, has a 2 GB size limit, and can dramatically increase the size of a database. Likewise, storing unstructured data in the file system requires the DBA to overcome several difficulties. For example:

- Files have a unique naming system that allows hundreds, if not thousands of files to be kept track of and requires very careful management of the folders to store the data.
- Security is a problem and often requires using NTFS permissions to keep people from accessing the files inappropriately.
- The DBA has to perform separate backups of the database and the files
- Problems can occur when outside files are modified or moved and the database is not updated to reflect this.

To help resolve these problems, SQL Server 2008 has introduced what is called FILESTREAM storage, essentially a hybrid approach that combines the best features of the previous two options.

Benefits of FILESTREAM

FILESTREAM storage is implemented in SQL Server 2008 by storing VARBINARY(MAX) binary large objects (BLOBs) outside of the database and in the NTFS file system. While this sounds very similar to the older method of storing unstructured data in the file system and pointing to it from a column, it is much more sophisticated. Instead of a simple link from a column to an outside file, the SQL Server Database Engine has been integrated with the NTFS file system for optimum performance and ease of administration. For example, FILESTREAM data uses the Windows OS system cache for caching data instead of the SQL Server buffer pool. This allows SQL Server to do what it does best: manage structured data, and allows the Windows OS to do what it does best: manage large files. In addition, SQL Server handles all of the links between database columns and the files, so we don't have to.

In addition, FILESTREAM storage offers these additional benefits:

- Transact-SQL can be used to SELECT, INSERT, UPDATE, DELETE FILESTREAM data.
- By default, FILESTREAM data is backed up and restored as part of the database file. If you want, there is an option available so you can backup a database without the FILESTREAM data.
- The size of the stored data is only limited by the available space of the file system. Standard VARBINARY(MAX) data is limited to 2 GB.

Limitations of FILESTREAM

As you might expect, using FILESTREAM storage is not right for every situation. For example, it is best used under the following conditions:

- When the BLOB file sizes average 1MB or higher.
- When fast read access is important to your application.
- When applications are being built that use a middle layer for application logic.
- When encryption is not required, as it is not supported for FILESTREAM data.

If your application doesn't meet the above conditions, then using the standard VARBINARY(MAX) data type might be your best option.

If you are used to storing binary data inside your database, or outside your database (but using pointers inside the database that point to the binary files), then you will find using FILESTREAM storage to be substantially different. You will want to thoroughly test your options before implementing one option or the other, in any new applications you build.

How to Implement FILESTREAM Storage

Enabling SQL Server to use FILESTREAM data is a multiple-step process, which includes:

1. Enabling the SQL Server instance to use FILESTREAM data
2. Enabling a SQL Server database to use FILESTREAM data
3. Creating FILESTREAM-enabled columns in a table, by specifying the "VARBINARY(MAX) FILESTREAM" data type.

The first step can be performed using the SQL Server 2008 Configuration Manager (demoed here), or by using the `sp_filestream_configure` system stored procedure.

To enable FILESTREAM storage at the instance level, start the SQL Server 2008 Configuration Manager, click on SQL Server Services in the left window, and then in the right window, right-click on the SQL Server instance you want to enable FILESTREAM storage on. Choose Properties, then click on the FILESTREAM tab, and the dialog box shown in Figure 1 appears:

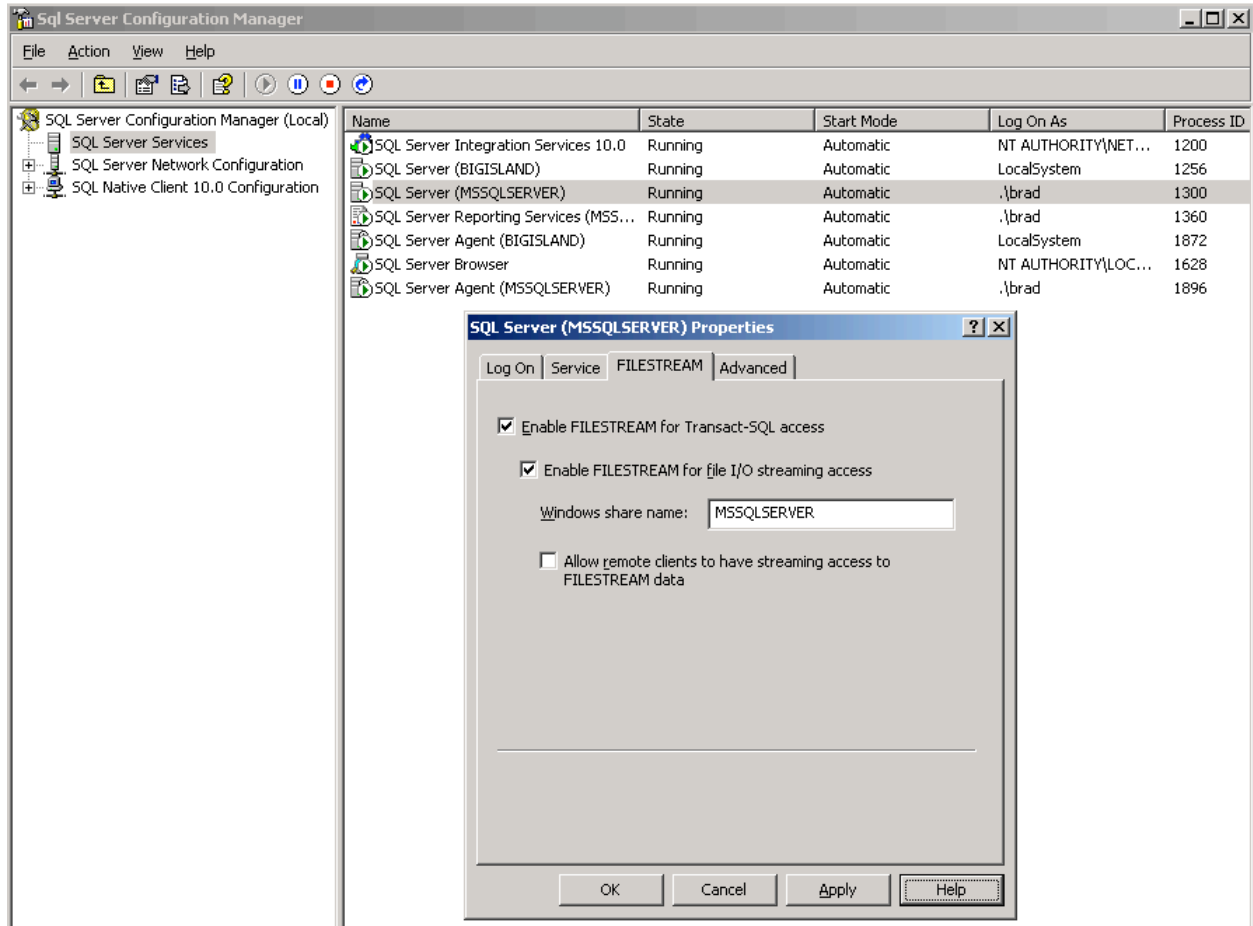


Figure 1: Enabling FILESTREAM storage offers several options.

