



Quick answers to common problems

# Microsoft SQL Server 2012 Performance Tuning Cookbook

80 recipes to help you tune SQL Server 2012 and achieve optimal performance

Ritesh Shah

Bihag Thaker

**[PACKT]** enterprise   
PUBLISHING professional expertise distilled

# Microsoft SQL Server 2012 Performance Tuning Cookbook

80 recipes to help you tune SQL Server 2012 and achieve optimal performance

**Ritesh Shah**

**Bihag Thaker**



BIRMINGHAM - MUMBAI

# **Microsoft SQL Server 2012 Performance Tuning Cookbook**

Copyright © 2012 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: July 2012

Production Reference: 1160712

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-84968-574-0

[www.packtpub.com](http://www.packtpub.com)

Cover Image by Asher Wishkerman ([a.wishkerman@mpic.de](mailto:a.wishkerman@mpic.de))

# Credits

## **Authors**

Ritesh Shah  
Bihag Thaker

## **Reviewers**

Satya SK Jayanty  
Maria Zakourdaev  
Michael Zilberstein

## **Acquisition Editor**

Dhwani Devater

## **Lead Technical Editor**

Kedar Bhat

## **Technical Editors**

Apoorva Bolar  
Madhuri Das  
Merin Jose

## **Copy Editor**

Brandt D'Mello

## **Project Coordinator**

Sai Gamare

## **Proofreader**

Lesley Harrison

## **Indexer**

Monica Ajmera Mehta

## **Graphics**

Manu Joseph  
Valentina Dsilva

## **Production Coordinator**

Shantanu Zagade

## **Cover Work**

Shantanu Zagade

# About the Authors

**Ritesh Shah** is a data professional with over 10 years of experience with using Microsoft technology, from SQL Server 2000 to the latest version. He has worked with various technologies, from Visual Basic 6.0 to .NET Framework 4.0. He has deployed many medium-scale as well as large-scale projects, using Microsoft technology.

He shares his knowledge on his blog, [SQLHub.com](http://SQLHub.com), and also helps the community, using different portals, such as [BeyondRelational.com](http://BeyondRelational.com), [Experts-Exchange.com](http://Experts-Exchange.com), and Asp.Net forum.

# Acknowledgement

It is really truer than ever that this is not an individual effort. The Packt team worked with me the whole time, so a really big thanks goes to them, especially Sai, Kedar, Apoorva, Madhuri, and many more. I cannot forget to mention Dhvani from the Packt team, as she is the one who presented the idea of this book to me. Seriously, I wouldn't have been able to author this book alone, so thanks should go to Mr. Bihag Thaker, as well, as he agreed to co-author this book with me and has worked even harder on it than I have myself.

I am really honored to have Satya, Michael, and Maria as the technical reviewers for this book. They are all well-known personalities in the world of SQL Server.

Apart from the team that worked on this book, I would also like to thank, on a personal note, two well-known personalities in the SQL Server community, who always inspire me to do more. In fact, they were the ones who diverted my interest from .NET technology to SQL Server. They are:

- ▶ Pinal Dave, who blogs at [SQLAuthority.com](http://SQLAuthority.com) and is an author of several SQL Server books. Currently, he is working as a Technology Evangelist at Microsoft.
- ▶ Jacob Sebastian, who blogs at [BeyondRelational.com](http://BeyondRelational.com) and is a SQL Server MVP, book author, well-known speaker in SQL Server technology, and much more.

Most important of all, my deepest gratitude goes to my parents, Mr. Ashwin Shah and Mrs. Divya Shah. It is because of their hard work, inspiration, and motivation that a small-town boy like me, who has grown up with very limited resources, has progressed so much in life, which in itself proves *where there's a will there's a way*. I would also like to thank my one-and-a-half-year-old son, Teerth, who used to often start crying at midnight, because of which I would lose my sleep and, not being able to get it back, started researching more on the subjects that helped me write this book. Finally, I would like to thank my wife, Alka Shah.

**Bihag Thaker** is a SQL Server enthusiast, an MCTS (SQL Server 2005), and an MCITP (SQL Server 2008), who has been working with SQL Server technology for the past few years. Initially he was into .NET technology, but his keen interest for SQL Server led him to be a database specialist.

He is currently working as a database administrator. He has worked on numerous performance tuning assignments and executed large-scale database migrations. He likes to share his knowledge and enjoys helping the SQL Server community. You will find him talking about SQL Server on his blog [MsSQLBlog.com](http://MsSQLBlog.com).

# Acknowledgement

I had never thought that the dream of writing my first book on SQL Server would come true so early, and I must give full credit for this to Mr. Ritesh Shah and Packt Publishing.

I would sincerely like to thank Packt Publishing, for showing their confidence in me and providing the invaluable opportunity of being a part of this book. Individuals at Packt whom I am deeply grateful to, are Kedar Bhat, Sai Gamare, Madhuri Das, Ashwin Shetty, Apoorva Bolar, and Dhvani Devater. They have been very co-operative and supportive at all the stages of this book. I am extremely thankful to Michael Zilberstein and Maria Zakourdaev, the technical reviewers, for their excellent work of getting the accuracy of the technical details of the book in perfect shape.

I find it difficult to express, in words, my gratitude, to Ritesh, who has shared the priceless gift of writing this book with me. This was not at all attainable without his continuous support. Apart from being a TechMate, Ritesh is an all-time great friend of mine, who is always willing to help the SQL Server community.

Two individuals to whom I am indebted and whose disciple I have always been, are Mr. Paresh Vora and Mr. Mukesh Devmurari. I have learnt a lot from them, and they are the reason I'm part of the IT community today.

Without my family support, a task such as writing a book would not have been achievable. I would like to heartily thank my parents, Mr. Kanaiyalal Thaker and Mrs. Hema Thaker. It is because of them that I exist, and I cherish their blessings, which are always with me. I am very thankful to my wife, Khyati, who has always stood by me, helped me at all times, and has even smilingly got me cups of coffee during my sleepless nights of writing!

Last but not the least, I would like to thank my friends who helped me directly or indirectly by giving me moral support.



# About the Reviewers

**Satya SK Jayanty** is a SQL Server MVP and Subject Matter Expert with consulting and technical expertise for D Bi A Solutions INc. Limited, with over 20 years of experience. His work experience includes a wide range of industries, including the stock exchange, insurance, tele-communications, financial, retail, and manufacturing sectors, among others.

He is a regular speaker and SME volunteer at major technology conferences such as Microsoft Tech-Ed (Europe, India, and North America), and SQL PASS (Europe and North America), SQL Bits - UK, and manages the Scottish Area SQL Server user group based in Scotland. He is also a moderator in a majority of web-based SQL Server forums (Microsoft Technet and [www.sql-server-performance.com](http://www.sql-server-performance.com)), writer, and contributing editor, and blogs at [www.sqlserver-qa.net](http://www.sqlserver-qa.net), [www.sql-server-performance.com](http://www.sql-server-performance.com), and [www.beyondrelational.com](http://www.beyondrelational.com) websites.

He is the author of *Microsoft SQL Server 2008 R2 Administration Cookbook*, Packt Publishing, and co-author of *SQL Server MVP Deep Dives, Volume 2*, Manning Publications.

**Maria Zakourdaev** has more than 10 years of experience with SQL Server. She is currently working with one of the most successful Israeli startup companies, called Conduit. She has extensive knowledge of Microsoft replication solutions, table partitioning, and advanced, query tuning techniques. Prior to Conduit she had worked with different companies, benchmarking different SQL Server features and flows, such as partitioning, data import, index impact on DML flows, star transformations in RDBMS, hierarchic queries, and custom OLAP-like aggregations. She was a speaker in Microsoft Teched (Israel) on the SQL Server track and is an active member of the Israel SQL Server Group.

**Michael Zilberstein** has more than 10 years of experience in the IT industry and database world, working with all the SQL Server versions from 6.5 to 2012 and with different Oracle versions as well. After working with several start-up companies during the first few years of his career, in 2007 Michael founded DBArt Ltd – SQL Server, a consulting services company.

Two of Michael's most distinctive interests (besides rappelling, homebrewing, playing chess, and reading history books) are performance tuning and architecture of large-scale systems. The biggest professional satisfaction for him is to take a young start-up company and build its product from schemas in scrapbook and Visio to a working and scalable terabyte-size system.

Michael is a frequent speaker at Israeli SQL Server Usergroup (ISUG) and other SQL Server events in Israel. He also writes a blog—[http://sqlblog.com/blogs/michael\\_zilberstein/default.aspx](http://sqlblog.com/blogs/michael_zilberstein/default.aspx).

# www.PacktPub.com

## **Support files, eBooks, discount offers and more**

You might want to visit [www.PacktPub.com](http://www.PacktPub.com) for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

## **Why Subscribe?**

- ▶ Fully searchable across every book published by Packt
- ▶ Copy and paste, print and bookmark content
- ▶ On demand and accessible via web browser

## **Free Access for Packt account holders**

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

## **Instant Updates on New Packt Books**

Get notified! Find out when new books are published by following @PacktEnterprise on Twitter, or the *Packt Enterprise* Facebook page.

