

Custom excerpt

Database security, performance, and more, plus
an exclusive offer for **50% off the full edition**



Inside **OUT**

The ultimate, in-depth reference
Hundreds of timesaving solutions
Supremely organized, packed
with expert advice

SQL Server 2017 Administration

William Assaf • Randolph West • Sven Aelterman • Mindy Curnutt

Foreword by Patrick LeBlanc, Microsoft

SQL Server 2008 and 2008 R2 end of support is coming

SQL Server 2008 and SQL Server 2008 R2 will no longer be supported by Microsoft starting in July 2019. Avoid challenges and vulnerabilities caused by end of support.



Why should you upgrade?



Mitigate risks with platform security and compliance

There will be no access to critical security updates, opening the potential for business interruptions and loss of data.



Upgrade to better cost efficiency

Maintaining legacy servers, firewalls, intrusion systems, and other tools can get expensive quickly.



Modernize to innovate

Grow your environments with data, analytics and the cloud.

More than an upgrade

With **SQL Server 2017** you don't just get an update—you get in-memory performance across workloads, mission-critical high availability, and built-in security features to help protect your data at rest and in motion.



#1 OLTP performance¹

#1 DW performance on 1TB², 10TB³, and 30TB⁴

#1 OLTP price/performance⁵

#1 DW price/performance on 1TB², 10TB³, and 30TB⁴

[→ Learn more about SQL Server 2017](#)

End of support options for SQL Server 2008 and 2008 R2

Take advantage of the Azure Hybrid Benefit

Save when you migrate your SQL Server 2008 or 2008 R2 workloads to Azure SQL Database with the Azure Hybrid Benefit for SQL Server. [Learn more.](#)

Extended Security Updates for on-premises environments

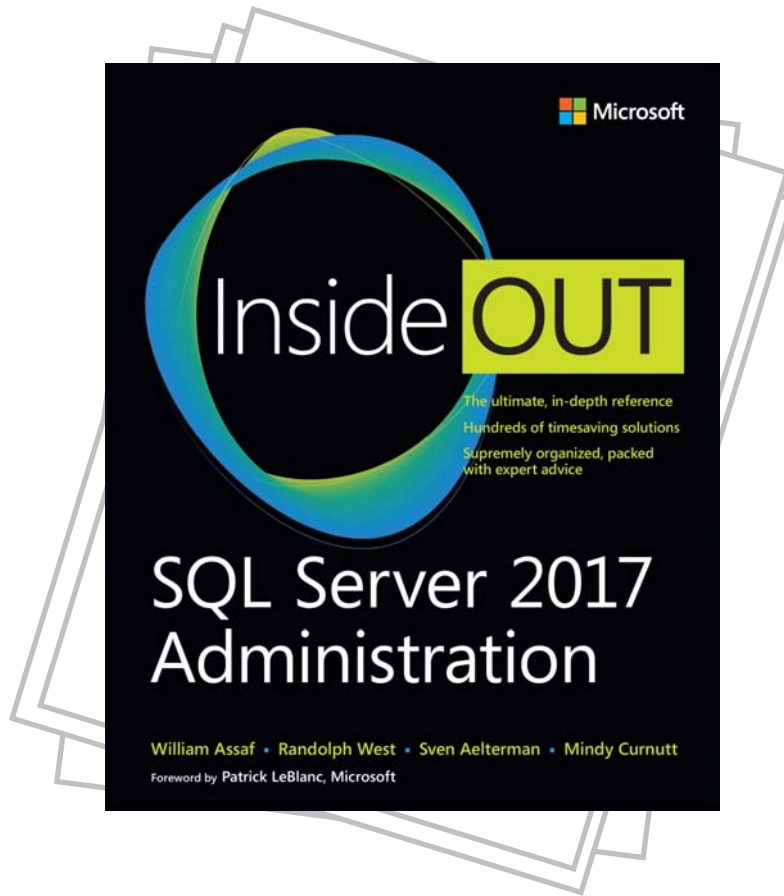
Customers with Software Assurance or subscription licenses may purchase Extended Security Updates for three years of security updates for SQL Server 2008 and 2008 R2. [Learn more.](#)

Free Extended Security Updates in Azure

Lift and shift your SQL Server 2008 workloads to Azure with no application code changes. This gives you more time to plan for end of support. [Learn more.](#)



[Learn more about SQL Server 2008 and 2008 R2 end of support](#)



Keep learning from the inside out—and save!

Save 50% when you purchase the complete eBook edition of *SQL Server 2017 Administration Inside Out* by William Assaf, Randolph West, Sven Aelterman, and Mindy Curnutt.

Visit microsoftpressstore.com/SQLAdmin to select title and use code **SQLADMIN** during checkout to apply discount. The eBook is delivered in EPUB, PDF, and MOBI to read on your preferred device.

Sign up to receive more special offers from Microsoft Press at microsoftpressstore.com/newsletters

Discount code valid on eBook purchase from microsoftpressstore.com and cannot be combined with another offer. Microsoft Press products are published, marketed, and distributed by Pearson.

SQL Server 2017 Administration Inside Out

William Assaf
Randolph West
Sven Aelterman
Mindy Curnutt

Published with the authorization of Microsoft Corporation by:
Pearson Education, Inc.

Copyright © 2018 by Pearson Education Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-1-5093-0521-6

ISBN-10: 1-5093-0521-1

Library of Congress Control Number: 2017961300

Printed and bound in the United States of America.

1 18

Trademarks

Microsoft and the trademarks listed at <https://www.microsoft.com> on the “Trademarks” webpage are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The authors, the publisher, and Microsoft Corporation shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or programs accompanying it.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Editor-in-Chief: Greg Wiegand

Acquisitions Editor: Trina MacDonald

Development Editor: Mark Renfrow

Technical Editor: Louis Davidson

Managing Editor: Sandra Schroeder

Senior Project Editor: Tracey Croom

Editorial Production: Octal Publishing, Inc.

Copy Editor: Octal Publishing, Inc.

Indexer: Octal Publishing, Inc.

Proofreader: Octal Publishing, Inc.

Cover Designer: Twist Creative, Seattle

*To David III
for inspiring and enabling STEM careers for many, including my own.*

—William Assaf

*To Marinus and Trixie
for putting up with the lies of “almost done,” and sharing my lap with a MacBook.*

—Randolph West

*To Ebony, Edward, and Sofia
in recognition of their sacrifices, support, and endless love.*

—Sven Aelterman

*To Chris
For believing in me more than I even believed in myself.*

—Mindy Curnutt

This page intentionally left blank



Contents at a glance

Chapter 1	Getting started with SQL Server tools.	1	Chapter 8	Understanding and designing tables	333
Chapter 2	Introducing database server components	45	Chapter 9	Performance tuning SQL Server	383
Chapter 3	Designing and implementing a database infrastructure	79	Chapter 10	Understanding and designing indexes.	429
	infrastructure		Chapter 11	Developing, deploying, and managing data recovery	459
Chapter 4	Provisioning databases	127	Chapter 12	Implementing high availability and disaster recovery	493
Chapter 5	Provisioning Azure SQL Database	197	Chapter 13	Managing and monitoring SQL Server.	557
Chapter 6	Administering security and permissions	241	Chapter 14	Automating SQL Server administration	607
Chapter 7	Securing the server and its data	291			

This page intentionally left blank



Table of contents

	Foreword	xvii
	Introduction	xix
	Who this book is forxix
	Assumptions about youxix
	How this book is organizedxx
	About the companion contentxxii
	Acknowledgmentsxxii
	Support and feedbackxxiv
	Errata & supportxxiv
	Stay in touchxxiv
Chapter 1	Getting started with SQL Server tools	1
	SQL Server setup	1
	Installing SQL Server by using the Installation Center	2
	Planning before an upgrade or installation	3
	Installing or upgrading SQL Server	6
	Tools and services installed with the SQL Server Database Engine	7
	Machine Learning Services	7
	Data Quality Services	7
	Command-line interface	9
	SQL Server Configuration Manager	11
	Performance and reliability monitoring tools	12
	Database Engine Tuning Advisor	12
	Extended events	13
	Management data warehouse	15
	SQL Server Reporting Services	18
	Installation	19
	Report Services Configuration Manager	20

SQL Server Management Studio	21
Releases and versions	21
Installing SQL Server Management Studio	22
Upgrading SQL Server Management Studio	22
Features of SQL Server Management Studio	23
Additional tools in SQL Server Management Studio	29
Error logs	32
Activity Monitor	33
SQL Server Agent	37
SQL Server Data Tools	41
SQL Server Integration Services	41
A note on deprecation	44

Chapter 2 Introducing database server components 45

Memory	45
Understanding the working set	46
Caching data in the buffer pool	46
Caching plans in the procedure cache	47
Lock pages in memory	47
Editions and memory limits	48
Central Processing Unit	49
Simultaneous multithreading	49
Non-Uniform Memory Access	50
Disable power saving everywhere	51
Storing your data	51
Types of storage	52
Configuring the storage layer	53
Connecting to SQL Server over the network	57
Protocols and ports	58
Added complexity with Virtual Local-Area Networks	58
High availability concepts	59
Why redundancy matters	60
Disaster recovery	60
Clustering	61
The versatility of Log Shipping	63
Always On availability groups	64
Read-scale availability groups	66
Distributed availability groups	67
Basic availability groups	67
Improve redundancy and performance with NIC teaming	67
Securing SQL Server	68
Integrated authentication and Active Directory	68
Azure Active Directory	71
Abstracting hardware with virtualization	73
Resource provisioning for VMs	74
When processors are no longer processors	75
The network is virtual, too	77
Summary	77

Chapter 3	Designing and implementing a database infrastructure	79
	Physical database architecture	79
	Data files and filegroups	80
	Recording changes in the transaction log	85
	Table partitioning	92
	Data compression	93
	Managing the temporary database	96
	Configuration settings	98
	Managing system usage by using Resource Governor	98
	Configuring the page file (Windows)	99
	Taking advantage of logical processors by using parallelism	100
	SQL Server memory settings	102
	Carving up CPU cores using an affinity mask	105
	File system configuration	107
	Azure and the Data Platform	110
	Infrastructure as a service	110
	Platform as a service	116
	Hybrid cloud with Azure	121
Chapter 4	Provisioning databases	127
	What to do before installing SQL Server	127
	Deciding on volume usage	127
	Important SQL Server volume settings	130
	SQL Server editions	131
	Installing a new instance	134
	Planning for multiple SQL Server instances	134
	Installing a SQL Server instance	134
	Installing options and features	137
	Installing other core features	142
	“Smart Setup”	146
	Setting up logging	147
	Automating SQL Server Setup by using configuration files	147
	Post-installation server configuration	151
	Post-installation checklist	151
	Installing and configuring features	164
	SSISDB initial configuration and setup	164
	SQL Server Reporting Services initial configuration and setup	165
	SQL Server Analysis Services initial configuration and setup	168
	Adding databases to a SQL Server instance	169
	Considerations for migrating existing databases	169
	Moving existing databases	175
	Creating a database	177
	Database properties and options	181
	Moving and removing databases	189
	Moving user and system databases	189
	Database actions: offline versus detach versus drop	191
	Single-user mode	195

Chapter 5	Provisioning Azure SQL Database	197
	Azure and database-as-a-service concepts	198
	Database-as-a-service	198
	Managing Azure: The Azure portal and PowerShell	199
	Azure governance	200
	Logical SQL Servers	201
	Cloud-first	202
	Database Transaction Unit	202
	Resource scalability	203
	Provisioning a logical SQL server	204
	Creating a server using the Azure portal	205
	Creating a server by using PowerShell	206
	Establishing a connection to your server	207
	Deleting a server	209
	Provisioning a database in Azure SQL Database	209
	Creating a database using the Azure portal	210
	Creating a database by using PowerShell	211
	Creating a database by using Azure CLI	212
	Creating a database by using T-SQL	213
	Selecting a pricing tier and service objective	213
	Scaling up or down	214
	Provisioning an elastic pool	214
	Limitations of Azure SQL Database	215
	Database limitations	215
	Other SQL Server services	216
	Overcoming limitations with managed instances	218
	Security in Azure SQL Database	218
	Security features shared with SQL Server 2017	219
	Server and database-level firewall	219
	Access control using Azure AD	222
	Role-Based Access Control	223
	Auditing and threat detection	224
	Preparing Azure SQL Database for disaster recovery	229
	Understanding default disaster recovery features	229
	Manually backing up a database	230
	Configuring geo-replication	232
	Setting up failover groups	235
	Using Azure Backup for long-term backup retention	237
	Moving to Azure SQL Database	239
Chapter 6	Administering security and permissions	241
	Logins and users	241
	Different types of authentication	242
	Solving orphaned SIDs	246
	Preventing orphaned SIDs	249
	Factors in securing logins	249
	Login security	254
	Contained databases	256

Permissions in SQL Server	257
Understanding Permissions for Data Definition Language and Data Manipulation Language	257
Modifying permissions	259
Granting commonly needed permissions	261
Ownership versus authorization	265
Understanding views, stored procedures, and function permissions	267
Understanding server roles	273
Understanding database roles	278
Using the Dedicated Administrator Connection	283
Moving SQL Server logins and permissions	285
Moving logins by using SQL Server Integration Services (SQL Server only)	286
Moving Windows-authenticated logins by using T-SQL (SQL Server only)	287
Moving SQL Server–authenticated logins by using T-SQL (SQL Server only)	287
Moving server roles by using T-SQL (SQL Server only)	288
Moving server permissions by using T-SQL (SQL Server only)	288
Moving Azure SQL Database logins	289
Other security objects to move	289
Alternative migration approaches	290
Chapter 7 Securing the server and its data	291
Introducing security principles and protocols	292
Securing your environment with defense in depth	292
The difference between hashing and encryption	294
A primer on protocols and transmitting data	296
Symmetric and asymmetric encryption	300
Digital certificates	301
Encryption in SQL Server	302
Data protection from the OS	303
The encryption hierarchy in detail	303
Using EKM modules with SQL Server	304
Master keys in the encryption hierarchy	306
Encrypting data by using TDE	308
Protecting sensitive columns with Always Encrypted	310
Securing data in motion	314
Securing network traffic with TLS	314
Row-level security	315
Dynamic data masking	317
Azure SQL Database	318
Auditing with SQL Server and Azure SQL Database	319
SQL Server Audit	319
Auditing with Azure SQL Database	326
Securing Azure infrastructure as a service	326
Network Security Group	327
User-defined routes and IP forwarding	328
Additional security features in Azure networking	330

Chapter 8	Understanding and designing tables	333
	Reviewing table design	333
	Generic data types	333
	Specialized data types	339
	Keys and relationships	345
	Constraints	346
	Sequences	347
	User-defined data types and user-defined types	350
	Sparse columns	352
	Computed columns	352
	Special table types	354
	System-versioned temporal tables	354
	Memory-optimized tables	357
	PolyBase external tables	361
	Graph tables	362
	Storing BLOBs	367
	Understanding FILESTREAM	368
	FileTable	369
	Table partitioning	370
	Horizontally partitioned tables and indexes	371
	Vertical partitioning	377
	Capturing modifications to data	377
	Using change tracking	378
	Using change data capture	380
	Comparing change tracking, change data capture, and temporal tables	381
Chapter 9	Performance tuning SQL Server	383
	Understanding isolation levels and concurrency	383
	Understanding how concurrent sessions become blocked	386
	Stating the case against READ UNCOMMITTED (NOLOCK)	390
	Changing the isolation level within transactions	391
	Understanding the enterprise solution to concurrency: SNAPSHOT	393
	Understanding on-disk versus memory-optimized concurrency	398
	Understanding delayed durability	400
	Delayed durability database options	401
	Delayed durability transactions	401
	Understanding execution plans	401
	Understanding parameterization and “parameter sniffing”	402
	Understanding the Procedure Cache	404
	Analyzing cached execution plans in aggregate	405
	Retrieving execution plans in SQL Server Management Studio	408
	Using the Query Store feature	413
	Initially configuring the query store	415
	Using query store data in your troubleshooting	416
	Understanding automatic plan correction	418

Understanding execution plan operators	419
Interpreting graphical execution plans	419
Forcing a parallel execution plan	425
Understanding parallelism	425
Chapter 10 Understanding and designing indexes	429
Designing clustered indexes	429
Choosing a proper clustered index key	429
The case against intentionally designing heaps	433
Designing nonclustered indexes	434
Understanding nonclustered index design	435
Creating “missing” nonclustered indexes	441
Understanding and proving index usage statistics	445
Designing Columnstore indexes	446
Demonstrating the power of Columnstore indexes	448
Using compression delay on Columnstore indexes	449
Understanding indexing in memory-optimized tables	449
Understanding hash indexes for memory-optimized tables	450
Understanding nonclustered indexes for memory-optimized tables	451
Moving to memory-optimized tables	451
Understanding other types of indexes	452
Understanding full-text indexes	452
Understanding spatial indexes	452
Understanding XML indexes	453
Understanding index statistics	453
Manually creating and updating statistics	454
Automatically creating and updating statistics	454
Important performance options for statistics	455
Understanding statistics on memory-optimized tables	456
Understanding statistics on external tables	457
Chapter 11 Developing, deploying, and managing data recovery	459
The fundamentals of data recovery	460
A typical disaster recovery scenario	460
Losing data with the RPO	462
Losing time with the RTO	463
Establishing and using a run book	463
An overview of recovery models	464
Understanding backup devices	470
Backup disk	470
Backup sets and media	470
Physical backup device	472
Understanding different types of backups	472
Full backups	473
Transaction log backups	474
Differential backups	475
File and filegroup backups	477
Additional backup options	477

Creating and verifying backups	478
Creating backups	479
Verifying backups	480
Restoring a database	482
Restoring a piecemeal database	486
Defining a recovery strategy	487
A sample recovery strategy for a DR scenario	488
Strategies for a cloud/hybrid environment	490
Chapter 12 Implementing high availability and disaster recovery	493
Overview of high availability and disaster recovery technologies in SQL Server	493
Understanding log shipping	494
Understanding types of replication	497
Understanding the capabilities of failover clustering	500
Understanding the capabilities of availability groups	503
Comparing HA and DR technologies	506
Configuring Failover Cluster Instances	507
Configuring a SQL Server FCI	510
Configuring availability groups	513
Comparing different cluster types and failover	514
Creating WSFC for use with availability groups	519
Understanding the database mirroring endpoint	520
Configuring the minimum synchronized required nodes	520
Choosing the correct secondary replica availability mode	521
Understanding the impact of secondary replicas on performance	522
Understanding failovers in availability groups	524
Seeding options when adding replicas	525
Additional actions after creating an availability group	529
Reading secondary database copies	531
Implementing a hybrid availability group topology	537
Configuring an availability group on Red Hat Linux	538
Installation requirements	538
Setting up an availability group	539
Setting up the cluster	545
Administering availability groups	548
Analyzing DMVs for availability groups	548
Analyzing wait types for availability groups	554
Analyzing extended events for availability groups	555
Alerting for availability groups	556
Chapter 13 Managing and monitoring SQL Server	557
Detecting database corruption	557
Setting the database's page verify option	557
Using DBCC CHECKDB	558
Repairing database data file corruption	560
Recovering the database transaction log file corruption	560
Database corruption in databases in Azure SQL Database	561

Maintaining indexes and statistics	561
Changing the Fill Factor property when beneficial	561
Monitoring index fragmentation	563
Rebuilding indexes	564
Reorganizing indexes	568
Updating index statistics	569
Reorganizing Columnstore indexes	571
Maintaining database file sizes	571
Understanding and finding autogrowth events	573
Shrinking database files	574
Monitoring databases by using DMVs	575
Sessions and requests	576
Understanding wait types and wait statistics	577
Reintroducing extended events	584
Viewing extended events data	586
Using extended events to detect deadlocks	589
Using extended events to detect autogrowth events	590
Securing extended events	591
Capturing Windows performance metrics with DMVs and data collectors	592
Querying performance metrics by using DMVs	592
Querying performance metrics by using Performance Monitor	595
Monitoring key performance metrics	596
Protecting important workloads using Resource Governor	600
Configuring the Resource Governor classifier function	601
Configuring Resource Governor pools and groups	602
Monitoring pools and groups	603
Understanding the new servicing model	604
Chapter 14 Automating SQL Server administration	607
Components of SQL Server automated administration	607
Database Mail	608
SQL Server Agent	612
Configuring SQL Server Agent jobs	612
Maintaining SQL Server	623
Basic “care and feeding” of SQL Server	623
Using SQL Server Maintenance Plans	625
Maintenance Plan report options	632
Covering databases with the Maintenance Plan	633
Building Maintenance Plans by using the design surface in SQL Server	634
Management Studio	
Backups on secondary replicas in availability groups	636
Strategies for administering multiple SQL Servers	638
Master and Target servers for SQL Agent jobs	638
SQL Server Agent event forwarding	642
Policy-Based Management	643
Evaluating policies and gathering compliance data	643
Using PowerShell to automate SQL Server administration	648

PowerShell basics.....	649
Installing the PowerShell SQLSERVER module.....	651
Using PowerShell with SQL Server.....	652
Using PowerShell with availability groups.....	656
Using PowerShell with Azure.....	660
Index.....	665
About the authors.....	679
About the Foreword author.....	680

Foreword

The world as we know it is being inundated with data. We live in a culture in which almost every individual has at least two devices, a smart phone, and a laptop or computer of some sort. Everything we do on these devices is constantly collecting, sharing, or producing data. This data is being used not only to help organizations make smarter decisions, but also to shape and transform how we as a society live, work, make decisions, and sometimes think.

This massive explosion can be attributed to the technological transformation that every business and nearly every industry is undergoing. Every click or purchase by an individual is now triggering some event that triggers another event that likely amounts to hundreds or possibly thousands of rows of data. Multiply this by every person in the world and now you have an unprecedented amount of stored data that no one could have ever imagined. Now, not only must organizations store this data, but also ensure that this data—this massive amount of data—is readily available for consumption at the click of a button or the swipe of a screen.

This is where the database comes into play. Databases are the backbone or back end to possibly every aspect of business today. Back when Ted Codd, the father of the relational database, came up with this seminal idea, he probably had no idea how widespread their use would be today. Initially, database usage was intended to store data and retrieve data. The primary purpose was to simply ensure the security, availability, and reliability of any information written by on-premises applications at varying scales.

Today, all of that has changed. Data must be available 24 hours per day, 7 days each week, primarily via the internet instead of just by way of on-premises applications. Microsoft SQL Server 2017 was designed with all of this in mind. It can support high-volume Online Transactional Processing (OLTP) databases and very large Online Analytical Processing (OLAP) systems out of the box. And, by taking advantage of Microsoft Azure, developers can grow and scale databases dynamically and transparently behind the scenes to accommodate planned and unplanned spikes in demand and resource utilization. In other words, the latest version of SQL Server was built to not only accommodate this new world of data, but to push the limits of what organizations are doing today and what they will be doing tomorrow and deeper into the future.

Close your eyes and imagine a world in which a DBA can configure a database system to automatically increase or decrease resource utilization based on end-user application usage. But that's not all. What if the relational database management system (RDBMS) could automatically tune performance based on usage patterns? All of this is now possible with SQL Server and Azure SQL Database. By using features such as the Query Store and Elastic Database Pools, DBAs can proactively design solutions that will scale and perform to meet any application Service-Level Agreement.

In addition to world-class performance, these databases also include security and high-availability features that are unparalleled to any other RDBMS. Organizations can build mission-critical secure applications by taking advantage of SQL Server out-of-the-box built-in features without purchasing additional software. These features are available both in the cloud and on-premises and can be managed using SQL Server Management Studio, SQL Server Data Tools, and SQL Operations Studio. All three tools are available to download for free, and will be familiar to DBAs and database developers.

Throughout this book, the authors highlight many of the capabilities that make it possible for organizations to successfully deploy and manage database solutions using a single platform. If you are a DBA or database developer looking to take advantage of the latest version of SQL Server, this book encompasses everything needed to understand how and when to take advantage of the robust set of features available within the product.

This book is based on the skills of a group of seasoned database professionals with several decades experience in designing, optimizing, and developing robust database solutions, all based on SQL Server technology. It is written for experienced DBAs and developers, aimed at teaching the advanced techniques of SQL Server.

SQL Server, Microsoft's core database platform, continues its maturity from supporting some of the smallest departmental tasks to supporting some of the largest RDBMS deployments in the world. Each release not only includes capabilities that enhance its predecessor, but also boasts features that rival and exceed those of many competitors.

This trend continues with SQL Server 2017. This release, just like all past releases, continues to add capabilities to an already sophisticated and reliable toolkit. Features include a secure, elastic, and scalable cloud system; advanced in-memory technologies; faster and consolidated management and development experiences; and continued growth and enhancements in the area of high availability and disaster recovery. In addition, concerted efforts have been focused on making the number one secure RDBMS in the world even more secure, by adding capabilities such as row-level security, Always Encrypted, and dynamic data masking. Finally, and as always, performance is at the center of this release. With enhancements to the Query Store, DBAs can take a more proactive approach to monitoring and tuning performance.

All in all, this book is sort of like an "Inside Out" look of each of the core components of SQL Server 2017, with a few excursions into the depths of some very specific topics. Each chapter first provides an overview of the topic and then delves deeper into that topic and any corresponding related topics. Although it's impossible to cover every detail of every Transact-SQL statement, command, feature or capability, this book provides you with a comprehensive look into SQL Server 2017. After reading each page of this book, you will be able to implement a cloud-based or on-premises scalable, performant, secure, and reliable database solution using SQL Server 2017.

Patrick LeBlanc, Microsoft

Introduction

The velocity of change for the Microsoft SQL Server DBA has increased this decade. The span between the releases of SQL Server 2016 and 2017 was only 16 months, the fastest new release ever. Gone are the days when DBAs had between three to five years to soak in and adjust to new features in the engine and surrounding technologies.

This book is written and edited by SQL Server experts with two goals in mind: to deliver a solid foundational skillset for all of the topics covered in SQL Server configuration and administration, and also to deliver awareness and functional, practical knowledge for the dramatic number of new features introduced in SQL Server 2016 and 2017. We haven't avoided new content—even content that stretched the boundaries of writing deadlines with late-breaking new releases. You will be presented with not only the “how” of new features, but also the “why” and the “when” for their use.

Who this book is for

SQL Server administration was never the narrow niche skillset that our employers might have suspected it was. Even now it continues to broaden, with new structures aside from the traditional rowstore, such as Columnstore and memory-optimized indexes, or new platforms such as Microsoft Azure SQL Database platform as a service (PaaS) and Azure infrastructure as a service (IaaS). This book is for the DBAs who are unafraid to add these new skillsets and features to their utility belt, and to give courage and confidence to those who are hesitant. SQL Server administrators should read this book to become more prepared and aware of features when talking to their colleagues in application development, business intelligence, and system administration.

Assumptions about you

We assume that you have some experience and basic vocabulary with administering a recent version of SQL Server. You might be curious, preparing, or accomplished with Microsoft Certifications for SQL Server. DBAs, architects, and developers can all benefit from the content provided in this book, especially those looking to take their databases to the cloud, to reach heights of performance, or to ensure the security of their data in an antagonistic, networked world.

This book mentions some of the advanced topics that you'll find covered in more detail elsewhere (such as custom development, business intelligence design, data integration, or data warehousing).

Book Features

These are the book's signature tips. In these tips, you'll get the straight scoop on what's going on with the software or service—inside information about why a feature works the way it does. You'll also find field-tested advice and guidance as well as details that give you the edge on deploying and managing like a pro.

How this book is organized

This book gives you a comprehensive look at the various features you will use. It is structured in a logical approach to all aspects of SQL Server 2017 Administration.

Chapter 1, "Getting started with SQL Server tools" gives you a tour of the tooling you need, from the installation media to the free downloads, not the least of which is the modern, rapidly evolving SQL Server Management Studio. We also cover SQL Server Data Tools, Configuration Manager, performance and reliability monitoring tools, provide an introduction to PowerShell, and more.

Chapter 2, "Introducing database server components," introduces the working vocabulary and concepts of database administration, starting with hardware-level topics such as memory, processors, storage, and networking. We then move into high availability basics (much more on those later), security, and hardware virtualization.

Chapter 3, "Designing and implementing a database infrastructure" introduces the architecture and configuration of SQL Server, including deep dives into transaction log virtual log files (VLFs), data files, in-memory Online Transaction Processing (OLTP), partitioning, and compression. We spend time with TempDB and its optimal configuration, and server-level configuration options. Here, we also cover running SQL Server in Azure virtual machines or Azure SQL databases as well as hybrid cloud architectures.

Chapter 4, "Provisioning databases" is a grand tour of SQL Server Setup, including all the included features and their initial installation and configuration. We review initial configurations, a post-installation checklist, and then the basics of creating SQL Server databases, including database-level configuration options for system and user databases.

Chapter 5, "Provisioning Azure SQL Database," introduces Microsoft's SQL Server database-as-a-service (DBaaS) offering. This Azure cloud service provides a database service with a very high degree of compatibility with SQL Server 2017. You will read about the concepts behind Azure SQL Database, learn how to create databases, and perform common management tasks for your databases.

Chapter 6, "Administering security and permissions" begins with the basics of authentication, the configuration, management, and troubleshooting of logins and users. Then, we dive into permissions, including how to grant and revoke server and database-level permissions and role membership, with a focus on moving security from server to server.

Chapter 7, “Securing the server and its data” takes the security responsibilities of the SQL Server DBA past the basics of authentication and permissions and discusses advanced topics including the various features and techniques for encryption, Always Encrypted, and row-level security. We discuss security measures to be taken for SQL Server instances and Azure SQL databases as well as the Enterprise-level SQL Server Audit feature.

Chapter 8, “Understanding and designing tables,” is all about creating SQL Server tables, the object that holds data. In addition to covering the basics of table design, we cover special table types and data types in-depth. In this chapter, we also demonstrate techniques for discovering and tracking changes to data.

Chapter 9, “Performance tuning SQL Server” dives deep into isolation and concurrency options, including READ COMMITTED SNAPSHOT ISOLATION (RCSI), and why your developers shouldn’t be using NOLOCK. We review execution plans, including what to look for, and the Query Store feature that was introduced in SQL Server 2016 and improved in SQL Server 2017.

Chapter 10, “Understanding and designing indexes” tackles performance from the angle of indexes, from their creation, monitoring, and tuning, and all the various forms of indexes at our disposal, past clustered and nonclustered indexes and into Columnstore, memory-optimized hash indexes, and more. We review indexes and index statistics in detail, though we cover their maintenance later on in Chapter 13.

Chapter 11, “Developing, deploying, and managing data recovery” covers the fundamentals of database backups in preparation for disaster recovery scenarios, including a backup and recovery strategy appropriate for your environment. Backups and restores in a hybrid environment, Azure SQL Database recovery, and geo-replication are important assets for the modern DBA, and we cover those, as well.

Chapter 12, “Implementing high availability and disaster recovery” goes beyond backups and into strategies for disaster recovery from the old (log shipping and replication) to the new (availability groups), including welcome new enhancements in SQL Server 2017 to support cross-platform and clusterless availability groups. We go deep into configuring clusters and availability groups on both Windows and Linux.

Chapter 13, “Managing and monitoring SQL Server” covers the care and feeding of SQL Server instances, including monitoring for database corruption, monitoring database activity, and index fragmentation. We dive into extended events, the superior alternative to traces, and also cover Resource Governor, used for insulating your critical workloads.

Chapter 14, “Automating SQL Server administration” includes an introduction to PowerShell, including features available in PowerShell 5.0. We also review the tools and features needed to automate tasks to your SQL Server, including database mail, SQL Server Agent jobs, Master/Target Agent jobs, proxies, and alerts. Finally, we review the vastly improved Maintenance Plans feature, including what to schedule and how.

About the companion content

We have included this companion content to enrich your learning experience. You can download this book's companion content from the following page:

<https://aka.ms/SQLServ2017Admin/downloads>

The companion content includes helpful Transact-SQL and PowerShell scripting, as mentioned in the book, for easy reference and adoption into your own toolbox of scripts.

Acknowledgments

From William Assaf:

I'd like to thank the influencers and mentors in my professional career who affected my trajectory, and to whom I remain grateful for technical and nontechnical lessons learned. In no particular order, I'd like to thank Connie Murla, David Alexander, Darren Schumaker, Ashagre Bishaw, Charles Sanders, Todd Howard, Chris Kimmel, Richard Caronna, and Mike Huguet. There's definitely a special love/hate relationship developed between an author and a tech editor, but I couldn't have asked for a better one than Louis Davidson. Finally, from user groups to SQLSaturdays to roadshow presentations to books, I am indebted to my friend Patrick Leblanc, who climbed the ladder and unflinchingly turned to offer a hand and a hug.

From Randolph West:

In June 2017, I told my good friend Melody Zacharias that I'd like to finish at least one of the many books I've started before I die. She suggested that I might be interested in contributing to this one. Piece of cake, I thought.

I have seven more gray hairs now. Seven!

I would like to thank Melody for recommending me in her stead, my husband for giving me space at the kitchen counter to write, and my dog Trixie for much needed distraction.

Trina, William, Louis, Sven and Mindy have been a great support as well, especially during the Dark Times.

This book would not be possible without the contributions of everyone else behind the scenes, too. Writing a book of this magnitude is a huge endeavour. (So help me if "endeavour" is the one word I get to spell the Canadian way!)

From Sven Aelterman:

I met William Assaf several years ago when I spoke at the Baton Rouge SQLSaturday. I have been back to this event many times since then and enjoyed preceding the Troy University Trojans' victory over Louisiana State University. (This just added in case the actual college football game doesn't make it in the history books. At least it will be recorded here.)

I am grateful for William's invitation to contribute two chapters to this book. William made a valiant attempt to prepare me for the amount of work "just" two chapters would be. Yet, I underestimated the effort. If it weren't for his support and that of Randolph West, technical editor Louis Davidson, editor Trina Macdonald, and even more people behind the scenes, the space for this acknowledgment might have been saved. They were truly a great team and valued collaborators. Without hesitation, I would go on the journey of book writing again with each of them.

My children, Edward and Sofia, and my wife, Ebony, have experienced firsthand that SQL Server can slow down time. "About two months" must have felt to them like months with 60 days each. I thank them for their patience while they had to share me with Azure and various table types. I hope that maybe my children will be inspired one day to become authors in their career fields.

Finally, I'd like to thank my coworkers at Troy University for inspiring me to do my best work. Working in a public higher education institution has some challenges, but the environment is so conducive to intellectual growth that it makes up for each challenge and then some.

From Mindy Curnutt:

I would like to thank Patrick LeBlanc for inviting me to participate in the creation of this book. Thanks also to Tracy Boggiano, for an amazing amount of help pulling together much of the chapter about automating administration. She's an MVP in my eyes! To everyone in the 2016-2017 TMW DBA Services "Team Unicorn": Eric Blinn, Lisa Bohm, Dan Andrews, Vedran Ikonic, and Dan Clemens, thank you for your proof reading and feedback. Thanks to my mom Barbara Corry for always swooping in to help with just about anything I needed. Of course, I couldn't have done any of this without the support of my husband, Chris Curnutt. He is always supportive despite long work hours, phone conversations with strange acronyms, and travel, he's also the love of my life. Last but not least, thanks to our two children, Riley and Kimball, who have supported and encouraged me in more ways than I can count.

Support and feedback

The following sections provide information on errata, book support, feedback, and contact information.

Errata & support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

<https://aka.ms/SQLServ2017Admin/errata>

If you discover an error that is not already listed, please submit it to us at the same page. If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *<https://support.microsoft.com>*.

Stay in touch

Let's keep the conversation going! We're on Twitter at *<http://twitter.com/MicrosoftPress>*.



Introducing database server components

Memory	45	High availability concepts	59
Central Processing Unit	49	Securing SQL Server	68
Storing your data	51	Abstracting hardware away with virtualization	73
Connecting to SQL Server over the network	57		

In this chapter, we cover the components that make up a typical database infrastructure. This chapter is introductory; the chapters that follow provide more detail about designing, implementing, and provisioning databases.

Although Microsoft SQL Server is new to Linux, Microsoft has, as much as possible, crafted it to work the same way that it does on Windows. We highlight places where there are differences.

No matter which configurations you end up using, there are four basic parts to a database infrastructure:

- Memory
- Processor
- Permanent storage
- Network

We also touch on a couple of high availability offerings, including improvements to availability groups in SQL Server 2017. We then look at an introduction to security concepts, including ways to access instances of SQL Server on-premises with Windows and Linux, and Microsoft Azure SQL Database. Finally, we take a brief look at virtualization.