When you enable FILESTREAM storage, you have several options. The first one is "Enable FILESTREAM for Transact-SQL access". This option must be selected if you want to use FILESTREAM storage. If you want to allow local WIN32 streaming access to FILESTREAM storage data, then you must also select the "Enable FILESTREAM for file I/O streaming access" option. In addition, selecting this option requires that you enter a Windows share name that specifies where you want the FILESTREAM data to be stored. And last, if you want to allow remote clients to access the FILESTREAM data, then you must select the "Allow remote clients to have streaming access to FILESTREAM data". Keep in mind that you only want to implement those options that you will use, as choosing additional options can increase server resource usage. Once you choose your options, click OK.

The next step is to open SQL Server Management Studio (SSMS) and run the following Transact-SQL code from a query window.

```
EXEC sp_configure filestream_access_level, 2
RECONFIGURE
```

FILESTREAM storage has now been enabled for the SQL Server instance.

The next step is to enable FILESTREAM storage for a particular database. You can do this when you first create a database, or after the fact using ALTER DATABASE. For this example, we will be creating a new database using Transact-SQL.

The Transact-SQL code used to create a FILESTREAM-enabled database looks like this:

```
CREATE DATABASE FILESTREAM_Database
ON
PRIMARY ( NAME = Data1,
          FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\FILESTREAM_Database.mdf'),
FILEGROUP FileStreamGroup CONTAINS FILESTREAM( NAME = FILESTREAM_Data,
          FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\FILESTREAM_Data')
LOG ON ( NAME = Log1,
        FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\FILESTREAM_Database.ldf')
GO
```

The above code looks similar to the code used to create a regular SQL Server database, except with the addition of a new filegroup to store the FILESTREAM data. In addition, when creating the FILESTREAM filegroup, you add the clause "CONTAINS FILESTREAM".

After the above code runs, and the database is created, a new sub-folder is created with the name of "FILESTREAM_Data", as shown in Figure 2. Notice that this sub-folder name is based on the name I assigned it in the above code.

Inside this newly created folder is a file called "filestream.hdr" and an empty sub-folder called \$FSLOG. It is very important that you do not delete, modify, or move the "filestream.hdr" file, as it is used to keep track of the FILESTREAM data.

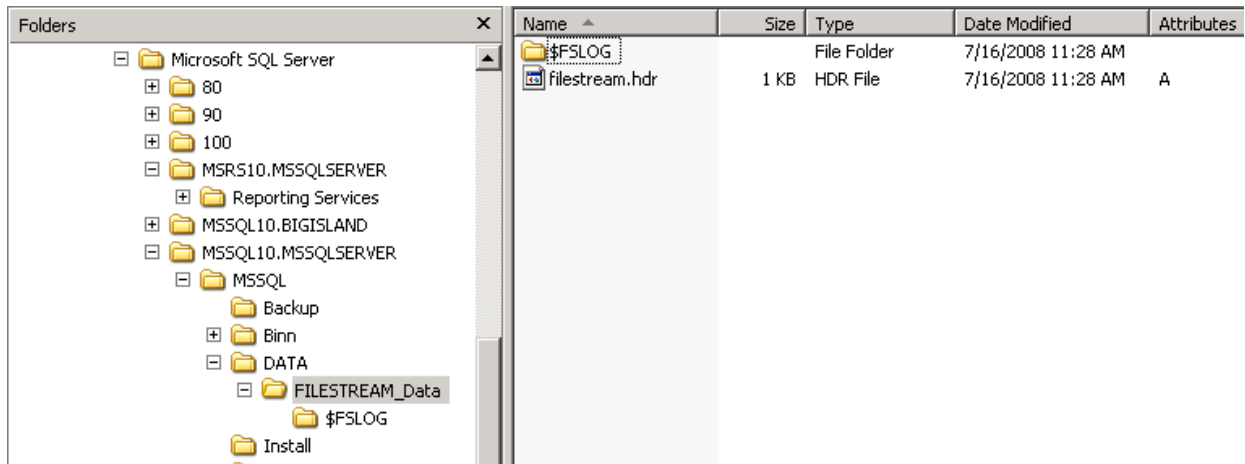


Figure 2: When a FILESTREAM-enabled database is created, additional sub-folders, and a system file, are created.

At this point, our database is FILESTREAM-enabled, and you begin adding new tables that include the **VARBINARY(MAX)** data type. The only difference between creating a standard **VARBINARY(MAX)** column in a table and a FILESTREAM-enabled **VARBINARY(MAX)** column is to add the keyword FILESTREAM after the **VARBINARY(MAX)**. For example, to create a very simple table that can store FILESTREAM data, you can use code similar to this:

```
CREATE TABLE dbo.FILESTREAM_Table
(
```

```

DATA_ID UNIQUEIDENTIFIER ROWGUIDCOL NOT NULL UNIQUE ,
DATA_Name varchar(100) ,
Catalog VARBINARY(MAX) FILESTREAM,
)

```

This simple table includes three columns, of which the last one, named "Catalog" can store FILESTREAM data. At this point, you can SELECT, INSERT, UPDATE, and DELETE FILESTREAM data similarly as you would any column in a SQL Server table.

When you create FILESTREAM-enabled columns in tables, and begin adding FILESTREAM data, additional subfolders will be created, under the folder created when FILESTREAM was enabled for the database, as shown in **Figure 3**:

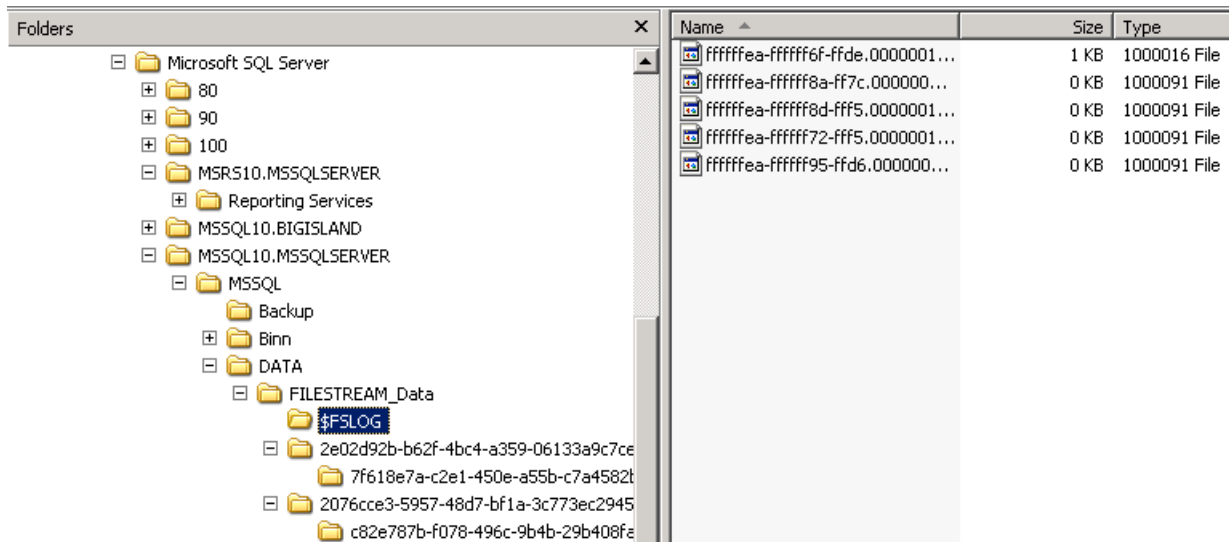


Figure 3: As you create new FILESTREAM-enabled tables, new sub-folders are created.

As you can imagine, you do not want to change any of these folders or the files inside of them. They are all managed by SQL Server.

Summary

If you find any of these interesting, I suggest you begin experimenting with them, learning how they work, along with their various advantages and disadvantages, before you decide to use them when building new applications. Also keep in mind that all of these new data types only work with SQL Server 2008 and that they are not backward-compatible. Because of this, they are of best use when creating new applications.

CHAPTER 9 EXTENDED EVENTS

Traditionally, SQL Server has provided many log-based tools to help DBAs identify and troubleshoot problems. There are SQL Server log files, Windows OS Event log files, System Monitor logs, Profiler traces, and more. In SQL Server 2008, there has been the addition of the Performance Data Collector and SQL Server Audit. Some of the tools have overlapping functionality, but for the most part each tool offers its own individual benefits.

As new SQL Server troubleshooting tools have been developed over the years, several problems have evolved. They include:

1. The various tools use different methods to collect, analyze and store data. This means that DBAs have to use multiple tools to identify and troubleshoot problems.
2. An offshoot of using dissimilar tools is that the data is stored in different formats, making it more difficult for DBAs to gather all relevant troubleshooting data together in a central location so it can be analyzed as a whole.
3. SQL Server does not run independently. It runs as part of a larger system that includes the Windows OS and the applications accessing it. A few of the tools provide some insight into SQL Server, the Windows OS, and the application but the implementation is not comprehensive or well-integrated. For example, Windows OS Event logs include SQL Server and OS events. System Monitor includes both OS and SQL Server performance counters, and Profiler captures SQL Server and application data. While all of this is great, it is still very difficult for DBAs to see the big picture, making it much more difficult for them to identify and troubleshoot many problems.

SQL Server 2008 begins to tackle the problem of providing a single, centralized way to collect, store, and analyze data for SQL Server, the Windows OS, and applications.

One strand of this effort is the introduction of the Performance Data Collector, covered in Chapter 5. Essentially, it offers a repository for storing performance data collected from selected performance counters and DMVs over time. In its current form, it is a long way from a total solution that would solve all of the problems described above, although it does have the potential to act as a central repository of all types of SQL Server data.

In addition to this, SQL Server 2008 introduces SQL Server Extended Events (Extended Events). The goal of Extended Events is to provide a generic event-handling system that is part of the database engine and that makes it much easier to capture, store and act on disparate troubleshooting data from SQL Server, the Windows OS, and applications. With all the data in one place, it is much easier to analyze holistically, helping DBAs see the bigger picture, and making troubleshooting easier.

An Overview of Extended Events

To be up front, the implementation of Extended Events in SQL Server 2008 is very much a version 1.0 attempt. It doesn't do everything a DBA would like to see it do, and it is not particularly easy to implement, as you will see in the examples found later in this chapter. On the other hand, Extended Events is the engine that is used to support the new SQL Server Audit feature in SQL Server 2008, so we know it has potential. As I understand it from talking with Microsoft product managers, Extended Events will be enhanced in the next version of SQL Server, helping to bring us closer to the ideal solution DBAs need to identify and troubleshoot SQL Server problems at the SQL Server,

Windows OS, and application levels. Until then, DBAs will have to rely on homegrown or third-party tools to accomplish this task.

Essentially, Extended Events is a generic event-handling system that can be used to:

- Collect pre-defined event data from SQL Server (254 events can be captured).
- Correlate different types of SQL Server events with one another.
- Correlate SQL Server events with Windows OS events.
- Correlate SQL Server events with database application events.

Some of the above is handled by the SQL Server Extended Events Engine, and some of it is done in cooperation with Event Tracing for Windows (ETW).

For those not familiar with ETW, it is a feature built into the Windows OS that allows developers to create their own traces of OS events. In fact, the events you see in the OS Event Logs come from ETW. In addition, ETW allows developers to add event tracing to the applications they write.

If you combine the SQL Server Extended Events features with the features offered by ETW, you have a framework that allows events from SQL Server, the Windows OS, and from applications to be captured, stored, and analyzed as a whole. But for the typical DBA who is not a Windows developer, all of the functionality described above is not within close reach.

Because of this, our focus in this chapter is on what a DBA can do using Extended Events, and that is to create an Extended Events session that can be used to collect event data from SQL Server. For now, we will leave the OS and application events to Windows developers.

Benefits and Limitations of Extended Events

Some of the benefits of using Extended Events for collecting troubleshooting events over using other SQL Server tools include:

- Events are handled by the Extended Events Engine, not with a variety of different tools.
- Events can be stored in a common format, making them easier to combine and correlate with one another.
- SQL Server Extended Events can be combined with ETW events from the OS and applications (assuming this has been created by a developer).
- The overhead incurred by using Extended Events is less than the overhead incurred by other tools.

On the other hand, the disadvantages of using Extended Events instead of using current troubleshooting tools include:

- The types of events that can be collected are not as comprehensive as the events that can be captured by other tools.
- There is no GUI interface to set up, manage, or to analyze results. Creating and managing Extended Events sessions are created and managed with Transact-SQL, and you are left completely on your own to analyze the data.
- Extended Events is a little rough around the edges, while many of the other available troubleshooting tools have been around a long time and are more polished. For example, many DBAs may choose to continue using Profiler to collect the same events that can be captured by Extended Events, even though using Extended Events uses fewer resources to capture the same data.