# Table of Contents

<b>Preface</b>	<b>1</b>
<b>Chapter 1: Mastering SQL Trace Using Profiler</b>	<b>7</b>
Introduction	7
Creating a trace or workload	9
Filtering events	19
Detecting slow running and expensive queries	27
Creating trace with system stored procedures	34
<b>Chapter 2: Tuning with Database Engine Tuning Advisor</b>	<b>45</b>
Introduction	45
Analyzing queries using Database Engine Tuning Advisor	46
Running Database Engine Tuning Advisor for workload	51
Executing Database Tuning Advisor from command prompt	59
<b>Chapter 3: System Statistical Functions, Stored Procedures, and the DBCC SQLPERF Command</b>	<b>63</b>
Introduction	63
Monitoring system health using system statistical functions	64
Monitoring with system stored procedure	69
Monitoring log space usage statistics with DBCC command	75
<b>Chapter 4: Resource Monitor and Performance Monitor</b>	<b>79</b>
Introduction	79
Monitoring of server performance	80
Monitoring CPU usage	86
Monitoring memory (RAM) usage	90

<b>Chapter 5: Monitoring with Execution Plans</b>	<b>95</b>
Introduction	95
Working with estimated execution plan	97
Working with actual execution plan	100
Monitoring performance of a query by SET SHOWPLAN_XML	103
Monitoring performance of a query by SET STATISTICS XML	108
Monitoring performance of a query by SET STATISTICS IO	112
Monitoring performance of a query by SET STATISTICS TIME	116
Including and understanding client statistics	118
<b>Chapter 6: Tuning with Execution Plans</b>	<b>121</b>
Introduction	121
Understanding Hash, Merge, and Nested Loop Join strategies	122
Finding table/index scans in execution plan and fixing them	128
Introducing Key Lookups, finding them in execution plans, and resolving them	133
<b>Chapter 7: Dynamic Management Views and Dynamic Management Functions</b>	<b>145</b>
Introduction	145
Monitoring current query execution statistics	147
Monitoring index performance	155
Monitoring performance of TempDB database	164
Monitoring disk I/O statistics	172
<b>Chapter 8: SQL Server Cache and Stored Procedure Recompilations</b>	<b>179</b>
Introduction	179
Monitoring compilations and recompilations at instance level using Reliability and Performance Monitor	182
Monitoring recompilations using SQL Server Profiler	188
<b>Chapter 9: Implementing Indexes</b>	<b>197</b>
Introduction	197
Increasing performance by creating a clustered index	198
Increasing performance by creating a non-clustered index	206
Increasing performance by covering index	212
Increasing performance by including columns in an index	216
Improving performance by a filtered index	219
Improving performance by a columnstore index	222
<b>Chapter 10: Maintaining Indexes</b>	<b>231</b>
Introduction	231
Finding fragmentation	232
Playing with Fill Factor	234

Enhance index efficiency by using the REBUILD index	237
Enhance index efficiency by using the REORGANIZE index	240
How to find missing indexes	241
How to find unused indexes	244
Enhancing performance by creating an indexed view	247
Enhancing performance with index on Computed Columns	252
Determining disk space consumed by indexes	258
<b>Chapter 11: Points to Consider While Writing Queries</b>	<b>261</b>
Introduction	261
Improving performance by limiting the number of columns and rows	262
Improving performance by using sargable conditions	265
Using arithmetic operator wisely in predicate to improve performance	267
Improving query performance by not using functions on predicate columns	270
Improving performance by Declarative Referential Integrity (DRI)	273
"Trust" your foreign key to gain performance	277
<b>Chapter 12: Statistics in SQL Server</b>	<b>283</b>
Introduction	283
Creating and updating statistics	284
Effects of statistics on non-key column	292
Find out-of-date statistics and get it correct	296
Effect of statistics on a filtered index	299
<b>Chapter 13: Table and Index Partitioning</b>	<b>303</b>
Introduction	303
Partitioning a table with RANGE LEFT	304
Partitioning a table with RANGE RIGHT	311
Deleting and loading bulk data by splitting, merging, and switching partitions (sliding window)	319
<b>Chapter 14: Implementing Physical Database Structure</b>	<b>333</b>
Introduction	333
Configuring data file and log file on multiple physical disks	334
Using files and filegroups	342
Moving the existing large table to separate physical disk	346
Moving non-clustered indexes on separate physical disk	350
Configuring the tempdb database on separate physical disk	354
<b>Chapter 15: Advanced Query Tuning Hints and Plan Guides</b>	<b>359</b>
Introduction	359
Using NOLOCK table query hint	360
Using FORCESEEK and INDEX table hint	363

Optimizing a query using an object plan guide	367
Implementing a fixed execution plan using SQL plan guide	371
<b>Chapter 16: Dealing with Locking, Blocking, and Deadlocking</b>	<b>381</b>
Introduction	381
Determining long-running transactions	382
Detecting blocked and blocking queries	384
Detecting deadlocks with SQL Server Profiler	388
Detecting deadlocks with Trace Flag 1204	395
<b>Chapter 17: Configuring SQL Server for Optimization</b>	<b>399</b>
Introduction	399
Configuring SQL Server to use more processing power	400
Configuring memory in 32 bit versus. 64 bit	403
Configuring "Optimize for Ad hoc Workloads"	405
Optimizing SQL Server instance configuration	410
<b>Chapter 18: Policy-based Management</b>	<b>415</b>
Introduction	415
Evaluating database properties	416
Restricting database objects	422
<b>Chapter 19: Resource Management with Resource Governor</b>	<b>427</b>
Introduction	427
Configuring Resource Governor with SQL Server Management Studio	429
Configuring Resource Governor with T-SQL script	436
Monitoring Resource Governor	442
<b>Index</b>	<b>447</b>

# Preface

*Microsoft SQL Server 2012 Performance Tuning Cookbook* is divided into three major parts—Performance Monitoring, Performance Tuning, and Performance Management—that are mandatory for dealing with performance in any capacity.

*Microsoft SQL Server 2012 Performance Tuning Cookbook* offers a great way to manage performance with effective, concise, and practical recipes. You will learn how to diagnose performance issues, fix them, and take precautions to avoid common mistakes.

Each recipe given in this book is an individual task that will address different performance aspects to take your SQL Server's Performance to a higher level.

The first part of this book covers monitoring with SQL Server Profiler, DTA, system statistical functions, SPs with DBCC commands, Resource Monitor, Reliability and Performance Monitor, and execution plans.

The second part of the book offers execution plan, dynamic management views and dynamic management functions, SQL Server Cache, stored procedure recompilations, indexes, important ways to write effective T-SQL, statistics, table and index partitioning, advanced query tuning with query hints and plan guide, dealing with locking, blocking, and deadlocking, and configuring SQL Server for optimization to boost performance.

The third and final part gives you knowledge about performance management with the help of policy based management and management with Resource Governor.



## What this book covers

*Chapter 1, SQL Server Profiler*, teaches you how to create and start your first SQL Trace, limit the trace data and capture only the events which are of interest, detect slow running and expensive queries, and create a trace with system stored procedures.

*Chapter 2, Tuning with Database Engine Tuning Advisor*, covers how to analyze queries using Database Engine Tuning Advisor, how to run Database Engine Tuning Advisor for Workload, and how to execute Database Tuning Advisor from the command prompt.

*Chapter 3, System Statistical Functions, System Stored Procedures, and DBCC SQLPERF Command*, starts with the monitoring of system health using system statistical functions and later on covers the monitoring of SQL Server processes and sessions with system stored procedures, and log space usage statistics with the DBCC SQLPERF command.

*Chapter 4, Resource Monitor and Performance Monitor*, teaches you how to do quick monitoring of server performance, followed by monitoring of CPU and memory (RAM) usage.

*Chapter 5, Monitoring with Execution Plans*, includes recipes for working with Estimated Execution Plan and Actual Execution Plan, monitoring the performance of queries by SET SHOWPLAN\_XML, SET STATISTICS XML, and SET STATISTICS IO, finding the execution time of a query by SET STATISTICS TIME, and including and understanding Client Statistics.

*Chapter 6, Tuning with Execution Plans*, explains the Hash, Merge, and Nested Loop Join strategies, teaches how to find table/index scans in execution plans and how to fix them, introduces Key Lookups, and explains how to find them in execution plans and resolve them.

*Chapter 7, Dynamic Management Views and Dynamic Management Functions*, includes recipes to monitor current query execution statistics, manage and monitor index performance, monitor the TempDB database's performance with database-related dynamic management views, and monitor disk I/O statistics.

*Chapter 8, SQL Server Cache and Stored Procedure Recompilations*, covers monitoring of compilations and recompilations at instance level, using Reliability and Performance Monitor, and monitoring of recompilations using SQL Server Profiler.

*Chapter 9, Implementing Indexes*, explains how to improve performance by creating a clustered index, by creating a non-clustered index, by covering index, by including columns in an index, by a filtered index, and by a columnstore index.

*Chapter 10, Maintaining Indexes*, includes recipes to find fragmentation, to enhance index efficiency by using the REBUILD and REORGANIZE index, to find missing and unused indexes, to enhance performance by creating indexed views and creating an index on Computed Columns, and to determine disk space consumed by indexes.

*Chapter 11, Points to Consider While Writing Query*, covers how to improve performance by limiting the number of columns and rows and by using sargable conditions, how to use arithmetic operators wisely in predicate to improve performance, how to improve query performance by not using functions on predicate columns, how to improve performance by Declarative Referential Integrity (DRI), and how to gain performance by trusting your foreign key.

*Chapter 12, Statistics in SQL Server*, explains how to create and update statistics, effects of statistics on non-key columns, how to find out-of-date statistics and correct them, and effects of statistics on a filtered index.

*Chapter 13, Table and Index Partitioning*, covers partitioning of table with RANGE LEFT and RANGE RIGHT, and deleting and loading of bulk data by splitting, merging, and switching partitions (sliding window).

*Chapter 14, Implementing Physical Database Structure*, includes recipes for configuring a data file and log file on multiple physical disks, using files and filegroups, moving an existing large table to a separate physical disk, moving non-clustered indexes to a separate physical disk, and configuring the TempDB database on a separate physical disk.

*Chapter 15, Advanced Query Tuning: Hints and Plan Guides*, includes recipes for using the NOLOCK table query hint, using the FORCESEEK and INDEX table hints, optimizing a query using an object plan guide, and implementing a fixed execution plan using a SQL plan guide.

*Chapter 16, Dealing with Locking, Blocking, and Deadlocking*, covers determining long-running transactions, detecting blocked and blocking queries, detecting deadlocks with SQL Server Profiler, and detecting deadlocks with Trace Flag 1204.

*Chapter 17, Configuring SQL Server for Optimization*, includes recipes for configuring SQL Server to use more processing power, configuring memory in 32-bit versus 64-bit, configuring "Optimize for Ad hoc Workloads", and optimizing SQL Server instance configuration.

*Chapter 18, Policy Based Management*, explains how to evaluate database properties and restrict database objects.

*Chapter 19, Management with Resource Governor*, includes recipes for configuring Resource Governor with SQL Server Management Studio and T-SQL script, and monitoring Resource Governor.