Memory

SQL Server is designed to use as much memory as it needs, and as much as you give it. By default, the upper limit of memory that SQL Server can access, is limited only by the physical Random Access Memory (RAM) available to the server, or the edition of SQL Server you're running, whichever is lower.

Understanding the working set

The physical memory made available to SQL Server by the operating system (OS), is called the *working set*. This working set is broken up into several sections by the SQL Server memory manager, the two largest and most important ones being the *buffer pool* and the *procedure cache* (also known as the *plan cache*).

In the strictest sense, “working set” applies only to physical memory. However, as we will see shortly, the buffer pool extension blurs the lines.

We look deeper into default memory settings in Chapter 3, in the section, “Configuration settings.”

Caching data in the buffer pool

For best performance, you cache data in memory because it’s much faster to access data directly from memory than storage.

The buffer pool is an in-memory cache of 8-KB data pages that are copies of pages in the database file. Initially the copy in the buffer pool is identical, but changes to data are applied to this buffer pool copy (and the transaction log) and then asynchronously applied to the data file.

When you run a query, the Database Engine requests the data page it needs from the Buffer Manager, as depicted in Figure 2-1. If the data is not already in the buffer pool, a page fault occurs (an OS feature that informs the application that the page isn’t in memory). The Buffer Manager fetches the data from the storage subsystem and writes it to the buffer pool. When the data is in the buffer pool, the query continues.

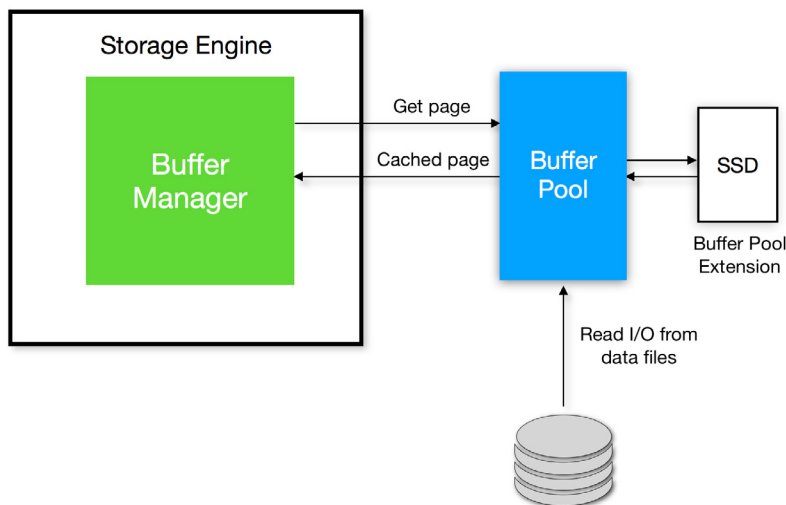


Figure 2-1 The buffer pool and the buffer pool extension.

The buffer pool is usually the largest consumer of the working set because that's where your data is. If the amount of data requested for a query exceeds the capacity of the buffer pool, the data pages will spill to a drive, either using the buffer pool extension or a portion of TempDB.

The buffer pool extension makes use of nonvolatile storage to extend the size of the buffer pool. It effectively increases the database working set, forming a bridge between the storage layer where the data files are located and the buffer pool in physical memory.

For performance reasons, this should be solid-state storage, directly attached to the server.

- To see how to turn on the buffer pool extension, read the section "Configuration settings" in Chapter 3. To learn more about TempDB, read the section "Physical database architecture," also in Chapter 3.

Caching plans in the procedure cache

Generally speaking, the procedure cache is smaller than the buffer pool. When you run a query, the Query Optimizer compiles a query plan to explain to the Database Engine exactly how to run the query. To save time, it keeps a copy of that query plan so that it doesn't need to compile the plan each time the query runs. It is not quite as simple as this, of course (plans can be removed, and trivial plans are not cached, for instance), but it's enough to give you a basic understanding.

The procedure cache is split into various cache stores by the memory manager, and it's also here where you can see if there are single-use query plans that are polluting memory.

- For more information about cached execution plans, read Chapter 9 or visit <https://blogs.msdn.microsoft.com/blogdoezequiel/2014/07/30/too-many-single-use-plans-now-what/>.

Lock pages in memory

Turning on the *Lock pages in memory* (LPIM) policy means that Windows will not be able to trim (reduce) SQL Server's working set.

Locking pages in memory ensures that Windows memory pressure cannot rob SQL Server of resources or shunt SQL Server memory into the Windows Server system page file, dramatically reducing performance. Windows doesn't "steal" memory from SQL Server flippantly; it is done in response to memory pressure on the Windows Server. Indeed, all applications can have their memory affected by pressure from Windows.

On the other hand, without the ability to relieve pressure from other applications' memory demands or a virtual host's memory demands, LPIM means that Windows cannot deploy

enough memory to remain stable. Because of this concern, LPIM cannot be the only method to use to protect SQL Server's memory allocation.

The controversy of the topic is stability versus performance, in which the latter was especially apparent on systems with limited memory resources and older operating systems. On larger servers with operating systems since Windows Server 2008, and especially virtualized systems, there is a smaller but nonzero need for this policy to insulate SQL Server from memory pressure.

The prevailing wisdom is that the LPIM policy should be turned on by default for SQL Server 2017, provided the following:

- The server is physical, not virtual. See the section "Sharing more memory than we have (overcommit)" later in this chapter.
- Physical RAM exceeds 16 GB (the OS needs a working set of its own).
- Max Server Memory has been set appropriately (SQL Server can't use everything it sees).
- The Memory\Available Mbytes performance counter is monitored regularly (to keep some memory free).

If you would like to read more, Jonathan Kehayias explains this thinking in a Simple Talk article (<https://www.simple-talk.com/sql/database-administration/great-sql-server-debates-lock-pages-in-memory/>).

Editions and memory limits

Since SQL Server 2016 Service Pack 1, many Enterprise edition features have found their way into the lower editions. Ostensibly, this was done to allow software developers to have far more code that works across all editions of the product.

Although some features are still limited by edition (high availability, for instance), features such as Columnstore and In-Memory OLTP are turned on in every edition, including Express. However, only Enterprise edition can use all available physical RAM for these features. Other editions are limited.

Inside OUT

In-Memory OLTP considerations

In-Memory OLTP requires an overhead of at least double the amount of data for a memory-optimized object. For example, if a memory-optimized table is 5 GB in size, you will need at least 10 GB of RAM available for the exclusive use of that table. Keep this in mind before turning on this feature in the Standard edition.

With Standard edition, as well, take care when using memory-optimized table-valued functions because each new object will require resources. Too many of them could starve the working set and cause SQL Server to crash.

You can read more at Microsoft Docs at <https://docs.microsoft.com/sql/relational-databases/in-memory-oltp/requirements-for-using-memory-optimized-tables>.

Central Processing Unit

The Central Processing Unit, or CPU, and often called the “brain” of a computer, is the most important part of a system. CPU speed is measured in hertz (Hz), or cycles per second. Current processor speed is measured in GHz, or billions of cycles per second.

Modern systems can have more than one CPU, and each CPU in turn can have more than one CPU core (which, in turn, might be split up into virtual cores).

For a typical SQL Server workload, single-core speed matters. It is better to have fewer cores with higher clock speeds than more cores with lower speeds, especially for non-Enterprise editions.

With systems that have more than one CPU, each CPU might be allocated its own set of memory, depending on the physical motherboard architecture.

Simultaneous multithreading

Some CPU manufacturers have split their physical cores into virtual cores to try to eke out even more performance. They do this via a feature called *simultaneous multithreading* (SMT). Intel calls this Hyper-Threading, so when you buy a single Intel® Xeon® CPU with 20 physical cores, the OS will see 40 virtual cores, because of SMT.

SMT becomes especially murky with virtual machines (VMs) because the guest OS might not have any insight into the physical versus logical core configuration.

SMT should be turned on for physical database servers. For virtual environments, you need to take care to ensure that the virtual CPUs are allocated correctly. See the section “Abstracting hardware away with virtualization” later in this chapter.

Non-Uniform Memory Access

CPUs are the fastest component of a system, and they spend a lot of time waiting for data to come to them. In the past, all CPUs would share one bank of RAM on a motherboard, using a shared bus. This caused performance problems as more CPUs were added because only one CPU could access the RAM at a time.

Multi-Channel Memory Architecture tries to resolve this by increasing the number of channels between CPUs and RAM, to reduce contention during concurrent access.

A more practical solution is for each CPU to have its own local physical RAM, situated close to each CPU socket. This configuration is called Non-Uniform Memory Access (NUMA). The advantages are that each CPU can access its own RAM, making processing much faster. However, if a CPU needs more RAM than it has in its local set, it must request memory from one of the other CPUs in the system (called *foreign memory access*), which carries a performance penalty.

SQL Server is NUMA-aware. In other words, if the OS recognizes a NUMA configuration at the hardware layer, where more than one CPU is plugged in, and each CPU has its own set of physical RAM (see Figure 2-2), SQL Server will split its internal structures and service threads across each NUMA node.

Since SQL Server 2014 Service Pack 2, the Database Engine automatically configures NUMA nodes at an instance level, using what it calls *soft-NUMA*. If more than eight CPU cores are detected (including SMT cores), *soft-NUMA* nodes are created automatically in memory.

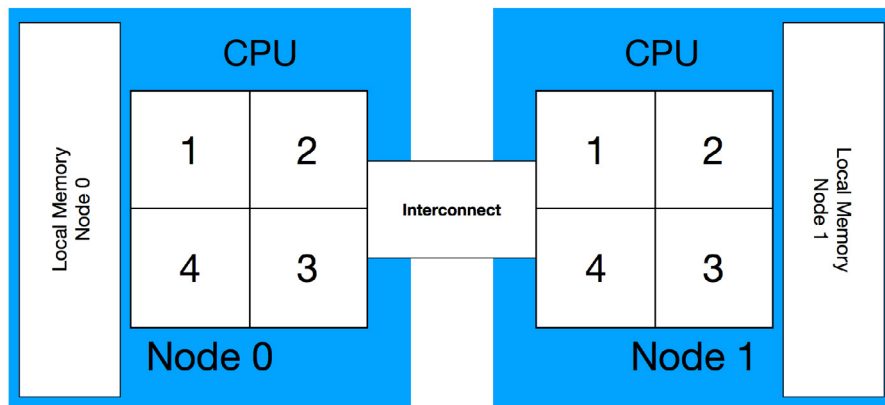


Figure 2-2 Two-socket NUMA configuration.

Inside OUT

Core counts and editions

SQL Server Standard edition has an artificial limit of 24 CPU physical cores that it can use. For instance, if a system contains two 16-core CPUs, for a total of 32 cores, Standard edition will need to be licensed for all 32 cores, even though it won't use eight of them.

Additionally, the NUMA distribution will be unbalanced because SQL Server will use the first 16 cores on the first CPU, and eight from the second CPU, unless you configure the SQL Server CPU usage using the affinity settings (for more information on how to do this, see the section "Configuration settings" in Chapter 3).

Be careful when choosing the hardware and edition for your SQL Server installation. If you're planning to install several VMs on one system, a better option would be Enterprise edition, licensed for all cores on the hardware. This would automatically cover all SQL Server VMs that you install on that hardware.

Disable power saving everywhere

Modern systems can use power saving settings to reduce the amount of electricity used by a server. Although this is good for the environment, it is bad for query performance because the CPU core speed might be reduced to save energy.

For all operating systems running SQL Server, turn on High Performance at the OS level, and double-check that High Performance is set at the BIOS level, as well. For dedicated VM hosts, this will require downtime to make the change.

Storing your data

When data is not in memory, it is *at rest*, and must be saved somewhere. Storage technology has evolved rapidly over the past few years, so we no longer think of storage as a mechanical hard drive containing one or more spinning metal disks with a magnetic surface. But, old habits die hard, and colloquially we still refer to a nonvolatile storage subsystem as "the disk," even if it might take another form. In this book, however, we refer to it as a "drive."

In the context of SQL Server, the storage subsystem should have low latency, so that when the database engine accesses the drive to perform reads and writes, those reads and writes should complete as quickly as possible. In the following list, we present some commonly used terms with respect to storage devices.

- **Drive.** The physical storage device. This might be a mechanical drive, a solid-state drive with the same form-factor as a mechanical drive, or a card that plugs directly into the motherboard.
- **Volume.** A logical representation of storage, as viewed by the OS. This might be one drive, part of a drive, or a logical section of a storage array. On Microsoft Windows, a volume usually gets its own drive letter or mount point.
- **Latency.** Measured in milliseconds, latency is how long it takes for data to be read from a drive (seconds per read), and written to a drive (seconds per write).
- **IOPS.** Input/output operations per second, or IOPS, is the number of reads and writes per second. A storage device might have differing performance depending on whether the IOPS are sequential or random. IOPS are directly related to latency by means of the queue depth.
- **Queue depth.** The number of outstanding read and write requests in a storage device's request queue. The deeper the queue depth, the faster the drive.

SQL Server performance is directly related to storage performance. The move toward virtualization and shared storage arrays has placed more emphasis on random data access patterns. Low latency and high random IOPS will thus benefit the average SQL Server workload.

In the next two chapters, we go into more detail about the preferred storage configuration for SQL Server.

Types of storage

Nonvolatile storage can be split up into two main areas: *mechanical and solid-state*.

Mechanical hard drives

Traditional spinning disks have a built-in latency, called *seek time*, due to their shape and physical nature. The read/write head is mounted on an arm that must scan the surface of the disk as it spins, seeking a particular area to perform the I/O operation. If the data on the spinning disk is fragmented, it can take longer to access because the head must skip around, finding data or free space.

The standard interface for mechanical drives is Serial ATA (SATA) or Serial Attached SCSI (SAS).

As spinning disks increase in capacity, the tracks between data become narrower, which causes performance to decrease, and increases the likelihood of mechanical failure or data corruption. The limits are pushed because of the rotational energy in the disk itself, so there is a physical speed limit to the motor.

In other words, mechanical disks grow bigger but slower and more prone to failure.

Solid-state drives

Solid-state technology, which makes use of flash memory, eliminates seek time entirely because the path to each cell where the data is stored is almost instantaneous. This is what makes solid-state storage so much faster than mechanical storage.

Solid-state storage devices can take many different forms. The most common in consumer devices is a 2.5-inch enclosure with a SATA interface, which was common with mechanical laptop drives. This accommodates a drop-in replacement of mechanical storage.

In server architecture, however, flash memory can take several forms. For local storage, they make use of the Peripheral Component Interconnect Express (PCIe) interface and plug directly into the motherboard. An example of this is Non-Volatile Memory Express (NVMe).

As the technology evolves, the performance will only improve as capacity grows. Solid state is not perfect though; data can be written to a particular cell only a certain number of times before it fails. You might have experienced this yourself with thumb drives, which tend to fail after heavy usage. Algorithms to balance writes across cells, called *wear-leveling*, help to extend the lifespan of a solid-state device.

Another problem with flash memory is *write-amplification*. On a mechanical drive, if a file is overwritten, the previous file is marked for deletion, but is not actually deleted from the disk surface. When the drive needs to write to that area again, it overwrites the location without removing what was there before.

Solid-state drives must erase the location in question before writing the new data, which has a performance impact. The size of the cells might also require a larger area to be erased than the file itself (if it is a small file), which compounds the performance impact. Various techniques exist to mitigate write amplification, but this does reduce the lifespan of flash memory.

The performance problems with mechanical disks, and the lifespan problems with both mechanical and solid-state drives, can be mitigated by combining them into drive arrays, to reduce the risk of failure by balancing the load and increase performance.

Configuring the storage layer

Nonvolatile storage can stand alone, in the form of Direct-Attached Storage, or be combined in many ways to provide redundancy or consolidation, perhaps even offering different levels of performance in order to manage costs better. For example, archive data might not need to be stored on the fastest available drive if it is accessed infrequently.

Direct-Attached Storage

Direct-Attached Storage (DAS) is plugged directly into the system accessing it. Also called local storage, it can comprise independent mechanical hard drives, solid-state drives, tape drives for backups, CD and DVD-ROM drives, or even enclosures containing storage arrays.

DAS has a lower latency than a Storage-Area Network or Network-Attached Storage (more on these later in the chapter) because there is no network to traverse between the system and the storage. However, it cannot be shared with other systems, unless the local file system is shared across the network using a protocol such as Server Message Block (SMB) 3.0.

For SQL Server, DAS comprising flash storage (solid-state) is preferred for TempDB, which is also supported (and recommended) in a Failover Cluster Instance. You can also use DAS for the buffer pool extension.

- To see how you should best configure TempDB, see the section “Configuration settings” in Chapter 3.

Storage arrays and RAID

Combining drives in an enclosure with a controller to access each drive, without any thought to redundancy or performance, is called *JBOD* (colloquially, “just a bunch of disks”). These drives might be accessed individually or combined into a single volume.

When done correctly, combining drives into an array can increase overall performance and/or lower the risk of data loss should one or more of the drives in the array fail. This is called Redundant Array of Independent Disks (RAID).

RAID offers several levels of configuration, which trade redundancy for performance. More redundancy means less raw capacity for the array, but this can reduce data loss. Faster performance can bring with it data loss.

Striping without parity (RAID 0) uses multiple drives to improve raw read/write performance, but with zero redundancy. If one drive fails, there is significant chance of catastrophic data loss across the entire array. JBOD configurations that span across drives fall under this RAID level.

Mirroring (RAID 1) uses two drives that are written to simultaneously. Although there is a slight write penalty because both drives must save their data at the same time, and one might take longer than the other, the read performance is nearly double that of a single drive because both drives can be read in parallel (with a small overhead caused by the RAID controller selecting the drive and fetching the data). Usable space is 50 percent of raw capacity, and only one drive in the array can be lost and still have all data recoverable.

Striping with parity (RAID 5) requires an odd number of three or more drives, and for every single write, one of the drives is randomly used for parity (a checksum validation). There is a larger write penalty because all drives must save their data and parity must be calculated and persisted. If a single drive is lost from the array, the other drives can rebuild the contents of the lost drive, based on the parity, but it can take some time to rebuild the array. Usable space is calculated as the number of drives minus one. If there are three drives in the array, the usable space is the sum of two of those drives, with the space from the third used for parity (which is evenly distributed over the array). Only one drive in the array can be lost and still have full data recovery.

Combinations of the base RAID configurations are used to provide more redundancy and performance, including RAID 1+0 (also known as RAID 10), RAID 0+1, and RAID 5+0 (also known as RAID 50).

In RAID 1+0, two drives are configured in a mirror (RAID 1) for redundancy, and then each mirror is striped together (RAID 0) for performance reasons.

In RAID 0+1, the drives are striped first (RAID 0), and then mirrored across the entire RAID 0 set (RAID 1). Usable space for RAID 0+1 and 1+0 is 50 percent of the raw capacity.

To ensure full recovery from failure in a RAID 1+0 or 0+1 configuration, an entire side of the mirror can be lost, or only one drive from each side of the mirror can be lost.

In RAID 5+0, a number of drives (three or more) is configured in a RAID 5 set, which is then striped (with no parity) with at least one other RAID 5 set of the same configuration. Usable space is $(x - 1) / y$, where x is the number of drives in each nested RAID 5 set, and y is the number of RAID 5 sets in this array. If there are nine drives, six of them are usable. Only one drive from each RAID 5 set can be lost with full recovery possible. If more than one drive in any of the RAID 5 sets is lost, the entire 5+0 array is lost.

SQL Server requires the best performance from a storage layer as possible. When looking at RAID configurations, RAID 1+0 offers the best performance and redundancy.

NOTE

Some database administrators tend to believe that RAID is an alternative to backups, but it does not protect 100 percent against data loss. A common backup medium is digital tape, due to its low cost and high capacity, but more organizations are making use of cloud storage options, such as Microsoft Azure Archive Storage and Amazon Glacier, for long-term, cost-effective backup storage solutions. Always make sure that you perform regular SQL Server backups that are copied securely off-premises.

Centralized storage with a Storage-Area Network

A Storage-Area Network (SAN) is a network of storage arrays that can comprise tens, hundreds, or even thousands of drives (mechanical or solid-state) in a central location, with one or more RAID configurations, providing block-level access to storage. This reduces wasted space, and allows easier management across multiple systems, especially for virtualized environments.

Block-level means that the OS can read or write blocks of any size and any alignment. This offers the OS a lot of flexibility in making use of the storage.

You can carve the total storage capacity of the SAN into logical unit numbers (LUNs), and each LUN can be assigned to a physical or virtual server. You can move these LUNs around and resize them as required, which makes management much easier than attaching physical storage to a server.

The disadvantage of a SAN is that you might be at the mercy of misconfiguration or a slow network. For instance, the RAID might be set to a level that has poor write performance, or the blocks of the storage are not aligned appropriately.

Storage administrators might not understand specialized workloads like SQL Server, and choose a performance model that satisfies the rest of the organization to reduce administration overhead but which penalizes you.

Inside OUT

Fibre Channel versus iSCSI

Storage arrays might use Fibre Channel (FC) or Internet Small Computer Systems Interface (iSCSI) to connect systems to their storage.

FC can support data transfer at a higher rate than iSCSI, which makes it better for systems that require lower latency, but it comes at a higher cost for specialized equipment.

iSCSI uses standard TCP/IP, which makes it potentially cheaper because it can run on existing network equipment. You can further improve iSCSI throughput by isolating the storage to its own dedicated network.

Network-Attached Storage

Network-Attached Storage (NAS), is usually a specialized hardware appliance connected to the network, typically containing an array of several drives, providing file-level access to storage.

Unlike the SAN's block-level support, NAS storage is configured on the appliance itself, and file sharing protocols (such as SMB, Common Internet File System [CIFS] and Network File System [NFS]) are used to share the storage over the network.

NAS appliances are fairly common because they provide access to shared storage at a much lower monetary cost than a SAN. You should keep in mind security considerations regarding file-sharing protocols.

Storage Spaces

Windows Server 2012 and later support Storage Spaces, which is a way to manage local storage in a more scalable and flexible way than RAID.

Instead of creating a RAID set at the storage layer, Windows Server can create a virtual drive at the OS level. It might use a combination of RAID levels, and you can decide to combine different physical drives to create performance tiers.

For example, a server might contain 16 drives. Eight of them are spinning disks, and eight are solid state. You can use Storage Spaces to create a single volume with all 16 drives, and keep the active files on the solid-state portion, increasing performance dramatically.

SMB 3.0 file share

SQL Server supports storage located on a network file share that uses the SMB 3.0 protocol or higher because it is now fast and stable enough to support the storage requirements of the Database Engine (performance and resilience). This means that you can build a Failover Cluster Instance (see the section on this later in the chapter) without shared storage such as a SAN.

Network performance is critically important, though, so we recommend a dedicated and isolated network for the SMB file share, using network interface cards that support Remote Direct Memory Access (RDMA). This allows the SMB Direct feature in Windows Server to create a low-latency, high-throughput connection using the SMB protocol.

SMB 3.0 might be a feasible option for smaller networks with limited storage capacity and a NAS, or in the case of a Failover Cluster Instance without shared storage. For more information, read Chapter 12.

Connecting to SQL Server over the network

We have covered a fair amount about networking just discussing the storage layer, but there is far more to it. In this section, we look at what is involved when accessing the Database Engine over a network, and briefly discuss Virtual Local-Area Networks.

Unless a SQL Server instance and the application accessing it is entirely self-contained, database access is performed over one or more network interfaces. This adds complexity with

authentication, given that malicious actors might be scanning and modifying network packets in flight.

CAUTION

Ensure that all TCP/IP traffic to and from the SQL Server is encrypted. For applications that are located on the same server as the SQL Server instance, this is not required if you're using the Shared Memory Protocol.

SQL Server 2017 requires strict rules with respect to network security, which means that older versions of the connectors or protocols used by software developers might not work as expected.

Transport Security Layer and its forerunner, Secure Sockets Layer, (together known as TLS/SSL, or just SSL), are methods that allow network traffic between two points to be encrypted. (For more information, see Chapter 7.) Where possible, you should use newer libraries that support TLS encryption. If you cannot use TLS to encrypt application traffic, you should use IPSec, which is configured at the OS level.

Protocols and ports

Connections to SQL Server are made over the Transport Control Protocol (TCP), with port 1433 as the default port for a default instance. Some of this is covered in Chapter 1, and again in Chapter 7. Any named instances are assigned random ports by the SQL Server Configuration Manager, and the SQL Browser service coordinates any connections to named instances. It is possible to assign static TCP ports to named instances by using the Configuration Manager.

There are ways to change the default port after SQL Server is installed, through the SQL Server Configuration Manager. We do not recommend changing the port, however, because it provides no security advantage to a port scanner, but some network administration policies require it.

Networking is also the foundation of cloud computing. Aside from the fact that the Azure cloud is accessed over the internet (itself a network of networks), the entire Azure infrastructure, which underlies both infrastructure-as-a-service (virtual machines with Windows or Linux running SQL Server) and platform-as-a-service (Azure SQL Database) offerings, is a virtual fabric of innumerable components tied together with networking.

Added complexity with Virtual Local-Area Networks

A Virtual Local-Area Network (VLAN) gives network administrators the ability to logically group machines together even if they are not physically connected through the same network switch.

It makes it possible for servers to share their resources with one another over the same physical LAN, without interacting with other devices on the same network.

VLANs work at a very low level (the data link layer, or OSI Layer 2), and are configured on a network switch. A port on the switch might be dedicated to a particular VLAN, and all traffic to and from that port is mapped to a particular VLAN by the switch.

High availability concepts

With each new version of Windows Server, terminology and definitions tend to change or adapt according to the new features available. With SQL Server now supported on Linux, it is even more important to get our heads around what it means when we discuss high availability.

At its most basic, high availability (HA) means that a service offering of some kind (for example, SQL Server, a web server, an application, or a file share) will survive an outage of some kind, or at least fail predictably to a standby state, with minimal loss of data and minimal downtime.

Everything can fail. An outage might be caused by a failed hard drive, which could in turn be a result of excessive heat, excessive cold, excessive moisture, or a datacenter alarm that is so loud that its vibrational frequency damages the internal components and causes a head crash.

You should be aware of other things that can go wrong, as noted in the list that follows; this list is certainly not exhaustive, but it's incredibly important to understand that assumptions about hardware, software, and network stability are a fool's errand:

- A failed network interface card
- A failed RAID controller
- A power surge or brownout causing a failed power supply
- A broken or damaged network cable
- A broken or damaged power cable
- Moisture on the motherboard
- Dust on the motherboard
- Overheating caused by a failed fan
- A faulty keyboard that misinterprets keystrokes
- Failure due to bit rot
- Failure due to a bug in SQL Server

- Failure due to poorly written code in a file system driver that causes drive corruption
- Capacitors failing on the motherboard
- Insects or rodents electrocuting themselves on components (this smells really bad)
- Failure caused by a fire suppression system that uses water instead of gas
- Misconfiguration of a network router causing an entire geographical region to be inaccessible
- Failure due to an expired SSL or TLS certificate
- Running a DELETE or UPDATE statement without a WHERE clause (human error)

Why redundancy matters

Armed with the knowledge that everything can fail, you should build in redundancy where possible. The sad reality is that these decisions are governed by budget constraints. The amount of money available is inversely proportional to the amount of acceptable data loss and length of downtime. For business-critical systems, however, uptime is paramount, and a highly available solution will be more cost effective than being down, considering the cost-per-minute to the organization.

It is nearly impossible to guarantee zero downtime with zero data loss. There is always a trade-off. The business decides on that trade-off, based on resources (equipment, people, money), and the technical solution is in turn developed around that trade-off. The business drives this strategy using two values called the Recovery Point Objective and Recovery Time Objective, which are defined in a Service-Level Agreement (SLA).

Recovery Point Objective

A good way to think of Recovery Point Objective (RPO) is “How much data are you prepared to lose?” When a failure occurs, how much data will be lost between the last transaction log backup and the failure? This value is usually measured in seconds or minutes.

Recovery Time Objective

The Recovery Time Objective (RTO) is defined as how much time is available to bring the environment up to a known and usable state after a failure. There might be different values for HA and disaster recovery scenarios. This value is usually measured in hours.

Disaster recovery

HA is not disaster recovery (DR). They are often grouped under the same heading (HA/DR), mainly because there are shared technology solutions for both concepts, but HA is about

keeping the service running, whereas DR is what happens when the infrastructure fails entirely. DR is like insurance: you don't think you need it until it's too late. HA costs more money, the shorter the RPO.

NOTE

A disaster is any failure or event that causes an unplanned outage.

Clustering

Clustering is the connecting of computers (nodes) in a set of two or more nodes, that work together and present themselves to the network as one computer.

In most cluster configurations, only one node can be active in a cluster. To ensure that this happens, a quorum instructs the cluster as to which node should be active. It also steps in if there is a communication failure between the nodes.

Each node has a vote in a quorum. However, if there is an even number of nodes, to ensure a simple majority an additional witness must be included in a quorum to allow for a majority vote to take place.

Inside OUT

What is Always On?

Always On is the name of a group of features, which is akin to a marketing term. It is not the name of a specific technology. There are two separate technologies that happen to fall under the Always On label, and these are addressed a little later in this chapter. The important thing to remember is that "Always On" does not mean "availability groups," and there is a space between "Always" and "On."

Windows Server Failover Clustering

As Microsoft describes it:

"Failover clusters provide high availability and scalability to many server workloads. These include server applications such as Microsoft Exchange Server, Hyper-V, Microsoft SQL Server, and file servers. The server applications can run on physical servers or virtual machines. [Windows Server Failover Clustering] can scale to 64 physical nodes and to 8,000 virtual machines." ([https://technet.microsoft.com/library/hh831579\(v=ws.11\).aspx](https://technet.microsoft.com/library/hh831579(v=ws.11).aspx)).

The terminology here matters. Windows Server Failover Clustering is the name of the technology that underpins a Failover Cluster Instance (FCI), where two or more Windows Server Failover

Clustering nodes (computers) are connected together in a Windows Server Failover Clustering resource group and masquerade as a single machine behind a network endpoint called a Virtual Network Name (VNN). A SQL Server service that is installed on an FCI is cluster-aware.

Linux failover clustering with Pacemaker

Instead of relying on Windows Server Failover Clustering, SQL Server on a Linux cluster can make use of any cluster resource manager. Microsoft recommends using Pacemaker because it ships with a number of Linux distributions, including Red Hat and Ubuntu.

Inside OUT

Node fencing and STONITH on Linux

If something goes wrong in a cluster, and a node is in an unknown state after a set time-out period, that node must be isolated from the cluster and restarted or reset. On Linux clusters, this is called *node fencing*, following the STONITH principle (“Shoot the Other Node in the Head”). If a node fails, STONITH will provide an effective, if drastic manner of resetting or powering-off a failed Linux node.

Resolving cluster partitioning with quorum

Most clustering technologies make use of the quorum model, to prevent a phenomenon called *partitioning*, or “split brain.” If there is an even number of nodes, and half of these nodes go offline from the view of the other half of the cluster, and vice versa, you end up with two halves thinking that the cluster is still up and running, and each with a primary node (split brain).

Depending on connectivity to each half of the cluster, an application continues writing to one half of the cluster while another application writes to the other half. A best-case resolution to this scenario would require rolling back to a point in time before the event occurred, which would cause loss of any data written after the event.

To prevent this, each node in a cluster shares its health with the other nodes using a periodic heartbeat. If more than half do not respond in a timely fashion, the cluster is considered to have failed. Quorum works by having a simple majority vote on what constitutes “enough nodes.”

In Windows Server Failover Clustering, there are four types of majority vote: Node, Node and File Share, Node and Disk, and Disk Only. In the latter three types, a separate witness is used, which does not participate in the cluster directly. This witness is given voting rights when there is an even number of nodes in a cluster, and therefore a simple majority (more than half) would not be possible.

Always On FCIs

You can think of a SQL Server FCI as two or more nodes with shared storage (usually a SAN because it is most likely to be accessed over the network).

On Windows Server, SQL Server can take advantage of Windows Server Failover Clustering to provide HA (the idea being minimal downtime) at the server-instance level, by creating an FCI of two or more nodes. From the network's perspective (application, end users, and so on), the FCI is presented as a single instance of SQL Server running on a single computer, and all connections point at the VNN.

When the FCI starts, one of the nodes assumes ownership and brings its SQL Server instance online. If a failure occurs on the first node (or there is a planned failover due to maintenance), there are at least a few seconds of downtime, during which the first node cleans up as best it can, and then the second node brings its SQL Server instance online. Client connections are redirected to the new node after the services are up and running.

Inside OUT

How long does the FCI failover take?

During a planned failover, any dirty pages in the buffer pool must be written to the drive; thus, the downtime could be longer than expected on a server with a large buffer pool. You can read more about checkpoints in Chapter 3 and Chapter 4.

On Linux, the principle is very similar. A cluster resource manager such as Pacemaker manages the cluster, and when a failover occurs, the same process is followed from SQL Server's perspective, in which the first node is brought down and the second node is brought up to take its place as the owner. The cluster has a virtual IP address, just as on Windows. You must add the virtual network name manually to the DNS server.

- You can read more about setting up a Linux cluster in Chapter 11.

FCIs are supported on SQL Server Standard edition, but are limited to two nodes.

The versatility of Log Shipping

SQL Server Transaction Log Shipping is an extremely flexible technology to provide a relatively inexpensive and easily managed HA and DR solution.

The principle is as follows: a primary database is in either the Full or Bulk Logged recovery model, with transaction log backups being taken regularly every few minutes. These transaction log backup files are transferred to a shared network location, where one or more secondary servers restore the transaction log backups to a standby database.

If you use the built-in Log Shipping Wizard in SQL Server Management Studio, on the Restore tab, click Database State When Restoring Backups, and then choose the No Recovery Mode or Standby Mode option (<https://docs.microsoft.com/sql/database-engine/log-shipping/configure-log-shipping-sql-server>).

If you are building your own log shipping solution, remember to use the RESTORE feature with NORECOVERY, or RESTORE with STANDBY.

If a failover occurs, the tail of the log on the primary server is backed up the same way (if available—this guarantees zero data loss of committed transactions), transferred to the shared location, and restored after the latest regular transaction logs. The database is then put into RECOVERY mode (which is where crash recovery takes place, rolling back incomplete transactions and rolling forward complete transactions).

As soon as the application is pointed to the new server, the environment is back up again with zero data loss (tail of the log was copied across) or minimal data loss (only the latest shipped transaction log was restored).

Log Shipping is a feature that works on all editions of SQL Server, on Windows and Linux. However, because Express edition does not include the SQL Server Agent, Express can be only a witness, and you would need to manage the process through a separate scheduling mechanism. You can even create your own solution for any edition of SQL Server, using Azure Blob Storage and AzCopy.exe, for instance.

Always On availability groups

As alluded to previously, this is generally what people mean when they incorrectly say “Always On.” However, its official name is Always On availability groups. In shorthand, you can refer simply to these as availability groups (or AGs).

What is an availability group, anyway? In the past, SQL Server offered database mirroring and failover clustering as two distinct HA offerings. However, with database mirroring officially deprecated since SQL Server 2012, coinciding with the introduction of availability groups, it is easier to think of availability groups as a consolidation of these two offerings as well as Log Shipping thrown in for good measure.

Inside OUT

What was database mirroring?

Database mirroring worked at the database level by maintaining two copies of a single database across two separate SQL Server instances, keeping them synchronized with a steady stream of active transaction log records.

Availability groups provide us with the ability to keep a discrete set of databases highly available across one or more nodes in a cluster. They work at the database level, as opposed to an entire server-instance level, like FCIs do.

Unlike the cluster-aware version of SQL Server, when it installed as part of an FCI, SQL Server on an availability group is installed as a standalone instance.

An availability group (on Windows Server through Windows Server Failover Clustering, and on Linux through a cluster resource manager like Pacemaker) operates at the *database level only*. As depicted in Figure 2-3, it is a set of one or more databases in a group (an *availability replica*) that are *replicated* (using Log Shipping) from a *primary replica* (there can be only one primary replica), to a maximum of eight *secondary replicas*, using synchronous or asynchronous *data synchronization*. Let's take a closer look at each of these:

- **Synchronous data synchronization.** The log is hardened (the transactions are committed to the transaction log) on every secondary replica *before* the transaction is committed on the primary replica. This guarantees zero data loss, but with a potentially significant performance impact. It can be costly to reduce network latency to a point at which this is practical for highly transactional workloads.
- **Asynchronous data synchronization.** The transaction is considered committed as soon as it is hardened in the transaction log on the primary replica. If something were to happen before the logs are hardened on all of the secondary replicas, there is a chance of data loss, and the recovery point would be the most recently committed transaction that made it successfully to all of the secondary replicas. With delayed durability turned on, this can result in faster performance, but higher risk of data loss.