Extended Events Sessions

Essentially, an Extended Events session is created to collect and store pre-defined events. In many ways, it is similar to a Profiler Trace. For example, Extended Events sessions can be created to perform any of the following tasks (among many others):

- Troubleshoot excess CPU utilization
- Troubleshoot memory problems
- Troubleshoot poorly performing queries
- Troubleshoot blocking locks
- Troubleshoot deadlocking
- Troubleshoot I/O problems
- Troubleshoot Service broker problems
- Troubleshoot wait states

Do these sound familiar to you? That's right, they sound familiar because you can do similar tasks using the various tools that already come with SQL Server. The main difference is that instead of using separate tools to collect and store the data, an Extended Events session is used instead.

Extended Events sessions are created and managed using Transact-SQL DDL statements. They include:

- **CREATE EVENT SESSION:** Used to create a new Extended Event session, the CREATE EVENT SESSION statement allows you to assign the event session a name, add events that you want to track, add a predicate to filter out events you don't want to see, specify the target where you want event data to be sent, among other options.
- **ALTER EVENT SESSION:** ALTER EVENT SESSION is used to modify a currently existing event session. It is also used to start and stop sessions.
- **DROP EVENT SESSION:** If you no longer need an event session you have created, you delete it using the DROP EVENT SESSION.

If you want to create and manage Extended Event sessions, you will need to create and maintain your own scripts that create new event sessions, change existing event sessions, stop and start event sessions, and remove event sessions that are no longer needed.

Once an Extended Events session has been created, run, and the data collected, it needs to be analyzed. The Extended Events Engine is not designed for analyzing data, so how you do this is left up to you. One of the options available to you is to extract the data captured by an Extended Events session and to store it in a database for analysis. You can analyze the data using SELECT statements, or by creating Reporting services reports.

There are no GUI-based tools to observe Extended Events sessions, but you can use five different catalog views to find metadata on the events you have created, and there are nine DMVs you can use to monitor session metadata and session data. As with creating and managing Extended Events, you will need to create your own scripts to perform these tasks.

Now that we have taken a high-level look at Extended Events session, let's take a look at a simple example of how can to create and use Extended Events.

Creating an Extended Events Session

Let's create an Extended Events session designed to capture XML deadlock reports, which can be used by a DBA to help identify the code involved in a deadlock. If you are familiar with the SQL

Server 2005 Profiler, it has the ability to capture a very similar event. I chose this example to demonstrate how similar Extended Events and Profiler Events are (in its current incarnation). In fact, the next version of SQL Server will probably replace Profiler's current "engine" with the Extended Events "engine". While there are many reasons for this, one of the key ones is that the Extended Events "engine" is much more lightweight.

The first step to creating an Extended Events session is to use the CREATE EVENT SESSION command, where we need to specify, at a minimum, three things:

1. The name of the Extended Event session
2. The name of the event(s) we want to capture
3. Where the event data is to be sent (called the target)

The code to create this Extended Events session is:

```
CREATE EVENT SESSION deadlock_trace
ON SERVER
    ADD EVENT sqlserver.xml_deadlock_report
    ADD TARGET package0.etw_classic_sync_target
    (SET default_etw_session_logfile_path =
     N'C:\traces\deadlock_trace.etl' )
```

The first part of this statement specifies the name of the Extended Events session we want to create:

```
CREATE EVENT SESSION deadlock_trace
ON SERVER
```

The second part of this statement specifies what events are to be captured:

```
ADD EVENT sqlserver.xml_deadlock_report
```

Currently, there are about 254 events that can be captured, many of which are similar to Profiler events. While I have only added one event above, you can add as many events as you want. In this particular case, we are capturing an event that fires when a deadlock event occurs, providing us with an XML-formatted report of what happened⁴.

The last part of this statement tells the Extended Events session how and where to store the events that are captured:

⁴ If you want to see a list of all the Extended Events available, run this code:

```
USE msdb
SELECT name,
       description
FROM   sys.dm_xe_objects
WHERE  object_type = 'event'
ORDER BY name
```

```
ADD TARGET package0.etw_classic_sync_target
    (SET default_etw_session_logfile_path =
        N'C:\traces\deadlock_trace.etl' )
```

You can choose from six different targets. In this case I have chosen the ETW target, which means that the events I capture are stored on disk in the ETW format. I selected this target because it is fairly easy to extract the data I want from the format, and, if desired, it allows me to correlate this data with other ETW formatted data I may have collected elsewhere. As part of this command, I also specify the location and name of the physical file.

To keep this example simple, I have limited the above CREATE EVENT SESSION statement to only the required options. There are many other options available to perform many other tasks, such as filtering out events you don't want, that are not shown in the above statement.

Once you are happy with the statement, execute it, and the Extended Events session is saved. Once saved, you can turn it on or off as required. The Transact-SQL code to turn this session on is as follows:

```
ALTER EVENT SESSION deadlock_trace
ON SERVER
STATE = start
```

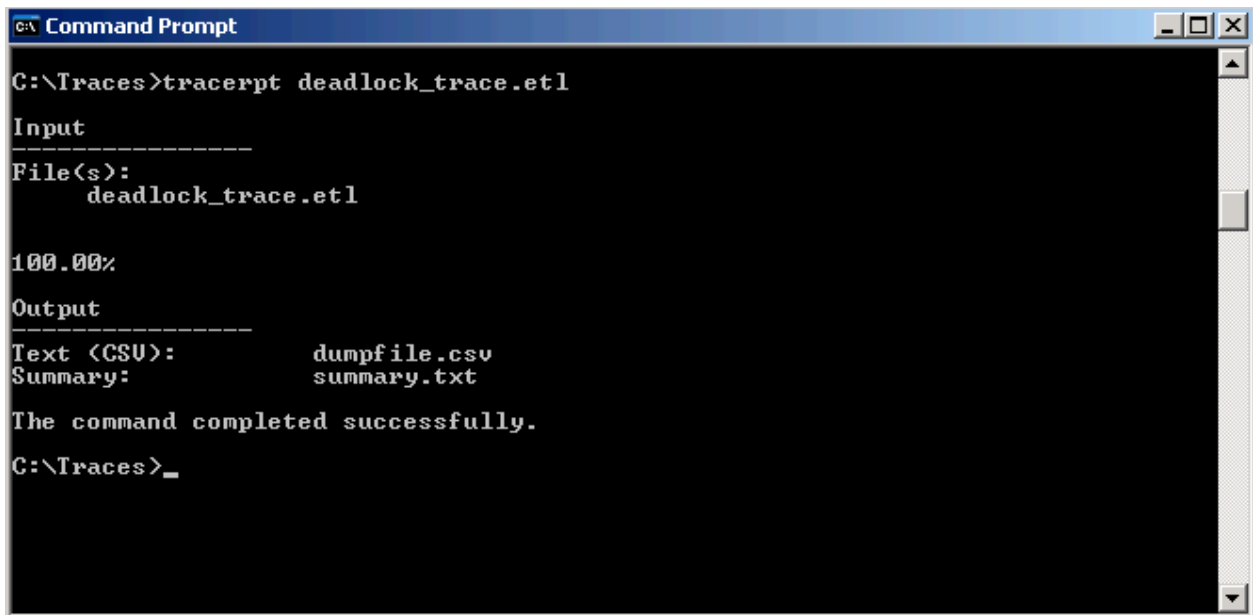
And the code to turn this session off is:

```
ALTER EVENT SESSION deadlock_trace
ON SERVER
STATE = stop
```

Having started the session, collected some data, and turned the session off, you can now view the data that was captured. The data is stored in the file and path specified in the ADD TARGET definition. In this example, we specified that the data be stored in the ETW format (which is an unreadable binary format), so we need a way to convert it into a readable format.