## What you need for this book

To work with the examples given in the book, you must have the following infrastructure:

- ▶ SQL Server Denail CTP version 3 or higher, or SQL Server 2012 RTM
- ▶ The AdventureWorks2012 database, which can be freely downloaded from the following link: <http://msftdbprodsamples.codeplex.com/releases/view/55330>
- ▶ A Windows administrator login and/or a SQL server login with the sysAdmin privilege

## Who this book is for

*Microsoft SQL Server 2012 Performance Tuning Cookbook* is aimed at SQL Server Database Developers, DBAs, and Database Architects who are working in any capacity to achieve optimal performance. Basic knowledge of SQL Server is expected, and professionals who want to get hands-on with performance tuning and have not worked on tuning the SQL Server for performance will find this book helpful.

## Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "You may notice some `TextData` appearing multiple times in a trace for a single execution of a T-SQL statement."

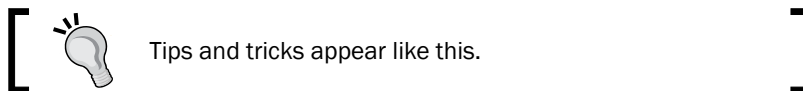
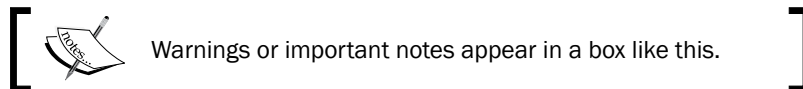
A block of code is set as follows:

```
--creating table for demonstration
CREATE TABLE ordDemo (OrderID INT IDENTITY, OrderDate DATETIME, Amount
MONEY, Refno INT)
GO
```

Any command-line input or output is written as follows:

```
dta -D AdventureWorks2012 -s adventureworks2012FromDTA5 -S WIN-
SLYJ9UY3PKD\DENALICTP3 -E -if D:\test.sql -F -of D:\DTA.sql
```

New terms and important words are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Connect object explorer with server and move to **Management** | **Policy Management** | **Policies**".



## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book title through the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website, or added to any list of existing errata, under the Errata section of that title.

## Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

## Questions

You can contact us at [questions@packtpub.com](mailto:questions@packtpub.com) if you are having a problem with any aspect of the book, and we will do our best to address it.



# 1

## Mastering SQL Trace Using Profiler

In this chapter we will cover:

- ▶ Creating a trace or workload
- ▶ Filtering events
- ▶ Detecting slow running and expensive queries
- ▶ Creating trace with SQL Trace system stored procedures

### Introduction

Welcome to the world of Performance Monitoring and Tuning with SQL Server 2012!

Let's assume that you are a database administrator in your organization. What, if one day one of your colleagues from your IT department calls you right away and complains that the production database server has abruptly started to run very slowly and applications that are accessing the production database are not responding the way they should? The issue needs immediate attention and for that you are required to investigate the issue and fix it in timely manner. What will be your approach to look at the problem and solve it? How would you be able to analyze the situation and identify where the problem is? What actions would you take once a particular problem is recognized in order to resolve it?

Installing and upgrading database servers, managing and maintaining database servers, managing database security, implementing disaster recovery plan, capacity planning, managing high-availability of databases, and performance tuning of databases and SQL server are some of the responsibilities of a DBA. Amongst these responsibilities, performance tuning of the database server is one of the prime responsibilities of DBA. The most common reason is, companies offering IT services are often engaged in signing **Service Level Agreements (SLAs)** and as per their SLAs they are committed to provide a certain level of services and up-time. Any additional down-time than what is allowed as per SLAs can cause them money loss or business loss. Even companies not engaged in SLAs might lose business because of their poor software systems caused by poor database systems. This is one of the reasons why skilled DBAs are required to keep the database performance up-to date by monitoring and tuning database performance.

In database centric application environment, it is very common for any DBA to face such database related performance issues at different levels. By means of different levels, it implies that performance problem can be found at query level, database level, server level or application level. There can be a number of reasons for a database centric application to be performing poorly. The troubleshooting skills and expertise in performance tuning of a DBA are tested out in recognizing such factors behind the performance degradation and taking the necessary corrective steps.

The first step towards performance tuning is monitoring. In data platform, monitoring something is the process of analyzing and identifying something. So, until you monitor something, you can't know for sure what and where the problem is. Until you know what and where the problem is, you can't analyze the problem. And until you can analyze the problem, you can't solve a problem! This also means that unless you understand performance monitoring, you cannot master performance tuning in a true sense. Thus, performance tuning always comes after performance monitoring. This is the reason why we have a few opening chapters that specifically concentrates on performance monitoring.

The troublesome situation that was just described earlier needs thorough monitoring and systematic analysis in order to identify the root problem accurately before a problem can be solved.

**SQL Server Profiler** is the most common but powerful tool for monitoring and auditing an instance of SQL server. By using this tool, a DBA is able to solve a large number of different types of database performance issues whether it is a query issue, index issue, locking issue or database, or server configuration issue. It is the tool that essentially any DBA must know. So, SQL Server Profiler will be the subject of this first chapter.

## Creating a trace or workload

If you have never worked with SQL Server Profiler, this recipe will teach you how to create and start your first SQL Trace. There is some detailed information on SQL Trace in *There's more...* section of this recipe. This will help you in appreciating rest of the recipes quite easily, which employs SQL Trace in remaining chapters. The section covers the information that will help you in mastering core concepts of SQL Trace and thus mastering SQL Server Profiler. There are no major changes in SQL Server Profiler 2012 documented. In SQL Server 2012, the architecture and functionality of SQL Server Profiler is almost identical to that of SQL Server 2008.

### Getting ready

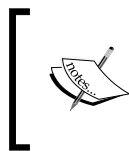
In this recipe, we will create our first trace with SQL Server Profiler. The following are the prerequisites that you should fulfil:

- ▶ An instance of SQL Server 2012 Developer or Enterprise Evaluation edition.
- ▶ An SQL Server Login account with administrative rights.
- ▶ Sample AdventureWorks2012 database on the instance of SQL Server. For more details on how to install AdventureWorks2012 database, please refer to the *Introduction* section of this book.

### How to do it...

To create a new trace, follow the steps provided here.

1. Start SQL Server Profiler. To start SQL Server Profiler, navigate through **Start | All Programs | Microsoft SQL Server 2012 Program Group | Performance Tools | SQL Server Profiler**.
2. Select **New Trace...** from the **File** menu. In the **Connect to Server** dialog box, provide connection details of SQL Server hosting AdventureWorks2012 database and click on **Connect**.

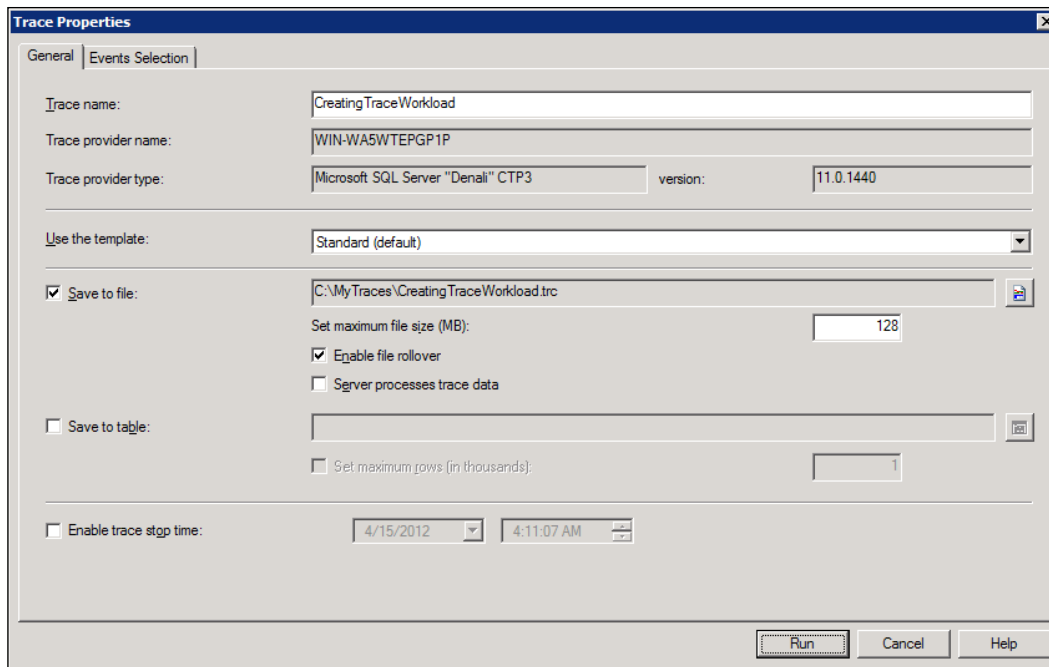


Login name that you use to connect SQL Server Profiler must have the **ALTER TRACE** permission otherwise you will receive an error and cannot start a trace session.

3. In the **General** tab of the **Trace Properties** dialog box, specify **CreatingTraceWorkload** as trace name. Use the **Standard (default)** trace template for the **Use the template:** option.



4. Check the checkbox **Save to file:** and specify a path and file name in the **Save As** dialog box and then click on **Save**.
5. Keep **Enable file rollover** checked and **Set maximum file size (MB):** to its default value, that is, 128. The following screenshot shows the **General** tab of the **Trace Properties** dialog box:



The screenshot shows the 'Trace Properties' dialog box with the 'General' tab selected. The 'Trace name' is 'CreatingTraceWorkload', 'Trace provider name' is 'WIN-WA5WTEPGP1P', and 'Trace provider type' is 'Microsoft SQL Server "Denali" CTP3'. The 'version' is '11.0.1440'. The 'Use the template' dropdown is set to 'Standard (default)'. The 'Save to file' checkbox is checked, with the file path 'C:\MyTraces\CreatingTraceWorkload.trc' and a file icon button. The 'Set maximum file size (MB)' is set to '128'. The 'Enable file rollover' checkbox is checked, and the 'Server processes trace data' checkbox is unchecked. The 'Save to table' checkbox is unchecked, with a table icon button. The 'Set maximum rows (in thousands)' is set to '1'. The 'Enable trace stop time' is set to '4/15/2012' and '4:11:07 AM'. At the bottom are 'Run', 'Cancel', and 'Help' buttons.