Inside OUT

What is delayed durability?

Starting in SQL Server 2014, delayed durability (also known as lazy commit) is a storage optimization feature that returns a successful commit before transaction logs are actually saved to a drive. Although this can improve performance, the risk of data loss is higher because the transaction logs are saved only when the logs are flushed to a drive asynchronously. To learn more, go to <https://docs.microsoft.com/sql/relational-databases/logs/control-transaction-durability>.

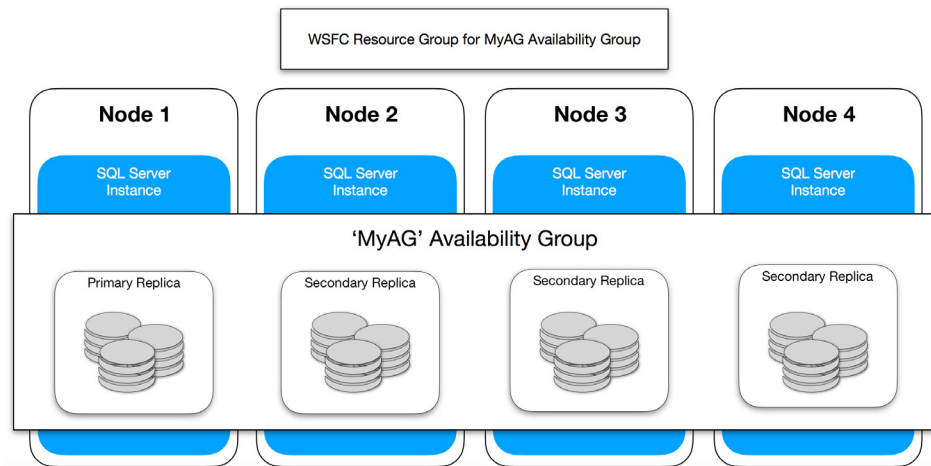


Figure 2-3 A Windows Server Failover Clustering cluster with four nodes.

You can use read-only secondary replicas for running reports and other operations that reduce the load on the primary replica. This also includes backups and database consistency checks, but you must also perform these on the primary replica when there is a low-usage period or planned maintenance window.

If the primary replica fails, one of the secondary replicas is promoted to the primary, with a few seconds of downtime while the databases run through crash recovery, and minimal data loss.

Read-scale availability groups

SQL Server 2017 introduces a new architecture that allows for multiple read-only secondary replicas, but does not offer HA. The major difference is that a read-scale availability group does not have a cluster resource manager.

What this allows is reduced contention on a business-critical workload by using read-only routing or connecting directly to a readable secondary replica, without relying on a clustering infrastructure on Windows or Linux.

- For more information, go to Microsoft Docs at <https://docs.microsoft.com/sql/database-engine/availability-groups/windows/read-scale-availability-groups>.

Distributed availability groups

Instead of having an availability group on one cluster, a distributed availability group can span two separate availability groups, on two separate clusters (Windows Server Failover Clustering or Linux, each cluster can run on a different OS) that are geographically separated. Provided that these two availability groups can communicate with each other, you can configure them in a distributed availability group. This allows a more flexible DR scenario, plus it makes possible multisite replicas in geographically diverse areas.

The main difference from a normal availability group, is that the configuration is stored in SQL Server, not the underlying cluster. With a distributed availability group, only one availability group can perform data modification at any time, even though both availability groups have a primary replica. To allow another availability group to write to its primary replica database requires a manual failover, using `FORCE_FAILOVER_ALLOW_DATA_LOSS`.

Basic availability groups

SQL Server Standard edition supports a single-database HA solution, with a limit of two replicas. The secondary replica does not allow backups or read access. Although these limits can be frustrating, they do make it possible to offer another kind of HA offering with Standard edition.

- For more information, go to Microsoft Docs at <https://docs.microsoft.com/sql/database-engine/availability-groups/windows/basic-availability-groups-always-on-availability-groups>.

Improve redundancy and performance with NIC teaming

NIC teaming, also known as *link aggregation*, uses two or more network interfaces to improve redundancy (failover), or increase the available bandwidth (bandwidth aggregation). In the Microsoft space, this is also called *load balancing and failover support* (LBFO). NIC teaming can work at the network-card level, where two or more NICs are combined into a virtual NIC on a server, or on a network switch level, where two or more network ports are aggregated.

When traffic encounters the aggregated network ports, the switch will know which port is the least busy at that time, and direct the packet to one of the other ports. This is how network load balancing works. There might be one or more servers behind each port, where the load

balancing is distributed to multiple servers. Otherwise they might just be connected to a single server with multiple NICs, used just for redundancy, so that if one network interface card on the server fails, the server remains available.

Securing SQL Server

Security is covered in more depth in Chapter 6, and Chapter 7, so what follows is a basic overview of server access security, not a discussion about permissions within SQL Server.

When connecting to SQL Server on Windows or Linux, or SQL Database in Azure, security is required to keep everyone out except the people who need access to the database.

Active Directory, using Integrated Authentication, is the primary method for connecting to SQL Server on a Windows domain. When you sign in to an Active Directory domain, you are provided a token that contains your privileges and permissions.

This is different from SQL Server Authentication, however, which is managed directly on the SQL Server instance and requires a user name and password to travel over the network.

Integrated authentication and Active Directory

Active Directory covers a number of different identity services, but the most important is Active Directory Domain Services, which manages your network credentials (your user account) and what you can do on the network (access rights). Having a network-wide directory of users and permissions facilitates easier management of accounts, computers, servers, services, devices, file sharing, and so on.

In this type of environment, SQL Server would be managed as just another service on the network, and the Active Directory Domain Service would control who has access to that SQL Server instance. This is much easier than having to manage per-server security, which is time consuming, difficult to troubleshoot, and prone to human error.

Inside OUT

Linux and Active Directory

SQL Server 2017 on Linux supports integrated authentication using Active Directory. For more information, read the Microsoft Docs article titled “Active Directory Authentication with SQL Server on Linux,” which is available at <https://docs.microsoft.com/sql/linux/sql-server-linux-active-directory-authentication>.

Authenticating with Kerberos

Kerberos is the default authentication protocol used in a Windows Active Directory domain; it is the replacement of NT LAN Manager (NTLM).

Kerberos ensures that the authentication takes place in a secure manner, even if the network itself might not be secure, because passwords and weak hashes are not being transferred over the wire. Kerberos works by exchanging encrypted tickets verified by a Ticket Granting Server (TGS; usually the domain controller).

A service account that runs SQL Server on a particular server, under an Active Directory service account, must register its name with the TGS, so that client computers are able to make a connection to that service over the network. This is called a *Service Principal Name*.

CAUTION

NTLM is the authentication protocol on standalone Windows systems and is used on older operating systems and older domains. You can also use NTLM as a fallback on Active Directory domains for backward compatibility.

The NTLM token created during the sign-in process consists of the domain name, the user name, and a one-way hash of the user's password. Unfortunately, this hash is considered cryptographically weak and can be cracked (decrypted) in a few seconds by modern cracking tools. It is incumbent on you to use Kerberos where at all possible.

Understanding the Service Principal Name

As shown in Figure 2-4, when a client logs into a Windows domain, it is issued a ticket by the TGS. This ticket is called a ticket-granting ticket (TGT), but it's easier to think of it as the client's credentials. When the client wants to communicate with another node on the network (for example, SQL Server), this node (or "principal") must have a Service Principal Name (SPN) registered with the TGS.

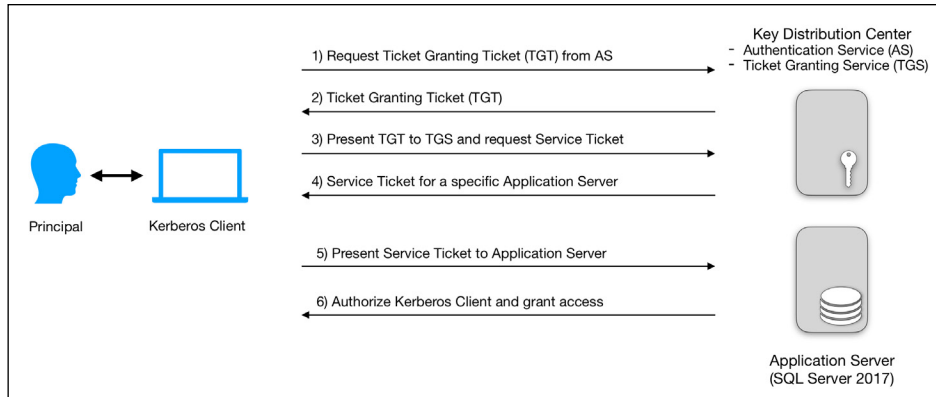


Figure 2-4 How Kerberos authentication works.

It is this SPN that the client uses to request access. After a verification step, a ticket and session key is sent from the TGS, to both the SQL Server and the client, respectively. When the client uses the ticket and session key on the SQL Server, the connection is authenticated by the SQL Server using its own copy of the session key.

For SQL Server to use Kerberos authentication instead of the older and less-secure NTLM, the Windows domain account that runs the SQL Server service, must register the SPN with the domain controller. Otherwise, the authentication will fall back to NTLM, which is far less secure. The easiest way to achieve this is to give the service account *Write ServicePrincipalName* permission in Active Directory Domain Service. To configure an SPN manually, you must use the *Setspn.exe* tool (built in to Windows).

Accessing other servers and services with delegation

Kerberos delegation allows an application (such as SQL Server, or Internet Information Services) to reuse end-user credentials to access a different server. This is intended to solve the so-called “double-hop issue,” in which the TGS verifies only the first hop, namely the connection between the client and the registered server. In normal circumstances, any additional connections (the second hop) would require reauthentication.

Delegation impersonates the client by sending the client’s TGT on the client’s behalf. This in turn causes the TGS to send tickets and session keys to the original server and the new server, allowing authentication. Because the original connection is still authenticated using the same TGT, the client now has access to the second server.

For delegation to work, the service account for the first server must be trusted for delegation, and the second server must be in the same Active Directory forest or between forests with the appropriate trust relationship.

Azure Active Directory

Azure Active Directory (Azure AD) is concerned with identity management for internet-based (and on-premises) services, which use HTTP and HTTPS to access websites and web services, without the hierarchy associated with on-premises Active Directory.

You can use Azure AD for user and application authentication; for example, to connect to Azure SQL Database or Microsoft Office 365. There are no Organizational Units or Group Policy Objects. You cannot join a machine to an Azure AD domain, and there is no NTLM or Kerberos authentication. Instead, protocols like OAuth, OpenID Connect (based on OAuth 2.0), SAML, and WS-Federation are used.

You can authenticate (prove who you are), which then provides authorization (permission, or claims) to access certain services, and these services might not even be controlled by the service that authenticated you. Think back to network credentials. On an on-premises Active Directory, your user credentials know who you are (authentication), and what you can do (authorization).

Protocols like OpenID Connect blur these lines, by extending an authorization protocol (what you can do) into an authentication protocol, as well (who you are). Although this works in a similar manner to Kerberos, whereby an authorization server allows access to certain internet services and applications, permissions are granted with *claims*.

Asserting your identity by using claims

Claims are a set of “assertions of information about the subject that has been authenticated” (<https://docs.microsoft.com/azure/active-directory/develop/active-directory-authentication-scenarios#claims-in-azure-ad-security-tokens>).

Think of your user credentials as a security token that indicates who you are based on how you were authenticated. This depends on the service you originally connected to (i.e., Facebook, LinkedIn, Google, Office 365, or Twitter).

Inside that user object is a series of properties, or attributes, usually in the form of key–value pairs. Each set of attributes, or claims, is dependent on the authentication service used.

Authentication services like Azure AD might restrict the amount of information permissible in a user object, to provide the service or application just enough information about you to prove who you are, and give you access to the service you’re requesting, without sharing too much about you or the originating authentication service.

Federation and single sign-on

Federation is a fancy word that means an independent collection of websites or services that can share information between them using claims. An authentication service allows you to sign in on one place (LinkedIn, Facebook, or Microsoft) and then use that identity for other services controlled by other entities.

This is what makes claims extremely useful. If you use a third-party authentication service, that third party will make certain information available in the form of claims (key–value pairs in your security token) that another service to which you’re connecting can access, without needing to sign in again, and without that service having access into the third-party service.

For example, suppose that you use LinkedIn to sign in to a blogging service so that you can leave a comment on a post. The blogging service does not have any access to your LinkedIn profile, but the claims it provides might include a URL to your profile image, a string containing your full name, and a second URL back to your profile.

This way, the blogging service does not know anything about your LinkedIn account, including your employment history, because that information is not in the claims necessary to leave a blog post comment.

Logging in to Azure SQL Database

Azure SQL Database uses three levels of security to allow access to a database. First is the firewall, which is a set of rules based on origin IP address or ranges and allows connections to only TCP port 1433.

The second level is authentication (proving who you are). You can either connect by using SQL Authentication, with a username and password (like connecting to a contained database on an on-premises SQL Server instance), or you can use Azure AD Authentication.

Microsoft recommends using Azure AD whenever possible, because it does the following (according to <https://docs.microsoft.com/azure/sql-database/sql-database-aad-authentication>):

- Centralizes user identities and offers password rotation in a single place
- Eliminates storing passwords by enabling integrated Windows authentication and other forms of authentication supported by Azure AD
- Offers token (claims-based) authentication for applications connecting to Azure SQL Database

The third level is authorization (what you can do). This is managed inside the Azure SQL database, using role memberships and object-level permissions, and works exactly the same way as it would with an on-premises SQL Server instance.

► You can read more about SQL Server security in Chapters 6 and 7.

Abstracting hardware with virtualization

Hardware abstraction has been around for many years, and, in fact, Windows NT was designed to be hardware independent. Taking this concept even further, virtualization abstracts the entire physical layer behind what's called a *hypervisor*, or *Virtual Machine Manager* (VMM) so that physical hardware on a host system can be logically shared between different VMs, or guests, running their own operating systems.

To a guest OS, the VM looks like normal hardware and is accessed in the same way.

As of this writing, there are two main players in the virtualization market: Microsoft Hyper-V and VMware.

Inside OUT

What is the cloud?

Cloud technology is just another virtualized environment, but on a much larger scale. Millions of servers are sitting in datacenters all over the world, running tens or hundreds of VMs on each server. The hypervisor and service fabric (the software that controls and manages the environment) is what differentiates each cloud vendor.

The move to virtualization has come about because physical hardware in many organizations is not being used to its full potential, and systems might spend hundreds of hours per year sitting idle. By consolidating an infrastructure, namely putting more than one guest VM on the same physical host, you can share resources between these guests, reducing the amount of waste and increasing the usefulness of hardware.

Certain workloads and applications are not designed to share resources, and misconfiguration of the shared resources by system administrators might not take these specialized workloads into account. SQL Server is an excellent example of this, given that it is designed to make use of all the physical RAM in a server by default.

If the resources are allocated incorrectly from the host level, contention between the guests takes place. This phenomenon is known as the *noisy neighbor*, in which one guest monopolizes

resources on the host, and the other guests are negatively affected. With some effort on the part of the network administrators, this problem can be alleviated.

The benefits far outweigh the downsides, of course. You can move VMs from one host to another in the case of resource contention or hardware failure, and some hypervisors can orchestrate this without even shutting down the VM.

It is also much easier to take snapshots of virtualized file systems, which you can use to clone VMs. This can reduce deployment costs and time when deploying new servers, by “spinning up” a VM template, and configuring the OS and the application software that was already installed on that virtual hard drive.

Over time, the cost benefits become more apparent. New processors with low core counts are becoming more difficult to find. Virtualization makes it possible for you to move physical workloads to VMs (now or later) that have the appropriate virtual core count, and gives you the freedom to use existing licenses, thereby reducing cost.

- ▶ David Klee writes more on this in his article “Point Counterpoint: Why Virtualize a SQL Server?” available at <http://www.davidklee.net/2017/07/12/point-counterpoint-why-virtualize-a-sql-server>.

Resource provisioning for VMs

Setting up VMs requires understanding their anticipated workloads. Fortunately, as long as resources are allocated appropriately, a VM can run almost as fast as a physical server on the same hardware, but with all of the benefits that virtualization offers.

It makes sense, then, to overprovision resources for many general workloads.

Sharing more memory than you have (overcommit)

You might have 10 VMs running various tasks such as Active Directory Domain Controllers, DNS servers, file servers, and print servers (the plumbing of a Windows-based network, with a low RAM footprint), all running on a single host with 16 GB of physical RAM.

Each VM might require 4 GB of RAM to perform properly, but in practice, you have determined that 90 percent of the time, each VM can function with 1 to 2 GB RAM each, leaving 2 to 3 GB of RAM unused per VM. You could thus *overcommit* each VM with 4 GB of RAM (for a total of 40 GB), but still see acceptable performance, without having a particular guest swapping memory to the drive as a result of low RAM, 90 percent of the time.

For the remaining 10 percent of the time, for which paging unavoidably takes place, you might decide that the performance impact is not sufficient to warrant increasing the physical RAM on the host. You are therefore able to run 10 virtualized servers on far less hardware than they would have required as physical servers.

CAUTION

Because SQL Server makes use of all the memory it is configured to use (limited by edition), it is not good practice to overcommit memory for VMs that are running SQL Server. It is critical that the amount of RAM assigned to a SQL Server VM is available for exclusive use by the VM, and that the Max Server Memory setting is configured correctly (see Chapter 3).

Provisioning virtual storage

In the same way that you can overcommit memory, so too can you overcommit storage. This is called *thin provisioning*, in which the VM and guest OS are configured to assume that there is a lot more space available than is physically on the host. When a VM begins writing to a drive, the actual space used is increased on the host, until it reaches the provisioned limit.

This practice is common with general workloads, for which the space requirements grow predictably. An OS like Windows Server might be installed on a guest with 127 GB of visible space, but there might be only 250 GB of actual space on the drive, shared across 10 VMs.

For specialized workloads like SQL Server and Microsoft SharePoint (which is underpinned by SQL Server anyway), thin provisioning is not a good idea. Depending on the performance of the storage layer and the data access patterns of the workload, it is possible that the guest will be slow due to drive fragmentation or even run out of storage space (for any number of reasons, including long-running transactions, infrequent transaction log backups, or a growing TempDB).

It is therefore a better idea to use thick provisioning of storage for specialized workloads. That way the guest is guaranteed the storage it is promised by the hypervisor, and is one less thing to worry about when SQL Server runs out of space at 3 AM on a Sunday morning.

When processors are no longer processors

Virtualizing CPUs is challenging because the CPU works by having a certain number of clock cycles per second (which we looked at earlier in this chapter). For logical processors (this refers to the physical CPU core, plus any logical cores if SMT is turned on), every core shares time slices, or *time slots*, with each VM. Every time the CPU clock ticks over, that time slot might be used by the hypervisor or any one of the guests.

Just as it is not recommended to overprovision RAM and storage for SQL Server, you should not overprovision CPU cores either. If there are four quad-core CPUs in the host (four CPU sockets populated with a quad-core CPU in each socket), this means that there are 16 cores available for use by the VMs (32 when accounting for SMT).

Inside OUT

Virtual CPUs and SMT (Hyper-Threading)

Even though it is possible to assign as many virtual CPUs as there are logical cores, we recommend that you limit the number of vCPUs to the number of physical cores available (in other words, excluding SMT) because the number of execution resources on the CPU itself is limited to the number of physical cores.

Virtual CPU

A virtual CPU (vCPU) maps to a logical core, but in practice, the time slots are shared evenly over each core in the physical CPU. A vCPU will be more powerful than a single core because the load is parallelized across each core.

One of the risks of mixing different types of workloads on a single host is that a business-critical workload like SQL Server might require all the vCPUs to run a large parallelized query. If there are other guests that are using those vCPUs during that specific time slot and the CPU is over-committed, SQL Server's guest will need to wait.

There are certain algorithms in hypervisors that allow vCPUs to cut in line and take over a time slot, which results in a lag for the other guests, causing performance issues. Assume that a file server has two logical processors assigned to it. Further assume that on the same host, a SQL Server has eight logical processors assigned to it. It is possible for the VM with fewer logical processors to "steal" time slots because it has a lower number of logical processors allocated to it.

There are several ways to deal with this, but the easiest solution is to keep like with like. Any guests on the same host should have the same number of virtual processors assigned to them, running similar workloads. That way, the time slots are more evenly distributed, and it becomes easier to troubleshoot processor performance. It might also be practical to reduce the number of vCPUs allocated to a SQL Server instance so that the time slots are better distributed.

CAUTION

A VM running SQL Server might benefit from fewer vCPUs. If too many cores are allocated to the VM, it could cause performance issues due to foreign memory access because SQL Server might be unaware of the underlying NUMA configuration. Remember to size your VM as a multiple of a NUMA node size.

You can find more information on VMware's blog at <https://blogs.vmware.com/vsphere/2012/02/vspherenuma-loadbalancing.html>.

The network is virtual, too

Whereas before, certain hardware devices might be used to perform discrete tasks, such as network interface cards, routers, firewalls, and switches, these tasks can be accomplished exclusively through a software layer, using virtual network devices.

Several VMs might share one or more physical NICs on a physical host, but because it's all virtualized, a VM might have several virtual NICs mapped to that one physical NIC.

This allows a number of things that previously might have been cumbersome and costly to set up. Software developers can now test against myriad configurations for their applications without having to build a physical lab environment using all the different combinations.

With the general trend of consolidating VMs, virtual networking facilitates combining and consolidating network devices and services into the same environment as the guest VMs, lowering the cost of administration and reducing the need to purchase separate hardware. You can replace a virtualized network device almost immediately if something goes wrong, and downtime is vastly reduced.

Summary

SQL Server now runs on Linux, but for all intents and purposes, it's the same as the Windows version, and many of the same rules apply.

Whether running on physical or virtual hardware, databases perform better when they can be cached in memory as much as possible and are backed by persistent storage that is redundant, and has low latency and high random IOPS.

As data theft becomes more prevalent, consider the security of the database itself, the underlying OS and hardware (physical or virtual), the network, and the database backups, too.

When considering strategies for SQL Server HA and DR, design according to the organization's business requirements, in terms of the RPO and RTO. Chapter 11 and Chapter 12 cover this in depth.

This page intentionally left blank



Introducing security principles and protocols.....	292	Auditing with SQL Server and Azure SQL Database.....	319
Encryption in SQL Server.....	302	Securing Azure infrastructure as a service	326
Securing data in motion.....	314		

In recent years, security has become incredibly important to organizations of all sorts, in all industries and government entities, as well. All you need to do is to pay attention to the news to see that the number of leaks and hacks of sensitive information is increasing almost daily.

IT organizations around the world—not just in Europe—should consider the implementation of a European privacy law known as the *General Data Protection Regulation* (GDPR; effective May 25, 2018) as a wake-up call to review how they handle and manage customer information.

Continuing on from Chapter 6, which focused on authorization, this chapter covers features in SQL Server and the underlying operating system (OS) that help you to secure your server and the databases that reside on it.

We begin with what it means to encrypt data. We then move on to understanding how networks transmit and secure data. We conclude with the different features in SQL Server and Microsoft Azure SQL Database that can help you to achieve a secure environment.

Defense in depth means combining different features and strategies to protect your data as much as possible. We show how this strategy can protect your data during regular operations as well as minimize the fallout should your data be stolen.

At the OS level, the defensive strategies for Windows and Linux are similar. But because entire books already have been written on securing these platforms, this chapter will look at OS security only from a high level and focus mainly on securing your data with SQL Server 2017 and Azure SQL Database.

Introducing security principles and protocols

Security is about finding a balance between the value of your data and the cost of protecting it. Ultimately, the organization makes this call, but at least you have the technical tools available to undertake these measures to protect your data.

SQL Server implements a number of security principles through cryptography and other means, which you can use to build up layers of security to protect your environment.

Computer cryptography is implemented through some intense mathematics that use very large prime numbers. However, even if you're wary of math, you need not be afraid in this chapter: we don't delve that deeply into it, although we do cover some terminology that might sound scary.

This section explains various security principles and goes into some detail about encryption. It also covers network protocols and how cryptography works. This will aid your understanding of how SQL Server and network security protects your data.

Securing your environment with defense in depth

Securing a SQL Server environment (or for that matter, a cloud infrastructure, including Azure SQL Database) requires a number of protections that work together to make it difficult for an attacker to get in, snoop around, steal or modify data, and then get out.

Defense in depth is about building layers of protection around your data and environment.

Perimeter security should include logical and physical segmentation; for example, keeping sensitive servers and applications on a separate part of the network, perhaps off-premises in a separate datacenter or in the Azure cloud. You would then want to protect these connections; for example, by using a Virtual Private Network (VPN).

You should have a firewall and other network defenses to protect against *external network attacks*. From a physical aspect, don't let just anyone plug a laptop into an unattended network point, or allow them to connect to your corporate wireless network and have access to the production environment.

From within the network, you need to implement *authentication* (who you are) and *authorization* (what you can do), preferably through Active Directory.

NOTE

Integrated authentication with Active Directory is supported on Linux.

On the *servers* themselves, you should ensure that the file system is locked down, that SQL Server permissions are set correctly, and that file shares (if any) are secured, and using the latest sharing protocols.

On the *application* side, you can implement coding practices that protect against things like SQL injection attacks, and you can implement encryption in your database (and backup files).

Inside OUT

What is SQL injection?

One of the most prevalent attack vectors for a database is to manipulate the software application or website to attack the underlying database.

SQL injection is a technique that exploits applications that do not sanitize input data. A carefully crafted Uniform Resource Identifier (URI) in a web application, for example, can manipulate the database in ways that a naïve application developer is not expecting.

If a web application exposes database keys in the Uniform Resource Locator (URL), for example, an industrious person could carefully craft a URL to read protected information from a table by changing the key value. An attacker might be able to access sensitive data or modify the database itself by appending Transact-SQL (T-SQL) commands to the end of a string to perform malicious actions on a table or database.

In a worst-case scenario, a SQL injection attack would take a few seconds, the entire database could be exfiltrated (data removed without your knowledge), and you might hear about it only when your organization is blackmailed or sensitive data is leaked.

You can avoid SQL injection easily by ensuring that all data input is escaped, sanitized, and validated. To be very safe, all SQL Server queries should use parameterization.

You can read more about defending against SQL injection attacks on Microsoft Docs at <https://docs.microsoft.com/sql/relational-databases/security/sql-injection>.

The Open Web Application Security Project (OWASP) is also an excellent resource to identify and defend against potential vulnerabilities, including SQL injection. You can visit the OWASP website at <https://www.owasp.org>.

The difference between hashing and encryption

In a security context, data that is converted in a repeatable manner to an unreadable, fixed-length format using a cryptographic algorithm and that *cannot* be converted back to its original form is said to be *hashed*.

Data that is converted to an unreadable form that *can* be converted back to its original form using a *cryptographic key* is said to be *encrypted*.

Cryptographic algorithms can be defeated in certain ways, the most common being *brute-force* and *dictionary* attacks. Let's take a quick look at each one:

- **Brute-force attack.** In a brute-force attack, the attacking code checks every possible combination of a password, passphrase, or encryption key against the hashing or encryption service, until it finally arrives at the correct value. Depending on the type of algorithm and the length of the password, passphrase, or key, this can take a few milliseconds, to as long as millions of years (yes, you read that correctly).
- **Dictionary attack.** A dictionary attack is a lot faster to perform, so a malicious actor would attempt this first. Dictionary attacks take a list of words from a dictionary (which can include common words, passwords, and phrases) and use these against the hashing or encryption service. Dictionary attacks take advantage of the fact that human beings are bad at remembering passwords and tend to use common words.

As computers become more powerful and parallelized, the length of time to run a brute-force attack continues to decrease. Countermeasures do exist to protect against some of these attacks, and some encryption systems cannot be defeated by a brute-force attack. These countermeasures are beyond the scope of this book, but it is safe to say that sufficiently complex algorithms and long encryption keys will take several years to compromise.

Hashing

A *cryptographic hash function* (an algorithm) takes *variable-length data* (usually a password) and applies a mathematical formula to convert it to a fixed size, or *hash value*.

This is the recommended method of securing passwords. When a password has been hashed correctly, it cannot be decrypted into its original form. Used with a random *salt* (a random string applied along with the hash function), this results in passwords that are impossible to reconstruct, even if the same password is used by different people.

To validate a password, it must be hashed using the same hash function again, with the same salt, and compared against the stored hash value.

Because hash values have a fixed size (the length depends on the algorithm used), there is a possibility that two sets of data (two different passwords) can result in the same hash value. This

is called a *hash collision*, and it is more likely to occur with shorter hash value lengths. This is why longer hashes are better.

NOTE

Make sure that you use passwords that are at least 15 characters in length and, preferably, more than 20 characters. If you use a password manager, you don't need to memorize passwords, and brute-force attacks take exponentially longer for each additional character you choose. Don't be shy about using phrases or sentences either. The password length matters more than its complexity.

Inside OUT

Why should I use a salt, and what is a rainbow table?

If you don't use a random salt, the same hash value will be created each time the hash function is applied against a particular password. Additionally, if more than one person uses the same password, the same hash value will be repeated.

Imagine that a malicious actor has a list of the most commonly used passwords and knows which hash function you used to hash the passwords in your database. This person could build a catalog of possible hash values for each password in that list. This catalog is called a *rainbow table*.

It becomes very simple to just look up the hash values in your database against the rainbow table and deduce which password was used. Thus, you should always use a random salt when hashing passwords in your database. Rainbow tables become all but useless in this case.

Encryption

Data encryption is the process of converting human-readable data, or *plain text*, into an encrypted form by applying a cryptographic algorithm called a key (the *cipher*) to the data. This process makes the encrypted data (the *ciphertext*) unreadable without the appropriate key to unlock it. Encryption facilitates both the secure transmission and storage of data.

Over the years, many ciphers have been created and subsequently defeated (*cracked*) because those algorithms were considered weak. In many cases, this is because both CPUs and Graphics Processor Units (GPUs) have become faster and more powerful, reducing the length of time it takes to perform brute-force and other attacks. In other cases, the implementation of the cryptographic function was flawed, and attacks on the implementation itself have been successful.

Inside OUT

Why are GPUs used for cracking passwords?

A GPU is designed to process identical instructions (but not necessarily the same data) in parallel across hundreds or thousands of cores, ostensibly for rendering images on a display many times per second.

This coincides with the type of work required to crack passwords through brute force, because those thousands of cores can each perform a single arithmetic operation per clock cycle through a method called *pipelining*.

Because GPUs can operate at billions of cycles per second (GHz), this results in hundreds of millions of hashes per second. Without a salt, many password hashes can be cracked in a few milliseconds, regardless of the algorithm used.

A primer on protocols and transmitting data

Accessing data from an Azure SQL database or SQL Server database involves the transmission of data over a network interface, which you need to do in a secure manner. A *protocol* is a set of instructions for transmitting that information over a specific network port.

A *Transmission Control Protocol (TCP) port* is one of 65,535 possible connections to a networked device; in this case, the device is a server running Windows or Linux. It is always associated with an IP address and a protocol.

Official and unofficial standards over the years have resulted in a set of commonly used ports. For instance, TCP ports 1433 and 1434 are reserved for SQL Server, whereas TCP ports 80 and 443 are reserved for HTTP and HTTPS, respectively. TCP port 22 is reserved for Secure Shell (SSH), User Datagram Protocol (UDP) port 53 is used for Domain Name Services (DNS), and so on.

The internet protocol suite

To discuss security on a network, you need to understand cryptographic protocols. To discuss the network itself, you need to discuss the biggest network of them all: the internet.

The internet is a network of networks (it literally means “between networks”) which transmits data using a suite of protocols, including TCP, which sits on top of *Internet Protocol (IP)*. TCP/IP is the most common network protocol stack in use today. Most of the services on the internet, as well as local networks, rely on TCP/IP.

NOTE

The full internet protocol suite comprises TCP, IP, Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP), UDP, and Internet Group Management Protocol (IGMP). All of these are required to implement the full TCP/IP stack.

IP is a connectionless protocol, meaning that each individual unit of transfer, also known as a *network packet* or *datagram*, contains the data itself—the *payload*—and a *header* that indicates where it came from and where it needs to go (the routing information).

IP network packets are delivered using a “best effort” model, meaning that they might be delivered out of order, with no delivery guarantee at all. This low overhead makes the protocol fast and allows packets to be sent to several recipients at once (*multicast* or *broadcast*).

TCP provides the necessary instructions for reliability, sequencing (the order of packets), and data integrity. If a packet is not received by the recipient, or a packet is received out of order, TCP can resubmit the data again, using IP as its delivery mechanism.

Versions of IP in use today Version 4 of the Internet Protocol (IPv4) has a 32-bit address space, which provides nearly 4.3 billion addresses (2^{32} , or approximately 4.3×10^9). Unfortunately, when this version was first proposed in September 1981, very few people predicted that the internet would be as large and important as it is today. With billions of humans online, and billions of devices connected, the available IPv4 address space is all but depleted.

- You can read the Internet Protocol, Version 4 Specification, known as Internet Engineering Task Force Request For Comments #791, at <https://tools.ietf.org/html/rfc791>.

Tricks like Network Address Translation (NAT), which uses private IP addresses behind a router with a single valid public IP address representing that entire network, have held off the depletion over the years, but time and address space has run out.

Version 6 of the Internet Protocol (IPv6), has an address space of 128 bits which provides more than 340 undecillion addresses (2^{128} , or approximately 3.4×10^{38}). This number is so staggeringly huge that, even with networks and devices being added every minute, including the upward trend of the Internet of Things, each of these devices can have its own unique address on the internet, without ever running out of addresses.

- You can read the Internet Protocol, Version 6 Specification, known as Internet Engineering Task Force Request For Comments #8200, at <https://tools.ietf.org/html/rfc8200>.

Inside OUT

What is the Internet of Things?

Until recently, computing devices such as servers, desktop computers, laptops, and mobile devices have been the only devices connected to the internet.

Today, a huge variety of objects embedded with electronics are finding their way online, including coffee machines, security cameras, home automation systems, vehicle trackers, heart monitors, industrial measurement devices, and many, many more.

Ignoring the fact that many of these devices should not have publicly accessible internet addresses in the first place, the growth trend is exponential, and IPv6 is making this massive growth possible.

Cloud platforms such as Azure have services dedicated to managing the communication and data requirements of these devices, including an Azure SQL database.

Making sense of an IP address An IP address is displayed in a human-readable notation but is binary under the hood:

- **IPv4.** The address is broken up into four subclasses of decimal numbers, each subclass ranging from 0 to 255, and separated by a decimal point. For example, 52.178.167.109 is a valid IPv4 address.
- **IPv6.** The address is broken up into eight subclasses of hexadecimal numerals, each subclass being four digits wide, and separated by a colon. If a subclass contains all zeroes, it can be omitted. For example, 2001:d74f:e211:9840:0000:0000:0000:0000 is a valid IPv6 address that can be simplified to 2001:d74f:e211:9840:: with the zeroes omitted (note the double-colon at the end to indicate the omission).

Adoption of IPv6 across the internet is taking decades, so a hybrid solution is currently in place by which IPv4 and IPv6 traffic is shared across IPv6 and IPv4 devices, respectively. If that doesn't sound like enough of a headache, let's add routing into the mix.

Finding your way around the internet

Routing between networks on the internet is performed by the Border Gateway Protocol (BGP), which sits on top of TCP/IP.

BGP is necessary because there is no map of the internet. Devices and entire networks appear and disappear all the time. BGP routes billions of network packets through millions of routers based on a best guess scenario. Packets are routed based on trust: routers provide information to one another about the networks they control, and BGP implicitly trusts that information.

BGP is thus not secure, because it was designed solely to fix the scalability of the internet, which was (and still is) growing exponentially. It was a “quick fix” that became part of the fabric of the infrastructure long before security was a concern.

Efforts to secure BGP have been slow. It is therefore critical to assume that your own internet traffic will be hijacked at some point. If this happens, proper cryptography can prevent third parties from reading your data.

A brief overview of the World Wide Web

A lot of people conflate the World Wide Web (the web) with the internet, but the web is a single component of the greater internet, along with email (and other services that have seemingly faded into obscurity but are still in use today, such as File Transfer Protocol and Voice over IP).

NOTE

Based on publicly available information, Microsoft processes around 500 billion emails per month through its various services, including Microsoft Office 365 and Outlook Mail (the web version).