There are several options for converting ETW-formatted data into other forms, but one of the most common ways is to use a command called TRACERPT, which is run at the command line. This command (which is part of Windows 2003 and Windows 2008) offers many options for extracting and formatting ETW-formatted data to make it readable. In this case, we want the data turned into an easy to read text format, so choose the default comma-separated values (CSV) format. Equally, we could choose to format the data in XML or HTML, by adding appropriate parameters to the command line.

By running the TRACERPT command on the "deadlock_trace.etl" file, as shown in Figure 1, the data is extracted from the ETW file and converted into a readable CSV file:



```
C:\Traces>tracert deadlock_trace.etl

Input
-----
File(s):
    deadlock_trace.etl

100.00%

Output
-----
Text (CSV):      dumpfile.csv
Summary:         summary.txt

The command completed successfully.

C:\Traces>_
```

Figure 1: The TRACERPT command is used to translate an ETW file into a CSV file.

In fact, when the TRACERPT is run against the "deadlock_trace.etl" file, two new files are created. The "Summary.txt" file includes information about the conversion process and is not really important. The second file, "dumpfile.csv" includes the event(s) that were captured by the Extended Events session.

The "dumpfile.csv", opened in Notepad, looks as shown in **Figure 2:**

```

Event Name,      Type,      TID,      Clock-Time, Kernel(ms),  User(ms),  User Data
EventTrace,     Header, 0x00000050, 128608251876377488, 590, 6020, 131072, 3362048
xml_deadlock_report, 0, 0x000000610, 128608252202846928, 30, 160, 0,
<victim-list>
<victimProcess id="process4c5f000"/>
<process-list>
<process id="process4c5f000" taskpriority="0" logused="248" waitresource="KEY: 7:72057594075480064"
<executionStack>
<frame procname="" line="10" stmtstart="56" sqlhandle="0x02000000d118bd24f0ac1bf99179b2282c838ck
</frame>
<frame procname="" line="10" stmtstart="308" stmtend="486" sqlhandle="0x020000004188191f7cc433c5
</frame>
</executionStack>
<inputbuf>
BEGIN TRAN

--Update: Run 2nd (both examples)
UPDATE sales.[SalesTerritory]
SET salesytd = salesytd + 1
WHERE territoryID = 1;

--Update
UPDATE sales.[SalesTaxRate]
SET taxrate = taxrate + 0.05
WHERE taxtype = 1;

COMMIT TRAN </inputbuf>
</process>
<process id="process4c5ec70" taskpriority="0" logused="1640" waitresource="KEY: 7:7205759407567667"
<executionStack>
<frame procname="" line="2" stmtstart="38" sqlhandle="0x020000001c34a713ac13079b3ea6334fe649c0ec
</frame>
<frame procname="" line="2" stmtstart="46" stmtend="230" sqlhandle="0x020000008839a5368ab89832c5
</frame>
</executionStack>
<inputbuf>
--Update Two: Run 3rd
UPDATE sales.[SalesTerritory]
SET salesytd = salesytd + 1
WHERE territoryID = 1;

COMMIT TRAN </inputbuf>
</process>
</process-list>
<resource-list>
<keylock hobtid="72057594075480064" dbid="7" objectname="" indexname="" id="lock8dc2440" mode="X"
<owner-list>
<owner id="process4c5ec70" mode="X"/>
</owner-list>
<waiter-list>
<waiter id="process4c5f000" mode="U" requestType="wait"/>
</waiter-list>
</keylock>
<keylock hobtid="72057594075676672" dbid="7" objectname="" indexname="" id="lock8dd1b40" mode="X"
<owner-list>
<owner id="process4c5f000" mode="X"/>
</owner-list>
<waiter-list>
<waiter id="process4c5ec70" mode="X" requestType="wait"/>
</waiter-list>
</keylock>
</resource-list>
</deadlock>
</deadlock-list>
", 0, 0

```

Figure 2: This is the results of our trace, which shows an XML Deadlock Report.

What we see in Figure 2 is the XML report of a single deadlock that occurred when we ran the Extended Event session. If you have seen the SQL Server Profiler XML Deadlock Report, you will recognize the output, as it is almost identical. At this point, I can analyze the deadlock event and determine what code contributed to the problem.

If you are experiencing a lot of deadlock problems, and need to track them over time, you could take the Extended Events captured and import them into a database for additional analysis. This is the same for any of the 254 or so Extended Events that are currently available.

Summary

In this brief chapter, we have learned that Extended Events is a general event-handling system that has been incorporated into SQL Server 2008. To take advantage of it in its current form, DBAs can create Extended Events sessions that allow them to collect a wide variety of internal SQL Server events. In addition, if set up correctly, data collected by an Extended Events session can be combined with events from the Windows OS and from applications, allowing the DBA to see a holistic view of related troubleshooting data.

CHAPTER 10: CHANGE DATA CAPTURE

As DBAs, one of our goals is to separate OLTP (On-Line Transaction Processing) and OLAP (On-Line Analytical Processing) activity, preferably onto different servers. This is because performing both types of activities in the same database, or even in different databases on the same server, can lead to performance problems. In order to accomplish this goal, DBAs must figure out the best way to move transactional data from OLTP servers to servers designated for OLAP activity so that the data remains more or less synchronized. This sounds simple, but can prove to be difficult in practice.

One solution is to simply copy the entire OLTP database to another server where OLAP activity can be performed on it. A slightly more sophisticated solution would be to transform all the data as it is being moved from the OLTP database to another server so that it is easier to perform OLAP activity on the data. While these solutions can and do work, they often become impractical when large amounts of data are involved.

Instead of moving all of the data, over and over again between an OLTP server and an OLAP server, it is much more efficient if only the changes are moved from server to server, so that the data on the OLTP server and the OLAP server are kept synchronized. This requires fewer resources and can boost performance. The larger the databases involved, the bigger the performance gain when only changed data is moved between servers.

In SQL Server 2005 and earlier, DBAs had several options when it came to moving changes from an OLTP server to an OLAP server. These included using replication, timestamp columns, triggers, complex queries, as well as expensive third-party tools. None of these are easy to implement, and many of them use a lot of server resources, negatively affecting the performance of the OLTP server.