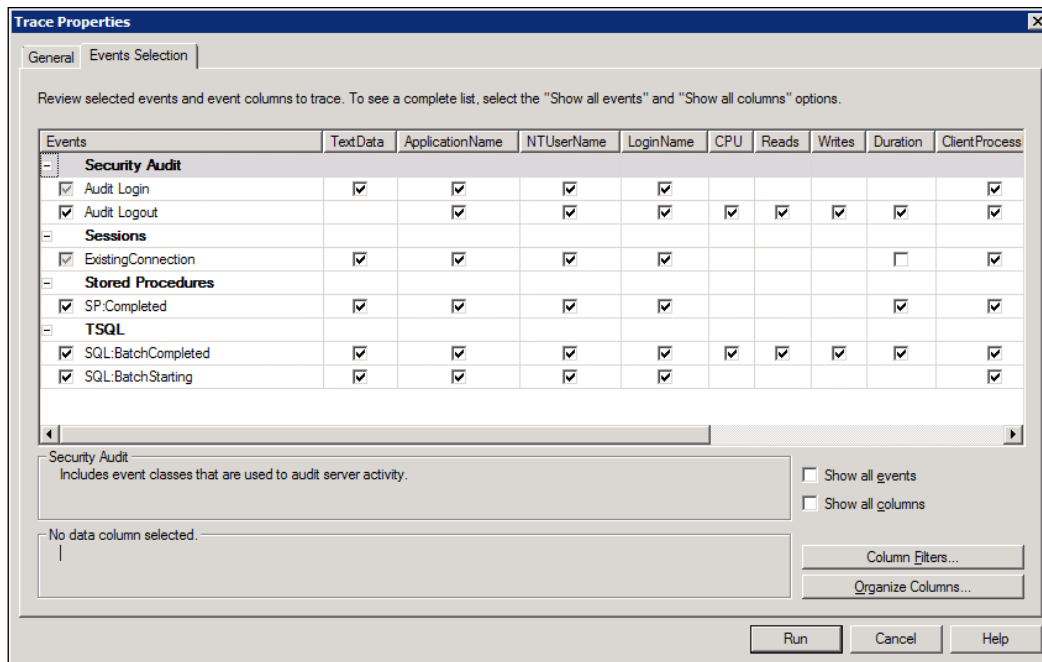


In the **Trace Properties** dialog box, there is a checkbox option in the **General** tab with the caption **Server processes trace data**, to specify whether trace data should be processed on the server. If not checked, trace data is processed at the client side.

When trace data is processed at the client side, it is possible for some events to be missed if the server load is high. If this option is checked, then trace data is processed on the server and all the events included in trace definition are guaranteed to be captured without miss. However, this guarantee comes with performance penalty, because processing trace data on server has an impact on the performance of SQL Server, and hence enabling this option is not recommended on production server.

Also, running SQL Server Profiler on production server itself should be avoided as running SQL Server Profiler is resource consuming. Instead, you should run SQL Server Profiler from a client computer and connect it to your SQL Server from there.

6. Click on the **Events Selection** tab. On this screen, the events that are predefined for the **Standard (default)** trace template are selected and shown in grid. Check the **Show all events** check box to show all events.
7. Navigate through the **Events** list until you find **Stored Procedures** event category. Expand **Stored Procedures** event category if it is collapsed. Uncheck the checkbox for **RPC:Completed** event and check the checkbox for **SP:Completed** event. Uncheck the **Show all events** checkbox to show only selected events. The screen should now look as shown in following screenshot:



8. Click on the **Run** button to start the trace.
9. Now open SQL Server Management Studio and establish a connection to the same SQL Server.
10. In query window, type the sample T-SQL statements as shown in following script and then execute them by pressing the **F5** key:

```
USE AdventureWorks2012
GO

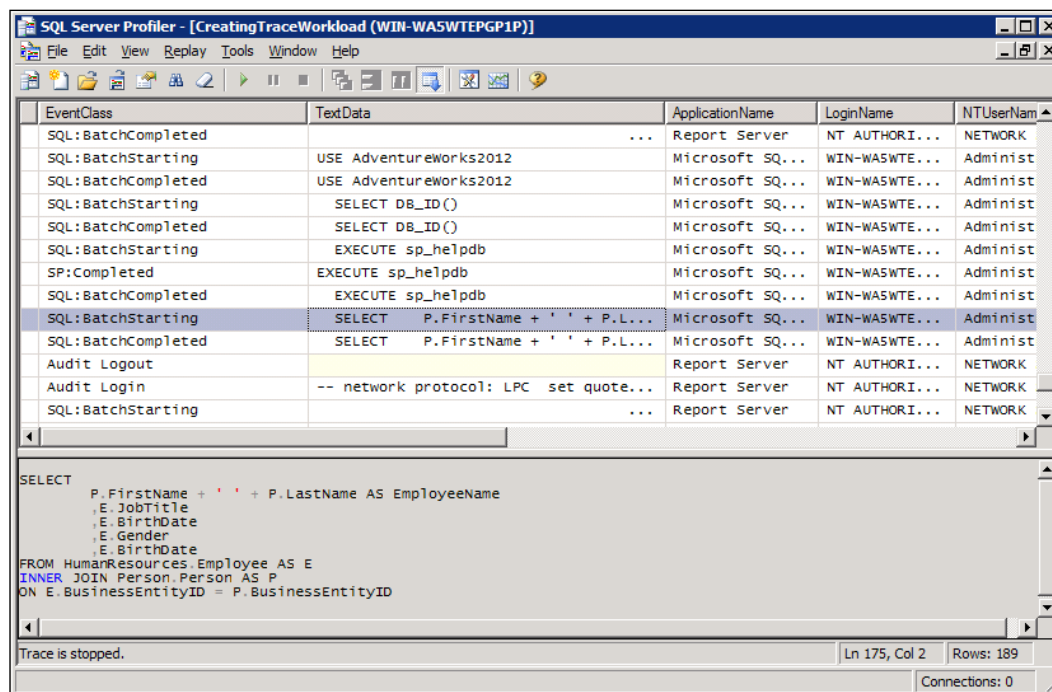
SELECT DB_ID()
GO

EXECUTE sp_helpdb
```

GO

```
SELECT
    P.FirstName + ' ' + P.LastName AS EmployeeName
    ,E.JobTitle
    ,E.BirthDate
    ,E.Gender
    ,E.BirthDate
FROM HumanResources.Employee AS E
INNER JOIN Person.Person AS P
ON E.BusinessEntityID = P.BusinessEntityID
GO
```

- Now switch to the **SQL Server Profiler** window and stop the trace by clicking **Stop selected trace** button in toolbar. Observe the events captured in the trace. The following screenshot shows the captured events that are displayed in SQL Server Profiler:



## How it works...

We started to configure a trace by setting a few trace properties. To demonstrate how we can use one of the in-built trace templates to get a quick start, we used the default trace template **Standard (default)** in this example. When this template is used, the following events are selected by default:

- ▶ Audit Login
- ▶ Audit Logout
- ▶ ExistingConnection
- ▶ RPC:Completed
- ▶ SQL:BatchCompleted
- ▶ SQL:BatchStarting



You may notice some `TextData` appearing multiple times in a trace for a single execution of a T-SQL statement. For instance, in the previous example, you will notice two events for `SELECT DB_ID()` statement even if we executed it only once. These two entries here do not represent two executions of the said statement. Rather, they represent two different related events associated to one single execution of the statement. For example, both events `SQL:BatchStarting` and `SQL:BatchCompleted` raised for a single execution of batch containing `SELECT DB_ID()` statement and they both show the same T-SQL command in `TextData` data column. This depends upon what events you have selected in trace definition.

In the **Trace Properties** dialog box, we have set the maximum file size for our trace to 128 MB. Option **Enable file rollover** was also enabled by default. Enabling this option is helpful while working with large amount of trace data.

When large amount of event data is captured, the trace file can grow very quickly and become very large. Enabling the **Enable file rollover** option can prevent a trace file from becoming very large by limiting a file to the maximum file size specified. When the file size is reached to the maximum file size specified, SQL Server creates a new roll-over file with the same name appended with a suffix of an incremental number for the same trace. Thus, when we have this option enabled and the size of trace data is greater than maximum file, we have multiple trace files for the same trace.

In this example, we are saving our trace file as `C:\MyTraces\CreatingTraceWorkload.trc`. A trace can also be started without having to save the trace data. In case a trace was started in this way without enabling the **Save to file:** checkbox, SQL Server manages to keep the captured data in queue temporarily. The unsaved trace data can be saved later on as well after gathering the required data. This can be done with the `Save` or `Save As` command from the **File** menu. With the **Save As** command, we can save trace data in our desired format. Selecting the **Trace Table...** option in the `Save As` command, asks for the SQL Server connection details and destination table details where the trace data will be stored.

It's best to store the trace file on a separate disk other than the one which is used to store data files and log files of SQL server databases. Storing the trace file on the same physical disk where database files are stored can degrade the performance of normal I/O operations of other databases.



Configuring a trace by enabling the **Save to table** checkbox in the **Trace Properties** dialog box and saving trace data directly to trace table is less efficient. If you want your trace data to be saved in a trace table then consider saving the trace data first in a trace file; then export your trace data from trace file to trace table by opening the trace file in SQL Server Profiler and selecting the **Save As** command from the **File** menu with the **Trace Table...** option. When you want to save your trace in a trace table, always consider to save your trace in a separate database.

The **Events Selection** tab of **Trace Properties** dialog box displays the selected events only and does not show all events by default. So, we checked the **Show all events** option to list all the available events. Because we did not want to capture **RPC:Completed** event, we excluded this event by un-checking its checkbox from the event list and included **SP:Completed** event under **Stored Procedures** event category.

Once we finished configuring our trace, the trace was started. To demonstrate how the events are captured, we produced some events by executing a few T-SQL statements from another connection through SQL Server Management Studio.

In the final figure, we can see the trace data that is produced by the events included in trace definition. Look at the trace data that we captured. By looking at the values in different data columns, we can learn many different things. For example, for a given trace, by examining `LoginName`, `TextData`, and `HostName` we can tell who is running which query and from which machine. By examining `StartTime` and `EndTime` data columns we can determine when a particular query was executed and when it finished its execution.