The web uses the Hypertext Transport Protocol (HTTP), which sits on top of TCP/IP. A web server provides mixed media content (text, graphics, video, and other media) in Hypertext Markup Language (HTML) format, which is transmitted using HTTP and then interpreted and rendered by a web browser.

The web grew quickly for two reasons. First, the internet became commercialized after originally being an academic and military project for several decades. The web itself then became wildly popular because of the introduction of the first graphical web browser, NCSA Mosaic, in the 1990s. The spiritual successors to Mosaic were Netscape Navigator and Microsoft Internet Explorer, during a period of internet history known as the “browser wars.”

- You can learn more about the commercial beginnings of the web and the so-called “Internet Era,” by listening to the Internet History Podcast, available at <http://www.internethistorypodcast.com>.

Modern web browsers include Microsoft Edge, Google Chrome, Mozilla Firefox, and Apple Safari.

NOTE

The modern web browser is hugely complex, doing a lot more than rendering HTML, but for the purposes of this discussion and in the interest of brevity, we gloss over those extras.

How does protocol encryption fit into this?

The explosive adoption of the web in the 1990s created the need for secure transactions as public-facing organizations began to transition their sales online into electronic commerce, or *e-commerce*, ventures. Consumers wanted to use their credit cards safely and securely so that they could shop and purchase goods without leaving the comfort of their homes.

Remember that the internet is built on the Internet Protocol, which is stateless and has routing information in the header of every single packet. This means that anyone can place a hardware device (or software) in the packet stream, do something with the packet, and then pass it on (modified or not) to the destination, without the sender or recipient having any knowledge of this interaction. Because this is a fundamental building block of a packet-switching network, it's very difficult to secure properly.

As we discussed earlier, encryption transforms data into an unreadable format. Now, if someone connected to the same network were to intercept encrypted packets, that person couldn't see what you're doing. The payload of each packet would appear garbled and unreadable, unless this person has the key to decrypt it.

A secure version of HTTP was created by Netscape Communications in 1994, which was dubbed HTTPS (HTTP Secure, or HTTP over Secure Sockets Layer [SSL]). Over the years, the moniker of HTTPS has remained, but it has come to be known as HTTP over Transport Layer Security (TLS) as standards improved.

When we talk about data moving over the network, that usually means TCP/IP is involved, and we need to transmit that data securely.

Symmetric and asymmetric encryption

You can encrypt data in two ways: symmetric and asymmetric. Each has its advantages and disadvantages.

Symmetric encryption (shared secret)

A secret key, which is usually a password, passphrase, or random string of characters, is used to encrypt data with a particular cryptographic algorithm. This secret key is shared between the sender and the recipient, and both parties can encrypt and decrypt all content by using this secret key.

If the key is accidentally leaked to a third party, the encrypted data could be intercepted, decrypted, modified, and reencrypted again, without either the sender or recipient being aware of this. This type of attack is known as a *man-in-the-middle* attack.

Asymmetric encryption (public key)

Also known as public key encryption (PKE). A key-pair is generated, comprising a private key and a public key, and the public key can be widely distributed. The *public key* is used to encrypt data, and the *private key* is used to decrypt that data.

The advantage is that the private key never needs to be shared, which makes this method far more secure because only you can use your private key to decrypt the data. Unfortunately, asymmetric encryption does require a lot more processing power, plus both parties need their own key-pairs.

Inside OUT

What encryption method should I use for SQL Server?

For practical purposes, SQL Server manages the keys internally for both symmetric and asymmetric encryption.

Owing to the much larger overhead of asymmetric encryption, however, you should encrypt any data in SQL Server that you want you protect by using symmetric key encryption.

Using the encryption hierarchy, layers above the data can be protected using passwords or asymmetric keys (we discuss this in the next section).

Digital certificates

Public keys require discoverability, which means that they need to be made publicly available. If a sending party wants to sign a message for the receiving party, the burden is on the sender to locate the recipient's public key in order to sign a message.

For small-scale communications between two private entities, this might be done by sharing their public keys between each other.

For larger-scale communications with many senders and one recipient (such as a web or database server, for example), a certificate authority can provide the public key through a digital certificate, which the recipient (the website or database administrator) can install on the server directly.

This certificate serves as an electronic signature for the recipient, which includes its public key. The authority, known as a Certification Authority, is trusted by both the sender and the recipient, and the sender can verify that the recipient is indeed who it claims to be.

Digital certificates, also known as *Public Key Certificates*, are defined by the X.509 standard. Many protocols use this standard, including TLS and its predecessor, SSL.

- You can read more about how digital certificates and TLS relate to SQL Server and Azure SQL Database later in this chapter.

Certification Authority

A Certification Authority (CA) is an organization or entity that issues digital certificates, which include the name of the owner, the owner's public key, and start and expiration dates.

The certificate is automatically revoked after it expires, and the CA can revoke any certificate before then.

For the certificate to be trusted, the CA itself must be trustworthy. It is the responsibility of the CA to verify the owner's identity so that any certificates issued in that owner's name can be trusted.

In recent months, several CAs have lost their trustworthy status, either because their verification process was flawed or their signing algorithms were weak. Take care when choosing a CA for your digital certificates.

Encryption in SQL Server

Encryption is but one part of securing your environment. SQL Server provides a full encryption hierarchy, starting at the OS layer (including the network stack and file system), working all the way down the levels of the database, through to individual cells in a table.

Figure 7-1 shows this hierarchy.

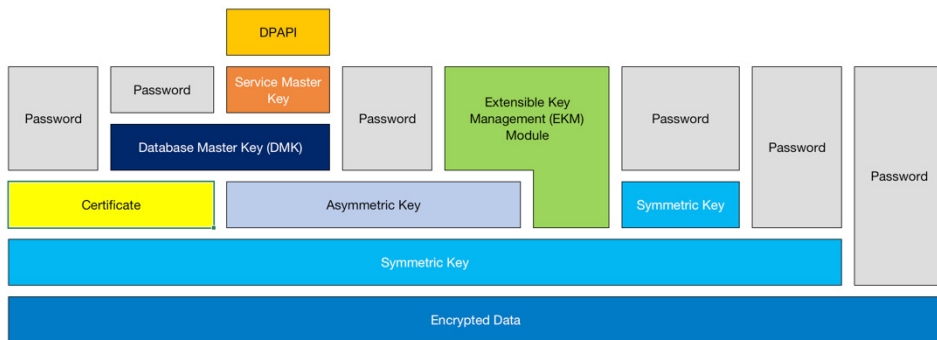


Figure 7-1 The SQL Server encryption hierarchy.

Data protection from the OS

At the top of the hierarchy, protecting everything below it, is the OS. Windows Server provides an Application Programming Interface (API) for system- and user-level processes to take advantage of data protection (encryption) on the file system.

In other words, SQL Server and other applications can make use of this data protection API to have Windows automatically encrypt data on the drive without having to encrypt data through other means.

SQL Server Enterprise edition uses the Data Protection API (DPAPI) for Transparent Data Encryption (TDE).

Inside OUT

How does data protection work for SQL Server on Linux?

The mechanism that Microsoft created for getting SQL Server to run on Linux and Docker containers, is called the Platform Abstraction Layer (PAL). It aligns all code specific to the OS in one place, forming a bridge with the underlying platform.

All APIs, including file system and DPAPIs, are included in the PAL. This makes SQL Server 2017 entirely platform agnostic.

To read more about the PAL, visit the official SQL Server Blog at <https://blogs.technet.microsoft.com/dataplatforminsider/2016/12/16/sql-server-on-linux-how-introduction/>.

The encryption hierarchy in detail

Each layer of the hierarchy protects the layer below it by using a combination of keys (asymmetric and symmetric) and certificates (refer to Figure 7-1).

Each layer in the hierarchy can be accessed by a password at the very least, unless an Extensible Key Management (EKM) module is being used. The EKM module is a standalone device that holds symmetric and asymmetric keys outside of SQL Server.

The Database Master Key (DMK) is protected by the Service Master Key (SMK), and both of these are symmetric keys. The SMK is created when you install SQL Server and is protected by the DPAPI.

If you want to use TDE on your database (see the section “Configuring TDE on a user database” later in this chapter), it requires a symmetric key called the Database Encryption Key (DEK), which is protected by an asymmetric key in the EKM module or by a certificate through the DMK.

This layered approach helps to protect the data from falling into the wrong hands.

There are two considerations when deciding how to secure a SQL Server environment, which you can implement independently.

- **Data at rest.** In the case of TDE, this is decrypting the data on a drive as it is read into the buffer pool, and encrypting the data as it is flushed to a drive from the buffer pool. (You could also encrypt your storage layer independently from SQL Server, but this does not form part of the encryption hierarchy.)
- **Data in motion.** Protecting the data during transmission over a network connection. Any network protocols and APIs involved must support encrypting and decrypting the data as it moves in and out of the buffer pool.

Data is in motion from the moment it is read from or written to the buffer pool in SQL Server or Azure SQL Database. Between the buffer pool and the underlying storage, data is considered to be at rest.

NOTE

TDE encrypts database backup files along with the data and transaction log files. However, the TDE feature is available only with the SQL Server Enterprise edition and Azure SQL Database.

Using EKM modules with SQL Server

Organizations might choose to take advantage of a separate security appliance called a Hardware Security Module (HSM) or EKM device to generate, manage, and store encryption keys for the network infrastructure outside of a SQL Server environment.

SQL Server can make use of these keys for internal use. The HSM/EKM device can be a hardware appliance, a USB device, a smart card, or even software, as long as it implements the Microsoft Cryptographic Application Programming Interface (MCAP) provider.

EKM is an advanced SQL Server setting and is turned off by default. To use the key or keys from an HSM/EKM device, you need to turn on EKM by using the `sp_execute 'EKM provider enabled'` command with the appropriate parameter. Then, the device must be registered as an EKM module for use by SQL Server.

After the HSM/EKM device creates a key for use by SQL Server (for TDE, for instance), the device exports it securely into SQL Server via the MCAP) provider.

The module might support different types of authentication (Basic or Other), but only one of these types can be registered with SQL Server for that provider.

If the module supports Basic authentication (a user name and password combination), SQL Server uses a credential to provide transparent authentication to the module.

Inside OUT

What is a credential?

In SQL Server, a credential is a record of authentication information that the Database Engine uses to connect to external resources.

These credentials provide security details for processes to impersonate Windows users on a network, though they can also be used to connect to other services like Azure Blob Storage and, of course, an HSM/EKM device.

Credentials that will be used by all databases can be created in the master database by using the `CREATE CREDENTIAL` command, or per individual database using the `CREATE DATABASE SCOPED CREDENTIAL` command.

Chapter 6 contains more information on logins, and Chapter 13 goes into more detail about credentials.

- To read more about EKM in SQL Server, go to <https://docs.microsoft.com/sql/relational-databases/security/encryption/extensible-key-management-ekm>.

Cloud security with Azure Key Vault

You can use Azure Key Vault in addition to, or as a drop-in replacement of, a traditional HSM/EKM device. SQL Server can use Key Vault on-premises or running in a VM in the cloud.

Key Vault is implemented as an EKM provider inside SQL Server, using the SQL Server Connector (a standalone Windows application) as a bridge between Key Vault and the SQL Server instance.

To make use of Key Vault, you must create the vault, along with a valid Azure Active Directory (Azure AD) first.

Begin by registering the SQL Server service principal name in Azure AD. After the service principal name is registered, you can install the SQL Server Connector and turn on EKM in SQL Server.

- You can read more about service principal names and Kerberos in Chapter 2.

You must then create a login that SQL Server will use for accessing Key Vault, and then map that login to a new credential that contains the Key Vault authentication information.

- A step-by-step guide for this process is available on Microsoft Docs at <https://docs.microsoft.com/sql/relational-databases/security/encryption/setup-steps-for-extensible-key-management-using-the-azure-key-vault>.

Master keys in the encryption hierarchy

Since SQL Server 2012, both the SMK and DMK are symmetric keys encrypted using the Advanced Encryption Standard (AES) cryptographic algorithm. AES is faster and more secure than Triple Data Encryption Standard (3DES), which was used in SQL Server prior to 2012.

Note, however, that when you upgrade from an older version of SQL Server—those that were encrypted using 3DES—you must regenerate both the SMK and DMK to upgrade them to AES.

The SMK

The SMK is at the top of the encryption hierarchy in SQL Server. It is automatically generated the first time the SQL Server instance starts, and it is encrypted by the DPAPI in combination with the local machine key (which itself is created when Windows Server is installed). The key is based on the Windows credentials of the SQL Server service account and the computer credentials. (On Linux, the local machine key is likely embedded in the PAL when SQL Server is installed.)

Inside OUT

What is the difference between DES, 3DES, and AES?

Data Encryption Standard (DES) was a symmetric key algorithm developed in the 1970s, with a key length of 56 bits (2^{56} possible combinations). It has been considered cryptographically broken since 1998. In 2012 it was possible to recover a DES key in less than 24 hours if both a plain-text and cipher-text pair were known.

Its successor, 3DES, applies the DES algorithm three times (each time with a different DES key) to each block of data being encrypted. However, with current consumer hardware, the entire 3DES keyspace can be searched, making it cryptographically weak.

AES (Advanced Encryption Standard) uses keys that are 128, 192, or 256 bits in length. Longer keys are much more difficult to crack using brute-force methods, so AES is considered safe for the foreseeable future. It also happens to be much faster than 3DES.

If you need to restore or regenerate an SMK, you first must decrypt the entire SQL Server encryption hierarchy, which is a resource-intensive operation. You should perform this activity only in a scheduled maintenance window. If the key has been compromised, however, you shouldn't wait for that maintenance window.

CAUTION

It is essential that you back up the SMK to a file and then copy it securely to an off-premises location. Losing this key will result in total data loss if you need to recover a database or environment.

To back up the SMK, you can use the T-SQL script shown that follows, but be sure to choose a randomly generated password. The password will be required for restoring or regenerating the key at a later stage. Keep the password separate from the SMK backup file so that they cannot be used together if your secure backup location is compromised. Ensure that the folder on the drive is adequately secured. After you back up the key, transfer and store it securely in an off-premises location.

```
BACKUP SERVICE MASTER KEY TO FILE = 'c:\SecureLocation\service_master_key'  
    ENCRYPTION BY PASSWORD = '<UseAReallyStrongPassword>';  
GO
```

The DMK

(Refer back to Figure 7-1 to see how the DMK is protected by the SMK.)

The DMK is used to protect asymmetric keys and private keys for digital certificates stored in the database. A copy of the DMK is stored in the database for which it is used as well as in the master database. The copy is automatically updated by default if the DMK changes. This allows SQL Server to automatically decrypt information as required. A DMK is required for each user database that will make use of TDE.

CAUTION

Don't forget to back up the DMK to a file, as well, and copy it securely to an off-premises location.

It is considered a security best practice to regenerate the DMK periodically to protect the server from brute-force attacks. The idea is that it will take longer for a brute-force attack to break the key than the length of time for which the key is in use.

For example, suppose that you encrypt your database with a DMK in January of this year. In July, you regenerate the DMK, which will cause all keys for digital certificates to be reencrypted with the new key. If anyone had begun a brute-force attack on data encrypted with the previous DMK, all results from that attack will be rendered useless by the new DMK.

You can back up the DMK by using the T-SQL script that follows. The same rules apply as with backing up the SMK (choose a random password, store the file off-premises, and keep the password and backup file separately). This script assumes that the master key exists.

```
USE WideWorldImporters;
GO
BACKUP MASTER KEY TO FILE = 'c:\SecureLocation\wwi_database_master_key'
    ENCRYPTION BY PASSWORD = '<UseAReallyStrongPassword>';
GO
```

- You can read more about the SMK and DMK on Microsoft Docs at <https://docs.microsoft.com/sql/relational-databases/security/encryption/sql-server-and-database-encryption-keys-database-engine>.

Encrypting data by using TDE

Continuing with our defense-in-depth discussion, an additional way to protect your environment is to encrypt data at rest, namely the database files (and when TDE is turned on, all backups of that database).

There are third-party providers, including storage vendors, that provide excellent on-disk encryption for your Direct-Attached Storage (DAS) or Storage-Area Network (SAN), as a file system solution or at the physical storage layer. Provided that your data and backups are localized to this particular solution, and no files are copied to machines that are not encrypted at the file-system level, this might be an acceptable solution for you.

However, if you have the Enterprise edition of SQL Server, you can use TDE, which encrypts the data, transaction log, and backup files at the file-system level by using a DEK.

If someone manages to acquire these files via a backup server, Azure Blob Storage archive, or by gaining access to your production environment, that person will not be able to simply attach the files or restore the database without the DEK.

The DEK is a symmetric key (shared secret) that is secured by a certificate stored in the master database. If using HSM/EKM or Key Vault, the DEK is protected by an asymmetric key in the EKM module, instead. The DEK is stored in the boot record of the protected database (page 0 of file 1) so that it is easily available during the recovery process.

NOTE

TDE is invisible to any applications that use it. No changes are required in those applications to take advantage of TDE for the database.

In the data file, TDE operates at the page level, because all data files are stored as 8-KB pages. Before being flushed from the buffer pool, the contents of the page are encrypted, the checksum is calculated, and then the page is written to the drive. When reading data, the 8-KB page is read from the drive, decrypted, and then the contents are placed into the buffer pool.

NOTE

Even though encryption might to some degree increase the physical size of the data it is protecting, the size and structure of data pages is not affected. Instead, the number of pages in the data file might increase.

For log files, the contents of the log cache are also encrypted before writing to and reading from the drive.

- To read more about checkpoint operations and active virtual log files (VLFs) in the transaction log, refer to Chapter 3.

Backup files are simply the contents of the data file, plus enough transaction log records to ensure that the database restore is consistent (redo and undo records of active transactions when the backup is taken). In other words, the contents of new backup files are encrypted by default after TDE is turned on.

Configuring TDE on a user database

To use TDE on SQL Server Enterprise edition, you need to create a DMK if you don't already have one.

Verify that it is safely backed up and securely stored off-premises. If you have never backed up the DMK, you will be warned by the Database Engine after using it that it has not yet been backed up. If you don't know where that backup is, back it up again. This is a crucial detail to using TDE (or any encryption technology).

Next, you will create a digital certificate or use one that you have acquired from a CA. In the next example, the certificate is created on the server directly.

Then, you create the DEK, which is signed by the certificate and encrypted using a cryptographic algorithm of your choice.

Although you do have a choice of algorithm, we recommend AES over 3DES for performance and security reasons, and you have a choice of three AES key sizes: 128, 192, or 256 bits. Remember that larger keys are more secure but will add additional overhead when encrypting data. If you plan to rotate your keys every few months, you can safely use 128-bit AES encryption because no brute-force attack (using current computing power) should be able to attack a 128-bit key in the months between key rotations.

After you create the DEK, you turn on encryption on the database. The command completes immediately, but the process will take place in the background because each page in the database will need to be read into the buffer pool, encrypted, and flushed to the drive.

CAUTION

Turning on TDE on a user database will automatically turn on TDE for TempDB, as well, if it is not already on. This can add overhead that adversely affects performance for unencrypted databases that make use of TempDB. If you want to turn off TDE on TempDB, all user databases must have it turned off first.

The script that follows provides a summary of the steps to turn on TDE:

```
USE master;
GO
-- Remember to back up this Database Master Key once it is created
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<UseAReallyStrongPassword>';
GO
CREATE CERTIFICATE WideWorldServerCert WITH SUBJECT = 'WWI DEK Certificate';
GO
USE WideWorldImporters;
GO
CREATE DATABASE ENCRYPTION KEY
    WITH ALGORITHM = AES_128
    ENCRYPTION BY SERVER CERTIFICATE WideWorldServerCert;
GO
ALTER DATABASE WideWorldImporters SET ENCRYPTION ON;
GO
```

Verifying whether TDE is turned on for a database

To determine which databases are encrypted with TDE, you can issue the following command:

```
SELECT name, is_encrypted FROM sys.databases;
```

If a user database is encrypted, the `is_encrypted` column value for that database will be set to 1. TempDB will also show a value of 1 in this column.

Protecting sensitive columns with Always Encrypted

Although TDE is really useful for encrypting the entire database at the file-system level, it doesn't prevent database administrators and other users from having access to sensitive information within the database.

The first rule of storing sensitive data is that you should avoid storing it altogether when possible. Credit card information makes sense in a banking system, but not in a sales database, for instance.

NOTE

Many third-party systems can encrypt your data securely, as well, but are beyond the scope of this chapter. It is good to keep in mind that there is a small but inherent risk in storing encryption keys with data, as SQL Server does. Your organization must balance that risk against the ease of managing and maintaining those keys.

If you must store sensitive data, Always Encrypted protects how data is *viewed* at the column level. It works with applications that use particular connection types (*client drivers*; see the next section) to interact with SQL Server. These client drivers are protected by a digital certificate so that only specific applications can view the protected data.

Always Encrypted was introduced in SQL Server 2016 and has been available on all editions since SQL Server 2016 Service Pack 1. To use this feature, the database makes use of two types of keys: *column encryption keys* and *column master keys* (discussed shortly).

The encryption used by Always Encrypted is one of two types:

- **Deterministic encryption.** This is the same as generating a hash value without a salt. The same encrypted value will always be generated for a given plain-text value. This is useful for joins, indexes, searching, and grouping, but it makes it possible for people to guess what the hash values represent.
- **Randomized encryption.** This is the same as generating a hash value with a salt. No two of the same plain-text values will generate the same encrypted value. Although this does improve security of the data, it does not permit joins, indexes, searching, and grouping for those encrypted columns.

For values that are not expected to participate in joins or searches, you can safely use randomized encryption. Choose deterministic encryption for values like social security numbers and other government-issued values because it helps for searching and grouping.

Because the whole intention of Always Encrypted is to prevent unauthorized persons from viewing data (including database administrators), you should generate the keys elsewhere and store them in a trusted key store (in the the operating system's key store for the database server and the application server, or an EKM module such as Key Vault), away from the database server. The person who generates the keys should not be the same person who is administering the database.

Client application providers that support Always Encrypted

The following providers currently support Always Encrypted:

- .NET Framework 4.6 or higher

- Microsoft JDBC Driver 6.0 or higher
- ODBC Driver 13.1 for SQL Server or higher

It is anticipated that .NET Standard will be supported in the near future.

The connection between the Database Engine and application is made by using a client-side encrypted connection. Each provider has its own appropriate method to control this setting:

- **.NET Framework.** Set the Column Encryption Setting in the connection string to `enabled`, or configure the `SqlConnectionStringBuilder.ColumnEncryptionSetting` property to `SqlConnectionColumnEncryptionSetting.Enabled`.
- **JDBC.** Set the `columnEncryptionSetting` to `Enabled` in the connection string, or configure the `SQLServerDataSource()` object with the `setColumnEncryptionSetting("Enabled")` property.
- **ODBC.** Set the `ColumnEncryption` connection string keyword to `Enabled`, use the `SQL_COPT_SS_COLUMN_ENCRYPTION` preconnection attribute, or through the Data Source Name (DSN) using the `SQL_COLUMN_ENCRYPTION_ENABLE` setting.

Additionally, the application must have the `VIEW ANY COLUMN MASTER KEY DEFINITION` and `VIEW ANY COLUMN ENCRYPTION KEY DEFINITION` database permissions in order to view the Column Master Key and Column Encryption Key.

The Column Master Key and Column Encryption Key

The Column Master Key (CMK) protects one or more Column Encryption Keys (CEK).

The CEK is encrypted using AES encryption and is used to encrypt the actual column data. You can use the same CEK to encrypt multiple columns, or you can create a CEK for each column that needs to be encrypted.

Metadata about the keys (but not the keys themselves) is stored in the database's system catalog views:

- `sys.column_master_keys`
- `sys.column_encryption_keys`

This metadata includes the *type* of encryption and *location* of the keys, plus their *encrypted values*. Even if a database is compromised, the data in the protected columns cannot be read without access to the secure key store.

- To read more about considerations for key management, go to <https://docs.microsoft.com/sql/relational-databases/security/encryption/overview-of-key-management-for-always-encrypted>.

Using the Always Encrypted Wizard

The easiest way to configure Always Encrypted is by using the Always Encrypted Wizard in SQL Server Management Studio. As noted previously, you need to have the following permissions before you begin:

- VIEW ANY COLUMN MASTER KEY DEFINITION
- VIEW ANY COLUMN ENCRYPTION KEY

If you plan on creating new keys, you also need the following permissions:

- ALTER ANY COLUMN MASTER KEY
- ALTER ANY COLUMN ENCRYPTION KEY

In SQL Server Management Studio, in Object Explorer, right-click the name of the database that you want to configure. In the Always Encrypted Wizard, in the pane on the left, click Tasks, and then, on the Tasks page, click Encrypt Columns.

On the Column Selection page, choose the a column in a table that you want to encrypt, and then select the encryption type (deterministic or randomized). If you want to decrypt a previously encrypted column, you can choose Plaintext here.

On the Master Key Configuration page, you can create a new key by using the local OS certificate store or by using a centralized store like Key Vault or an HSM/EKM device. If you already have a CMK in your database, you can use it, instead.

NOTE

Memory-optimized and temporal tables are not supported by this wizard, but you can still encrypt them by using Always Encrypted.

- You can read more about Always Encrypted on Microsoft Docs at <https://docs.microsoft.com/sql/relational-databases/security/encryption/always-encrypted-database-engine>.

Securing data in motion

Data in motion is data that SQL Server provides over a network interface. Protecting data in motion requires a number of considerations, from the perimeter security, to cryptographic protocols for the communication itself, and the authorization of the application or process accessing the data.

This section first goes into more detail about network encryption with TLS, which operates on the network itself, and then dives into row-level security and data masking. The latter features do not make use of encryption, but form part of your defense-in-depth strategy to protect data in motion from prying eyes.

Unlike Always Encrypted, which encrypts data at rest and only decrypts it when being read, row-level security and data masking hide or show data depending on who's asking for it and how it is queried.

Securing network traffic with TLS

We touched briefly on TLS earlier in this chapter in the discussion about TCP/IP, but we did not go into much detail. Now, it's time we look at it more closely.

So, what is TLS, and how does it affect SQL Server and Azure SQL Database? The name is revealing. TLS is a security layer on top of a transport layer, or in technical terms, a *cryptographic protocol*. As we pointed out at the beginning of this chapter, most networks use the TCP/IP protocol stack. In other words, TLS is designed to secure the traffic on TCP/IP-based networks.

How does TLS work?

With TLS protection, before two parties can exchange information, they need to mutually agree on the encryption key and the cryptographic algorithm to use, which is called a *key exchange* or *handshake*. TLS works with both symmetric and asymmetric encryption, which means that the encryption key could be a shared secret or a public key (usually with a certificate).

After the key exchange is done, the handshake is complete, and a secured communication channel allows traffic between the two parties to flow. This is how data in motion is protected from external attacks.

NOTE

Remember that longer keys mean better security. Public keys of 1,024 bits (128 bytes) are considered short these days, so some organizations now prefer 2,048-bit, or even 4,096-bit public key certificates for TLS.

A brief history of TLS

Just as earlier cryptographic protocols have been defeated or considered weak enough that they will eventually be defeated, so too have SSL and its successor, TLS, had their challenges:

- The prohibition of SSL 2.0 is covered at <https://tools.ietf.org/html/rfc6176>.
- Known attacks on TLS are available at <https://tools.ietf.org/html/rfc7457>.

TLS 1.2 was defined in 2008, and is the latest public version. It is vulnerable to certain attacks, like its predecessors, but as long as older encryption algorithms are not used (for instance 3DES, RC4, and IDEA), it is good enough for the moment.

Where possible, you should be using TLS 1.2 everywhere. SQL Server ships with TLS 1.0, 1.1, and 1.2 support out of the box, so you will need to turn off 1.0 and 1.1 at the OS level to ensure that you use TLS 1.2.

- You can see how to turn off older versions of TLS in the Microsoft Knowledge Base article at <https://support.microsoft.com/help/3135244>.

As of this writing, TLS 1.3 is a draft specification.

NOTE

Although we do not recommend 3DES for TLS, you can still use 3DES lower in the SQL Server security hierarchy for securing DEKs because these are protected by the SMK, the DMK, and a Certificate, or entirely by an HSM/EKM module like Key Vault.

Row-level security

Protecting the network itself is good and proper, but this does not protect assets within the network from, for example, curious people snooping on salaries in the HR database. Or, suppose that your database contains information for many customers, and you want only customers to view their own data, without having knowledge of other data in the same tables.

Row-level security performs at the database level to restrict access through a security policy, based on group membership or execution context. It is functionally equivalent to a WHERE clause.

Access to the rows in a table is protected by an inline table-valued function, which is invoked and enforced by the security policy.

The function checks whether the user is allowed to access a particular row, while the security policy attaches this function to the table. So, when you run a query against a table, the security policy applies the predicate function.

There are two types of security policies supported by row-level security, both of which you can apply simultaneously:

- Filter predicates, which limit the data that can be seen
- Block predicates, which limits the actions a user can take on data

Hence, a user might be able to see rows, but cannot insert, update, or delete rows that look like rows they can see. This concept is covered in more detail in the next section.

CAUTION

There is a risk of information leakage if an attacker writes a query with a specially crafted WHERE clause and, for example, a divide-by-zero error, to force an exception if the WHERE condition is true. This is known as a *side-channel attack*. It might be wise to limit the ability of users to run ad hoc queries when using row-level security.

Filtering predicates for read operations

You can silently filter rows that are available through read operations. The application has no knowledge of the other data that is filtered out.

Filter predicates affect all read operations (this list is taken directly from the official documentation at <https://docs.microsoft.com/sql/relational-databases/security/row-level-security>):

- **SELECT.** Cannot view rows that are filtered.
- **DELETE.** Cannot delete rows that are filtered.
- **UPDATE.** Cannot update rows that are filtered. It is possible to update rows that will be subsequently filtered. (The next section covers ways to prevent this.)
- **INSERT.** No effect (inserting is not a read operation). Note, however, that a trigger could cause unexpected side effects in this case.

Blocking predicates for write operations

These predicates block access to write (or *modification*) operations that violate the predicate. Block predicates affect all write operations:

- **AFTER INSERT.** Prevents inserting rows with values that violate the predicate. Also applies to bulk insert operations.
- **AFTER UPDATE.** Prevents updating rows to values that violate the predicate. Does not run if no columns in the predicate were changed.
- **BEFORE UPDATE.** Prevents updating rows that currently violate the predicate.
- **BEFORE DELETE.** Blocks delete operations if the row violates the predicate.

Dynamic data masking

Data masking works on the premise of limiting exposure to data by obfuscation. Without requiring too many changes to the application or database, it is possible to mask *portions* of columns to prevent lower-privilege users from seeing, for example, full credit card numbers and other sensitive information.

The mask is defined in the column definition of the table, using `MASKED WITH (FUNCTION = [type])` syntax (and you can add masking after table creation by using `ALTER COLUMN` syntax).

There are four types of masks that are available:

- **Default.** The column is masked according to the data type (not its default value). Strings will use “XXXX” (fewer if the length is less than four characters); numerics will use a zero value; dates will use midnight on January 1st, 1900; and binary will use a single byte binary equivalent of zero.
 - **Email.** Only the first letter and the trailing domain suffix is not masked; for example, “aXXX@XXXXXXX.com”.
 - **Random.** This replaces a numeric data type with a random value between a range you specify.
 - **Custom String.** Only the first and last letters are not masked. There is a custom padding string in the middle, which you specify.
- You can read more about dynamic data masking, including samples of how to set it up, at <https://docs.microsoft.com/sql/relational-databases/security/dynamic-data-masking>.

Limitations with masking data

Dynamic data masking has some significant limitations. It does not work on Always Encrypted columns, nor `FILESTREAM` or `COLUMN_SET` column types. Additionally, `GROUP BY` and `WHERE` clauses are excluded, as are `INSERT` and `UPDATE` statements. Computed columns are also excluded, but if the computed column depends on a masked column, the computed column inherits that mask and returns masked data. Finally, a masked column cannot be used as a `FULLTEXT` index key.

CAUTION

It is possible to expose masked data with carefully crafted queries. This can be performed by using a brute-force attack or using inference based on the results. If you are using data masking, you should also limit the ability of the user to run ad hoc queries and ensure that their permissions are sound.

Azure SQL Database

All of the security features discussed thus far work equally on SQL Server and Azure SQL Database, namely TDE, Always Encrypted, row-level security and dynamic data masking.

That's great if you're just comparing SQL Server to Azure SQL Database, but there are some features unique to Azure SQL Database that are worth looking at, which we'll do in the next section. But keep in mind that because Azure features and products are always changing, this is only a brief overview.

Azure SQL Database Threat Detection

The risks of having a publicly accessible database in the cloud are numerous. To help protect against attacks, you can activate Threat Detection, which runs 24 hours per day on each of your Azure SQL Database servers (called nodes) for a monthly fee. This service notifies you by email whenever atypical behavior is detected.

Some of the interesting threats include SQL injection attacks and potential vulnerabilities as well as unfamiliar database access patterns, including unfamiliar logins or access from unusual locations. Each notification includes possible causes and recommendations to deal with the event.

Threat Detection ties into the Azure SQL Audit log (discussed in the next section); thus, you can review events in a single place and decide whether each one was expected or malicious.

Although this does not prevent malicious attacks (over and above your existing protections), you are given the necessary tools to mitigate and defend against future events. Given how prevalent attacks like SQL injection are, this feature is very useful in letting you know if that type of event has been detected.

You can turn on Threat Detection through the Azure portal, or through PowerShell.

- To read more on configuring Azure SQL Database Threat Detection with PowerShell, go to <https://docs.microsoft.com/azure/sql-database/scripts/sql-database-auditing-and-threat-detection-powershell>.

Built-in firewall protection

Azure SQL Database is secure by default. All connections to your database environment pass through a firewall. No connections to the database are possible until you add a rule to the firewall to allow access.

To provide access to all databases on an Azure SQL server, you must add a server-level firewall rule through the Azure portal or through PowerShell with your IP address as a range.

- To read more about protecting your Azure SQL Database, see Chapter 5.

Auditing with SQL Server and Azure SQL Database

Auditing is the act of tracking and recording events that occur in the Database Engine.

Since SQL Server 2016 Service Pack 1, the Audit feature is available in all editions, as well as in Azure SQL Database. Chapter 5 covers configuring auditing in Azure SQL Database in depth.

SQL Server Audit

There is a lot going on in the Database Engine. SQL Server Audit uses extended events to give you the ability to track and record those actions at both the instance and database level.

NOTE

Although extended events carry minimal overhead, it is important that you carefully balance auditing against performance impact. Use targeted auditing by only capturing the events that are necessary to fulfil your audit requirements.

- You can read more about extended events in Chapter 13.

Audits are logged to event logs or audit files. An event is initiated and logged every time the audit action is encountered, but for performance reasons, the audit target is written to asynchronously.

The permissions required for SQL Server auditing are complex and varied, owing to the different requirements for reading from and writing to the Windows Event Log, the file system, and SQL Server itself.

Requirements for creating an audit

To keep track of events (called actions), you need to define a collection, or *audit*. The actions you want to track are collected according to an *audit specification*. Recording those actions is done by the *target* (destination).

- **Audit.** The SQL Server audit object is a collection of server actions or database actions (these actions might also be grouped together). Defining an audit creates it in the off state. After it is turned on, the destination receives the data from the audit.
- **Server audit specification.** This audit object defines the actions to collect at the instance level or database level (for all databases on the instance). You can have multiple Server Audits per instance.
- **Database audit specification.** You can monitor audit events and audit action groups. Only one database audit can be created per database per audit. Server-scoped objects must not be monitored in a database audit specification.

- **Target.** You can send audit results to the Windows Security event log, the Windows Application event log, or an audit file on the file system. You must ensure that there is always sufficient space for the target. Keep in mind that the permissions required to read the Windows Application event log are lower than the Windows Security event log, if using the Windows Application event log.

An audit specification can be created only if an audit already exists.

- ▶ To read more about audit action groups and audit actions, go to <https://docs.microsoft.com/sql/relational-databases/security/auditing/sql-server-audit-action-groups-and-actions>.

Inside OUT

What if an audit shuts down the instance or prevents SQL Server from starting?

SQL Server can be shut down by a failure in the audit. You will find an entry in the log saying `MSG_AUDIT_FORCED_SHUTDOWN`. You can start SQL Server in single-user mode using the `-m` option at the command line, which will write an entry to the log saying `MSG_AUDIT_SHUTDOWN_BYPASSED`.