In SQL Server 2008 (Enterprise Edition only), Microsoft has introduced a new feature called Change Data Capture (CDC). It is designed to make it much easier and less resource intensive to move changed data from an OLTP server to an OLAP server. In a nutshell, CDC captures and records INSERT, UPDATE, and DELETE activity in an OLTP database and stores it in a form that is easily consumed by an application, such as an SSIS package. This package is used to take the data and apply it to an OLAP server, allowing both servers to keep their data synchronized.

Change Data Capture Architecture

Figure 1 presents a high-level overview of the major components of the CDC architecture, and at how the various components work together. Notice that the diagram is divided into two sections. The top section represents an OLTP server and the bottom section represents an OLAP server.

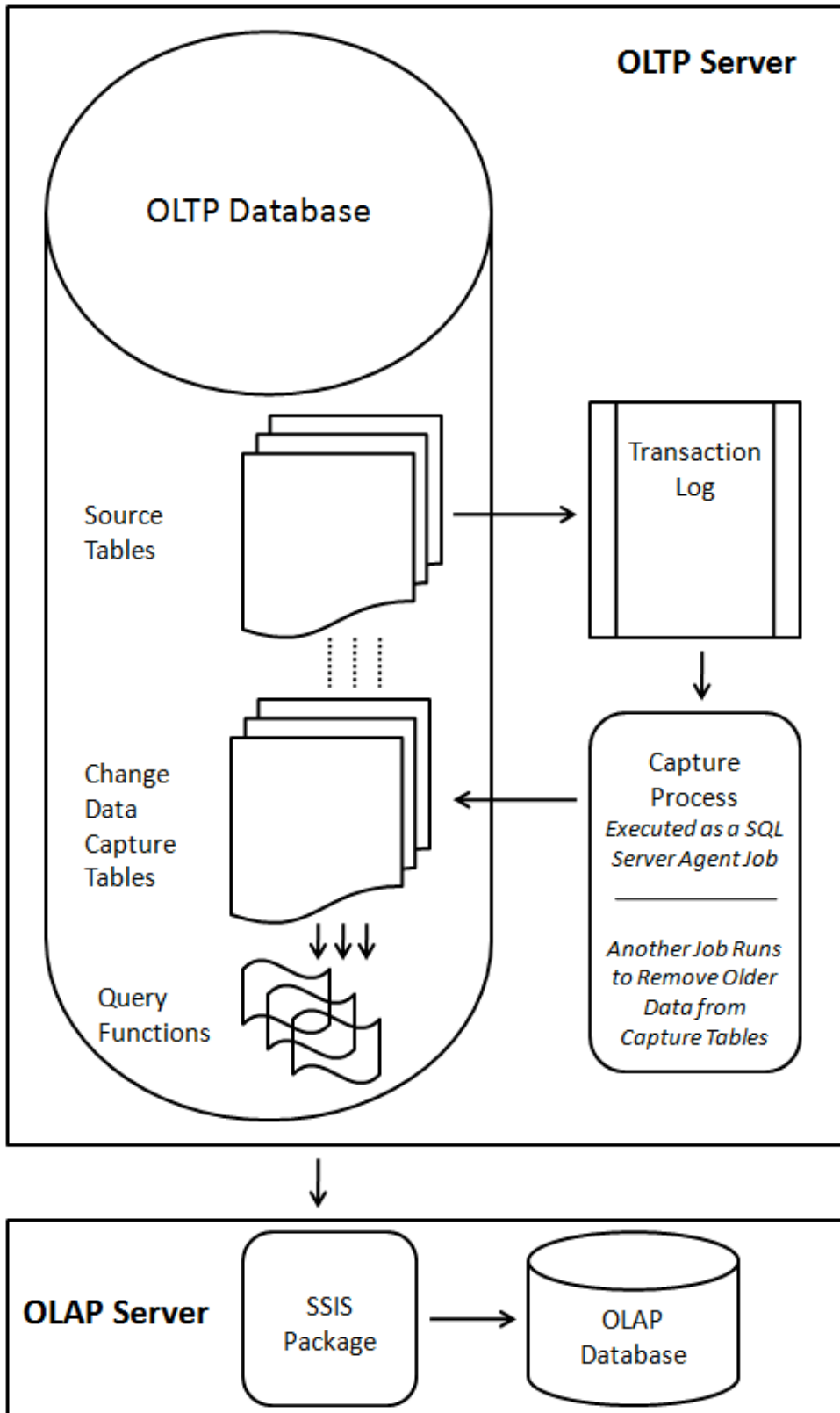


Figure 1: High-level view of Change Data Capture architecture.

Let's start with the OLTP server, as this is where CDC does its work.

Source Tables

When SQL Server 2008 is first installed, CDC is turned off by default so the first step is to enable it at the database level. If you want to turn it on for more than one database, you must enable it for each. Next, CDC must be enabled at the table level, on a table-by-table basis. Each table that is CDC-enabled is known as a **Source table**. In figure 1, a single OLTP database has been enabled for CDC, and three **Source tables** have been enabled for CDC.

Change Data Capture Tables

When CDC has been enabled for a database, along with one or more tables, several things happen under the covers. For example, for every Source table enabled for CDC, an associated **Change Data Capture table** is created, which is used to store the changes made in the Source table, along with some metadata used to track the changes.

In our example, since three of our Source tables have been enabled for CDC, three CDC tables have been created.

CDC tables aren't like the tables we are used to working with. Normally, when you want to view data from a table, you run a SELECT statement directly against the table. This is not the case with CDC tables as you can't run statements directly against them. Instead, in order to view the data from a CDC table, you have to run SELECT against a **Change Data Capture query function**.

Change Data Capture Query Functions

For each of the Source tables enabled for CDC, one or two **CDC query functions** are created, depending on how the tables were enabled:

- **cdc.fn_cdc_get_all_changes_capture_instance** – this query function is always created and is used to return all changes from a CDC table.
- **cdc.fn_cdc_get_net_changes_capture_instance** - this query function is optionally created when enabling the Source table for CDC, and can be used to only return the net changes from a CDC table.

In order to view the contents of a CDC Table, you run a SELECT statement against its associated CDC query function, and the results are displayed just as any other query.

In our example in figure 1, only one CDC query function has been created for each of the three CDC tables.

Capture Instances

When a CDC table and any related CDC query functions are created, they make up what is called a **capture instance**. A capture instance is created for every Source table that is enabled for CDC. Capture instances are given a name so that they can be distinguished from one another. For example, if the table named **sales.products** is CDC-enabled, the capture instance created is named **sales_products**. In turn, the name of the CDC table is referred to as **sales_products**, and the names of the two associated CDC query functions are: **cdc.fn_cdc_get_all_changes_sales_products** and **cdc.fn_cdc_get_net_changes_sales_products**. In our example, since three Source tables have been CDC-enabled, three new capture instances have been created.