#### Pausing and Stopping a trace

Once a trace is started, it can be either paused or stopped. To do this, select the **Run Trace**, **Pause Trace**, and **Stop Trace** commands from the **File** menu or click on the corresponding shortcut command buttons on standard toolbar.

**Pausing and resuming trace:** When a trace is paused, event data stops from being captured temporarily. Once a trace is paused, it can be resumed by starting it again. Restarting a trace resumes and continues to capture event data again without wiping out any previously captured trace data.

**Stopping and restarting trace:** When a trace is stopped, event data stops from being captured. If a trace is stopped, it can be restarted by starting it again. Restarting a stopped trace starts to capture event data again; but any previously captured trace data is lost.

Remember that we cannot change the **Trace Properties** of a trace while it is running. To do this, we must have to pause or stop the trace.

## There's more...

This section covers some essential information on SQL Trace that you must know if you want to master SQL Tracing. It is advised that even if you are an advanced user, you do not skip this section.

### Some background of SQL Trace

Follow this section in order to have an in-depth understanding of SQL Trace and its architecture.

#### SQL Trace terms and concepts

Understanding the SQL Trace and its architecture by knowing its related terms and concepts is a prerequisite for working with SQL Server Profiler effectively. This section discusses the basic terminologies and concepts of SQL Trace in brief.

#### SQL Trace

**SQL Trace** is an event monitoring and capturing engine that comes with SQL Server. It provides the capability to capture the database events with event data and create traces that can be used for performance analysis afterwards.

#### SQL Server Profiler

SQL Server Profiler is a graphical user interface tool for working with SQL Trace. Behind the scene, it uses the same SQL Trace engine, but additionally provides graphical user interface to the user for working with traces. SQL Server Profiler provides functionalities, such as displaying collected event data on its graphical interface, saving traces either in a file or in an SQL Server table, opening previously saved traces, extracting T-SQL statements from a trace, and many more. Finding and analyzing long running or costly queries, finding deadlocks and their related information, looking for which indexes are scanned, and looking for database connection requests are some of the practical applications of SQL Server Profiler.

#### Event

In context of SQL Trace terminology, an **event** is the happening of a database activity that takes place within an instance of SQL Server. Execution of an ad-hoc query or T-SQL batch, a call to stored procedure, an attempt to log in or log out from database server are a few examples that raise specific SQL Server events.

#### Event class

An **event class** describes a specific type of event. There are many different types of events that can occur within the database engine and each type of event is represented by an event class. Audit Login, Audit Logout, SP:Completed, SP:Recompile, SQL:BatchCompleted, Lock:Deadlock are some of the examples of event classes. To get list of all available event classes, you can query `sys.trace_events` catalog view.

### Event category

An event category is a subset of related event classes. Each event class belongs to a particular event category and each event category includes a subset of specific type of event classes. Locks, performance, scans, and stored procedures are some examples of the event categories. To get list of all available event categories, you can query `sys.trace_categories` catalog view. You can join `sys.trace_events` and `sys.trace_categories` catalog views on `category_id` column to make correlation between the two views.

### Data column

A **data column** is an attribute that represents a particular characteristic of an event class. For example, event class `SQL:BatchCompleted` can have different characteristics, such as `TextData`, `LoginName`, `Duration`, `StartTime`, `EndTime`, and so on, where `TextData` represents T-SQL statement(s) whose execution raises a particular event. These characteristics of event classes are represented by different data columns.

### Trace

A session that performs the activity of capturing database events and collecting events' data is typically called a **trace**. Loosely, the term Trace is also used by database professionals to refer the *Trace Data* that has been collected previously during a trace session and saved in a trace file or SQL Server table.

### Trace properties and Trace definition

A set of configured settings for a trace that defines how event data should be collected or saved and which event classes or data columns should be collected as a part of trace data is called Trace properties or a Trace definition.

### Filter

A **filter** is an optional logical condition that can be applied to a trace to limit the resulting trace data by capturing only the required trace events for which the filter condition is satisfied. For example, in a trace definition we can specify a filter condition so that SQL Trace collects event data only for a specific database by applying a filter on either `DatabaseID` data column or `DatabaseName` data column.

### Trace file

This is a file with the extension `.trc` in which the captured trace data is saved.

### Trace table

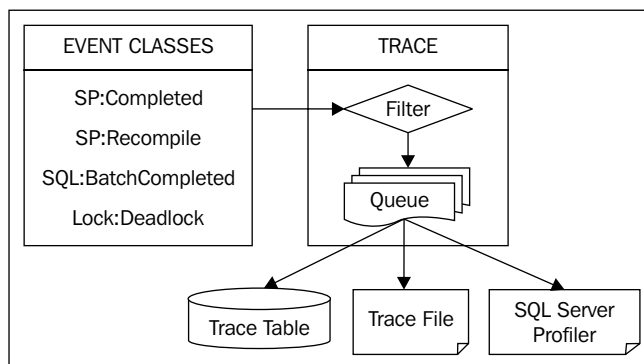
A table in SQL Server database in which the captured trace data is stored is a **trace table**.

### Trace template

A file which saves the pre-configured trace definitions is called a **Trace Template**. This can be reused for creating new traces.

## Architecture of SQL Trace

After learning the basic SQL Trace terms and concepts, it will be easier to understand the following architectural diagram of SQL Trace:



When events are raised in SQL Server database engine, SQL Trace captures event data only for those event classes that are included in trace definition and for which filter conditions if specified any are satisfied. Once the event data is captured, it is queued and then sent to its specified target location. The target location can be a Trace file, Trace table, or SQL Server Profiler. Trace data can also be viewed only in SQL Server Profiler without the need of saving a trace.

After understanding the basic concepts of SQL Trace, working with SQL Server Profiler and traces should be an easy task. As this is our first recipe of the book where we learn how to create a trace or workload with SQL Server Profiler, let's first discuss something about trace and workload.

## Trace and workload

We now know that a trace is a session during which the events are captured and event data is collected. SQL Server supports few formats for saving this collected trace data. We can save trace data in one of the following formats:

- ▶ A trace file with `.trc` extension name
- ▶ A trace file in XML format with `.xml` extension name
- ▶ A trace table in an SQL Server database

A **trace** contains a series of events and every event has its associated event data. All the events of a trace and their event data collectively form trace data for a trace file. Data columns associated with trace events form the event data. T-SQL statements whose execution causes the events to be raised are also a part of this event data under `TextData` data column and are themselves included in trace data.



A **workload** or workload file basically contains a series of T-SQL statements. A T-SQL script is an example of a workload file. Because trace data also contains a series of T-SQL statements as a part of event data (as **TextData** Column), they are also used as workloads. Thus, a T-SQL script, trace file (.trc or .xml), trace table, all can be considered as workload. In other words, a trace file is also a workload file. This workload can be used to re-run on a database for workload or performance analysis. Usually, a workload file is provided as input file to **Database Engine Tuning Advisor (DTA)** for a tuning session. You will learn more about Database Engine Tuning Advisor in *Chapter 2, Tuning with Database Engine Tuning Advisor*.

### Commonly-used event classes

The following list gives brief descriptions of commonly used event classes:

- ▶ **Audit Login**: This event occurs when a user connects and logs in to SQL Server
- ▶ **Audit Logout**: This event occurs when a users disconnects and logs out from SQL Server
- ▶ **RPC:Starting**: This event occurs when a **Remote Procedure Call (RPC)** starts executing
- ▶ **RPC:Completed**: This event occurs when a **Remote Procedure Call (RPC)** completes its execution
- ▶ **SQL:BatchStarting**: This event occurs when a T-SQL batch starts executing
- ▶ **SQL:StmtStarting**: This event occurs when a statement inside a T-SQL batch starts executing
- ▶ **SQL:StmtCompleted**: This event occurs when a statement inside a T-SQL batch completes its execution
- ▶ **SQL:BatchCompleted**: This event occurs when a T-SQL batch completes its execution
- ▶ **SP:Starting**: This event occurs when a stored procedure starts executing
- ▶ **SP:StmtStarting**: This event occurs when a statement inside a stored procedure starts executing
- ▶ **SP:StmtCompleted**: This event occurs when a statement inside a stored procedure completes its execution
- ▶ **SP:Completed**: This event occurs when a stored procedure completes its execution

### Commonly-used data columns

The following list gives brief descriptions of commonly used event classes:

- ▶ **ApplicationName**: This data column represents the name of the client application causing a trace event to occur
- ▶ **DatabaseID**: This data column represents the internal system assigned ID of the database for which a trace event occurs

- ▶ **DatabaseName:** This data column represents the name of the database for which a trace event occurs
- ▶ **HostName:** This data column represents the name of the host or computer where the client component connecting to SQL Server causes a trace event to occur
- ▶ **LoginName:** This data column represents the name of the login under whose security context, particular T-SQL statement(s) executes that causes trace event to occur
- ▶ **ObjectID:** This data column represents the internal system assigned ID of an object for which a trace event occurs
- ▶ **ObjectName:** This data column represents the name of an object for which a trace event occurs
- ▶ **SessionLoginName:** This data column represents the name of the login who initiated the connection and under whose security context a trace event occurs
- ▶ **SPID:** This data column represents the Server Process ID or Session ID of the connection which causes a trace event to occur



For a complete list of event classes and data columns of SQL Trace with their description, you can refer product documentation for SQL Server 2012 at [msdn.microsoft.com/en-us/library/bb418432\(v=sql.10\).aspx](http://msdn.microsoft.com/en-us/library/bb418432(v=sql.10).aspx).

## Filtering events

Running a trace which is configured to collect large number of events is not best practice. While collecting trace data, SQL Trace itself can introduce overhead and affect the performance of SQL Server if trace is configured to collect too much trace information. This also depends on whether the trace is server-side trace or client-side trace. If the trace is client-side using profiler, then the performance overhead can be greater.

Also, if large number of trace data is captured, the size of the trace file immediately grows very big and it becomes a difficult job for us to look for the right data in the trace. Therefore, any unnecessary or irrelevant trace data should not be collected.

This is the reason why we should consider limiting the resulting trace data and capturing only the events which are of our interest. For this, we should identify what trace data we need to look at and based upon that we should identify the filters that are applied to our trace.