An audit initiation failure also can prevent SQL Server from starting. In this case, you can use the `-f` command-line option to start SQL Server with minimal configuration (which is also single-user mode).

In minimal configuration or single-user mode, you will be able to remove the offending audit that caused the failure.

Creating a server audit in SQL Server Management Studio

Verify that you are connected to the correct instance in SQL Server Management Studio. Then, in Object Explorer, expand the Security folder. Right-click the Audits folder, and then, on the shortcut menu that opens, select New Audit.

In the Create Audit dialog box that opens, configure the settings to your requirements, or you can leave the defaults as is. Just be sure to enter in a valid file path if you select File in the Audit Destination list box. We also recommend that you choose an appropriate name to enter into the Audit Name box (the default name is based on the current date and time).

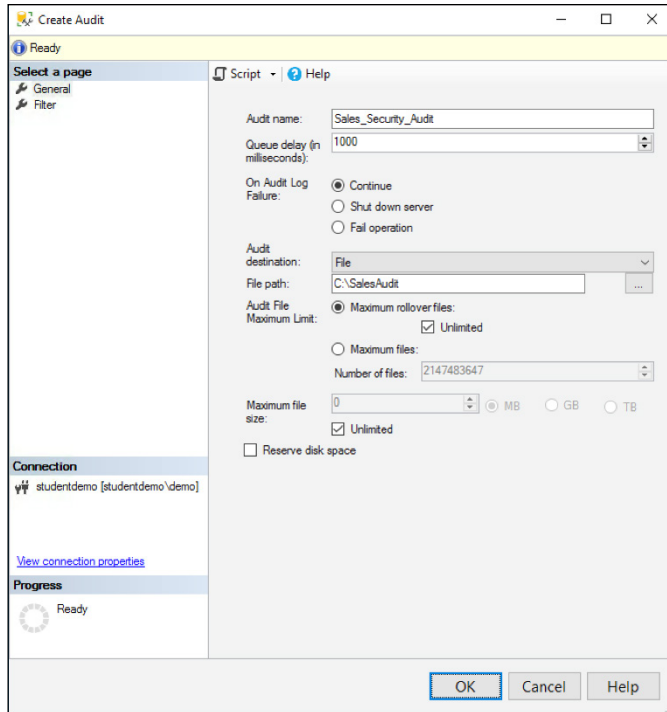


Figure 7-2 Creating an audit in SQL Server Management Studio.

Remember to turn on the audit after it is created. It will appear in the Audit folder, which is within the Security folder in Object Explorer. To do so, right-click the newly created audit, and then, on the shortcut menu, click Enable Audit.

Create a server audit by using T-SQL

The server audit creation process can be quite complex, depending on the destination, file options, audit options, and predicates. As just demonstrated, you can configure a new audit by using SQL Server Management Studio, and then create a script of the settings before clicking OK, which produces a T-SQL script, or you can do it manually.

- To read more about creating a server audit in T-SQL visit <https://docs.microsoft.com/sql/t-sql/statements/create-server-audit-transact-sql>.

To create a server audit in T-SQL, verify that you are connected to the appropriate instance, and then run the code in Listing 7-4. (You'll need to change the audit name and file path accordingly.) Note that the next example also sets the audit state to ON. It is created in the OFF state by default.

This audit will not have any effect until an audit specification and target are also created.

```
USE master;
GO
-- Create the server audit.
CREATE SERVER AUDIT Sales_Security_Audit
    TO FILE (FILEPATH = 'C:\SalesAudit');
GO
-- Enable the server audit.
ALTER SERVER AUDIT Sales_Security_Audit
    WITH (STATE = ON);
GO
```

Create a server audit specification in SQL Server Management Studio

In Object Explorer, expand the Security folder. Right-click the Server Audit Specification folder, and then, on the shortcut menu, click New Server Audit Specification.

In the Create Server Audit Specification dialog box (Figure 7-3), in the Name box, type a name of your choosing for the audit specification. In the Audit list box, select the previously created server audit. If you type a different value in the Audit box, a new audit will be created by that name.

Now you can choose one or more audit actions, or audit action groups.

- A full list of audit actions and audit action groups is available at <https://docs.microsoft.com/sql/relational-databases/security/auditing/sql-server-audit-action-groups-and-actions>.

NOTE

If you have selected an audit group action, you cannot select Object Class, Object Schema, Object Name, and Principal Name, because the group represents multiple actions.

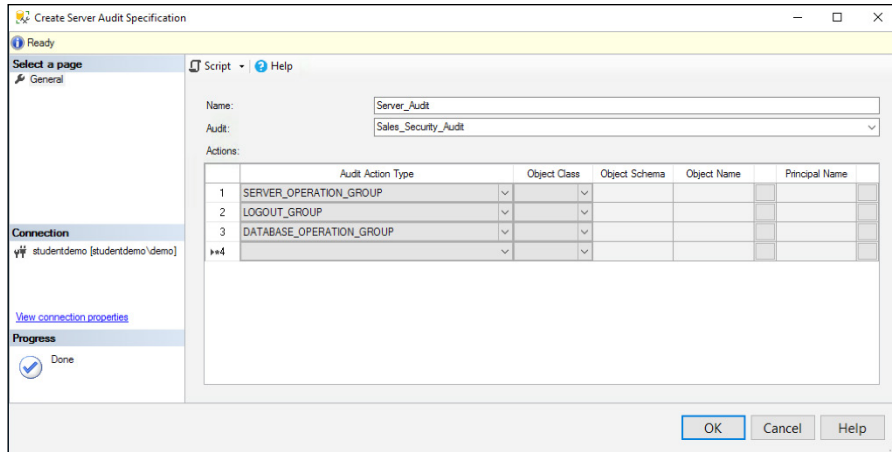


Figure 7-3 Creating a Server Audit Specification in SQL Server Management Studio.

Remember to turn on the server audit specification after you create it, by using the context menu.

Create a server audit specification by using T-SQL

In much the same way as you create the audit itself, you can create a script of the configuration from a dialog box in SQL Server Management Studio, or you can create the specification manually, as shown in the script that follows. Note that the server audit specification refers to a previously created audit.

```
USE [master];
GO
-- Create the server audit specification.
CREATE SERVER AUDIT SPECIFICATION Server_Audit
FOR SERVER AUDIT Sales_Security_Audit
    ADD (SERVER_OPERATION_GROUP),
    ADD (LOGOUT_GROUP),
    ADD (DATABASE_OPERATION_GROUP),
WITH (STATE = ON);
GO
```

Creating a database audit specification in SQL Server Management Studio

As you would expect, the location of the database audit specification is under the database security context.

In Object Explorer, expand the database on which you want to perform auditing, and then expand the Security folder. Right-click the Database Audit Specifications folder, and then, on the shortcut menu, click New Database Audit Specification. Remember again to use the context menu to turn it on.

Figure 7-4 shows an example of capturing SELECT and INSERT operations on the Sales.CustomerTransactions table by the dbo user.

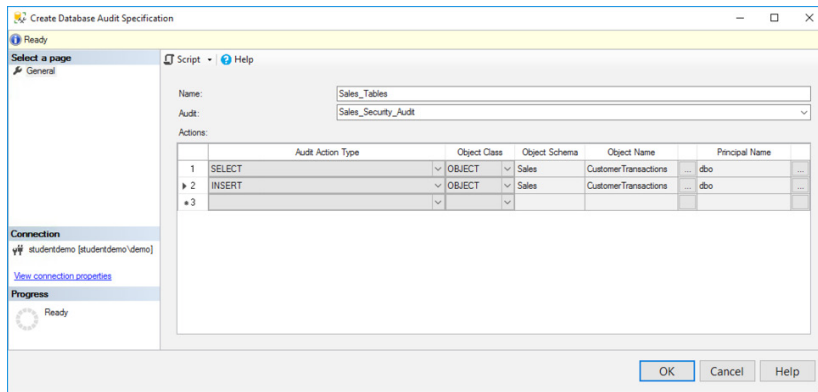


Figure 7-4 Creating a database audit specification in SQL Server Management Studio.

Creating a database audit specification by using T-SQL

Again, verify that you are in the correct database context. Create the database audit specification by referring to the server audit that was previously created, and then specify which database actions you want to monitor, as demonstrated in the next example.

The destination is already specified in the server audit, so as soon as this is turned on, the destination will begin logging the events as expected.

```
USE WideWorldImporters;
GO
-- Create the database audit specification.
CREATE DATABASE AUDIT SPECIFICATION Sales_Tables
    FOR SERVER AUDIT Sales_Security_Audit

    ADD (SELECT, INSERT ON Sales.CustomerTransactions BY dbo)
    WITH (STATE = ON);
GO
```

Viewing an audit log

You can view audit logs either in SQL Server Management Studio or in the Security Log in the Windows Event Viewer. This section describes how to do it by using SQL Server Management Studio.

NOTE

To view any audit logs, you must have CONTROL SERVER permission.

In Object Explorer, expand the Security folder, and then expand the Audits folder. Right-click the audit log that you want to view, and then, on the shortcut menu, select View Audit Logs.

Note that the Event Time is in UTC format. This is to avoid issues regarding time zones and daylight savings.

Figure 7-5 shows two audit events that have been logged. In the first, the audit itself has been changed (it was turned on). The second event is a SELECT statement that was run against the table specified in the database audit specification example presented earlier.

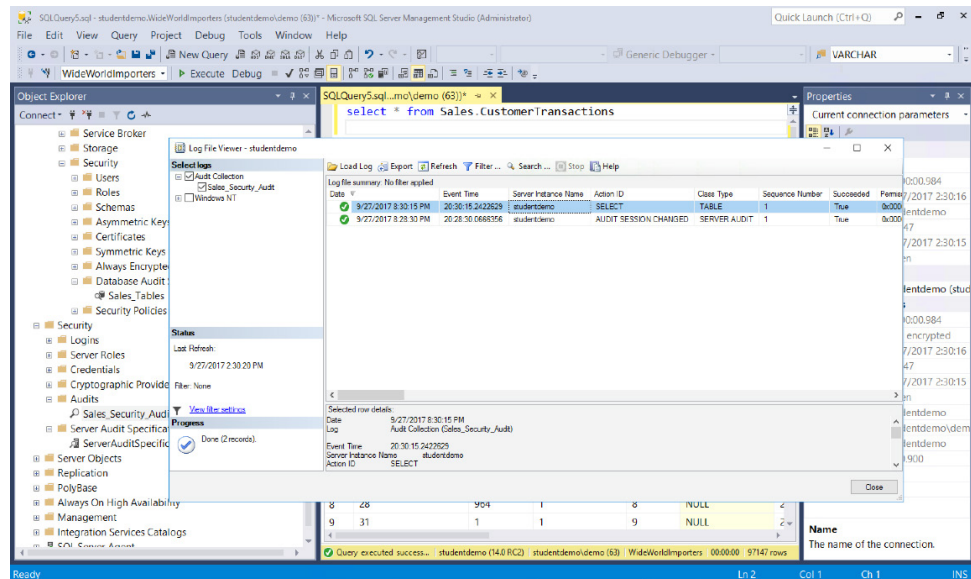


Figure 7-5 File Viewer dialog box for viewing a SQL Server audit.

There are many columns in the audit that you cannot see in Figure 7-5, notable among them are Server Principal ID (SPID), Session Server Principal Name (the logged-in user), and the Statement (the command that was run). The point here being that you can capture a wealth of information.

NOTE

You can also view the audit log in an automated manner by using the built-in T-SQL system function `sys.fn_get_audit_file`, though the data is not formatted the same way as it is through the File Viewer in SQL Server Management Studio. See more at <https://docs.microsoft.com/sql/relational-databases/system-functions/sys-fn-get-audit-file-transact-sql>.

Auditing with Azure SQL Database

With Azure SQL Database auditing, you can track database activity and write it to an audit log in an Azure Blob storage container, in your Azure Storage account (you are charged for storage accordingly).

This helps you to remain compliant with auditing regulations as well as see anomalies (as discussed earlier in the section “Azure SQL Database Threat Detection”) to give you greater insight into your Azure SQL Database environment.

Auditing gives you the ability to retain an audit trail, report on activity in each database, and analyze reports, which includes trend analysis and security-related events. You can define server-level and database-level policies. Server policies automatically cover new and existing databases.

If you turn on server auditing, that policy applies to any databases on the server. Thus, if you also turn on database auditing for a particular database, that database will be audited by both policies. You should avoid this unless retention periods are different or you want to audit for different event types.

- You can read more about Azure SQL Database auditing in Chapter 5.

Securing Azure infrastructure as a service

Infrastructure as a service (IaaS), or SQL Server running on an Azure VM, is secured in much the same way as the on-premises product. Depending on the edition, you can use TDE, Always Encrypted, row-level security, and dynamic data masking.

With Azure IaaS, setting up a VM in a resource group is secure by default. If you want to allow connections from outside of your Azure virtual network, you need to allow not only the connection through the OS firewall (which is on by default in Windows Server), but you also can control connections through a Network Security Group.

In addition to that, you can control access through a network appliance, such as a firewall or NAT device. This provides finer-grained control over the flow of network traffic in your virtual network, which is needed to set up Azure ExpressRoute, for example (Chapter 3 covers this in some detail).

Network Security Group

A Network Security Group (NSG) controls the flow of traffic in and out of the entirety (or part) of an Azure virtual network *subnet*.

Inside OUT

What is a subnet?

A subnet, short for subnetwork, is a logical separation of a larger network into smaller sections, making the network easier to manage and secure.

Subnetting can be vastly complex and is definitely beyond the scope of this book. There are subnet calculators online that you should refer to if you're doing this yourself. Because Azure Virtual Networks make use of subnets, here is a high-level overview.

Subnets are identified by a *network ID*, which is rendered in *network prefix notation* (also known as CIDR, or Classless Interdomain Routing). You will recognize this as a network address in IPv4 format followed by a prefix of /8, /16, or /24, and so on. The lower (*shorter*) the prefix, the more addresses are available.

This is a shorthand for the IP addresses that are available in that subnet, with the network address as the starting value. For example, 192.168.1.0/24 means that there are 256 possible addresses, starting at 192.168.1.1, up to and including 192.168.1.254. All subnets reserve the first address (in this case, 192.168.1.0) for the network identifier, and the last address (in this case, 192.168.1.255) for the broadcast address.

In the Azure classic deployment model, an NSG would provide security for an individual virtual machine. With the Azure Resource Manager deployment model, an NSG can provide security for an entire subnet, which affects all the resources in that subnet (see Figure 7-6). If you require more control, you can associate the NSG with an individual network interface card (NIC), thus restricting traffic further.

NOTE

When creating a VM using the Azure Resource Manager, it will come with at least one virtual NIC, which in turn, you manage through an NSG. This is an important distinction from the classic provider (in which the NSG worked at the VM level) because individual NICs can belong to different NSGs, which provides finer control over the flow of network traffic on individual VMs.

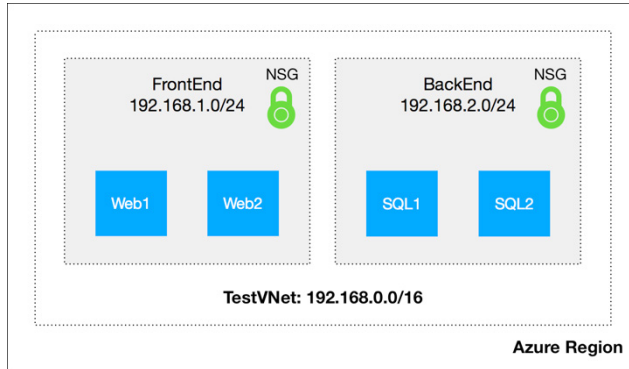


Figure 7-6 A typical virtual network, with each subnet secured by a security group.

As with typical firewalls, the NSG has rules for incoming and outgoing traffic. When a packet hits a port on the virtual network or subnet, the NSG intercepts the packet and checks whether it matches one of the rules. If the packet does not qualify for processing, it is discarded (dropped).

Rules are classified according to source address (or range) and destination address (or range). Depending on the direction of traffic, the source address could refer to inside the network or outside on the public internet.

This becomes cumbersome with more complex networks, so to simplify administration and provide flexibility, you can use service tags to define rules by service name instead of IP address. Storage, SQL and Traffic are currently supported, with more to come in the future.

You can also use default categories, namely VirtualNetwork (the IP range of all addresses in the network), AzureLoadBalancer (the Azure infrastructure load balancer), and Internet (IP addresses outside the range of the Azure Virtual Network).

- You can read more about Azure Virtual Network security and service tags at <https://docs.microsoft.com/azure/virtual-network/security-overview>.

User-defined routes and IP forwarding

As a convenience to Azure customers, all VMs in an Azure Virtual Network are able to communicate with one another by default, irrespective of the subnet in which they reside. This also holds true for virtual networks connected to your on-premises network by a VPN, and for Azure VMs communicating with the public internet (including those running SQL Server).

- You can read more about Virtual Private Networks in Chapters 2 and 3.

In a traditional network, communication across subnets like this requires a gateway to control (route) the traffic. Azure provides these *system routes* for you automatically.

You might decide that this free-for-all communication is against your network policy and that all traffic from your VMs should first be channeled through a network appliance (such as a firewall or NAT device). Virtual appliances are available in the Azure Marketplace at an additional cost, or you could configure a VM yourself to run as a firewall.

A user-defined route with IP forwarding makes this happen. With a user-defined route, you create a subnet for the virtual appliance and force traffic from your existing subnets or VMs through the virtual appliance.

In Microsoft's own words:

"[t]o allow a VM to receive traffic addressed to other destinations, you must enable IP Forwarding for the VM. This is an Azure setting, not a setting in the guest operating system." (<https://docs.microsoft.com/azure/virtual-network/virtual-networks-udr-overview>)

CAUTION

With user-defined routes, you cannot control how traffic *enters* the network from the public internet. They only control how traffic *leaves* a subnet, which means that your virtual appliance must be in its own subnet. If you want to control traffic flow from the public internet as it enters a subnet, use a Network Security Group, instead.

Until you create a routing table (by user-defined route), subnets in your Virtual Network rely on system routes. A user-defined route adds another entry in the routing table, so a technique called Longest Prefix Match (LPM) kicks in to decide which is the better route to take, by selecting the most specific route (the one with the longest prefix). As seen earlier in Figure 7-6, a /24 prefix is longer than a /16 prefix, and a route entry with a higher prefix takes precedence.

If two entries have the same LPM match, the order of precedence is as follows:

- User-defined route
- BGP route
- System route

Remember BGP? It's used for ExpressRoute. As we mentioned in Chapter 3, ExpressRoute is a VPN service by which you can connect your Azure Virtual Network to your on-premises network, without going over the public internet. You can specify BGP routes to direct traffic between your network and the Azure Virtual Network.

Additional security features in Azure networking

There are additional features for improving the management and security of an Azure Virtual Network, as it relates to SQL Server or Azure SQL Database, which are worth discussing here. As of this writing, some of these features are still in preview.

Virtual network service endpoints

Service endpoints make it possible for you to restrict access to certain Azure services that were traditionally open to the public internet so that they are available only to your Azure Virtual Network, as illustrated in Figure 7-7.

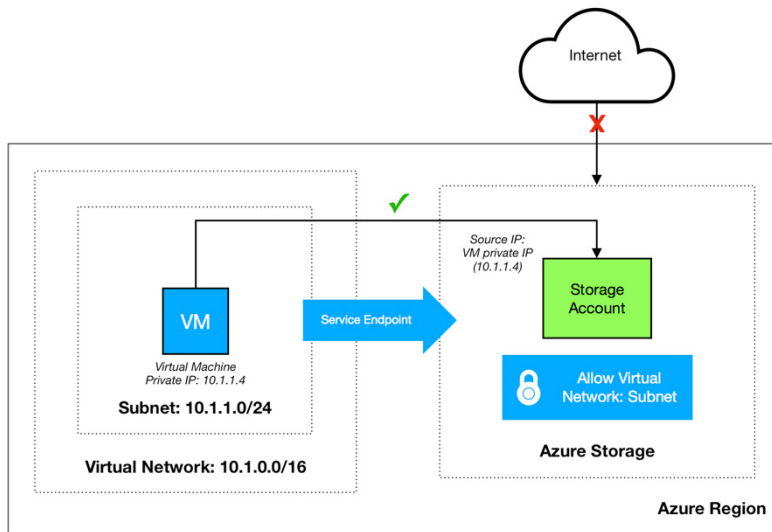


Figure 7-7 A service endpoint protecting an Azure Storage account.

Configurable through the Azure portal (or PowerShell), you can block public internet access to your Azure Storage and Azure SQL Database accounts. Additional service endpoints will be introduced in the future.

- To read more about Virtual Network service endpoints, go to <https://docs.microsoft.com/azure/virtual-network/virtual-network-service-endpoints-overview>.

Distributed-denial-of-service protection

Azure's protection against distributed-denial-of-service (DDoS) attacks for Virtual Networks has been improved, which is timely, given that attacks against publicly accessible resources are increasing in number and complexity. The basic service included in your subscription provides real-time protection by using the scale and capacity of the Azure infrastructure to mitigate attacks (see Figure 7-8).

For an additional cost, you can take advantage of built-in machine learning algorithms to protect against targeted attacks, with added configuration, alerting, and telemetry.

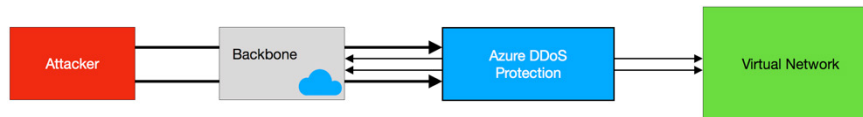


Figure 7-8 Azure DDoS protection defending a virtual network against attacks.

You also can use the Azure Application Gateway web application firewall to help protect against more sophisticated attacks.

Combined with Azure SQL Database auditing and NSGs, these features provide a comprehensive suite of protection against the latest threats.

- To read more about Azure DDoS protection, go to <https://azure.microsoft.com/services/ddos-protection>.

This page intentionally left blank



Understanding isolation levels and concurrency	383	Understanding automatic plan correction	418
Understanding delayed durability	400	Understanding execution plan operators	419
Understanding execution plans	401	Understanding parallelism	425
Using the Query Store feature	413		

In this chapter, we review the database concepts and objects most commonly associated with performance tuning the performance of objects within the Microsoft SQL Server database. We begin with a fundamental exploration of database isolation and its practical effects on queries. We then review the concepts of delayed durability and delayed durability transactions. Then, we explore *execution plans*, including ways to use them with the Query Store feature. We discuss execution plans in detail, what to look for when performance tuning, and how to control when they go parallel.

Entire books have been written on some of the sections in this chapter—we obviously can't go into that degree of detail here in a single chapter, but we do provide a deep enough discussion to jumpstart and accelerate your learning toward SQL Server performance tuning, including features added in SQL Server 2016 and 2017.

Understanding isolation levels and concurrency

It is important to have a fundamental understanding of *isolation levels*. These aren't just arcane keywords you study only when it is certification test time; they can have a profound effect on application performance, stability, and data integrity.

Understanding the differing impact of isolation levels on locking and blocking, and therefore on concurrency, is the key to understanding when you should use an isolation level different from the default of READ COMMITTED. Table 9-1 presents all of the isolation levels available in SQL Server.

Table 9-1 Isolation levels

Transaction isolation level	Allows dirty reads	Allows nonrepeatable reads	Allows phantom rows	Update conflicts possible
READ UNCOMMITTED	X	X	X	
READ COMMITTED		X	X	
REPEATABLE READ			X	
SERIALIZABLE				
READ COMMITTED SNAPSHOT (RCSI)		X	X	
SNAPSHOT				X

Inside OUT

What about READPAST?

READPAST is a table hint, not an isolation level, and you cannot set it at the session level. We discuss more about how and where you can set isolation levels later in this chapter.

But, READPAST can be useful in very specific circumstances, limited to when there are SQL Server tables used as “stack” or “queue,” with “first in, first out” architecture. READPAST does not place row-level locks on a table, and instead of being blocked by rows that are locked, it skips them. User transactions can fetch the “first” row in the stack that isn’t already being accessed.

In this way, a multithreaded process that is regularly looping through a table can read rows, afford to skip the rows currently being written to, and read them on the “next pass.” Outside of these limited scenarios, READPAST is not appropriate because it will likely return incomplete data.

When you are choosing an isolation level for a transaction in an application, you should consider primarily the transactional safety and business requirements of the transaction in a multiuser environment. The performance of the transaction should be a distant second priority when choosing an isolation level. Locking is not bad, it is the way that every transaction in SQL Server cooperates with others when dealing with disk-based tables. READ COMMITTED is generally a safe isolation level because it allows updates to block reads. In the default READ COMMITTED isolation level, reads cannot read uncommitted data and must wait for a transaction to commit or rollback. In this way, READ COMMITTED prevents a SELECT statement from accessing uncommitted data, a problem known as a *dirty read*. This is especially important during multistep transactions, in which parent and child records in a foreign key relationship must be created in the same transaction. In that scenario, reads should not access either table until both tables are updated.

READ COMMITTED does not ensure that row data and row count won't change between two SELECT queries in a multistep transaction. For some application scenarios, this might be acceptable or desired, but not for others. To avoid these two problematic scenarios (which we talk more about soon), you need to increase the transaction's isolation.

For scenarios in which transactions must have a higher degree of isolation from other transactions, escalating the isolation level of a transaction is appropriate. For example, if a transaction must process multistep writes and cannot allow other transactions to change data during the transaction, escalating the isolation level of a transaction is appropriate. Here are two examples:

In this example, REPEATABLE READ would block other transactions from changing or deleting rows needed during a multistep transaction. This phenomenon is called *nonrepeatable reads*. A nonrepeatable read returns different or fewer rows of data when attempting to read the same data twice, which is problematic to multistep transactions. Nonrepeatable reads can affect transactions with less isolation than REPEATABLE READ.

However, if the transaction in this example would need to ensure that the same number of rows in a result set is returned throughout a multistep transaction, the SERIALIZABLE isolation is necessary. It is the only isolation level that prevents other transactions from inserting new rows inside of a range of rows, a problem known as *phantom rows*.

The behavior just described is consistent with transactions affecting only a few rows. In these cases, the Database Engine is performing row and page-level locks to provide protection for transaction isolation. It is possible that REPEATABLE READ transactions could access a large number of rows and then escalate to use table locks, and then protect the transaction against phantom rows.

- For more on monitoring database locking and blocking, see Chapter 13.

Inside OUT

SQL Server doesn't have a time-out? Really?

That's correct, by default there is no time-out for a local request that is being blocked in SQL Server, although applications can report a "SQL time-out" if query run time surpasses their own time-out limitations.

By default, SQL Server will not cancel a request that is being blocked, but you can change this behavior for individual sessions. The value of the global variable @@LOCK_TIMEOUT is -1 by default, indicating that there is no time-out. You can change this for the current session by using the following statement:

```
SET LOCK_TIMEOUT n;
```

Where *n* is the number of milliseconds before a request is cancelled by SQL Server, returning error 1222, "Lock request time out period exceeded. The statement has been terminated." Take caution in implementing this change to SQL's default lock time-out, and try to fully understand the cause of the blocking first. If you change the lock time-out in code, ensure that any applications creating the sessions are prepared to handle the errors gracefully and retry.

SQL Server does have a configuration setting for a lock time-out for outgoing remote connections called Remote Query Timeout (s), which defaults to 600 seconds. This time-out applies only to connections to remote data providers, not to requests run on the SQL Server instance.

NOTE

You can declare isolation levels for transactions that read and write to both memory-optimized tables and disk-based tables. Memory-optimized tables do not use locks or latches; instead, they use row versioning to achieve the isolation and concurrency. Chapter 8 covers memory-optimized tables, and we discuss their use in high-transaction volume scenarios in Chapter 10.

Understanding how concurrent sessions become blocked

In this section, we review a series of realistic examples of how concurrency works in a multiuser application interacting with SQL Server tables. First, let's discuss how to diagnose whether a request is being blocked or blocking another request.

How to observe blocking

It's easy to find out live whether a request is being blocked. The dynamic management view `sys.dm_db_requests`, when combined with `sys.dm_db_sessions` on the `session_id` column, provides similar data plus much more information than the legacy `sp_who` or `sp_who2` commands, including the `blocked_by` column, as demonstrated here:

```
SELECT * FROM
sys.dm_exec_sessions s
LEFT OUTER JOIN sys.dm_exec_requests r ON r.session_id = s.session_id;
```

Now, let's review some example scenarios to detail exactly why and how requests can block one another in the real world. This is the foundation of concurrency in SQL Server and helps you understand the reason why `NOLOCK` appears to make queries perform faster. The examples that follow behave identically in SQL Server instances and databases in Microsoft Azure SQL Database

Understanding concurrency: two requests updating the same rows

Consider the following steps involving two writes, with each transaction coming from a different session. The transactions are explicitly declared by using the `BEGIN/COMMIT TRAN` syntax. In this example, the transactions are not overriding the default isolation level of `READ COMMITTED`:

1. A table contains only rows of `Type = 0` and `Type = 1`. Transaction 1 begins and updates all rows from `Type = 1` to `Type = 2`.
2. Before Transaction 1 commits, Transaction 2 begins and issues a statement to update `Type = 2` to `Type = 3`. Transaction 2 is blocked and will wait for Transaction 1 to commit.
3. Transaction 1 commits.
4. Transaction 2 is no longer blocked and processes its update statement. Transaction 2 then commits.

The result: The resulting table will contain records of `Type = 3`, and the second transaction will have updated records. This is because when Transaction 2 started, it waited, too, for committed data until after Transaction 1 committed.

Understanding concurrency: a write blocks a read

Next, consider the following steps involving a write and a read, with each transaction coming from a different session. In this scenario, an uncommitted write in Transaction 1 blocks a read in Transaction 2. The transactions are explicitly declared using the `BEGIN/COMMIT TRAN`

syntax. In this example, the transactions are not overriding the default isolation level of READ COMMITTED:

1. A table contains only records of Type = 0 and Type = 1. Transaction 1 begins and updates all rows from Type = 1 to Type = 2.
2. Before Transaction 1 commits, Transaction 2 begins and issues a SELECT statement for records of Type = 2. Transaction 2 is blocked and waits for Transaction 1 to commit.
3. Transaction 1 commits.
4. Transaction 2 is no longer blocked, and processes its SELECT statement. Rows are returned. Transaction 2 then commits.

The result: Transaction 2 returns records of Type = 2. This is because when Transaction 2 started, it waited for committed data until after Transaction 1 committed.

Understanding concurrency: a nonrepeatable read

Consider the following steps involving a read and a write, with each Transaction coming from a different session. In this scenario, Transaction 1 suffers a nonrepeatable read, as READ COMMITTED does not offer any protection against phantom rows or nonrepeatable reads. The transactions are explicitly declared using the BEGIN/COMMIT TRAN syntax. In this example, the transactions are not overriding the default isolation level of READ COMMITTED:

1. A table contains only records of Type = 0 and Type = 1. Transaction 1 starts and selects rows where Type = 1. Rows are returned.
2. Before Transaction 1 commits, Transaction 2 starts and issues an Update statement, setting records of Type = 1 to Type = 2. Transaction 2 is not blocked, and process immediately.
3. Transaction 1 again selects rows where Type = 1, and is blocked.
4. Transaction 2 commits.
5. Transaction 1 is immediately unblocked. No rows are returned. (No committed rows exist where Type=1.) Transaction 1 commits.

The result: The resulting table contains records of Type = 2, and the second transaction has updated records. This is because when Transaction 2 started, Transaction 1 had not placed any exclusive locks on the data, allowing for writes to happen. Because it is doing only reads, Transaction 1 would never have placed any exclusive locks on the data. Transaction 1 suffered from a nonrepeatable read: the same SELECT statement returned different data during the same multistep transaction.

Understanding concurrency: preventing a nonrepeatable read

Consider the following steps involving a read and a write, with each transaction coming from a different session. This time, we protect Transaction 1 from dirty reads and nonrepeatable reads by using the REPEATABLE READ isolation level. A read in the REPEATABLE READ isolation level will block a write. The transactions are explicitly declared by using the BEGIN/COMMIT TRAN syntax:

1. A table contains only records of Type = 0 and Type = 1. Transaction 1 starts and selects rows where Type = 1 in the REPEATABLE READ isolation level. Rows are returned.
2. Before Transaction 1 commits, Transaction 2 starts and issues an UPDATE statement, setting records of Type = 1 to Type = 2. Transaction 2 is blocked by Transaction 1.
3. Transaction 1 again selects rows where Type = 1. Rows are returned.
4. Transaction 1 commits.
5. Transaction 2 is immediately unblocked and processes its update. Transaction 2 commits.

The result: The resulting table will contain records of Type = 2. This is because when Transaction 2 started, Transaction 1 had placed read locks on the data it was selecting, blocking writes to happening until it had committed. Transaction 1 returned the same records each time and did not suffer a nonrepeatable read. Transaction 2 processed its updates only when it could place exclusive locks on the rows it needed.

Understanding concurrency: experiencing phantom reads

Consider the following steps involving a read and a write, with each transaction coming from a different session. In this scenario, we describe a phantom read:

1. A table contains only records of Type = 0 and Type = 1. Transaction 1 starts and selects rows where Type = 1 in the REPEATABLE READ isolation level. Rows are returned.
2. Before Transaction 1 commits, Transaction 2 starts and issues an INSERT statement, adding rows of Type = 1. Transaction 2 is not blocked by Transaction 1.
3. Transaction 1 again selects rows where Type = 1. More rows are returned compared to the first time the select was run in Transaction 1.
4. Transaction 1 commits.
5. Transaction 2 commits.

The result: Transaction 1 experienced a phantom read when it returned a different number of records the second time it selected from the table inside the same transaction. Transaction 1 had not placed any locks on the range of data it needed, allowing for writes in another transaction to happen within the same dataset. The phantom read would have occurred to Transaction 1 in any isolation level, except for `SERIALIZABLE`. Let's look at that next.

Understanding concurrency: preventing phantom reads

Consider the following steps involving a read and a write, with each transaction coming from a different session. In this scenario, we protect Transaction 1 from a phantom read.

1. A table contains only records of `Type = 0` and `Type = 1`. Transaction 1 starts and selects rows where `Type = 1` in the `SERIALIZABLE` isolation level. Rows are returned.
2. Before Transaction 1 commits, Transaction 2 starts and issues an `INSERT` statement, adding rows of `Type = 1`. Transaction 2 is blocked by Transaction 1.
3. Transaction 1 again Selects rows where `Type = 1`. The same number of rows are returned.
4. Transaction 1 commits.
5. Transaction 2 is immediately unblocked and processes its insert. Transaction 2 commits.

The result: Transaction 1 did not suffer from a phantom read the second time it selected from the table, because it had placed a lock on the range of rows it needed. The table now contains additional records for `Type = 1`, but they were not inserted until after Transaction 1 had committed.

Stating the case against `READ UNCOMMITTED (NOLOCK)`

Many developers and database administrators consider the `NOLOCK` table hint and the equivalent `READ UNCOMMITTED` isolation level nothing more than the turbo button on their 486DX. "We had performance problems, but we've been putting `NOLOCK` in all our stored procedures to fix it."

The effect of the table hint `NOLOCK` or the `READ UNCOMMITTED` isolation level is that no locks are taken inside the database, save for schema locks. (A query using `NOLOCK` could still be blocked by Data Definition Language [DDL] commands.) The resulting removal of basic integrity of the mechanisms that retrieve data can result in uncommitted data, obviously, but that is not usually enough to scare away developers. There are more good reasons to avoid the `READ UNCOMMITTED` isolation level, however.

The case against using the `READ UNCOMMITTED` isolation level is deeper than the performance and deeper than "data that has yet to be committed." Developers might counter that data is rarely ever rolled back or that the data is for reporting only. In production environments, these

are not sufficient grounds to justify the potential problems. The only situations in which READ UNCOMMITTED are an acceptable performance shortcut involve nonproduction systems, estimate-only counts, or estimate-only aggregations.

A query in READ UNCOMMITTED isolation level could return invalid data in the following real-world, provable ways:

- Read uncommitted data (dirty reads)
- Read committed data twice
- Skip committed data
- Return corrupted data
- Or, the query could fail altogether: “Could not continue scan with NOLOCK due to data movement.”