Capture and Cleanup Jobs

Besides the Change Data Capture tables and query functions that have been created in our example, two SQL Server Agent jobs are created: a Capture and a Cleanup Job.

The **Capture** job generally runs continuously and is used to move changed data to the CDC tables from the transaction log. The **Cleanup** job runs on a scheduled basis to remove older data from the CDC tables so that they don't get too large. By default, data older than three days is automatically removed from CDC tables by this job.

Now that we are familiar with the components of CDC, let's see how they work together.

How Change Data Capture Works

You will recall from figure 1 that three Source tables were CDC-enabled. At this point, when any INSERT, UPDATE, or DELETE statements are issued against any of these three Source tables, these changes are first written to the transaction log associated with the database. This is normal behavior and occurs whether or not CDC has been enabled. What is different, now that CDC has been turned on for the Source tables, is that the Capture Process (the SQL Server Agent Capture job) reads the changes from the transaction log and moves them into the appropriate CDC tables.

INSERTs and DELETEs each occupy one row of the CDC table, and UPDATEs occupy two rows: one row for the "before" data and one row for the "after" data. These changes continue to accumulate in the CDC tables until they are deleted by the Cleanup Job, when it makes its scheduled run.

At any time, the DBA can view (or extract) the change data from the CDC tables by running appropriate statements against the relevant CDC query functions. For example, let's say that a DBA has a job that executes an SSIS package once every four hours. When this SSIS package executes, it runs a statement against the appropriate CDC query function, which allows it to extract all of the changes made to a Source table since the last time the SSIS package was executed, and then moves the changes to a database on an OLAP server. This way, the data on the OLAP server is kept synchronized with the data from the OLTP server, with only a four hour delay. Of course, this is only one example. As the DBA, you have many options on how you might want to configure the data transfer between the two servers.

Implementing Change Data Capture for a Single Table

Now that we have a basic understanding of how CDC works, let's implement a simple example of it using a modified version of the AdventureWorks database. For this example, our goal is to enable a single table for CDC so that we can see how changes to the Source table are stored in the CDC table. In addition, we will look at how we can query the CDC table using a CDC query function to examine the changes that were captured.

The first step is to CDC-enable the AdventureWorks database. This is done by running the **sys.sp_cdc_enable_db** stored procedure inside the database to be CDC-enabled, as follows:

```
USE AdventureWorks
GO
EXEC sys.sp_cdc_enable_db
GO
```

Now that the AdventureWorks database is CDC-enabled, the next step is to CDC-enable each Source table for which we want to track changes. While we will only perform this for a single table

for this example, you can easily repeat this same process for each table in a database that you want to CDC-enable. This is done using the `sys.sp_cdc_enable_table` stored procedure:

```
USE AdventureWorks
GO

EXEC sys.sp_cdc_enable_table
@source_schema = N'Sales',
@source_name   = N'Store',
@role_name     = N'CDCRole'
```

The parameters are as follows:

- **@source_schema** refers to the schema name of the Source table you are enabling.
- **@source_name** refers to the name of the Source table you are enabling.
- **@role_name** refers to the security role that is created when CDC is enabled. This role can be ignored, or it can be used to assign permission to specific users so they can access the changed data using CDC query functions, without having to be a member of the **db_owner** role. If this role does not preexist, it will be created for you.

In our example above, we have CDC-enabled a Source table called **Sales.Store**, and have created a new role called **CDCRole**. Although you can't see it, an associated CDC table has been created, along with a CDC query function named **cdc.fn_cdc_get_all_changes_Sales_Store**.

Now that CDC has been enabled for the database and a table, let's make some changes to the Source table and then use the CDC query function to see what was captured in the CDC table. The following code INSERTs a row, DELETES a row, and UPDATES a row in the Source table:

```
INSERT INTO Sales.Store
VALUES (1556,N'The Bunny Shoppe', '277', GETDATE());

DELETE FROM Sales.Store
WHERE CustomerID = 2

UPDATE Sales.Store
SET Name = 'The Magic Shoppe'
WHERE CustomerID = 6
```

Now that some changes have been made to our Source table, let's see what data the CDC table contains. As mentioned before, the only way to view the rows in a Change Data Capture Table is use the CDC query function that was created when the Source table was CDC-enabled.

The generic appearance of this function is:

```
cdc.fn_cdc_get_all_changes_capture_instance ( from_lsn , to_lsn , '<row_filter_option>' )
```

The first thing to notice about the CDC query function above is that the first part of the function's name is:

```
cdc.fn_cdc_get_all_changes_
```

And that the last part of the function name is:

```
capture_instance
```

As we discussed earlier in this chapter, whenever a capture instance is created, it not only creates the related CDC table, it also creates a custom CDC query function that is based on the capture instance name. For our example, the capture instance name is:

```
Sales_Store
```

Given this, then the name of the CDC query function is:

```
cdc.fn_cdc_get_all_changes_Sales_Store
```

Keep in mind that every Source table that is CDC-enabled will have either one or two CDC query functions, and that they will be named differently, based on the capture instance name.

Our CDC query function has three parameters. The first parameter is:

```
from_lsn
```

This refers to the LSN (Log Sequence Number) that represents the low endpoint of the LSN range you want to include in your result set. A LSN is assigned to every entry made into the transaction log, in ascending order. This means that as changes are made to the Source table, and then made in the transaction log, each change is assigned an LSN. The LSN of every transaction is also recorded in the CDC table, and can be used as a way to select which changes you want to return using the CDC query function.

The second parameter is:

```
to_lsn
```

This is the LSN that represents the high endpoint of the LSN range you want included in the result set. The combination of the **from_lsn** and the **to_lsn** constitute the range of "changes" you want to return.

The third parameter is:

```
'<row_filter_option>'
```

The values of this parameter can be either:

- **All** - tells the function to return all changes within the specified LSN range, but only return a single UPDATE row that contains the new values.
- **All update old** - also returns all changes within the specified LSN range, but includes the two UPDATE rows, one with the before data and one with the after data.

Now, you may be asking yourself, how do you find the **from_lsn** and the **to_lsn** that are needed for the function's parameters? There are different ways this can be done, but for this example, we will keep it simple. In our case, we just want to return the oldest **from_lsn** and the newest **to_lsn** that is stored in the CDC table. In this case, two different functions are used for this purpose.

The first function is:

```
sys.fn_cdc_get_min_lsn('capture_instance_name')
```

This function is used to find the **begin_lsn**, where:

```
'capture_instance_name'
```

is the capture instance name of the Change Data Capture table.