Collecting large amount of trace data can affect the performance of SQL Server. So, before creating a trace, we should identify the type of analysis we want to perform on trace information. A single trace should not be created for multiple types of analysis. For each analysis type, a separate trace should be created until and unless different types of analysis explicitly need to be combined into single trace for performing correlative analysis. For example, rather than creating a single trace that collects both scan events and lock events for index scan analysis and object locking analysis respectively, we should consider creating two separate traces; one for collecting only scan events and another for collecting lock events only.

## Getting ready

In this recipe, we will see how to capture only those trace events that occurred for a specific database and from a specific SQL Server login.

Let's assume that sample database AdventureWorks2012 is our production database on our production server, which is hosting other databases also. One of the database users *James* complains that he faces some problems while running queries against database AdventureWorks2012. So, we want to trace his session only for database AdventureWorks2012. Because there are also other databases hosted on the same production server and many users are accessing AdventureWorks2012 database, we need to filter trace events based on session login name and database name in order to avoid any unwanted trace data from being collected.

To emulate this case practically, we need the following as prerequisites:

- ▶ An instance of SQL Server 2012 Developer or Enterprise Evaluation edition
- ▶ An SQL Server Login account with **sysadmin** rights
- ▶ The sample AdventureWorks2012 database on the instance of SQL Server. For more details on how to install AdventureWorks2012 database, please refer the *Introduction* section of this book.
- ▶ Two SQL Server logins named *James* and *Peter* with some permission on AdventureWorks2012 database.

## How to do it...

We will be performing three main actions in this example. These are as follows:

- ▶ Creating the required logins and users in the AdventureWorks2012 database (*James* and *Peter*)
- ▶ Creating a trace by applying filters on the `DatabaseName` and `SessionLoginName` data columns

- ▶ Executing sample queries from two separate connections belonging to *James* and *Peter* respectively and observing the trace data

Because two SQL Server logins named *James* and *Peter* with permissions on AdventureWorks2012 database are required, create them by performing the following steps:

1. Open SQL Server Management Studio.
2. Connect to the instance of SQL Server with login account having sysadmin rights.
3. Execute the following T-SQL script to create the logins and their corresponding users in the AdventureWorks2012 database for James and Peter:

```
--Creating Login and User in
--AdventureWorks2012 database for James
USE [master]
GO
CREATE LOGIN [James] WITH PASSWORD=N'JamesPass123'
,DEFAULT_DATABASE=[AdventureWorks2012]
,CHECK_EXPIRATION=OFF
,CHECK_POLICY=OFF
GO
USE [AdventureWorks2012]
GO
CREATE USER [James] FOR LOGIN [James]
GO
ALTER ROLE [db_owner] ADD MEMBER [James]
GO

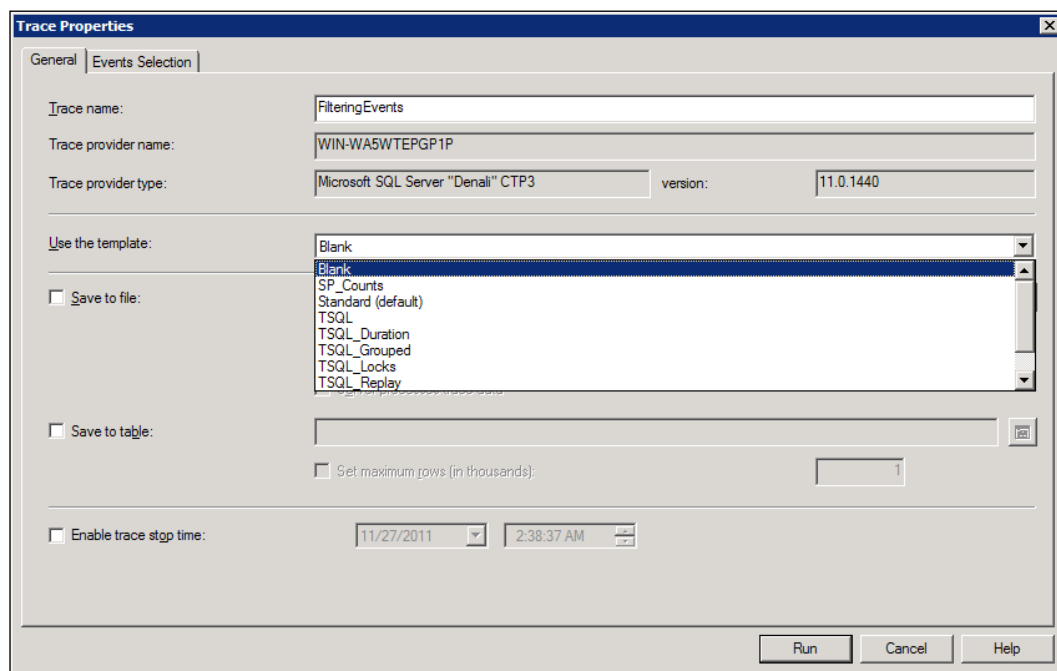
--Creating Login and User in AdventureWorks2012 database for Peter
USE [master]
GO
CREATE LOGIN [Peter] WITH PASSWORD=N'PeterPass123'
,DEFAULT_DATABASE=[AdventureWorks2012]
,CHECK_EXPIRATION=OFF
,CHECK_POLICY=OFF
GO
USE [AdventureWorks2012]
GO
CREATE USER [Peter] FOR LOGIN [Peter]
GO
ALTER ROLE [db_owner] ADD MEMBER [Peter]
GO
```



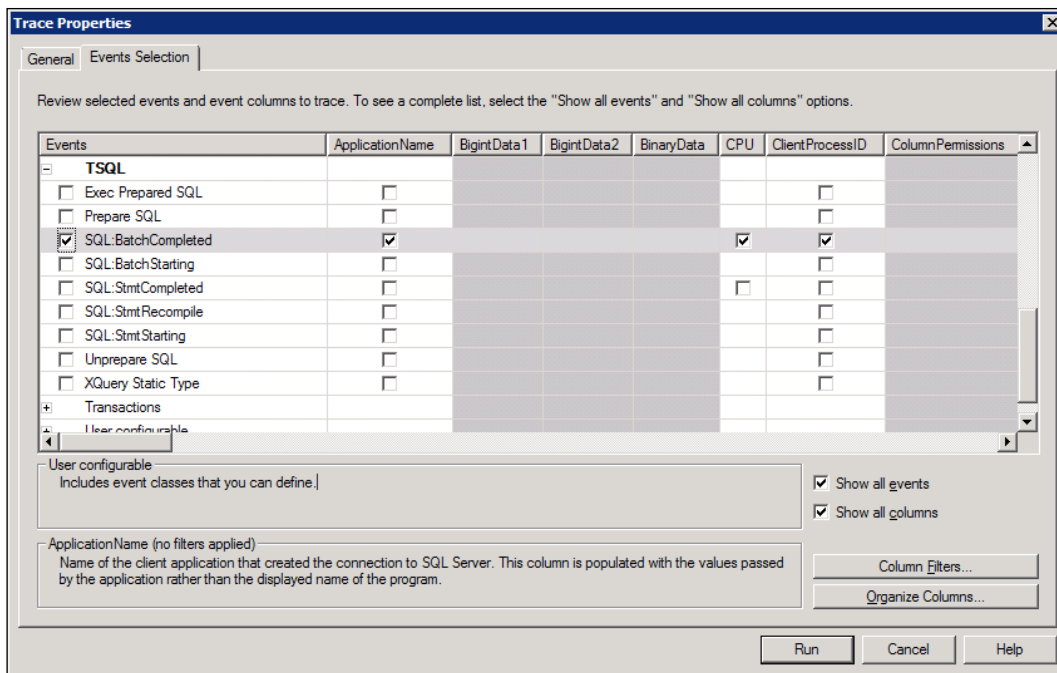
Notice the new command syntax in this script introduced in SQL Server 2012 for adding members to a role.

Now, we will create a trace and capture only events that occur for AdventureWorks2012 database from *James'* session only. To do this, follow these steps:

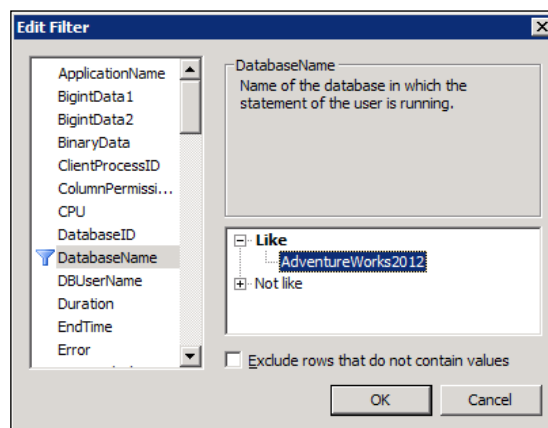
1. Start SQL Server Profiler.
2. Select **New Trace...** from the **File** menu. In the **Connect to Server** dialog box, provide connection details of SQL Server hosting the AdventureWorks2012 database and click on **Connect**.
3. In the **General** tab of **Trace Properties**, enter `FilteringEvents` as the **Trace name** and select **Blank** template for the **Use the template:** drop-down menu as shown in following:



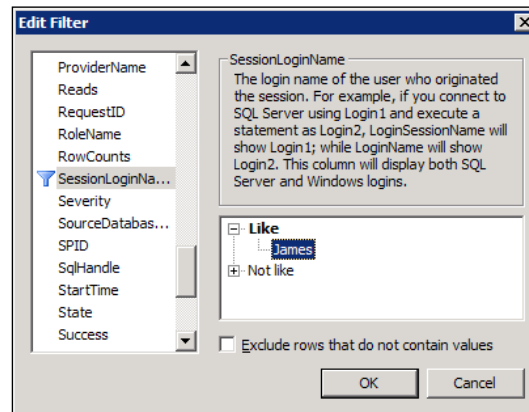
4. In **Events Selection** tab, check the checkbox for event class **SQL:BatchCompleted** under the **TSQL** event category as shown in following screenshot:



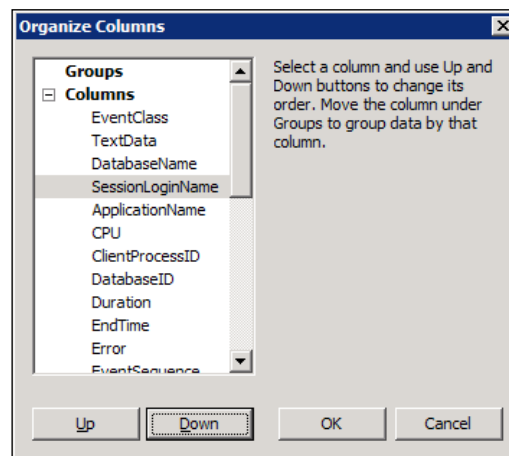
5. Click on **Column Filters...** button.
6. In the **Edit Filter** dialog box, select **DatabaseName** from the list of available data columns on the left. Expand the **Like** option and enter string value **AdventureWorks2012**; then press the **OK** button as shown in the following screenshot:



- In the **Edit Filter** dialog box, select **SessionLoginName** from the list of available data columns on the left. Expand the **Like** option and enter string value `James`; then press the **OK** button as shown in following screenshot:



- Click on the **Organize Columns...** button in **Events Selection** tab of **Trace Properties** dialog box. Select **TextData** data column and then keep clicking on **Up** button repeatedly to move the column up the order in the list, until the column appears as the second item, at the top of the list underneath **EventClass** data column. Do this same exercise also for the data columns **DatabaseName** and **SessionLoginName** so that the final order of the data columns should look like as shown in following screenshot. Press **OK** in the **Organize Columns** dialog box:



- Click on the **Run** button to run the trace in the **Trace Properties** dialog box.

Now, we will open two instances of SQL Server Management Studio one by one that connect to SQL Server with the logins *James* and *Peter* respectively and run a few queries.

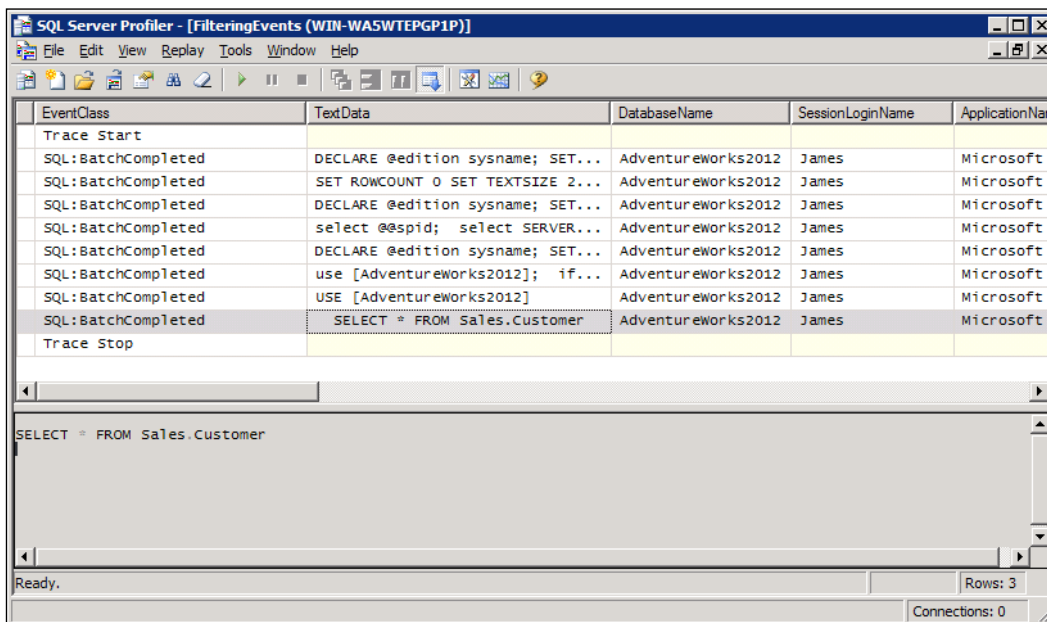
1. Open the first instance of SSMS and connect to SQL Server with the login credentials of *James*. In the query window, type and execute the T-SQL statements as shown in following script:
 

```
USE [AdventureWorks2012]
GO

SELECT * FROM [Sales].[Customer]
GO

USE [master]
GO

SELECT * FROM sys.databases
GO
```
2. Open a second instance of SSMS and connect to SQL Server with the login credentials of *Peter*. In the query window, type and execute the same T-SQL queries as shown in previous step.
3. Switch to SQL Server Profiler window that is running the trace. Examine the trace data as shown in following screenshot:





## How it works...

In this recipe, we first created two SQL Server logins and their corresponding users in AdventureWorks2012 database to demonstrate how to apply a trace filter based on a specific SQL Server login, so that the events belonging to SQL Server logins other than the one for which the filter condition on **SessionLoginName** is satisfied are not captured. We executed a T-SQL script to create logins and users for *James* and *Peter*. For a login/user, the script first creates an SQL Server login account by the executing T-SQL statement—**CREATE LOGIN**. It then creates a user in the AdventureWorks2012 database for that login and adds the user to the **db\_owner** database role by executing the T-SQL commands **CREATE USER** and **ALTER ROLE** respectively.

After creating logins and users, we started a new trace in SQL Server Profiler. We selected a **Blank** trace template and chose **SQL:BatchCompleted** event class as the only event that will be captured. Then we specified filters on **DatabaseName** and **SessionLoginName** data columns so that only the events which are occurred against AdventureWorks2012 database by user *James* are captured. We also organized the data columns in the **Organize Columns** dialog box, so that we can have better view of data columns we are interested in when trace data is displayed in SQL Server Profiler; we do not have to scroll much across the right side to see the values of **TextData**, **DatabaseName**, and **SessionLoginName**.



### Use of DatabaseID

We can alternatively use **DatabaseID** data column instead of **DatabaseName** to specify a filter on a particular database. For this, we must know system assigned ID value for a specific database. This value can be retrieved by either calling `DB_ID('AdventureWorks2012')` metadata function or querying `sys.databases` catalog view.

After starting the trace, we opened two instances of SSMS out of which one instance connects with the login *James* and another one connects with the login *Peter*. In both the instances of SSMS, we run a few sample queries against the AdventureWorks2012 and master database.

We can see the resulting trace data as shown in final screenshot. Notice that events belonging to login *Peter* and the events occurred for master database were not captured.

## There's more...

In a real world scenario, you may need to put filters on columns that are frequently used in trace filters to narrow down the data that you have to look at for troubleshooting. The following section lists some of data columns that are commonly used in trace filters:

- ▶ **ApplicationName:** A filter can be specified on this data column so that only trace events raised by a particular client application are captured

- ▶ **DatabaseID:** A filter can be specified on this data column so that only trace events raised for a specific database are captured
- ▶ **DatabaseName:** A filter can be specified on this data column so that only trace events raised for a specific database are captured
- ▶ **HostName:** A filter can be specified on this data column so that only trace events raised from a specific host or client machine are captured
- ▶ **LoginName:** A filter can be specified on this data column so that only trace events raised by a specific login are captured
- ▶ **ObjectID:** A filter can be specified on this data column so that only trace events raised for a specific object are captured
- ▶ **ObjectName:** A filter can be specified on this data column so that only trace events raised for a specific object are captured
- ▶ **SessionLoginName:** A filter can be specified on this data column so that only trace events raised by a specific login are captured
- ▶ **SPID:** A filter can be specified on this data column so that only trace events raised from a specific session connection are captured



**LoginName** and **SessionLoginName** may look identical at first. However, there is a small difference between them.

By using **EXECUTE AS** syntax in SQL Server, we can execute T-SQL statements in the same session under different security context other than the security context of the login who actually initiates the session/connection. For example, *James* can login to SQL Server and run a query under security context of *Peter* by using **EXECUTE AS** command. In this case, data column **SessionLoginName** returns **James**, while **LoginName** data column returns **Peter**. In other cases, where SQL Statements are not executed under different security context, data columns **SessionLoginName** and **LoginName** return the same value.

## Detecting slow running and expensive queries

Quite a few times, you may come across database related performance issues that are caused by slow running and expensive queries. Slow running queries or expensive queries are queries that have longer execution time and consume more hardware resources, such as CPU, memory, and disk I/O. For instance, suppose that you are working for an organization having an enterprise application environment with high degree of database transaction activity against single production database that is used to support many applications, it is usual to face database performance issues due to a poorly designed application or poorly written queries.

For example, an application that processes one record at a time and makes a round trip to SQL server for each record is an example of poorly designed application when it is possible to process multiple records in batch and send them to database server in one go. Similarly, a query can be considered to be poorly written if is not optimized for efficient read/write operations, generates sub-optimum execution plan, and takes longer to execute. One common example of a poorly written query is the one which processes records row-by-row, using cursor to perform a task that can be accomplished by a set-based query.

When there are a few hundreds of query requests per second coming from different applications hitting the same database continuously, how would you identify those slow running and expensive queries?

Of course, you can use Dynamic Management Views or Activity Monitor to perform such an investigation. However, SQL Profiler will give you more insight into the execution flow of different applications because you can see the actual order and sequence of incoming query requests in real-time along with their execution statistics that can help you in identifying the performance related issues caused by any possible loopholes in application logic.

## Getting ready

Remember that the objective of this recipe is not to teach you how to write efficient queries but instead how to identify expensive queries. Thus, for demonstration purposes, we ourselves will write a few expensive queries that take longer to execute in this example.

But before you can identify these slow running queries, you need to know what to look in SQL Server Profiler to identify those queries.

Whenever there is problem with the logic of the query, there is a possibility that the queries may start to take longer to execute as the database starts to grow. This results in holding locks on resources for a longer time, which can lead blockage to other queries. Poorly written queries also produce bad execution plans and can cause a high number of read/write operations that are expensive and take longer to execute.

So, when you are identifying long running queries, mostly you will be looking at time duration and CPU time that a query takes and the number of read/write operations that a query causes.