One final caveat: in SQL Server you cannot apply NOLOCK to tables when used in modification statements, and it ignores the declaration of READ UNCOMMITTED isolation level in a batch that includes modification statements; for example:

```
INSERT INTO dbo.testno1ock1 WITH (NOLOCK)
SELECT * FROM dbo.testno1ock2;
```

The preceding code will return the error:

```
Msg 1065, Level 15, State 1, Line 17
The NOLOCK and READUNCOMMITTED lock hints are not allowed for target tables of INSERT,
UPDATE, DELETE or MERGE statements.
```

However, this protection doesn't apply to the *source* of any writes, hence the danger.

This following code *is* allowed and is dangerous because it could write invalid data:

```
INSERT INTO testno1ock1
SELECT * FROM testno1ock2 WITH (NOLOCK);
```

Changing the isolation level within transactions

In addition to using the SET TRANSACTION ISOLATION LEVEL command, you can use table hints to override previously set behavior. Let's review the two ways by which you can change the isolation level of queries.

Using the transaction isolation level option

The SET TRANSACTION ISOLATION LEVEL command changes the isolation level for the current session, affecting all future transactions until the connection is closed.

But, you can change the isolation level of an explicit transaction after it is created, as long as you are not changing from or to the SNAPSHOT isolation level.

For example, the following code snippet is technically valid:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRAN
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
SELECT...
```

However, this snippet is invalid:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRAN
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
SELECT...
```

Doing so results in the following error:

```
Msg 3951, Level 16, State 1, Line 4
```

```
Transaction failed in database 'databasename' because the statement was run under snapshot isolation but the transaction did not start in snapshot isolation. You cannot change the isolation level of the transaction after the transaction has started.
```

In .NET applications, you should change the isolation level of each transaction when it is created. In Transact-SQL (T-SQL) code and stored procedures, you should change the execution plan of the session before creating an explicit transaction.

Using table hints to change isolation

You also can use isolation level hints to change the isolation level at the individual object level. This is an advanced type of troubleshooting that you shouldn't use commonly, because it increases the complexity of maintenance and muddies architectural decisions with respect to enterprise concurrency.

For example, you might have seen developers use NOLOCK at the end of a table, effectively (and dangerously) dropping access to that table into the READ COMMITTED isolation level:

```
SELECT col1 FROM dbo.Table (NOLOCK)
```

Aside from the unadvisable use of NOLOCK in the preceding example, using a table hint without WITH is deprecated syntax (since SQL Server 2008).

Aside from the cautionary NOLOCK, there are 20-plus other table hints that can have utility, including the ability for a query to use a certain index, to force a seek or scan on an index, or to override the query optimizer's locking strategy. We look at how to use UPDLOCK later in this chapter; for example, to force the use of the SERIALIZABLE isolation level.

All table hints should be considered for temporary and/or highly situational troubleshooting. They could make maintenance of these queries problematic in the future. For example, using the `INDEX` or `FORCESEEK` table hints could result in poor query performance or even cause the query to fail if the table's indexes are changed.

- For detailed information on all possible table hints, see the SQL Server documentation at <https://docs.microsoft.com/sql/t-sql/queries/hints-transact-sql-table>.

Understanding the enterprise solution to concurrency: SNAPSHOT

In the interest of performance, however, application developers too often seek to solve concurrency issues (reduce blocking) by using `READ UNCOMMITTED`. At first and at scale, the performance gains are too vast to consider other alternatives. But there is a far safer option, without the significant drawbacks and potential for invalid data and errors. Using row versioning with `READ_COMMITTED_SNAPSHOT (RCSI)` and/or the `SNAPSHOT` isolation level is the enterprise solution to performance issues related to concurrency.

`SNAPSHOT` isolation allows queries to read from the same rows that might be locked by other queries by using row versioning. The SQL Server instance's TempDB keeps a copy of committed data, and this data can be served to concurrent requests. In this way, `SNAPSHOT` allows access only to committed data but without blocking access to data locked by writes. By increasing the utilization and workload of TempDB for disk-based tables, performance is dramatically increased by increasing concurrency without the dangers of accessing uncommitted data.

Although row versioning works silently in the background, you access it at the statement level, not at the transaction or session levels. Each statement will have access to the latest committed row version of the data. In this way, `RCSI` is still susceptible to nonrepeatable reads and phantom rows. `SNAPSHOT` isolation uses row versions of affected rows throughout a transaction; thus, it is not susceptible to nonrepeatable reads and phantom rows.

As an example of `SNAPSHOT` in use internally, all queries run against a secondary readable database in an availability group are run in the `SNAPSHOT` isolation level, by design. The transaction isolation level and any locking table hints are ignored. This removes any concurrency conflicts between a read-heavy workload on the secondary database and the transactions arriving there from the primary database.

Understanding concurrency: accessing SNAPSHOT data

Consider the following steps involving a read and a write, with each transaction coming from a different session. In this scenario, we see that Transaction 2 has access to previously committed row data, even though those rows are being updated concurrently.

1. A table contains only records of Type = 1. Transaction 1 starts and updates rows where Type = 1 to Type = 2.
2. Before Transaction 1 commits, Transaction 2 sets its session isolation level to SNAPSHOT.
3. Transaction 2 issues a SELECT statement WHERE Type = 1. Transaction 2 is not blocked by Transaction 1. Rows where Type = 1 are returned. Transaction 2 commits.
4. Transaction 1 commits.
5. Transaction 2 again issues a SELECT statement WHERE Type = 1. No rows are returned.

The result: Transaction 2 was not blocked when it attempted to query rows that Transaction 1 was updating. It had access to previously committed data, thanks to row versioning.

Implementing SNAPSHOT isolation

You can implement SNAPSHOT isolation level in a database in two different ways. Turning on SNAPSHOT isolation simply allows for the use of SNAPSHOT isolation and begins the process of row versioning. Alternatively, turning on RCSI changes the default isolation level to READ COMMITTED SNAPSHOT. You can implement both or either. It's important to understand the differences between these two settings, because they are not the same:

- READ COMMITTED SNAPSHOT configures optimistic concurrency for reads by overriding the default isolation level of the database. When turned on, all queries will use RCSI unless overridden.
- SNAPSHOT isolation mode configures optimistic concurrency for reads and writes. You must then specify the SNAPSHOT isolation level for any transaction to use SNAPSHOT isolation level. It is possible to have update conflicts with SNAPSHOT isolation mode that will not occur with READ COMMITTED SNAPSHOT.

The statement to implement SNAPSHOT isolation in the database is simple enough, but is not without consequence. Even if no transactions or statements use the SNAPSHOT isolation level, behind the scenes, TempDB begins storing row version data for disk-based tables. (Memory-optimized tables have row-versioning built in and don't need TempDB.) The Database Engine maintains previous versions for changing data in TempDB regardless of whether that data is currently being accessed by user queries. Here's how to implement SNAPSHOT isolation:

```
ALTER DATABASE databasename SET ALLOW_SNAPSHOT_ISOLATION ON;
```

All transactions will continue to use the default READ COMMITTED isolation level, but you now can specify the use SNAPSHOT isolation at the session level or in table hints, as shown in the following example:

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

Alternatively, or in conjunction with `ALLOW_SNAPSHOT_ISOLATION`, you can turn on RCSI as the new default isolation level in a database. Here's how to turn on RCSI:

```
ALTER DATABASE databasename SET READ_COMMITTED_SNAPSHOT ON;
```

You can set both of the preceding database settings independently of each other. Setting `ALLOW_SNAPSHOT_ISOLATION` is not required to turn on `READ_COMMITTED_SNAPSHOT`, and vice versa. Similarly, these settings are not tied to the `MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT` database setting to promote memory-optimized table access to SNAPSHOT isolation.

- We discuss memory-optimized tables in greater detail in Chapter 10.

For either of the previous `ALTER DATABASE` statements to succeed, no other transactions can be open in the database. It might be necessary to close other connections manually or to put the database in `SINGLE_USER` mode. Either way, we do not recommend that you perform this change during production activity.

NOTE

Do not change the `READ_COMMITTED_SNAPSHOT` database option if you have any memory-optimized tables set to `DURABILITY = SCHEMA_ONLY`. All rows in the table will be lost. You should move the contents of the table to a more durable table before changing `READ_COMMITTED_SNAPSHOT` to `ON` or `OFF`.

Be aware and prepared for the increased utilization in the TempDB, both in the demand and space requirements. To avoid autogrowth events, increase the size of the TempDB data and log files and monitor their size. Although you should try to avoid autogrowth events by growing the TempDB data file(s) yourself, you should also verify that your TempDB file autogrowth settings are appropriate.

- For more information on file autogrowth settings, see Chapter 4.

Should the TempDB exhaust all available space on its drive volume, SQL will be unable to row-version records for transactions, and will terminate them with SQL Server error 3958. SQL Server will also issue errors 3967 and 3966 as the oldest row versions are removed from the TempDB to make room for new row versions needed by newer transactions.

NOTE

Prior to SQL Server 2016, `READ COMMITTED SNAPSHOT` and `SNAPSHOT` isolation levels were not supported with Columnstore indexes. Beginning with SQL Server 2016, `SNAPSHOT` isolation and Columnstore indexes are fully compatible.

Understanding updates in SNAPSHOT isolation level

Transactions that read data in SNAPSHOT isolation or RCSI will have access to previously committed data instead of being blocked, when data needed is being changed. This is important to understand and could result in an update statement experiencing a concurrency error. The potential for update conflicts is real and you need to understand it. In the next section, we review ways to mitigate the risk.

For example, consider the following steps, with each transaction coming from a different session. In this example, Transaction 2 fails due to a concurrency conflict or “write-write error”:

1. A table contains many records, each with a unique ID. Transaction 1 begins a transaction in the READ COMMITTED isolation level and performs an update on the row where ID = 1.
2. Transaction 2 sets its session isolation level to SNAPSHOT and issues a statement to update the row where ID = 1.
3. Transaction 1 commits first.
4. Transaction 2 immediately fails with SQL error 3960.

The result: Transaction 1’s update to the row where ID = 1 succeeded. Transaction 2 immediately failed with the following error message:

```
Msg 3960, Level 16, State 2, Line 8
```

```
Snapshot isolation transaction aborted due to update conflict. You cannot use snapshot isolation to access table 'dbo.AnyTable' directly or indirectly in database 'WideWorldImporters' to update, delete, or insert the row that has been modified or deleted by another transaction. Retry the transaction or change the isolation level for the update/delete statement.
```

The transaction for Transaction 2 was rolled back, marked uncommittable. Let’s try to understand why this error occurred, what to do, and how to prevent it.

In SQL Server, SNAPSHOT isolation uses locks to create blocking but doesn’t block updates from colliding for disk-based tables. It is possible to error when committing an update statement, if another transaction has changed the data needed for an update during a transaction in SNAPSHOT isolation level.

For disk-based tables, the update conflict error will look like the Msg 3960 that we saw a moment ago. For queries on memory-optimized tables, the update conflict error will look like this:

```
Msg 41302, Level 16, State 110, Line 8
```

```
The current transaction attempted to update a record that has been updated since this transaction started. The transaction was aborted.
```

The preceding error can occur with `ALLOW_SNAPSHOT_ISOLATION` turned on if transactions are run in `SNAPSHOT` isolation level.

Even though optimistic concurrency of snapshot isolation level (and also memory-optimized tables) increases the potential for update conflicts, you can mitigate these by doing the following:

- When running a transaction in `SNAPSHOT` isolation level, it is crucial to avoid using any statements that place update locks to disk-based tables inside multistep explicit transactions.

Similarly, always avoid multistep transactions with writes when working with memory-optimized tables, regardless of isolation level.

- Specifying the `UPDLOCK` table hint can have utility at preventing update conflict errors for long-running `SELECT` statements. The `UPDLOCK` table hints places pessimistic locks on rows needed for the multistep transaction to complete. The use of `UPDLOCK` on `SELECT` statements with `SNAPSHOT` isolation level is not a panacea for update conflicts, and it could in fact create them. Frequent select statements with `UPDLOCK` could increase the number of update conflicts with updates. Regardless, your application should handle errors and initiate retries when appropriate.

If two concurrent statements use `UPDLOCK`, with one updating and one reading the same data, even in implicit transactions, an update conflict failure is possible if not likely.

- Avoid writes altogether while in `SNAPSHOT` isolation mode. Change the transaction isolation level back to `READ COMMITTED` before running an `UPDATE` statement, and then back to `SNAPSHOT` if desired.

Specifying table granularity hints such as `ROWLOCK` or `TABLOCK` can prevent update conflicts, although at the cost of concurrency. The second update transaction must be blocked while the first update transaction is running—essentially bypassing `SNAPSHOT` isolation for the write. If two concurrent statements are both updating the same data in `SNAPSHOT` isolation level, an update conflict failure is likely for the statement that started second.

Using memory-optimized tables in `SNAPSHOT` isolation level

`SNAPSHOT` isolation is supported for memory-optimized tables, but not with all of the different ways to place a query in `SNAPSHOT` isolation. There are only ways to ensure that memory-optimized tables use `SNAPSHOT` isolation:

- Turn on the `MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT` database option. This promotes access to all memory-optimized tables in the database up to `SNAPSHOT` isolation level if the current isolation level is not `REPEATABLE READ` or `SERIALIZABLE`. It will promote the isolation level to `SNAPSHOT` from isolation levels such as `READ UNCOMMITTED`

and READ COMMITTED. This option is off by default, but you should consider it because you otherwise cannot use the READ UNCOMMITTED or SNAPSHOT isolation levels for a session including memory-optimized tables.

- You can specify SNAPSHOT isolation with table hints (see the section “Using table hints to change isolation” earlier in this chapter). Note that only for memory-optimized tables can use this SNAPSHOT table hint, not disk-based tables.

You cannot, for example, include memory-optimized tables in a session that begins with SET TRANSACTION ISOLATION LEVEL SNAPSHOT, even if MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT = ON or you specify the SNAPSHOT table hint.

Inside OUT

Which isolation level does my .NET application use?

Be aware that by default the .NET System.Transaction infrastructure uses the SERIALIZABLE isolation level, the safest but least practical choice. SERIALIZABLE provides the most isolation for transactions, so by default .NET transactions do not suffer from dirty reads, nonrepeatable reads, or phantom rows.

You might find, however, that SERIALIZABLE transactions are being frequently blocked and at the source of blocking, and that reducing the isolation of certain transactions would result in better performance. Evaluate the potential risk of nonrepeatable reads and phantom rows for each new .NET transaction, and reduce the isolation level to REPEATABLE READ or READ COMMITTED only where appropriate, and following guidance throughout this chapter, do not use the READ UNCOMMITTED isolation level in any production code.

For applications with high transactional volume, consider also using SNAPSHOT isolation level to increase concurrency.

You can set the isolation level of any transaction when it is begun by setting the IsolationLevel property of the TransactionScope class. You can also default a new database connection’s isolation level upon creation. Remember, however, that you cannot change the isolation level of a transaction after it has begun.

Understanding on-disk versus memory-optimized concurrency

Queries using memory-optimized tables (initially called Project Hekaton prior to the release of SQL 2014) can perform significantly faster than queries based on the same data in disk-based tables. Memory-optimized tables can improve the performance of frequently written-to tables by up to 40 times over disk-based tables.

When in the aforementioned scenarios we use the words “prevents” or “protection,” we mean locking, and this applies only to on-disk tables, not memory-optimized tables. When a transaction has rows or a range of rows locked, any other transaction’s writes in that range are blocked and wait patiently, queueing up to proceed as soon as the locks are released. Although SQL Server allows requests to wait and be blocked forever, the applications generating the request might easily time out under a minute of waiting.

In the case of memory-optimized tables, locking isn’t the mechanism that ensures isolation. Instead, the in-memory engine uses row versioning to provide row content to each transaction. In the in-memory engine, update operations create new rows in the in-memory data structure (actually a heap), that supplant older row versions. Similarly, delete operations create rows in a delta file, marking the row as deleted. Periodically, cleanup is done to merge the in-memory data structure and delta files to reduce the space used in memory, and in the case of tables with durable data, on a drive. If you are familiar with the data warehousing concept of a Slowly Changing Dimension (SCD), this is similar to an SCD Type II.

If two transactions attempt to update the same data at the same time, one transaction will immediately fail due to a concurrency error. Only one transaction can be in the process of updating or deleting the same row at a time. The other will fail with a concurrency conflict (SQL error 41302).

This is the key difference between the behavior of pessimistic and optimistic concurrency. Pessimistic concurrency uses locks to prevent write conflict errors, whereas optimistic concurrency uses row versions with acceptable risk of write conflict errors. On-disk tables offer isolation levels that use pessimistic concurrency to block conflicting transactions, forcing them to wait. Memory-optimized tables offer optimistic concurrency that will cause a conflicting transaction to fail.

In the case of a nonrepeatable read, SQL error 41305 will be raised. In the case of a phantom read, a SQL error 41325 will be raised. Because of these errors, applications that write to memory-optimized tables must include logic that gracefully handles and automatically retries transactions. They should already handle and retry in the case of deadlocks or other fatal database errors.

- For more information on configuring memory-optimized tables, see Chapter 8.
- We discuss more about indexes for memory-optimized tables in Chapter 10.

Understanding delayed durability

Delayed durability is a set of transaction features first introduced in SQL Server 2014. It allows for transactions to avoid synchronously committing to a disk; instead, committing only to memory and asynchronously committing to a disk. If this sounds dangerous to you, and opens the possibility to losing records in the event of a server shutdown, you are correct!

However, unless your SQL Server instance's databases are running in a synchronous availability group (and even then, chance exists for the databases to drop into asynchronous under pressure), you already face the likelihood in your database of losing recently written records in the event of a sudden server or drive failure.

So perhaps delayed durability's danger isn't so unfamiliar after all. Databases in Azure SQL Database also support delayed durability transactions, with the same caveat and expectations for data recovery. Some data loss is possible.

NOTE

Any SQL Server instance service shutdown, whether it be a planned restart or sudden failure, could result in delayed durability transactions being lost. This also applies to the failover of a failover cluster instance (FCI), availability group, or database mirror. Transaction log backups and log shipping will similarly contain only transactions made durable. You must be aware of this potential when implementing delayed durability.

NOTE

Distributed (DTC) and cross-database transactions are always durable.

A delayed durable transaction will be flushed to the disk whenever a threshold of delayed durability transactions builds up, or, whenever any other durable transaction commits in the same database. You also can force a flush of the transaction log with the system stored procedure `sp_flush_log`. Otherwise, the transactions are written to a buffer in-memory and kept away from using I/O resources until a log flush event. SQL Server manages the buffer, but makes no guarantees as to the amount of time a transaction can remain in buffer.

The delayed durability options, implemented either at the database level or at the transaction level, have application in very-high-performance workloads for which the bottleneck to write performance is the transaction log itself. By trading the possibility for new records to be written only to memory and lost in the event of a shutdown, you can gain a significant performance increase, especially with write-heavy workloads.

It's important to note that delayed durability is simply about reducing the I/O bottleneck of committing a massive quantity of writes to the transaction log. This has no effect on isolation (locking, blocking) or access to any data in the database that must be read to perform the write. Otherwise, delayed durability transactions follow the same rules as other transactions.

NOTE

Aside from the basic concept of an in-memory buffer, this topic is not related to memory-optimized tables. The `DELAYED_DURABILITY` database option is not related to the `DURABILITY` option when creating optimized tables.

Delayed durability database options

At the database level, you can set the `DELAYED_DURABILITY` option to `DISABLED` (default), `ALLOWED`, or `FORCED`.

The `FORCED` option obviously has implications on the entirety of the database, and you should consider it carefully with existing applications and databases. The `ALLOWED` option permits delayed durability transactions but has no effect on other transactions.

Delayed durability transactions

In the end, delayed durability is a transaction option with simple syntax. This syntax is necessary only when `DELAYED_DURABILITY = ALLOWED` in the current database.

It is supported for explicit transactions at the time they are committed by using the following sample syntax:

```
BEGIN TRAN
COMMIT TRAN WITH (DELAYED_DURABILITY=ON);
```

In the case of a natively compiled procedure, you can specify `DELAYED_DURABILITY` in the `BEGIN ATOMIC` block. Take, for example, this procedure in the `WideWorldImporters` database:

```
CREATE PROCEDURE [Website].[RecordColdRoomTemperatures_DD]
@SensorReadings Website.SensorDataList READONLY
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS OWNER
AS
BEGIN ATOMIC WITH
(
    TRANSACTION ISOLATION LEVEL = SNAPSHOT,
    LANGUAGE = N'English',
    DELAYED_DURABILITY = ON
)
BEGIN TRY
...

```

Understanding execution plans

Execution plans are a detailed explanation of the query optimizer's plan for processing any statement. Each time you run a statement, including batches with multiple statements, an execution plan is generated.

Execution plans inform the developer of the steps the Database Engine will take to retrieve data, from the tables, through the various transformation steps to sort, join, and filter data, and finally return or affect data. All statements create execution plans, including Data Manipulation Language (DML) and DDL.

Execution plans contain the cost and other metadata of each piece that it takes to process a query—from the data retrieval steps, joins, sorts, and more, and finally the DML or DDL operation itself. This data can be invaluable to developers and database administrators for tuning query performance.

The Procedure Cache, stored in the memory that SQL Server uses, contains query plans for statements that have been run. The Query Store is a powerful built-in repository in each database to track and trend runtime statistics over time.

Execution plans are generated for a query and reused when that exact same query text is called again. (The query text is first and always subjected to simplification, which removes redundancies, including using a code reduction technique called Constant Folding.) Queries will reuse the same plan only if every character of the query statement matches, including capitalization, whitespace, line breaks, and text in comments. There is one exception to this rule of query reuse, and that is when SQL Server parameterizes a query or stored procedure statement.

SQL Server does a smart job at sniffing for parts of a statement that could be parameterized to make a query's cached plan reusable. For example, a query that has a `WHERE` clause on a `LastName` field should be able to use the same execution plan whether it is searching for "Smith" or "Green."

Understanding parameterization and "parameter sniffing"

SQL Server parameterization occurs when the query optimizer detects values (such as the search criteria of a `WHERE` clause statement) that can be parameterized.

With parameterization, it's possible that two potentially helpful or potentially problematic conditions can occur:

- You can reuse a query plan for multiple queries for which the query text is exactly the same, except for parameterized values.
- The same query could use the same execution plan for two different values of a `WHERE` clause, resulting in vastly different performance.

For example, the following two query statements in the `WideWorldImporters` database will be parameterized and use the same query plan. (This also means that both queries could

be affected by the same Query Store forced plan; more on that later.) The first query returns 13 rows, the second returns 1,055 rows:

```
SELECT ppo.OrderDate, ppo.PurchaseOrderID, pol.PurchaseOrderLineID, ppo.[SupplierID]
FROM [Purchasing].[PurchaseOrders] AS ppo
INNER JOIN [Purchasing].[PurchaseOrderLines] AS pol
    ON ppo.PurchaseOrderID = pol.PurchaseOrderID
INNER JOIN [Purchasing].[Suppliers] AS s ON s.SupplierID = ppo.SupplierID
WHERE ppo.SupplierID = 5
```

```
SELECT ppo.OrderDate, ppo.PurchaseOrderID, pol.PurchaseOrderLineID, ppo.[SupplierID]
FROM [Purchasing].[PurchaseOrders] AS ppo
INNER JOIN [Purchasing].[PurchaseOrderLines] AS pol
    ON ppo.PurchaseOrderID = pol.PurchaseOrderID
INNER JOIN [Purchasing].[Suppliers] AS s ON s.SupplierID = ppo.SupplierID
WHERE ppo.SupplierID = 4
```

In the WideWorldImporters database, we might see the same query plan for both statements results in quick performance for the smaller rowcount `SupplierID` and horrible performance for the larger rowcount.

If the larger rowcount query (`SupplierID = 4`) is run first and has its query plan cached, there isn't likely to be a problem. Both versions of the query will run well enough. If the smaller rowcount query (`SupplierID = 5`) is run first, its version of the plan will be cached. In this case, the plan is different, less efficient for very large row counts, and will be used for all versions of the parameterized statement.

Here are a few advanced troubleshooting avenues to alleviate this scenario:

- You can use the `OPTIMIZE FOR` query hint to demand that the query analyzer use a cached execution plan that substitutes a provided value for the parameters. You also can use `OPTIMIZE FOR UNKNOWN`, which instructs the query analyzer to optimize for the most common value, based on statistics of the underlying data object.
- The `RECOMPILE` query hint or procedure option does not allow the reuse of a cached plan, forcing a fresh query plan to be generated each time the query is run.
- You can use the Plan Guide feature (implemented via stored procedures) to guide the query analyzer to a plan currently in cache. You identify the plan via its `plan_handle`. For information on identifying and analyzing plans in `sys.dm_exec_cached_plans`, see the upcoming section, which contains a `plan_handle`.
- You can use the Query Store feature (implemented with a GUI in SQL Server Management Studio, and via stored procedures behind the scenes) to visually look at plan performance and force a query to use a specific plan currently in cache.

- ▶ For more information, see the section “Using the Query Store feature” later in this chapter.
- You could use the USE PLAN query hint to provide the entire XML query plan for any statement execution. This obviously is the least convenient option, and like other approaches that override the query analyzer, you should consider it an advanced and temporary performance tuning technique.

Understanding the Procedure Cache

New execution plans enter the Procedure Cache only when a new statement is run. If a procedure cache already contains a plan matching a previous run of the current statement, the execution plan is reused, saving valuable time and resources.

This is why complex statements can appear to run faster the second time they are run.

The Procedure Cache is empty when the SQL Server service starts and grows from there. SQL Server manages plans in the cache, removing them as necessary under memory pressure. The size of the Procedure Cache is managed by SQL Server and is inside the memory space configured for the server in the Max Server Memory configuration setting. Plans are removed based on their cost and how recently it has been used. Smaller, older plans and single-user plans are the first to be cleared.

Inside Out

If I run a statement only once, does SQL Server remember its plan?

By default, SQL Server adds an execution plan to the Procedure Cache the first time it is generated. You can view the number and size of cached execution plans with the dynamic management view `sys.dm_exec_cached_plans`. You might find that a large amount of space in the Procedure Cache is dedicated to storing execution plans that have been used only once. These single-use plans can be referred to as *ad hoc* execution plans, from the Latin, meaning “for this situation.”

If you find that a SQL Server instance is storing many single-use plans, as many do, selecting the server configuration option Optimize For Ad Hoc Queries will benefit performance. This option does not optimize ad hoc queries; rather, it optimizes SQL Server memory by storing an execution plan in memory only after the same query has been detected twice. Queries might then benefit from the cached plan only upon the third time they are run.

The following query provides the number of single-use versus multiuse query plans, and the space used to store both:

```
SELECT
    PlanUse = CASE WHEN p.usecounts > 1 THEN '>1' ELSE '1' END
,    PlanCount = COUNT(1)
,    SizeInMB = SUM(p.size_in_bytes/1024./1024.)
FROM sys.dm_exec_cached_plans p
GROUP BY CASE WHEN p.usecounts > 1 THEN '>1' ELSE '1' END;
```

Analyzing cached execution plans in aggregate

You can analyze execution plans in aggregate starting with the dynamic management view `sys.dm_exec_cached_plans`, which contains a `plan_handle`.

The `plan_handle` column contains a system-generated `varbinary(64)` string that can be joined to a number of other dynamic management views. As seen in the code example that follows, you can use the `plan_handle` to gather information about aggregate plan usage, plan statement text, and to retrieve the graphical execution plan itself. You might be used to viewing the graphical execution plan only after a statement is run in SQL Server Management Studio, but you can also analyze and retrieve plans by using the following query against a handful of dynamic management views (DMVs). These DMVs return data for all databases in SQL Server instances, and for the current database in Azure SQL Database.

Query cached plan stats

```
SELECT
    UseCount      = p.usecounts
,    PlanSize_KB = p.size_in_bytes / 1024
,    CPU_ms      = qs.total_worker_time/1000
,    Duration_ms = qs.total_elapsed_time/1000
,    ObjectType  = p.cacheobjtype + ' (' + p.objtype + ')'
,    DatabaseName = db_name(convert(int, pa.value))
,    txt.ObjectID
,    qs.total_physical_reads
,    qs.total_logical_writes
,    qs.total_logical_reads
,    qs.last_execution_time
,    StatementText = SUBSTRING (txt.[text], qs.statement_start_offset/2 + 1,
                                CASE WHEN qs.statement_end_offset = -1 THEN LEN
                                ELSE qs.statement_end_offset/2 - qs.statement_start_offset/2 + 1 END)
(CONVERT(nvarchar(max), txt.[text]))
```

```

, QueryPlan = qp.query_plan
FROM sys.dm_exec_query_stats AS qs
INNER JOIN sys.dm_exec_cached_plans p ON p.plan_handle = qs.plan_handle
OUTER APPLY sys.dm_exec_plan_attributes (p.plan_handle) AS pa
OUTER APPLY sys.dm_exec_sql_text (p.plan_handle) AS txt
OUTER APPLY sys.dm_exec_query_plan (p.plan_handle) AS qp
WHERE pa.attribute = 'dbid' --retrieve only the database id from sys.dm_exec_plan_
attributes
ORDER BY qs.total_worker_time + qs.total_elapsed_time DESC;

```

Note that the preceding query orders by a sum of the CPU time and duration, descending, returning the longest running queries first. You can adjust the `ORDER BY` and `WHERE` clauses in this query to hunt, for example, for the most CPU-intensive or most busy execution plans. Keep in mind that the Query Store feature, as detailed later in this chapter, will help you visualize the process of identifying the most expensive and longest running queries in cache.

As you can see in the previous query, you can retrieve a wealth of information from these five DMVs, including the statement within a batch that generated the query plan. The query plan appears as blue hyperlink in SQL Server Management Studio's Results To Grid mode, opening the plan as a new `.sqlplan` file. You can save and store the `.sqlplan` file for later analysis.

Permissions required to access cached plan metadata

The only permission needed to run the previous query in SQL Server is the server-level `VIEW SERVER STATE` permissions, which might be appropriate for developers to have access to in a production environment because it does not give them access to any data in user databases.

In Azure SQL Database, because of the differences between the Basic/Standard and Premium tiers, different permissions are needed. In the Basic/Standard tier, you must be the server admin or Azure Active Directory Admin to access objects that would usually require `VIEW SERVER STATE`. In Premium tier, you can grant `VIEW DATABASE STATE` in the intended database in Azure SQL Database to a user who needs permission to view the above DMVs.

Clearing the Procedure Cache

You might find that manually clearing the Procedure Cache is useful when performance testing or troubleshooting. Typically, you want to reserve this activity for nonproduction systems. There are a few strategies to clearing out cached plans in SQL Server.

To compare two versions of a query or the performance of a query with different indexes, you could clear the cached plan for the statement to allow for proper comparison. You can manually flush the entire Procedure Cache, or individual plans in cache, with the following database-scoped configuration command. The following command affects only the current database context, as opposed to the entire instance's procedure cache:

```
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
```

CAUTION

Avoid clearing the Procedure Cache in a live production environment during normal business hours. Doing so will cause all new statements to have their execution plans compiled, dramatically increasing processor utilization and potentially dramatically slowing performance.

This command was introduced in SQL Server 2016 and is effectively the same as the command `DBCC FREEPROCCACHE` within the current database context. It works in both SQL Server and Azure SQL Database. `DBCC FREEPROCCACHE` is not supported in Azure SQL Database.

You can use `DBCC FREEPROCCACHE` to clear the procedure cache of the SQL Server instance.

You can also remove a single plan from cache by identifying its `plan_handle` and then providing it as the parameter to the `DBCC FREEPROCCACHE` function. Perhaps this is a plan you would like to remove for testing or troubleshooting purposes that you have identified with the script in the previous section:

```
DBCC FREEPROCCACHE (0x06000700CA920912307B86
7DB701000001000000000000000000000000000000000000000000000000000000);
```

You could alternatively flush the cache by object type. This command clears cached execution plans that are the result of ad hoc statements and prepared statements (from applications):

```
DBCC FREESYSTEMCACHE ('SQL Plans');
```

The advantage of this statement is that it does not wipe the cached plans from “Programmability” database objects such as stored procedures, multistatement table-valued functions, scalar user-defined functions, and triggers. The following command clears the cached plans from those type of objects:

```
DBCC FREESYSTEMCACHE ('Object Plans');
```

Note that `DBCC FREESYSTEMCACHE` is not supported in Azure SQL Database.

You can also use `DBCC FREESYSTEMCACHE` to clear cached plans association to a specific Resource Governor Pool, as follows:

```
DBCC FREESYSTEMCACHE ('SQL Plans', 'poolname');
```

NOTE

Execution plans are not removed from cache for a database that is **OFFLINE**. Plan data is cleared from the Procedure Cache for databases dropped or detached from the SQL Server instance.

Retrieving execution plans in SQL Server Management Studio

There are three basic types of graphical execution plans to retrieve for a statement: Estimated, Actual, and Live. Let's review the differences, and how you can view them.

Estimate the execution plan

You can generate the estimated execution plan quickly and view it graphically from within SQL Server Management Studio by choosing the Display Estimated Execution Plan option in the Query menu, or pressing Ctrl+L. An estimated execution plan will return for the highlighted region, or for the entire file if no text is selected.

You can also retrieve an estimated graphical execution plan in T-SQL code by running the following statement:

```
SET SHOWPLAN_XML ON
```

The actual execution plan is returned as an XML string. In SQL Server Management Studio, in Grid mode, the results are displayed as a link. Click the link to open the plan graphically in SQL Server Management Studio. You can save the execution plan as a .sqlplan file by right-clicking in the neutral space of the plan window.

You can also configure the estimated text execution plan in code by running one of the following statements, which return the execution plan in one result set or two, respectively:

```
SET SHOWPLAN_ALL ON  
SET SHOWPLAN_TEXT ON
```

NOTE

Be aware that when any of the aforementioned three options are turned on, SQL Server will not run statements, only return estimated execution plans. Remember to turn off the SET SHOWPLAN_ option before you reuse the same session for other queries.

As expected, the estimated execution plan is not guaranteed to match the actual plan used when you run the statement, but it is a very reliable approximation. The query optimizer uses the same information for the estimate as it does for the actual plan when you run it.

One cause for any differences between the estimate and actual execution plans would be any reason for the plan to be recompiled between the estimate and actual plan generation, including if the plan was removed from the Procedure Cache.

To display information for individual steps, hover over a step in the execution plan. You can also click an object, and then open the Properties window by pressing F4 or, in the View menu, clicking Properties Window. You'll notice the estimated execution plan is missing some information that the actual plan returns. The missing fields are self-explanatory; for example, Actual Number Of Rows, Actual Number Of Batches, and Number of Executions.

Displaying the actual execution plan

You can generate the actual execution plan along with the statement's result set from within SQL Server Management Studio by choosing the Include Actual Execution Plan option in the Query menu, or pressing Control+M to turn on the setting. After turning on this setting, when you run a statement, you will see an additional tab along with the execution results.

You'll notice that returning the actual graphical execution plan adds some additional time to the execution. The actual execution plan will return as an additional tab in Management Studio.

You can also configure the actual graphical execution plan in T-SQL code, returning XML that can be viewed graphically in SQL Server Management Studio, by running the following statement:

```
SET STATISTICS XML ON
```

The actual execution plan is returned as an XML string. In SQL Server Management Studio, in Grid mode, the results display as a link.

Remember to turn off the SET STATISTICS option before you reuse the same session, if you don't want to get back the actual plan for every query you run on this connection.

You can save both the estimated and actual execution plans as a .sqlplan file by right-clicking the neutral space of the plan window.

Displaying live query statistics

You can generate and display a "live" version of the execution plan by using SQL Server Management Studio 2016. You can access live statistics on versions of SQL Server starting with SQL Server 2014. You turn on the Live Execution Statistics option in the Query menu of SQL Server Management Studio, as demonstrated in Figure 9-1.