The second function:

```
sys.fn_cdc_get_max_lsn()
```

is used to find the **to_lsn** value. Notice that it requires no parameters.

OK, let's put all of this together and see what rows were recorded in the Change Data Capture table, after performing the three DML statements above. To find out, let's run the following code:

```
USE AdventureWorks
GO

--declare variables to represent beginning and ending lsn
DECLARE @from_lsn BINARY(10), @to_lsn BINARY(10)
-- get the first LSN for table changes
SELECT @from_lsn = sys.fn_cdc_get_min_lsn('Sales_Store')
-- get the last LSN for table changes
SELECT @to_lsn = sys.fn_cdc_get_max_lsn()
-- get all changes in the range using "all update old" parameter
SELECT * FROM cdc.fn_cdc_get_all_changes_Sales_Store(@from_lsn, @to_lsn, 'all update old');
GO
```

The results returned are shown in **Figure 2**:

| | __\$start_lsn | __\$seqval | __\$operation | __\$update_mask | CustomerID | Name | SalesPersonID | ModifiedDate |
|---|-------------------------|-------------------------|---------------|-----------------|------------|--------------------------|---------------|-------------------------|
| 1 | 0x000000045000019280004 | 0x000000045000019280002 | 2 | 0x0F | 1556 | The Bunny Shoppe | 277 | 2008-08-05 16:49:13.980 |
| 2 | 0x000000045000019380006 | 0x000000045000019380004 | 1 | 0x0F | 2 | Progressive Sports | 283 | 2004-10-13 11:15:07.497 |
| 3 | 0x000000045000019380004 | 0x000000045000019380002 | 3 | 0x02 | 6 | Aerobic Exercise Company | 276 | 2004-10-13 11:15:07.497 |
| 4 | 0x000000045000019380004 | 0x000000045000019380002 | 4 | 0x02 | 6 | The Magic Shoppe | 276 | 2004-10-13 11:15:07.497 |

Figure 2: This is what is stored in the Change Data Capture table.

So let's take a look at the above code and see how it works. The line:

```
DECLARE @from_lsn BINARY(10), @to_lsn BINARY(10)
```

is used to declare two variables: one represents the **from_lsn** value and the other represents the **to_lsn** value. Notice that these values use the BINARY data type.

The next two statements are used to identify the **from_lsn** and **to_lsn** values:

```
SELECT @from_lsn = sys.fn_cdc_get_min_lsn('Sales_Store')
SELECT @to_lsn = sys.fn_cdc_get_max_lsn()
```

Now that we have the **from_lsn** and the **to_lsn** values, we can place them in our CDC query function:

```
SELECT * FROM cdc.fn_cdc_get_all_changes_Sales_Store(@from_lsn, @to_lsn, 'all update old');
```

Since I used the option "all update old" as part of the above query function, all the rows in the table were returned, included the row I INSERTED, the row I DELETED, and the row I UPDATED (both the before and after values).

As you might imagine, you could incorporate similar code inside a SSIS package that would periodically execute and get all of the changes from the CDC table, manipulate the data

appropriately, and then move it into a database on an OLAP server. Of course, the code needed to accomplish this goal is more complex than what you see above, but I think you get the idea.

Summary

Change Data Capture offers a new technology that makes it much easier than in previous versions of SQL Server to move changed data from an OLTP server to an OLAP server. Not only is setup and administration much easier than previously available technology, performance has been greatly enhanced. Change Data Capture should be seriously considered as you design new applications, or redesign older applications, whenever you need to move changed data from one database to another.

INDEX

- Activity Monitor, 10, 11, 12
- Audit
 - Creating a Server or Database Audit Specification, 72, 76
 - Creating the Audit Object, 72
 - Reviewing Audit Events, 72, 83
 - Starting the Audit, 72, 83
- Change Data Capture, 8, 103, 104, 105, 106, 107, 109, 110
 - Architecture, 103
 - Capture and Cleanup Jobs, 106
 - Capture Instances, 105
 - How it Works, 106
 - Implementing for a Single Table, 106
 - Query Functions, 105
 - Source Tables, 105
 - Tables, 105
- Compliance, 68
 - Checking Multiple Servers for Compliance, 31
- Data Compression, 7, 34, 36, 37, 38
 - Backup Compression, 34, 41
 - Demo, 36
 - Overview, 34
 - Page Level, 36
 - Row-Level, 36
- Data type
 - Benefits of FILESTREAM, 90
 - Date and Time types, 86
 - DATE data type, 87
 - DATETIME2, 86, 87
 - DATETIMEOFFSET, 86, 87, 88
 - filestream, 63, 86, 89, 90, 91, 92, 93, 94
 - Hierarchy, 86, 89
 - How to Implement FILESTREAM Storage, 91
 - Limitations of FILESTREAM, 90
 - Spatial types, 86, 88
- Data type Spatial types, 89
- Database Encryption, 66
 - Backup the Private Encryption Key and Certificate, 66
 - Create a Key, 65, 66
 - Create or Obtain a Certificate Protected by the Master Key, 65, 66
 - Creating a Master Key, 65
 - Turning TDE On, 65, 67
- Extended Events, 8, 52, 68, 95, 96, 97, 98, 99, 100, 101, 102
 - Benefits and Limitations, 96
 - Creating a Session, 98
 - Overview, 95
 - Sessions, 97
- IntelliSense, 7, 17, 18
- Multi-Server Queries, 10, 16
- Object Explorer, 10, 13, 15, 16
 - Details, 10, 13, 15
- Object Search, 10, 15, 16

- Performance Data Collector, 7, 52, 53, 54, 56, 57, 61, 95
 - How it works, 54
 - How to Configure, 53
 - Reports Available, 57
- Policy-based management
 - Creating the Policy, 27
 - Facets, 23, 24
 - Running the Policy, 29
 - Selecting Facets, 23, 24
 - Setting the required Property Conditions, 26
- Policy-Based Management, 7, 20, 21, 22, 23, 24, 28, 32, 33
 - How to Implement a policy, 24
 - How it Works, 23
 - How You Might Use, 21
 - What it Does, 20
- Resource Governor, 7, 9, 45, 46, 47, 48, 49, 51
 - Configuring, 48
 - Creating Resource Pools, 49
 - Creating the Classification Function, 50
 - Creating Workload Groups, 50
 - Enabling, 51
 - How it Works, 46
 - Uses, 45
- SQL Server Audit, 7, 68, 69, 70, 71, 74, 84, 85, 95
 - A Simple Auditing Example, 71
 - Advantages, 68
 - How it Works, 69
 - Limitations, 69
- Transparent Data Encryption, 7, 9, 42, 62, 63, 64, 65, 66, 67
 - How it Works, 63
 - How to Implement it, 65
 - Limitations, 63
 - Why Use it?, 62
- T-SQL Debugger, 10, 18