Therefore, in this recipe we will look at the following data columns:

- ▶ **CPU:** Amount of CPU processing time in milliseconds taken by an event
- ▶ **Duration:** Total amount of time in microseconds taken by an event
- ▶ **StartTime:** Time when an event starts
- ▶ **EndTime:** Time when an event ends
- ▶ **Reads:** Number of data pages that SQL Server has to read for an event
- ▶ **Writes:** Number of data pages that SQL Server has to write on disk for an event

The following are the prerequisites to do this recipe:

- ▶ An instance of SQL Server 2012 Developer or Enterprise Evaluation edition
- ▶ An SQL Server Login account with administrative rights
- ▶ Sample AdventureWorks2012 database on the instance of SQL Server

## How to do it...

Follow the steps provided here for this recipe:

1. Start SQL Server Profiler. To start SQL Server Profiler, navigate through **Start | All Programs | Microsoft SQL Server 2012 Program Group | Performance Tools | SQL Server Profiler**.
2. Select **New Trace...** from the **File** menu. In the **Connect to Server** dialog box, provide connection details of SQL Server hosting the AdventureWorks2012 database and click on **Connect**.
3. In the **General** tab of **Trace Properties**, specify `IdentifyingExpensiveQueries` as trace name and select **Blank** template for the **Use the template:** drop-down menu.
4. Check the checkbox **Save to file:** and specify a trace file name and location in the **Save As** dialog box.
5. In the **Events Selection** tab, check the checkbox for event class **SQL:BatchCompleted** under TSQL event category.
6. Click on the **Column Filters...** button.
7. In the **Edit Filter** dialog box, select **DatabaseName** from the list of available data columns on the left. Expand the **Like** option and enter string value `AdventureWorks2012`; then click on the **OK** button.
8. Click on **Organize Columns...** button in **Events Selection** tab of **Trace Properties** dialog box. Select **TextData** data column and then keep clicking the **Up** button repeatedly to move the column up the order in the list until the column appears as the second item at the top of the list underneath **EventClass** data column. Do this same exercise also for data columns, such as **CPU**, **Duration**, **StartTime**, **Endtime**, **Reads**, and **Writes** so that they appear underneath the **TextData** column. Press **OK** in the **Organize Columns** dialog box.
9. Open SQL Server Management Studio and connect to SQL Server.
10. Click on the **Run** button to run the trace in **Trace Properties** dialog box.
11. Type and execute the following T-SQL script. The script creates a stored procedure `usp_calculateOrderTotals` in AdventureWorks2012 database and a table `tbl_SampleData` by generating and inserting five million sample records:
 

```
USE [AdventureWorks2012]
GO
```

```
--Drop the stored procedure if it exists.
IF OBJECT_ID('[dbo].[usp_CalculateOrderTotals]') IS NOT NULL
    DROP PROCEDURE [dbo].[usp_CalculateOrderTotals]
GO
--Creates the stored procedure.
CREATE PROCEDURE [dbo].[usp_CalculateOrderTotals] AS
BEGIN
    CREATE TABLE [tempdb].[dbo].[#tbl_OrderTotals]
    (
        SRNo INT IDENTITY(1,1) PRIMARY KEY CLUSTERED
        ,OrderID INT
        ,OrderDate DATETIME
        ,CustomerName NVARCHAR(200)
        ,SalesPersonName NVARCHAR(200)
        ,OrderTotal NUMERIC(38,6)
    )

    DECLARE @SalesOrderID INT
    DECLARE @OrderDate DATETIME
    DECLARE @CustomerName NVARCHAR(200)
    DECLARE @SalesPersonName NVARCHAR(200)
    DECLARE @OrderTotal NUMERIC(38,6)

    DECLARE curSalesOrders CURSOR FAST_FORWARD FOR
    SELECT
        SOH.SalesOrderID
        ,SOH.OrderDate
        ,UPPER(P2.FirstName + ' ' + P2.LastName) AS CustomerName
        ,UPPER(P1.FirstName + ' ' + P1.LastName) AS SalesPersonName
    FROM [Sales].[SalesOrderHeader] AS SOH
    LEFT OUTER JOIN [Sales].[SalesPerson] AS SP
    ON SOH.SalesPersonID = SP.BusinessEntityID
    LEFT OUTER JOIN [Sales].[Customer] AS C
    ON SOH.CustomerID = C.CustomerID
    LEFT OUTER JOIN [Person].[Person] AS P1
    ON SP.BusinessEntityID = P1.BusinessEntityID
    LEFT OUTER JOIN [Person].[Person] AS P2
    ON C.PersonID = P2.BusinessEntityID

    OPEN curSalesOrders

    FETCH NEXT FROM curSalesOrders INTO
        @SalesOrderID
        ,@OrderDate
```

```

        ,@CustomerName
        ,@SalesPersonName

    WHILE @@FETCH_STATUS=0
    BEGIN

        SELECT @OrderTotal=SUM(LineTotal) FROM [Sales].[SalesOrderDetail]
        WHERE SalesOrderID = @SalesOrderID

        INSERT INTO [tempdb].[dbo].[#tbl_OrderTotals]
        VALUES
        (
            @SalesOrderID
            ,@OrderDate
            ,@CustomerName
            ,@SalesPersonName
            ,@OrderTotal
        )

        FETCH NEXT FROM curSalesOrders INTO
            @SalesOrderID
            ,@OrderDate
            ,@CustomerName
            ,@SalesPersonName

    END
    CLOSE curSalesOrders
    DEALLOCATE curSalesOrders

    SELECT * FROM [tempdb].[dbo].[#tbl_OrderTotals]
    ORDER BY OrderID DESC
END

GO
--Excutes stored procedure.
EXECUTE [dbo].[usp_CalculateOrderTotals]
GO
--Drop the table if it exists
IF OBJECT_ID('[dbo].[tblSampleData]') IS NOT NULL
    DROP TABLE [dbo].[tblSampleData]
GO
--Generate 5 million records and insert them into a table.
SELECT TOP 5000000 C1.*
INTO [dbo].[tblSampleData]
FROM sys.columns AS C1

```

```
CROSS JOIN sys.columns AS C2
CROSS JOIN sys.columns AS C3

GO
```

- After executing the previous script, switch to SQL Server Profiler and stop the trace. Notice the **CPU**, **Duration**, **StartTime**, **EndTime**, **Reads**, and **Write** columns. The following screenshot shows the trace after execution of the script:

EventClass	TextData	CPU	Duration	Reads	Writes	StartTime
Trace Start						2011-01-01 00:00:00
SQL:BatchCompleted	USE [Adventureworks2012]	0	0	0	0	2011-01-01 00:00:00
SQL:BatchCompleted	IF OBJECT_ID('[dbo].[usp_Calculat...]	0	8	115	8	2011-01-01 00:00:00
SQL:BatchCompleted	CREATE PROCEDURE [dbo].[usp_Calcu...	10	9	119	0	2011-01-01 00:00:00
SQL:BatchCompleted	EXECUTE [dbo].[usp_CalculateOrder...	3425	5891	296166	217	2011-01-01 00:00:00
SQL:BatchCompleted	IF OBJECT_ID('[dbo].[tblSampleData']...	10	7	219	2	2011-01-01 00:00:00
SQL:BatchCompleted	SELECT TOP 5000000 C1.* INTO db...	23103	25696	55369	35862	2011-01-01 00:00:00
Trace Stop						2011-01-01 00:00:00

SELECT TOP 5000000 C1.\*  
INTO dbo.tblSampleData  
FROM sys.columns AS C1  
CROSS JOIN sys.columns AS C2  
CROSS JOIN sys.columns AS C3

Trace is stopped. Ln 7, Col 2 Rows: 8 Connections: 0

Notice in the figure, how some of the **SQL:BatchCompleted** events caused high number of CPU usage counts, duration counts, and reads/writes counts. These queries are resource consuming and thus expensive queries.

## How it works...

We started a new trace in SQL Server Profiler. We selected **Blank** trace template and **SQL:BatchCompleted** event class that is the only event we wanted to capture. We then specified a trace filter on **DatabaseName** data column so that only the events which are occurred against AdventureWorks2012 database are captured.

We organized data columns in the **Organize Columns** dialog box so we can have a better view of data columns that we are interested in when trace data is displayed in SQL Server Profiler; we do not have to scroll much across the right side to see the values of **TextData**, **CPU**, **Duration**, **StartTime**, **EndTime**, **Reads**, and **Writes** data columns.



### Trace Filter on CPU or Duration

We could also have put a trace filter on **CPU** or **Duration** data column with > (greater than) operator in order to capture only those events whose CPU or duration count is higher than the value specified in trace filter. With this, let's say for example, if you want to find out the queries that are taking total execution time of 10 seconds or more, then you can define a filter on **Duration** column and only those queries running for 10 seconds or more will be captured.

After starting trace, we opened SSMS and connected to SQL Server. We then run sample script against AdventureWorks2012 database. The script creates and executes a sample stored procedure named [AdventureWorks2012].[dbo].[usp\_CalculateOrderTotals] that loops through a cursor to calculate the total for an order and inserts it in a temporary table. Looking at **CPU** and **Duration** data columns, it can be noticed that stored procedure took almost around six seconds to execute. Also, the **Reads** data column has high value and suggests that SQL Server had to read **296166** data pages to run this stored procedure. Higher the reads and writes counts are, slower the query will be. When the stored procedure [AdventureWorks2012].[dbo].[usp\_CalculateOrderTotals] is executed to retrieve the requested data with required columns along with the required calculation, it performed a read operation on the following tables:

- ▶ Sales.SalesOrderHeader
- ▶ Sales.SalesPerson
- ▶ Sales.Customer
- ▶ Person.Person
- ▶ #tbl\_OrderTotals

The script also generates five million sample records by cross joining sys.columns catalog view with itself multiple times and inserting the resulting data in tblSampleData table by SELECT...INTO command. This demonstrates how the writes count gets high when large amount of data is inserted. You can see that it caused **55369** reads and **35862** writes counts.

Remember that value in **CPU** data column is reported in milliseconds and the value in **Duration** data column is reported in microseconds. However, when SQL Server Profiler shows the value of **Duration** on its GUI, it shows the value in milliseconds by default. But when you save the trace in a trace file or trace table the value is stored in microseconds and not in milliseconds. Thus, for the **Duration** data column SQL Server behaves differently when it displays and stores the value.



You can change the way SQL Server displays the value of Duration so that it is reported in microsecond instead of millisecond on GUI if you wish so. You can change this setting from **Tools | Options....**