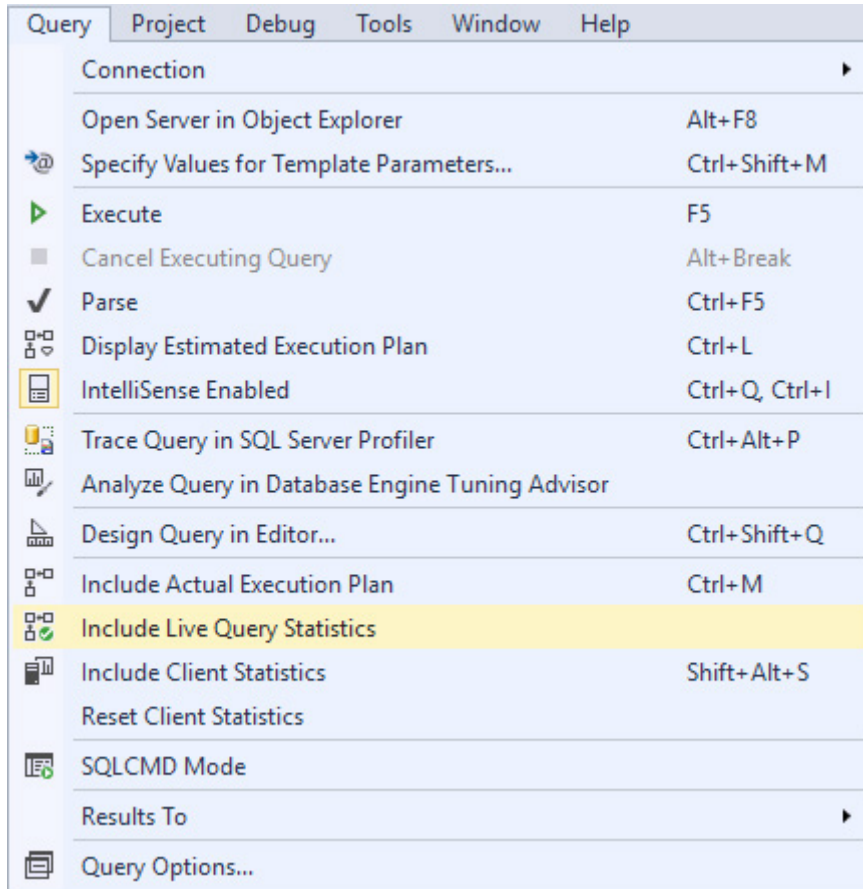


Figure 9-1 The Query menu in SQL Server Management Studio, with the Include Live Query Statistics option highlighted.

The Live Query Statistics window displays a hybrid version of the Estimated and Actual execution plans while the query is processing. If your query runs too quickly, you'll miss the dotted, moving lines and the various progress metrics including duration for each step and overall percentage completion. The percentage is based on the Actual rows processed currently incurred versus a total number of rows processed for that step.

The Live Query Statistics contains more information than the Estimated query plan, such as Actual Number Of Rows and Number Of Executions, but less than the Actual query plan. The Live Query Statistics does not display some data from the Actual Execution Plan, Actual Execution Mode, Number Of Rows Read, Actual Rebinds, and Actual Rebinds.

Notice in Figure 9-2 that returning the execution plan slows down the query, so be aware that the individual and overall execution durations measured will often be longer than when the query is run without the option to display Live Query Statistics.

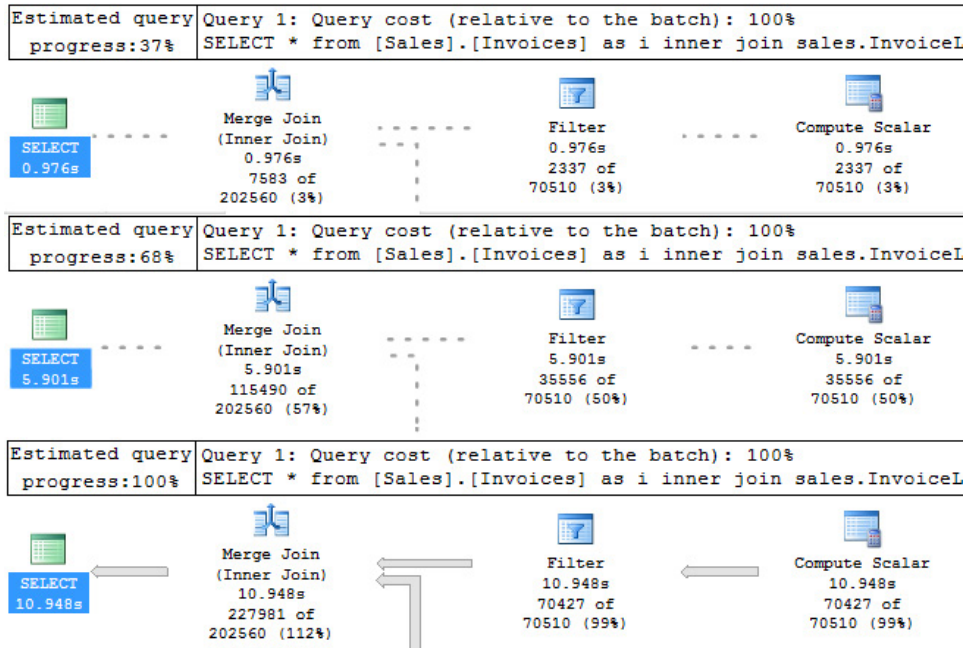


Figure 9-2 Three different screenshots of the Live Query Statistics, moments apart.

You might see that the total rows to be processed does not match total Estimated Number Of Rows for that step; rather, the multiple of that step's Estimated Number Of Rows and a preceding step's Estimated Number Of Rows. In Figure 9-2, the number of rows Estimated is less than the number of rows actually read.

Inside OUT

What's the difference between "Number Of Rows Read" and "Actual Number Of Rows"?

This is an important distinction, and it can tip you off to a significant performance issue.

Both are "Actual" values, but Actual Number Of Rows contains the number of values in the range of rows we expect to retrieve, and Number Of Rows Read contains the number of rows that were actually read. The difference could be significant to performance, and the solution is likely to change the query so that the predicate is narrower and/or better aligned with indexes on the table. Alternatively, you could add indexes to better fit the query predicates and make for more efficient searches.

One of the easiest ways to reproduce this behavior is with a wildcard search, for example in the WideWorldImporters sample database:

```
SELECT i.InvoiceID
FROM [Sales].[Invoices] as i
WHERE i.InvoiceID like '1%'
```

In the XML, in the node for the Index Scan, you will see:

```
<RunTimeInformation>
<RunTimeCountersPerThread Thread="0" ActualRows="11111" ActualRowsRead="70510"
...

```

Defined as "ActualRowsRead" in the XML of the plan, this value is displayed as "Number of Rows Read" in SQL Server Management Studio. Similarly, "ActualRows" is displayed as "Actual Number of Rows."

Permissions necessary to view execution plans

The user must have permissions to actually run the query, even if they are generating only an Estimated execution plan.

Retrieving the Estimated or Actual execution plan requires the SHOWPLAN permission in each database referenced by the query. The Live Query Statistics feature requires SHOWPLAN in each database, plus the VIEW SERVER STATE permission to see live statistics.

It might be appropriate in your environment to grant SHOWPLAN and VIEW SERVER STATE permissions to developers. However, the permission to execute queries against the production

database may not be appropriate in your regularly environment. If that is the case, there are alternatives to providing valuable execution plan data to developers without production access:

- Consider providing database developers with saved execution plan (.sqlplan) files for offline analysis.
 - Consider also configuring the dynamic data masking feature, which may already be appropriate in your environment for hiding sensitive or personally identifying information for users who are not sysadmins on the server. Do not provide UNMASK permission to developers; assign that only to application users.
- For more information on dynamic data masking, see Chapter 7.

Using the Query Store feature

First introduced in SQL Server 2016, the Query Store provides a practical history of execution plan performance. It can be invaluable for the purposes of investigating and troubleshooting sudden negative changes in performance, by allowing the administrator or developer to identify high-cost queries and the quality of their execution plans.

The Query Store is most useful for looking back in time toward the history of statement execution. The Query Store can also assist in identifying and overriding execution plans by using a feature similar to but different from the legacy plan guides feature.

Inside OUT

How should I force a statement to use a certain execution plan?

Your options for forcing a statement to follow a certain execution plan are either the older plan guides stored procedures or the newer Query Store interface (and its underlying stored procedures) to force an execution plan.

Both options are advanced options for temporary or diagnostic use only. Overriding the query optimizer's execution plan choice is an advanced performance tuning technique. It is most often necessitated by query parameter sniffing.

It is possible to create competing plan guides or Query Store forced plans. This is certainly not recommended because it could be extremely confusing. If you create compete plan guides or Query Store forced plans, it's likely you'll see the Query Store forced plan "win."

In case you are troubleshooting competing plan guides and Query Store forced plans, you can view any existing plan guides and forced query plans with the following DMV queries:

```
SELECT * FROM sys.plan_guides

SELECT *
FROM sys.query_store_query AS qsq
JOIN sys.query_store_plan AS qsp
      ON qsp.query_id = qsq.query_id
WHERE qsp.is_forced_plan = 1;
```

Finally, you could use the `USE PLAN` query hint to provide the entire XML query plan for any statement execution. This obviously is the least convenient option, and like other approaches that override the query analyzer, should be considered an advanced and temporary performance tuning technique.

Plan guides are used to override an otherwise complicated manual scripting exercise.

You see live Query Store data as it happens from a combination of both memory-optimized and on-disk sources. Query Store minimizes overhead and performance impact by capturing cached plan information to in-memory data structure. The data is “flushed” to disk at an interval defined by Query Store, by default 15 minutes. The Disk Flush Interval setting defines how much Query Store data could be lost in the event of an unexpected system shutdown.

NOTE

Cross-database queries are captured according to the query database context. In the following code example, the query’s execution would be captured in the Query Store of the `WideWorldImporters` database.

```
USE WideWorldImporters;
GO
SELECT * FROM
AdventureWorks.[Purchasing].[PurchaseOrders];
```

The Query Store is a feature that Microsoft delivered to the Azure SQL Database platform *first*, and then to the SQL Server product. In fact, Query Store is at the heart of the Azure SQL Database Advisor feature which provides automatic query tuning. The Query Store feature’s overhead is quite manageable, tuned to avoid performance hits, and is already in place on millions of databases in Azure SQL Database.

The `VIEW DATABASE STATE` permission is all that is needed to view the Query Store data.

Initially configuring the query store

The Query Store feature is identical between the two platforms, except for its default activation. Query Store is turned on automatically on Azure SQL Database, but it is not automatically on for new databases in SQL Server 2017, and it is not a setting that can be inherited by the model database.

You should turn on the Query Store on new production databases in SQL Server 2017 when you anticipate doing any performance tuning. You can turn on Query Store via the database Properties dialog box, in which Query Store is a page on the menu on the left. Or, you can turn it on via T-SQL by using the following command:

```
ALTER DATABASE [DatabaseOne] SET QUERY_STORE = ON;
```

Keep in mind that Query Store begins collecting when you activate it. You will not have any historical data when you first turn on the feature on an existing database, but you will begin to immediately see data for live database activity.

The Query Store Capture Mode default setting of All includes all queries. You might soon realize that this setting does not filter out ad hoc queries, even if you selected the Optimize For Ad Hoc Queries option in the system configuration. Change this setting to Auto because the additional data of one-use plans might not be useful, and can reduce the amount of historical data can be retained.

NOTE

The Query Store data is stored in the user database. It is backed up and restored along with the database.

The Query Store retains data up to two limits: a Max Size (500 MB by default), and a “Stale Query Threshold” time limit of Days (30 by default). If Query Store reaches its Max Size, it will clean up the oldest data. Because Query Store data is saved on a drive, its historical data is not affected by the commands we looked at earlier in this chapter to clear the Procedure Cache, such as DBCC FREEPROCACHE.

You should keep the Size Based Cleanup Mode set to the default Auto. If not, when the Max Size is reached, Query Store will stop collecting data and enter “Read Only” mode, which does not collect new data. If you find that the Query Store is not storing more historical days of data than your Stale Query Threshold setting in days, increase the Max Size setting.

NOTE

Starting with SQL Server Management Studio 17.3, you can also see wait stats on existing reports.

Using query store data in your troubleshooting

Query Store has several built-in dashboards, shown in Figure 9-3, to help you examine query performance and overall performance over recent history.

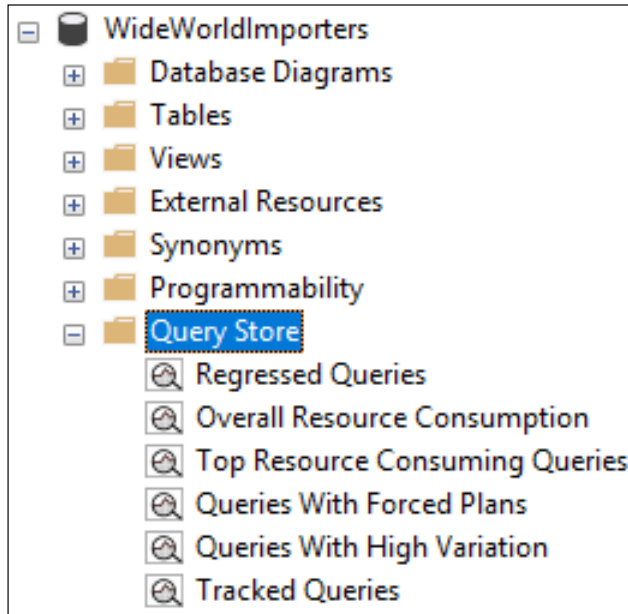


Figure 9-3 The SQL Server Object Explorer list of built-in dashboards available for Query Store in SQL Server Management Studio 2017.

With SQL Server Management Studio 2017, you can view more dashboards in SQL Server 2016 databases than you could in SQL Server Management Studio 2016, including Queries With Forced Plans and Queries With High Variation.

You can also write your own reports against the collection of system DMVs that present Query Store data to administrators and developers by using the `VIEW DATABASE STATE` permission. You can view the six-view schema of well-documented views and their relationships at <https://docs.microsoft.com/sql/relational-databases/performance/how-query-store-collects-data#views>.

On many of the dashboards, there is a button with a crosshairs symbol, as depicted in Figure 9-4. If a query seems interesting, expensive, or is of high value to the business, you can click this button to view a new screen that tracks the query when it's running as well as various plans identified for that query.

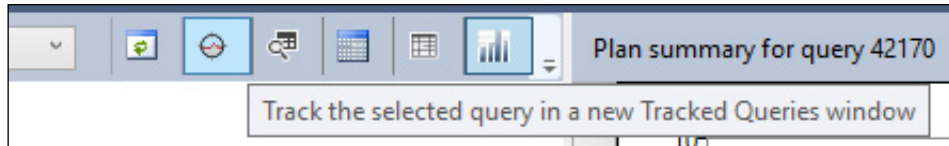


Figure 9-4 The Query Store tool bar at the top of the screen on many of the dashboards, in this example, the tool bar for the Regressed Queries report.

You can also review the various plans for the same statement, compare the plans, and if necessary, force your chosen plan into place. Compare the execution of each plan by CPU Time, Duration, Logical Reads, Logical Writes, Memory Consumption, and Physical Reads.

Most of all, the Query Store can be valuable by informing you when a query started using a new plan. You can see when a plan was generated and the nature of the plan; however, the cause of the plan's creation and replacement is not easily answered, especially when you cannot correlate to a DDL operation. Query plans can become invalidated automatically due to large changes in statistics due to data inserts or deletes, changes made to other statements in the stored procedure, changes to any of the indexes used by the plan, or manual recompilation due to the RECOMPILE option.

Forcing a statement (see Figure 9-5) to use a specific execution plan via the Query Store is not a recommended common activity. You should use this only for specific performance cases, problematic queries demanding unusual plans, workarounds for other unresolvable index or performance scenarios. Note that if the forced plan is invalid, such as an index changing or being dropped, SQL Server will move on without the forced plan without warning or error, though Query Store will still show that the plan is being forced for that statement.

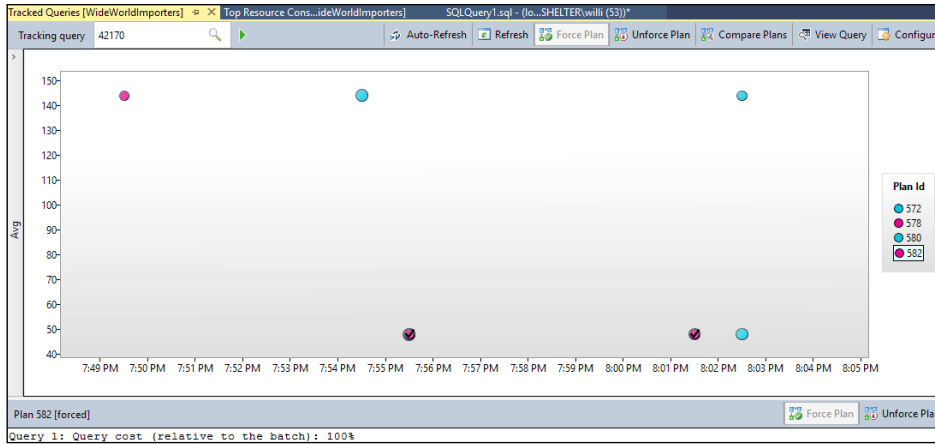


Figure 9-5 The Query Store has recorded the execution results of the query. Note that one plan has been Forced (using the Force Plan button) for this statement and is displayed with a check mark.

Understanding automatic plan correction

SQL Server 2017 introduces a new feature called Automatic Plan Tuning, originally developed for the Azure SQL Database platform. It is capable of detecting and reverting plan regression.

You could use Query Store in 2016 to identify a query that has regressed in performance, and manually force a past execution plan into use. Now in SQL Server 2017, the database can be configured to detect plan regression and take this action automatically. The sample syntax for enabling automatic plan correction is below:

```
ALTER DATABASE [WideWorldImporters] SET AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = ON );
```

Currently, `FORCE_LAST_GOOD_PLAN` is the only option for automatic plan tuning.

The DMV `sys.dm_db_tuning_recommendations` captures plan recommendations based on query performance regression. This doesn't happen immediately – the feature has an algorithm that requires several executions before regression is identified. When a recommendation appears in `sys.dm_db_tuning_recommendations`, it includes a large amount of diagnostic data, including a plain-language “reason” explanation for the recommendation to be generated, and a block of JSON data containing diagnostic information. A sample query to parse this data is available at <https://docs.microsoft.com/sql/relational-databases/automatic-tuning/automatic-tuning>.

Understanding execution plan operators

After you have a graphical execution plan in front of you, you can begin to understand how the statement is processed.

To display information for individual steps, position your pointer over a step in the execution plan. You can also click an object, and then open the Properties window by pressing F4 or, in the View menu, clicking Properties Window. You'll notice that the information returned estimate and actual values for some metrics, including Number of Rows and Executions. Look for differences here; they can indicate an inefficient execution plan and the source of a poor performing query. Your query might be suffering from a poorly chosen plan because of the impact of parameter sniffing or stale, inaccurate index statistics. (We discussed parameter sniffing earlier in this chapter, and discuss index statistics in Chapter 10.)

However, notice that some values, like Cost information, contain only Estimated values, even when you are viewing the Actual execution plan. This is because the operator costs aren't sourced separately, they are generated the same way for both Estimated and Actual plans, and do not change based on statement execution. Furthermore, cost is not just comprised entirely of duration. You might find that some statements far exceed others in terms of duration, but not in cost.

There are even known plan presentation issues (as recent as SQL Server Management Studio 17.1) that might sometimes result in a sum of Operator Costs that do not add up to 100 percent, specifically in the presence of the concatenation operator.

Interpreting graphical execution plans

In the next list, we review some of the most common things to look for as you review execution plans in SQL Server Management Studio. You can also choose to review execution plans with a well-known third-party tool called Plan Explorer, which is a free download from <https://www.sentryone.com/>.

In this section, it is assumed that you will have access to the Actual execution plan, as not all the information within will exist in the Estimated plan.


Start in the upper left

The upper-left operator will reflect the basic operation that the statement performed. For example, Select, Delete, Update, or Insert for DML statements. This operator might contain warnings or other items that require your immediate attention. These might show up with a small yellow triangle warning icon, with additional detail when you position your pointer on the operator.

Click the upper-left operator, and then press F4 to open the Properties window, or open the Properties window from the View menu in SQL Server Management Studio. In this list are a couple other things to look for. You'll see warnings repeated in here, along with additional aggregate information.

ATTENTION



Yellow triangles () indicate something that should grab your attention. The alert could tip you off to an implicit conversion—a data type mismatch that could be costly! Investigate any warnings reported before moving on.

Look also for the Optimization Level, which ideally says FULL. If the Optimization Level was TRIVIAL, the plan bypassed the query optimizer altogether because it was too straightforward. The plan contained only a simple Scan or Seek operation the only other operator, perhaps. If not FULL or TRIVIAL, this is something to investigate.

Look next for the presence of a value for Reason For Early Termination, which indicates the query optimizer spent too long on attempting to build the perfect execution plan, and gave up, sometimes literally returning the self-explanatory value, Good Enough Plan Found. If the reason is Time Out, the optimizer tried as many times as it could to find the best plan before deciding, taking the best plan available, which might not be “good enough.” If you see this case, consider simplifying the query, especially reducing the use of functions, and by potentially modifying the underlying indexes. Finally, if you see the reason is Memory Limit Exceeded, this is a rare and critical error indicating severe memory pressure on the SQL Server instance.

In the Query Cached Plan Stats script sample shown in the section “Analyzing cached execution plans in aggregate” earlier in this chapter, in which we queried the procedure cache for plan statistics, you can add some code to search only for queries that have a Reason For Early Termination. In the execution plan XML, the Reason For Early Termination will show in a node `StatementOptmEarlyAbortReason`. Before the `WHERE` clause, add this line:

```
CROSS APPLY sys.dm_exec_text_query_plan(p.plan_handle, qs.statement_start_offset,
qs.statement_end_offset) AS tqp
```

And before the `ORDER BY` in the script, add this line:

```
and tqp.query_plan LIKE '%StatementOptmEarlyAbortReason%'
```

Next, scroll right, then read from right to left

Graphical execution plans build from sources (rightmost objects), and apply operators to join, sort, and filter data from right to left, eventually arriving at the leftmost operator. In the rightmost objects, you'll see Scans, Seeks, and Lookups of different types. You might find some quick, straightforward insight into how the query is using indexes.

Seek operations are best for when you're looking for a needle or needles in a much larger haystack. They are generally the most efficient operators to see, and can rarely be improved by additional indexes. Keep an eye out for Seeks that are accompanied by Lookups, however. They'll likely appear one on top of the other in the graphical execution plan. Row Lookups indicate that although the optimizer used a seek, it needed a second pass at the table in the form of a Lookup on another object, perhaps the clustered index. Key Lookups (on clustered indexes) and RID Lookups (on heaps) are expensive and inefficient, and likely can be eliminated from the execution plan with the modification to an existing nonclustered index. Lookups are very efficient when looking up a small number of rows, but very inefficient for larger number of rows. In high-cost or high-importance queries, Key Lookups can represent a significant cost, one that is easily resolvable with a nonclustered index.

- For an example, see the section "Designing nonclustered indexes" in Chapter 10.

Scan operations aren't great unless your query is intentionally performing a query that returns most of the rows out of a table. Scans are in fact that most efficient option for when an index does not provide an ordered dataset, but keep in mind, they do read all rows from the index. Without a nonclustered index with a well-designed key to enable a seek for the query, a scan might be the query optimizer's only option. Scans on nonclustered indexes are often better than scans of clustered indexes, in part due to what is likely a smaller key size. Test and compare the performance of a new or updated nonclustered index, created based on the predicates and outputs of Index Scans and Clustered Index Scans.

NOTE

Again, very few queries are important enough to deserve their own indexes. Think "big picture" when creating indexes. More than one query should benefit from any nonclustered indexes you create. Avoid redundant or overlapping nonclustered indexes. See Chapter 10 for more information on creating nonclustered indexes, including "missing" indexes.

Other types of scans include the following:

- **Table Scans.** These indicate that the table has no clustered index. We discuss why this is probably not a good idea in Chapter 10.

- **Remote Scans.** This includes any object that is preceded by “remote,” which is the same operation but over a linked server connection. Troubleshoot them the same way, but potentially by making changes to the remote server instead.
- **Constant Scans.** These appear when the query optimizer deals with scalar values, repeated numbers, and other “constants.” These are necessary operators for certain tasks and generally not actionable from a performance standpoint.
- **Columnstore Index Scans.** These are incredibly efficient operators, and likely will outperform a Clustered Index Scan or Index Seek where millions of rows, for example, must be aggregated. No need to create a nonclustered index to replace this operator.

NOTE

Since SQL Server 2016, Columnstore indexes are a viable option for read-write tables in a transactional system. In previous versions of SQL Server, nonclustered Columnstore indexes did not allow writes to the table, and so couldn't easily be adopted in transactional databases. If you aren't using them already to optimize large row count queries, considering adding them to your toolbox.

Furthermore, since SQL Server 2016 SP1, Columnstore indexes are even available to all edition licenses of SQL Server, even Express edition, though editions below Enterprise edition have limits to the amount of Columnstore cache in memory. For more information, visit <https://docs.microsoft.com/sql/sql-server/editions-and-components-of-sql-server-2017>.

The weight of the lines connecting operators isn't the full story

SQL Server dynamically changes the thickness of the gray lines to reflect the Actual Number Of Rows. You can get a visual idea of where the bulk of data is coming from by observing the pipes, drawing your attention to the places where performance tuning could have the biggest impact.

The visual weight and the sheer number of rows does not directly translate to cost, however. Look for where the pipe weight changes from light to heavy, or vice versa. Be aware of when thick pipes are joined or sorted.

Operator cost share isn't the full story either

When you run multiple queries, the cost of the query relative to the batch is displayed in the Query Execution Plan header, and within each plan, the batch cost relative to the rest of the operators in the statement is displayed. SQL Server uses a cost-based process to decide which query plan to use. Deciding to address only the highest-cost single operator in the execution plan might be a dead end, but generally you will find the highest cost operators on the right-most side of the execution plan.

In Figure 9-6, we can see that operator cost might not align with the amount of data. You should investigate performance tuning this execution plan using all of the information provided.

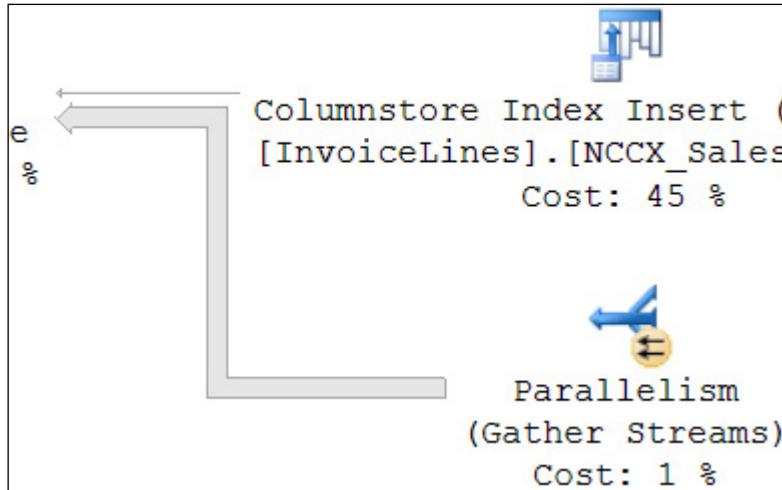


Figure 9-6 In this snippet from an execution plan, much of the cost is associated with the top operator, but more rows are moving on from the bottom operator.

Look for Join operators and understand the different algorithms

As you read from right to left, in a query of any complexity, you'll likely see the paths meet at a join operator. Two tables can be joined, obviously, but different indexes on the table can also meet in a join operator. If you find that a large portion of the cost of an execution plan spent in a Hash Match, Hash Join, Merge Join, or Nested Loop, take a look at what is being joined.

The Hash operators have the most overhead, with a temporary hash table created to bucketize and match rowdata. Merge Joins are the best for ordered data that streams processed data as it receives it. Nested Loops aren't as bad as they sound, but they are essentially the row-by-row comparison of one rowset against another. This can be very efficient for small, indexed datasets.

Each of the following could reduce the cost of a Join operator.

- There may be an opportunity to improve the indexing on the columns being joined, or perhaps, you have a join on a compound key that is incompletely defined. Perhaps you are unintentionally omitting part of the join key in the ON or WHERE clause of the query.

- In the case of a Merge Join, you may see a preceding Sort operator. This could be an opportunity to present the data already sorted according to how the Merge Join requires the data to be sorted. Perhaps changing the ASC/DESC property or the order of index key columns could remove the Sort operator.
- Make sure you that are filtering at the lowest level possible. Perhaps a WHERE clause could exist in a subquery instead of at the top level of the query, or in the definition of a common table expression (CTE) instead of in the lower query.
- Hash Match and Hash Join operators are the most expensive, but are the typically the most efficient for joining two large row sets, especially large unsorted datasets. Reducing the row counts going into the Hash Match or Hash Join could allow the query optimizer to use a less memory-intensive and less costly join operator. You could accomplish this perhaps by adding or modifying nonclustered indexes to eliminate Scan operators in favor of Seek operators.
- Nested Loops are often necessitated by Key Lookups and sometimes quite costly. They too are no longer necessary if a new nonclustered index is added to address the Key Lookup and make an accompanying Index Seek more capable.

Look for Parallel icons

The left-pointing pair of arrows in a yellow circle shown in Figure 9-7 indicate that your query has been run with a parallel-processing execution plan. We talk more about Parallelism later in this chapter, but the important thing here is to be aware that your query has gone parallel.

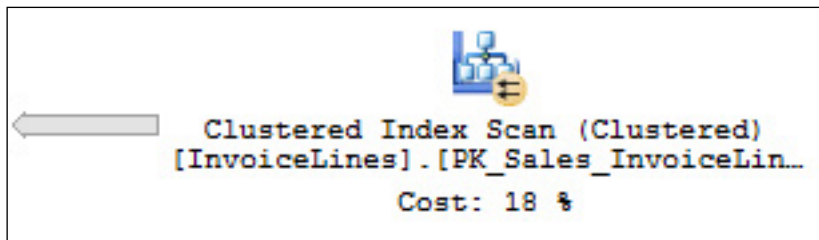


Figure 9-7 the parallel indicator on a Clustered Index Scan operator.

This doesn't mean that multiple sources or pipes are being read in parallel; rather, the work for individual tasks has been broken up behind the scenes. The query optimizer decided it was faster if your workload was split up and run into multiple parallel streams of records.

You might see one of the three different Parallelism operators—the distribute streams, gather streams, and repartition streams operators—each of which appear only for parallel execution plans.

Forcing a parallel execution plan

New to SQL Server 2017 (and also implemented in SQL Server 2016 CU2) is a query hint that can force a statement to compile with a parallel execution plan. This can be valuable in troubleshooting, or to force a behavior in the query optimizer, but is not usually a necessary or recommended option.

Appending the following hint to a query will force a parallel execution plan, which you can see using the Estimate or Actual execution plan output options:

```
OPTION(USE HINT('ENABLE_PARALLEL_PLAN_PREFERENCE'));
```

NOTE

The presence of certain system variables or functions can force a statement to compile to be serial, that is, without any parallelism. This behavior will override the new `ENABLE_PARALLEL_PLAN_PREFERENCE` option.

The `@@TRANCOUNT` system variable will force a serial plan, as will any of the built-in error reporting functions, including `ERROR_LINE()`, `ERROR_MESSAGE()`, `ERROR_NUMBER()`, `ERROR_PROCEDURE()`, `ERROR_SEVERITY()`, or `ERROR_STATE()`. Note that this pertains only to using these objects in a query. Using them in the same batch, such as in a `TRY ... CATCH` handler, will not affect the execution plans of other queries in the batch.

Understanding parallelism

We mentioned parallelism in execution plans earlier in this chapter. When SQL Server decides to split and stream data needed for requests into multiple threads, it uses more than one processor to get the job done. The number of different parallel threads used for the query is called the degree of parallelism. Because parallelism can never exceed the number of logical processors, naturally the maximum degree of parallelism (`MAXDOP`) is capped.

The default `MAXDOP` setting of 0 (allowing all processors to be used in a single statement) allows SQL Server to “go parallel” at will, and, sometimes, to a fault. Although queries may perform fastest in a vacuum going massively parallel, at scale the overuse of parallelism creates a multithreaded bottleneck. Split into too many different parts, queries slow down en masse as CPU utilization rises and SQL Server records increasing values in the `CXPACKET` wait type.

- We talk about `CXPACKET` here, but for more about wait type statistics, see Chapter 13.

Until SQL Server 2016, MAXDOP was a server-level setting, or a setting enforced at the query level, or a setting enforced to sessions selectively via the Resource Governor, an Enterprise edition feature. Since SQL server 2016, the MAXDOP setting is now available as a database-scoped configuration. You can also use the MAXDOP query hint in any statement to override the database or server level MAXDOP setting.

Another limit to parallelism, called the Cost Threshold for Parallelism (CTFP), enforces a minimum bar for query cost before a query can use a parallel execution plan. The higher the threshold, the fewer queries go parallel. This setting is fairly low by default, but its proper setting in your environment is quite dependent on the workload and processor count. More expensive queries usually benefit from parallelism more than simpler queries, so limiting the use of parallelism to the worst queries in your workload can help. Similarly, setting the CTFP too high could have an opportunity impact, as performance is limited, queries are executed serially, and CPU cores go underutilized. The CTFP is a server-level setting only.

If large queries are already a problem for performance and multiple large queries regularly run simultaneously, raising the CTFP might not solve the problem. In addition to the obvious solutions of query tuning and index changes, including the introduction of Columnstore indexes, use MAXDOP instead to limit very large queries.

When the CXPACKET wait is the predominant wait type experienced over time by your SQL Server, both MAXDOP and CTFP are dials to turn when performance tuning. You can also view the live and last wait types for a request using `sys.dm_exec_requests`. Make these changes in small, measured gestures, and don't overreact to performance problems with a small number of queries. Use the Query Store to benchmark and trend the performance of high-value and high-cost queries as you change configuration settings.

Another flavor of CPU pressure, and in some ways the opposite of the CXPACKET wait type, is the `SOS_SCHEDULER_YIELD` wait type. The `SOS_SCHEDULER_YIELD` is an indicator of CPU pressure, indicating that SQL Server had to share time or "yield" to other CPU tasks, which may be normal and expected on busy servers. Whereas CXPACKET is the SQL Server complaining about too many threads in parallel, the `SOS_SCHEDULER_YIELD` is the acknowledgement that there were more runnable tasks for the available threads. In either case, first take a strategy of reducing CPU-intensive queries and rescheduling or optimizing CPU-intensive maintenance operations. This is more economical than simply adding CPU capacity.

Inside OUT

How can I reduce the processor utilization during maintenance operations?

If processor utilization spikes and during maintenance operations such as index maintenance or integrity checks, you can force these to run serially. Although this can increase the duration of maintenance, other queries should be less negatively affected.

You can use the MAXDOP query hint at the end of index maintenance to force index rebuild steps to run serially. Combined with the ONLINE hint, an Enterprise edition feature, your scripted index maintenance might run longer but have a minimal impact of concurrent queries. You can also specify MAXDOP when creating indexes. You cannot specify a MAXDOP for the reorganize step.

```
ALTER INDEX ALL ON WideWorldImporters.Sales.Invoices REBUILD  
WITH (MAXDOP = 1, ONLINE = ON);
```

You can also turn on trace flag 2528 to disable parallelism server-wide for DBCC CHECKDB, DBCC CHECKFILEGROUP, and DBCC CHECKTABLE operations. Keep in mind these operations can take hours to complete on large databases, and might run longer if single-threaded.

This page intentionally left blank



Index

A

- ABORT_AFTER_WAIT** parameter 566
- access control**
 - role-based 223
 - single sign-on (SSO) 222
- actions** 585
- Active Directory Organizational Unit (OU)** 507
- Activity Monitor**
 - Active Expensive Queries section 36
 - Data File I/O section 36
 - overview of 33
 - Processes section 34
 - Recent Expensive Queries section 36
 - Resource Waits section 35
- actual execution plans** 408
- ad hoc queries** 105
- Advanced Encryption Standard (AES)** 306, 478
- affinity masks** 105
- alerts**
 - performance conditions 620
 - recommendations for 618
 - SQL Server events 619
 - WMI event alert conditions 621
- alphanumeric data types** 334
- ALTER ANY EVENT SESSION** permission 262
- ALTER ANY USER** permission 257
- ALTER AUTHORIZATION** 266
- ALTER TABLE** statements 352
- ALTER TRACE** permission 262
- Always Encrypted** 310
- Always On** availability groups 64, 255, 515, 636
- AlwaysOn_health** event session 555, 586
- antivirus** software, configuring 159
- approximate numeric types** 335
- articles** 497
- artificial intelligence** 138
- AS** clause 353
- asymmetric keys** 244, 477
- ASYNC_NETWORK_IO** 580
- AT TIME_ZONE** function 337
- auditing and threat detection**
 - auditing defined 319
 - Azure SQL Database 224, 331
 - SQL Server Audit 319
 - threat detection feature 318
- authentication**
 - authentication to SQL Server on Linux 245
 - Certificate-Based Authentication 244
 - integrated authentication and Active-Directory 68
 - Kerberos authentication 69
 - mixed mode 141
 - two-factor authentication 243
 - types of 242
- authorization, vs. ownership** 265
- autoclose** 184
- autocreate statistics** 184
- autogrowth** events 572, 590
- automatic checkpoints** 188
- automatic failovers** 524
- Automatic Plan Tuning** feature 418
- automation**
 - administering multiple SQL Servers 638
 - components of automated administration 607
 - maintaining SQL Server 623
 - SQL Server Agent 612
 - SQL Server Maintenance Plans 625
 - using PowerShell 648
- autoshrink** database setting 185, 575, 627
- availability groups (AGs)** 64
 - alerting 556
 - availability group administration 548
 - backups on secondary replicas in 636
 - basic availability groups 517
 - capabilities of 503
 - checklist 529
 - configuring 513
 - creating WSFC for use with 519
 - database mirroring endpoint 520
 - distributed availability groups 518
 - failovers in 524
 - full database and log backup 528
 - hybrid availability group topology 537
 - load balanced read-only routing 536
 - managing multiserver administration in 641
 - minimum synchronized required nodes 520
 - None option (clusterless) 517
 - ownership of 514
 - Powershell module and 656
 - RegisterAllProvidersIP and MultiSub-NetFailover 535
 - secondary replica availability mode 521, 531
 - seeding options when adding replicas 525
 - SQL Server Agent automation and 621
- Available Memory** 599
- Average Disk seconds per Read or Write** 597
- Azure Active Directory (Azure AD)**
 - authentication, integrated 244
 - authentication, password 244
 - authentication, universal 243
 - benefits of 71
 - hybrid cloud with 121
 - integrated authentication and 68
 - Kerberos authentication 69
 - Linux and 68
 - security offered by 72
- Azure Analysis Services** 217
- Azure Automation** 216
- Azure Backup** 237
- Azure Blob Storage** 111, 138, 231, 472
- Azure CLI, creating databases using** 212
- Azure Cloud Shell** 199
- Azure Data Factory** 217

Azure Data Lake 218

Azure ExpressRoute 125

Azure Key Vault 305

Azure portal

- creating databases using 210
- creating servers using 205
- PowerShell and 199, 206

Azure Resource Manager 327, 660

Azure Role-Based Access Control (RBAC) 224

Azure SQL Database

- auditing 319
- Azure governance 200
- Azure management 199
- benefits of 116, 197
- cloud-first approach 202
- compared to SQL Server 117
- database as a service concept 198
- database corruption handling 561
- Database Transaction Units (DTUs) 202
- disaster recovery preparation 229
- elastic database pools 118
- firewall protection 318
- hybrid availability group topology 537
- hybrid cloud with Azure 121
- limitations of 117, 215
- logical SQL Servers 201
- managed instances 218
- migrating logins from one server to another 289
- moving to 239
- other SQL Server services 216
- pricing tiers and service objectives 213
- provisioning, considerations for 197
- provisioning databases 209
- provisioning elastic pools 214
- provisioning logical SQL servers 204
- recovery strategies 491
- scalability 203, 214
- securing 326
- security considerations 218
- service tiers 118
- sharding databases with Split-Merge 121
- sign in security 72
- Threat Detection 318
- using PowerShell module with 660

Azure SQL Database Import Service 239

Azure SQL Data Warehouse, benefits of 117

Azure Stack 124

Azure Storage 114

Azure Virtual Machines (Azure VMs) 111

Azure Virtual Network (VNet) 124

B

Back Up Database task 632

backup disks 470

backups. *See also high availability (HA)*

- Azure Backup 237
- backup chains 466, 476
- backup creation and verification 478
- backup types 472
- backup vs. restore strategies 459
- DBCC CHECKDB and 558
- encrypting 478
- fundamentals of data recovery 460
- manual (Azure SQL Database) 230
- on secondary replicas in availability groups 636
- physical backup devices 470
- post-installation configuration settings 161
- RAID and 55
- recovery strategies 487
- restore strategies 459
- restoring 175
- scheduling 623, 631

backup sets 471

Backup-SQLDatabase cmdlet 653

BACPAC files 177, 230, 239

base table elimination. *See partition elimination*

basic availability groups 67, 517

Batch Mode execution 447

Batch Requests 599

bigint data type 336

binary data type 339

binary large objects (BLOBs) 367

blocked_by column 387

blocking 386. *See also isolation levels and concurrency*

boot page 83

Border Gateway Protocol (BGP) 298

bring-your-own-license (BYOL) VM 135

broadcast 297

broken recovery chains 638

brute-force attacks 294

B-tree structure 437

Buffer Cache Hit Ratio (BCHR) 598

Buffer Manager 46

buffer pools 46, 479

buffer pool extension 47

BUILTIN\Administrators group 254

bulkadmin server role 275

Bulk Changed Map (BCM) 83

Bulk Copy Program (BCP) 6, 9

bulk-logged recovery model 464, 469

Business Intelligence edition, appropriate use of 132

C

capital expenditures (CapEx) 198

cascading 346

Central Management Server (CMS) 26

Central Processing Unit (CPU)

- core counts and affinity masks 105

- core counts and editions 51
- core speed 49
- multiple 49
- Non-Uniform Memory Access 50
- power saving disablement 51
- simultaneous multithreading (SMT) 49, 75
- virtualizing CPUs 75
- Certificate-Based Authentication** 244
- Certification Authority (CA)** 302
- change data capture** 380
- CHANGETABLE function** 380
- change tracking** 378
- char column** 335
- check constraints** 347
- CHECKDB** 558
- checkpoint process** 89, 188
- CHECK_POLICY option** 251
- checksum verification** 84, 174, 187, 480, 557, 632
- claims** 71
- classification** 99
- cloud computing**
 - cloud-first approach 202
 - hybrid cloud with Azure 121
 - key features of 198
 - networking as foundation of 58
 - scalability 203
 - virtualization and 73
- clustered indexes**
 - case against intentionally designing heaps 433
 - choosing proper clustered index keys 429
 - design choices 432
 - function of 429
- clustering** 61
- Code Snippets Manager** 29
- collation** 181, 335
- colocation constraint** 548
- Column Encryption Keys (CEK)** 312
- Column Master Key (CMK)** 312
- Columnstore** 48, 102
- Columnstore indexes**
 - architecture of 447
 - Batch Mode execution 447
 - benefits of 446
 - clustered and nonclustered Columnstore indexes 447
 - compression delay 449
 - key improvements to 447
 - power of 448
 - reorganizing 571
- command-line interface** 9
- Common Language Runtime (CLR)** 339

- compatibility mode** 170, 182
- components.** *See* **database infrastructure**
- Compress Backup** 632
- compression delay** 449
- compression information (CI) structure** 95
- computed columns** 352
- concurrency, optimistic vs. pessimistic** 342, 399. *See also* **isolation levels and concurrency**
- Configuration Checker** 3, 136
- configuration settings**
 - affinity masks 105
 - file system 107
 - memory settings 102
 - page file (Windows) 99
 - parallelism 100
 - post-installation checklist 151
 - using Resource Governor 98
- CONNECT ALL DATABASE permission** 264
- constraints** 346
- contained databases** 183, 256
- CONTAINMENT** 256
- CONTROL SERVER/DATABASE permission** 265
- COPY_ONLY option** 474
- corruption**
 - detecting 557
 - recovering transaction log files 560
 - repairing 560
- Cost Threshold for Parallelism (CTFP)** 426
- crash recovery.** *See* **recovery**
- create custom server roles** 277
- CREATE SEQUENCE command** 348
- CREATE TABLE statement** 351
- CREATE TYPE statement** 350
- credentials** 305, 612
- credit card information** 310
- Cumulative Updates (CUs)** 604
- CXPACKET** 581
- CXPACKET wait** 426

D

- data analytics** 138
- database as a service (DBaaS)** 116, 198
- database availability groups (DAG)** 503
- database checkpoints** 88
- Database Encryption Key (DEK)** 303
- Database Engine** 24
- Database Engine Tuning Advisor** 12
- database infrastructure** 45–78, 79–126
 - Azure and the data platform 110
 - Central Processing Unit (CPU) 49
 - configuration settings 98
 - connecting to SQL Server over networks 57
 - data storage 51
 - high availability concepts 59
 - hybrid cloud 121
 - memory 45
 - physical database architecture 79–126
 - server access security 68
 - virtualization 73
- Database Mail**
 - configuration options 609
 - email history 610
 - key features 607
 - set up 608
 - test email 609
 - troubleshooting 610
- database management**
 - capturing Windows performance metrics 592
 - detecting database corruption 557
 - maintaining database file sizes 571
 - maintaining indexes and statistics 561
 - monitoring databases by using DMVs 575
 - Policy-Based Management (PBM) 643
 - product life cycle model 604
 - protecting important workloads 600
- Database Master Key (DMK)** 303, 307
- database mirroring** 64, 505, 520
- database ownership** 265
- database properties and options**
 - autoclose 184
 - autocreate statistics 184
 - collation 181
 - compatibility level 182
 - containment type 183
 - Database-Scoped Configurations 187
 - indirect checkpoints 188
 - page verify 187
 - Query Store 188
 - read-only 187
 - recovery model 182
 - reviewing database settings 181
 - single-user mode 195
 - Snapshot Isolation mode 186
 - Trustworthy setting 187
- database roles** 278
- databases**
 - considerations for migrating existing 169
 - contained databases 183, 256
 - creating 177
 - migrating master database 290
 - moving and removing 189

- moving existing 175
- physical database architecture 79
- properties and options 181
- provisioning Microsoft Azure SQL databases 197–240
- provisioning Microsoft SQL Server databases 127–196
- setting default for logins 250
- Database-Scoped Configurations** 173, 187
- database snapshots** 473
- Database Transaction Units (DTUs)** 117, 202
- Datacenter edition, appropriate use of** 132
- data collectors** 592
- data compression**
 - backup compression 96
 - dictionary compression 95
 - leaf-level vs. non-leaf-level pages 94
 - page compression 94
 - prefix compression 95
 - purpose of 93
 - row compression 93
 - Unicode compression 96
- Data Control Language (DCL)** 259, 378
- Data Definition Language (DDL)** 257, 378
- Data Definition Language (DDL) events** 555
- Data Encryption Standard (DES)** 306
- data files and filegroups**
 - backups 477
 - checkpoint process 89
 - checksum verification 84
 - data page types 82
 - extents, mixed and uniform 81
 - file unit allocation size 130
 - locating SQL Server files 190
 - maintaining database file sizes 571
 - memory-optimized objects 84
 - MinLSN and the active log 91
 - multiple instances of 80
 - partial recovery and 81
 - primary filegroup 80
 - restarting with recovery 91
 - separating SQL Server files 130
 - shrinking database files 574
- datagrams** 297
- data in motion, securing** 314
- Data Manipulation Language (DML)** 230, 257
- data masking** 317
- Data Migration Assistant** 4, 136
- Data Platform**
 - Azure Blob Storage 111
 - Azure VMs, performance optimization 111
 - Azure VMs, locating TempDB files on 116
 - bandwidth considerations 113
 - drive caching 114
 - infrastructure as a service (IaaS) 110
 - platform as a service (PaaS) 116
 - SQL Server data files 114
 - virtual hard drives (VHDs) 112
 - virtual machine sizing 115
- Data Profiling Task** 43
- Data Protection API (DPAPI)** 303
- Data Quality Client** 8
- Data Quality Server** 8
- Data Quality Services** 7
- data recovery**
 - backup creation and verification 478
 - backup types 472
 - backup vs. restore strategies 459
 - fundamentals of 460
 - physical backup devices 470
 - recovery strategies 487
- data storage** 51–57. *See also* **data files and filegroups**
 - commonly used terms 51
 - drives 52
 - Fibre Channel vs. iSCSI 56
 - IOPS (input/output operations per second) 52
 - latency 52
 - Network-Attached Storage (NAS) 56
 - nonvolatile storage disks vs. drives 51
 - overcommitting 75
 - queue depth 52
 - SMB 3.0 file share 57
 - Storage-Area Network (SAN) 56
 - storage arrays and RAID 54
 - storage layer configuration 53
 - Storage Spaces 57
 - types of 52
 - volumes 52
- date and time data types** 336
- date data type** 337
- datetime2 data type** 336
- datetime data type** 336
- datetimeoffset data type** 337
- Daylight Saving Time (DST)** 337
- day-to-day maintenance** 623
- db_accessadmin** role 280
- db_backupoperator** role 280
- DBCC CHECKDB** 481, 558, 624, 626
- DBCC CHECKDB REPAIR_ALLOW_DATA_LOSS** 560
- DBCC SHRINKFILE** 575
- dbcreator** server role 275
- db_datareader** role 280
- db_datawriter** permission 280
- db_ddladmin** role 281
- db_denydatareader** role 281
- db_denydatawriter** role 281
- db_owner** database role 279
- db_securityadmin** role 281
- deadlocks** 589
- decimal-point numbers** 336
- Dedicated** 99
- Dedicated Administrator Connection (DAC)** 283
- default constraints** 347
- defense-in-depth**
 - defined 291
 - securing your environment with 292
- delayed durability** 65, 85, 400
- deprecated features, identifying** 5, 44
- Developer edition, appropriate use of** 132
- dictionary attacks** 294
- differential backups** 475, 483
- differential bitmap** 474
- Differential Changed Map (DCM)** 83
- digital certificates** 301
- Direct-Attached Storage (DAS)** 53
- dirty reads** 385
- disaster recovery (DR).** *See also* **data recovery**
 - Azure SQL Database preparations 229
 - compared to high availability (HA) 60, 494, 506
 - configuring failover cluster instances for 502
 - overview of 493
 - typical scenario 460, 488
- diskadmin** server role 275
- Disk Usage report** 572
- distributed availability groups** 67
- distributed-denial-of-service (DDoS) attacks** 331
- distributors** 497
- DML statements** 365
- DMV (dynamic management views)** 548, 575, 592
- domain security groups** 261
- double-byte character sets (DBCS)** 335
- double-hop issue** 70
- drives** 52. *See also* **data storage**
 - mechanical hard drives 52
 - solid-state drives 53
 - types of 52
- drive starting offset** 131
- dynamic data masking** 317

dynamic management
 function (DMF) 563
 dynamic quorum management 509

E

e-commerce 300
Edition Upgrade Wizard 136
elastic database pools
 benefits of 118
 best use case for 119
 database consolidation 119
 elastic database jobs 120
 elastic database query 120
 multitenant architecture 119
elastic DTUs (eDTUs) 118
elasticity 198
emails 607, 632
emojis 335
encryption
 Always Encrypted 310
 backup encryption 478
 defined 294
 deterministic vs. randomized 311
 in SQL Server 302
 network security and 58
 process of 295
 symmetric and asymmetric 300
 transparent data encryption (TDE) 308
Enforce Password Policy 250
Enterprise edition, appropriate use of 131
Entity Framework 342
error logs 32
estimated execution plans 408
ESTIMATEONLY parameter 559
ETW (Event Tracing for Windows) 588
event counter 589
event forwarding 642
event-handling, extended events
 GUI 13
events 585
exact numeric types 335
execution plan operators
 Clustered Index Scans 421
 Columnstore Index Scans 422
 Constant Scans 422
 displaying individual steps 419
 Good Enough Plan Found 420
 Index Scans 421
 interpreting graphical execution plans 419
 Join operators 423
 Key Lookups 421
 lookup operations 421
 Memory Limit Exceeded 420

operator cost share 422
 Optimization Level 420
 ORDER BY 420
 Parallel icons 424
 Query Cached Plan Stats 420
 Reason For Early Termination 420
 Remote Scans 422
 RID Lookups 421
 rightmost objects 421
 Row Lookups 421
 scan operation 421
 seek operations 421
 Table Scans 421
 thickness of gray connector lines 422
 upper-left operator (basic operation) 420
 yellow triangles 420
execution plans
 analyzing cached execution plans in aggregate 405
 clearing the Procedure Cache 406
 enforcing 413
 parameterization and “parameter sniffing” 402
 permissions necessary to view execution plans 412
 permissions required to access cached plan metadata 406
 Procedure Cache 404
 purpose of 401
 retrieving 408
Export Registered Servers Wizard 25
Express edition, appropriate use of 132
ExpressRoute 125
Express with Advanced Services, appropriate use of 132
extended events
 AlwaysOn_health session 555
 autogrowth event detection 590
 benefits of 584
 deadlock detection 589
 page_split event identification 563, 591
 securing 591
 targets 587
 terminology used 585
 viewing event data 586
 XEvent Profiler tool 584
Extended Events GUI 13
Extensible Key Management (EKM) 303
external tables 361, 457
Extract, Transform, and Load (ETL) 378

F

FacetDomainControllerCheck 3

FacetWOW64PlatformCheck 3
Failover Cluster Instance (FCI) 57, 61, 500, 505, 507
failover groups 235
feature parity, identifying 5
Feature Selection page
 Machine Learning Services 139
 Oracle SE Java Runtime Environment (JRE) 138
 PolyBase Query Service 138
federation 72
fencing agents 539, 546
Fibre Channel (FC) 56
File Allocation Table (FAT) 107
file backups 477
filegroups. See data files and filegroups
file header page 83
FILEPROPERTY function 572
files. See data files and filegroups
file sharing protocols 57. *See also data storage*
FILESTREAM 339, 346, 367
file system, configuration settings 107
FileTable 369
File Transfer Protocol 299
Fill Factor property 561
filter drivers 481
filtered unique index 347
firewalls 219, 318, 328
flash memory 53
float data type 336
fn_hadr_backup_is_preferred_replica function 517
forced failovers 525
foreign keys 345
full database backups 473, 483
full recovery model 464, 468, 487
full-text indexes 452
Full-Text Search feature 452
function permissions 267

G

General Availability (GA) release 604
General Data Protection Regulation (GDPR) 291
General Distribution Releases (GDRs) 604
generic data types 333
geography data type 339
geometry data type 339
geo-replication 232
geo-restore 230, 492
Get-ChildItem cmdlet 654
Global Allocation Map (GAM) 83
globally unique identifier (GUID) 343

GO statement 351

Grant Perform Volume Maintenance

Task Privileges 139

graphical execution plans 419

graph tables 362

GUID Partition Table (GPT) 130

H

Hadoop nonrelational data 138

hard drives 52. *See also* data storage

HardMemoryLimit 154

Hardware Security Module (HSM) 304

hash indexes 450

hashing 294

headers 297

heap structures 433

hierarchical data 363

hierarchyid data type 339, 344

high availability (HA) 59–68

availability group administration 548

availability group alerting 556

availability group checklist 529

availability group configuration 513

availability group endpoints 520

availability groups and failovers 524

availability groups and WSFC 519

availability group seeding options 525

availability groups (AGs) 64, 503

clustering 61

defined 59

disaster recovery (DR) and 60, 494

effort and investment required for 493

failover clustering 500

full database and log backup 528

hybrid availability group topology 537

importance of redundancy 60

Linux failover clustering with Pacing-
maker 62

load balanced read-only routing 536

log shipping feature 494

NIC teaming 67

overview of 493

potential failure points 59

reading secondary database copies 531

Red Hat Enterprise Linux (RHEL) con-
figuration 538

replication 497

secondary replica availability mode 521, 636

SQL Server Transaction Log Shipping 63

Windows Server Failover Clustering 61, 507

High Performance settings 51

historic data and values 122, 354

HISTORY_RETENTION_PERIOD option 357

horizontal partitioning 92, 371

HTTP over Transport Layer Security (TLS) 300, 314

HTTPS (HTTP Secure/HTTP over Secure Sockets Layer [SSL]) 300

hybrid cloud

automated backups 123

benefits of 121

keeping cold data online and queryable 122

private cloud 124

private networking 124

recovery strategies 490

Hypertext Markup Language (HTML) 299

Hypertext Transport Protocol (HTTP) 299

Hyper-Threading. *See* simultaneous multithreading (SMT)

I

IMPERSONATE permission 264

Import Registered Servers Wizard 25

INCLUDE list 437

Index Allocation Map (IAM) 83

indexes

clustered index design 429

Columnstore indexes 446

filtered unique index 347

full-text indexes 452

hash indexes 450

hierarchyid data type and 344

index statistics 453

index usage statistics 445

locating hypothetical 13

maintaining indexes and statistics 561, 627

memory-optimized tables 449

Missing Indexes feature 441

monitoring index fragmentation 563

nonclustered index design 434

rebuilding 564

reorganizing 568

reorganizing Columnstore indexes 571

spatial indexes 452

updating index statistics 569

XML indexes 453

index maintenance 161, 624

indirect checkpoint 188

infrastructure as a service (IaaS) 110, 326

In-Memory OLTP 48

Insert Snippet option 29

installation

adding databases to SQL Servers 169

Installation Center 2, 135

Installation Tab 6

installing a new instance 134

installing or upgrading SQL Server 6

installing tools and services 7, 164

minimizing footprint 128

moving and removing databases 189

performance and reliability monitoring tools 12

platforms supported 1

post-installation server configuration 151

pre-installation considerations 3, 127, 134

smart setup 146

int data type 336

integrity checks 161, 624

integrity, guaranteeing 346

IntelliSense 29

interconnected data 362

Internet of Things 298

Internet Protocol (IP) 300

Internet Protocol (IPv4) 297

Internet Protocol (IPv6) 297

internet protocol suite 296

Internet Small Computer Systems Interface (iSCSI) 56

Invoke-Sqlcmd cmdlet 655

IO_COMPLETION 583

IOPS (input/output operations per second) 52

IP addresses 298

IP forwarding 328

isolation levels and concurrency

blocking of concurrent sessions 386

blocking, observing 387

default level 398

experiencing phantom reads 389

isolation levels, changing with table hints 392

isolation levels, changing within transactions 391

isolation levels, choosing 385

levels available 383, 384

nonrepeatable reads 388

nonrepeatable reads, preventing 389

- on-disk vs. memory-optimized concurrency 398
- preventing phantom reads 390
- READ UNCOMMITTED (NOLOCK) 390
- SNAPSHOT isolation level 393
- two requests updating the same rows 387
- writes blocks reads 387

J

- JBOD (just a bunch of disks) 54
- Join operators 423
- JSON-formatted data 341

K

- Kerberos 69
- keys, primary and foreign 345

L

- large object (LOB) data types 83, 339, 367
- latency 52
- LCK_M_* 581
- leaf-level pages 94, 562
- licensing 131, 135
- life cycle model 604
- link aggregation. *See* NIC teaming
- Linux
 - affinity masks on 107
 - authentication to SQL Server on 245
 - availability group configuration 538
- Live Data window 586
- live execution plan 409
- load balanced read-only routing 536
- load balancing and failover support (LBFO). *See* NIC teaming
- Local Server Groups 24
- local storage. *See* Direct-Attached Storage (DAS)
- Lock pages in memory (LPIM) 47, 100, 105, 160
- log backup chain 466, 483
- logging
 - Maintenance Plan report options 632
 - setting up 147
 - transaction log backups 474
 - viewing Azure SQL Database audit logs 227
- logical SQL Servers 201, 204
- logical unit numbers (LUNs) 56

logins and users

- authentication to SQL Server on Linux 245
- authentication types 242
- BUILTIN\Administrators group 254
- contained database 256
- DBA team logins 252
- login types 244
- moving SQL Server logins 285
- NT AUTHORITY\SYSTEM account 255
- orphaned SIDs 246
- sa login 254
- securing logins 249
- service accounts 255
- terminology 241

Log Sequence Number (LSN) 86

- log shipping feature 494

- Log Shipping Wizard 64

- log truncation 87

- LowMemoryLimit 154

M

- Machine Learning Server, limiting memory usage by 156

- Machine Learning Services 7, 139

- maintenance, day-to-day 623

Maintenance Plans

- Back Up Database task 631
- backups on secondary replicas in availability groups 636
- benefits of 625
- Check Database Integrity task 626
- covering databases with 633
- Execute SQL Server Agent Job task 631
- Execute T-SQL Statement task 632
- History Cleanup task 630
- Maintenance Cleanup task 630
- new database detection 633
- Rebuild Index task 628
- Reorganize Index task 627
- report options 632
- scheduling options 625
- Shrink Database task 627
- SQL Server Management Studio and 634
- Update Statistics 629
- when not to use 635

- Maintenance Plan Wizard 478, 623, 625

- Maintenance tab (Installation Center) 136

- managed instance 218

- management data warehouse 15–18

- accessing reports 18
- data collection set up 17
- installing 15

- Management/Error Logs node 32

- many-to-many relationships 363

- Master Boot Record (MBR) 130

- Master server (MSX) 638

- Master Server Wizard 640

- max degree of parallelism (MAXDOP) 101, 425

- MAXDOP option 566

- MAX_DURATION parameter 566

- Maximum Server Memory 152

- Max Server Memory 102

- Max Worker Threads 104

- mechanical hard drives 52

- MediaPathLength 3

- memory 45–49

- buffer pool cache 46
- Central Processing Unit (CPU) issues 49

- competition for among various services 154

- configuration settings 102

- editions and memory limits 48

- Lock pages in memory (LPIM) 47, 100, 105

- Non-Uniform Memory Access 50

- optimize for ad hoc workloads 105

- OS reservation calculation 103

- overcommitting 74

- post-installation settings 152

- procedure cache 47

- thread consumption 104

- upper limit available 45

- working set 46

- MEMORYCLERK_XE 584

- memory-optimized objects 84, 102

- memory-optimized tables 357, 397, 449, 456, 478, 629

- Memory Pages 598

- MemorySafetyMargin 156

- MemoryThreshold 156

- merge replication 499

metrics

- key performance metrics 596

- Performance Monitor (perfmon.exe) application 592

- querying using Performance Monitor 595

- querying with DMVs 592

- Microsoft Assessment and Planning (MAP) Toolkit 136

Microsoft Cryptographic Application Programming Interface (MCAP) 304

Microsoft Data Migration Assistant (DMA) 240

Microsoft Hyper-V 73

Microsoft Management Console 11

Microsoft Power BI 217

migration readiness, assessing 4, 169.
See also databases

Minimum Recovery LSN (MinLSN) 89, 91

Minimum Server Memory setting 154
minimum synchronized required nodes 520

Missing Indexes feature 441

mixed extents 81

mixed mode authentication 141, 249

monetary data 336

money data type 336

MSX/TSX feature 638

multicast 297

Multi-Channel Memory Architecture 50

Multi Server Administration options 639

MultiSubNetFailover 535

MUST_CHANGE option 251

N

Network Address Translation (NAT) 297

Network-Attached Storage (NAS) 56

networking

complexities created by 57

network security 58

protocols and ports 58

Virtual Local-Area Network (VLAN) 58

network interface card (NIC) 343

network packets 297

network routing 298

Network Security Groups (NSG) 327

NEWID() function 343

NEWSEQUENTIALID() function 343

NEXT VALUE FOR 349

NIC teaming 67

node-level fencing 546

NOINDEX parameter 559

NO_INFOMSGS parameter 559

noisy neighbor phenomenon 73

NOLOCK 387, 390

nonclustered indexes

benefits of 434

choosing proper nonclustered index keys 435

creating "missing" nonclustered indexes 441

designing 434

INCLUDE list 437

index usage statistics 445

memory-optimized tables 451

properties of good 434

purpose of 434

redundant indexes 436

non-leaf-level pages 94

Non-Uniform Memory Access (NUMA) 50

Non-Volatile Memory Express (NVMe) 53

NoRebootPackage 3

NORECOVERY option 482

normalization 345

NT AUTHORITY\SYSTEM account 255

NT File System (NTFS) 107, 130, 368

NT LAN Manager (NTLM) 69

nullable sparse columns 341, 352

numeric data types 334, 335

NVARCHAR(100) 350

NVARCHAR(4000) 345

O

Object Explorer 23, 27

object-relational mappers (ORMs) 342

on-disk concurrency 399

ONLINE keyword 564

Online Transaction Processing (OLTP) 102

Open Database Connectivity (ODBC) 9

Open Geospatial Consortium (OGC) 339

operational expenditures (OpEx) 198

optimistic concurrency 342, 399

Optimize For Ad Hoc Workloads 105, 160

OPTIMIZE FOR query hint 403

OPTIMIZE FOR UNKNOWN query hint 403

Oracle SE Java Runtime Environment (JRE) 138

Organizational Unit (OU) 507

overcommitting 74

ownership 265

ownership chains 265

P

Pacemaker 62, 546

package managers 540

Page Faults 599

page file (Windows) 99

Page Free Space (PFS) 83

PAGEIOLATCH_* 582

PAGELATCH_* 582

page-level corruption 84

Page Life Expectancy (PLE) 597

Page Reads 598

page splits 562, 591, 593

page verify option 84, 174, 557

parallelism

benefits and drawbacks of 100

Cost Threshold for Parallelism (CTFP) 100, 426

defined 425

forcing parallel execution plans 425

max degree of parallelism (MAXDOP) 101, 425

parallel plan operations 100

parameterization 402

parameter sniffing 402

PARTIAL. *See* CONTAINMENT

partial backups 477, 486

partial-restore sequence 486

partitioned views 93

partition elimination 92

partition switching 92

partitioning key 92

partitioning, preventing 62

passwords 250, 294

patches 152, 198

payloads 297

peer-to-peer replication 497

performance and reliability monitoring tools

Database Engine Tuning Advisor 12

Extended Events GUI 13

management data warehouse 15

Performance Monitor 592, 595

performance tuning

Automatic Plan Tuning feature 418

capturing metrics with DMVs and data collectors 592

delayed durability 400

execution plan operators 419

execution plans 401

isolation levels and concurrency 383

parallelism 425

Query Store feature 413

Peripheral Component Interconnect Express (PCIe) 53

permissions

authorization vs. ownership 265

database roles 278

Data Definition and Data Manipulation languages 257

Dedicated Administrator Connection (DAC) 283

granting commonly needed 261

logins and users 241

modifying 259

- moving logins and permissions 285
- necessary to view execution plans 412
- overlapping 260
- required to access cached plan meta-data 406
- securing permissions to interact with jobs 614
- server roles 273
- SQL Server 257, 285
- views, stored procedures, and function permissions 267
- worst practices 281
- pessimistic concurrency** 342, 399
- phantom rows** 385
- physical backup devices** 472
- physical database architecture** 79–98
 - data compression 93
 - data files and filegroups 80
 - file types 79
 - table partitioning 92
 - temporary database (TempDB) 96
- piecemeal databases** 486
- plan cache.** *See* **procedure cache**
- Plan Guide feature** 403
- plan_handle column** 405
- planned failovers** 524
- Planning tab (Installation Center)**
 - Configuration Checker tool 3, 136
 - Data Migration Assistant 4
 - Upgrade Advisor link 4
- Platform Abstraction Layer (PAL)** 303
- platform as a service (PaaS)** 116, 198
- point-in-time recovery** 468, 485
- Policy-Based Management (PBM)** 643
- PolyBase external tables** 361
- PolyBase Query Engine** 138
- Power BI** 217
- power options** 159
- power saving** 51
- PowerShell module** 199, 206
 - automation using 648
 - availability group automation 656
 - Backup-SQLDatabase cmdlet 653
 - cmdlets for 649
 - creating databases using 211
 - Get-ChildItem cmdlet 654
 - help information 650
 - installing 651
 - installing offline 652
 - Invoke-Sqlcmd cmdlet 655
 - Remove-Item cmdlet 654
 - using with Azure 660
- PowerShell Provider for SQL** 11
- predicates** 585
- Premium Storage** 112
- preproduction environments** 252
- primary keys** 345

- principal, defined** 241
- proactive maintenance** 623
- procedure cache** 47, 402, 404
- processadmin server role** 276
- production environments** 252
- product life cycle model** 604
- Product Updates page** 146
- Profiler tool** 13
- Project Hekaton** 398
- protocols**
 - Border Gateway Protocol (BGP) 298
 - defined 296
 - File Transfer Protocol 299
 - HTTP over Transport Layer Security (TLS) 300, 314
 - Hypertext Transport Protocol (HTTP) 299
 - Internet Protocol (IP) 296, 300
 - internet protocol suite 296
 - protocol encryption 300
 - Transmission Control Protocol (TCP) ports 296
 - versions of IP in use today 297
 - Voice over IP 299
 - X.509 standard 302
- Proxies** 612
- public database role** 281
- Public Key Certificates** 302
- public key encryption (PKE)** 301
- public server role** 276
- publishers** 497
- Pull subscriptions** 497
- Push subscriber models** 497

Q

- Query Optimizer** 47
- Query Store feature**
 - examining execution plans using 403
 - initially configuring 415
 - purpose of 413
 - turning on 188
 - using query store data in your troubleshooting 416
- queue depth** 52
- quorum model** 62, 508

R

- rainbow tables** 295
- Random Access Memory (RAM)** 45. *See also* **memory**
- random salts** 295
- READ COMMITTED** 385
- READ_COMMITTED_SNAPSHOT (RCSI)** isolation level 393
- READ_ONLY mode** 174, 187
- read-scale availability groups** 66

- READ UNCOMMITTED (NOLOCK)** 390
- real data type** 336
- RebootRequiredCheck** 4
- RECOMPILE query hint** 403
- recovery.** *See also* **data recovery**
 - checkpoint system 88
 - Grant Perform Volume Maintenance Task Privilege 139
 - Minimum Recovery LSN 89
 - recovery chains, preventing broken 638
 - recovery interval, setting 90
 - recovery model setting 174, 182, 464
 - restarting with recovery 91
 - strategies for 487
- Recovery Point Objective (RPO)** 60, 460, 462
- Recovery Time Objective (RTO)** 60, 90, 460, 463
- Red Hat Enterprise Linux (RHEL), availability group configuration** 538
- redundancy** 60
- Redundant Array of Independent Disks (RAID)** 54, 57
- redundant indexes** 436
- referential integrity** 346
- RegisterAllProvidersIP setting** 535
- regular maintenance** 623
- Remote Desktop Protocol (RDP)** 463
- Remote Direct Memory Access (RDMA)** 57
- Remove-Item cmdlet** 654
- REPAIR_ALLOW_DATA_LOSS parameter** 559
- Repair feature** 136
- REPAIR_REBUILD parameter** 559
- REPEATABLE READ** 385
- replication** 229, 240, 497, 636
- Report Services Configuration Manager** 20
- Resilient File System (ReFS)** 368
- Resource Governor** 98, 600
- resource pools** 98, 602
- RESOURCE_SEMAPHORE** 582
- restart recovery.** *See* **recovery**
- restore strategies** 459, 482
- RESTORE VERIFYONLY** 632
- RESUMABLE index rebuilds** 628
- RESUMABLE parameter** 566
- retention policy** 624
- ring_buffer data collection** 584, 588, 589, 594, 595
- Role-Based Access Control** 223
- routing** 298
- ROWGUIDCOL property** 499
- row identifier (RID)** 433

row-level security 315
 rowversion data type 341
 run books 463

S

sa login 254
 salts 295
 scalability 203
 schemas 341
 scientific notation 335
 secret keys 300
 Secure Sockets Layer (SSL) 58
 security admin permission 276
 security groups 261
 security identifier (SID) 172, 242, 246, 266
 security issues
 auditing 319
 Azure SQL Database 218
 Border Gateway Protocol (BGP) 298
 brute-force attacks 294
 Certification Authorities (CA) 302
 data transmission protocols 296
 defense-in-depth 292
 dictionary attacks 294
 digital certificates 301
 distributed-denial-of-service (DDoS) attacks 331
 encryption in SQL Server 302
 General Data Protection Regulation (GDPR) 291
 hashing vs. encryption 294
 logins and users 241
 moving security objects from one server to another 289
 moving SQL Server logins and permissions 285
 network security 58
 permissions in SQL Server 257
 permissions worst practices 281
 securing Azure infrastructure as a service 326
 securing data in motion 314
 security principles and protocols 292
 server access security 68
 SQL injection 293
 symmetric and asymmetric encryption 300
 seek time 52
 SELECT ALL USER SECURABLES permission 264
 SELECT INTO syntax 343
 SELECT statements 385
 sensitive data 311

sequences 347
 Serial ATA (SATA) 52
 Serial Attached SCSI (SAS) 52
 SERIALIZABLE isolation 385, 398
 serveradmin server role 276
 server components. *See* database infrastructure
 Server Configuration page
 Grant Perform Volume Maintenance Task Privilege 139
 SQL Server PolyBase Engine service 138
 server editions 131, 169
 Server Message Block (SMB) 54
 Server Registration feature 24
 server roles 273
 server volume alignment 127
 Service accounts 255
 Service Broker feature 244
 service endpoints 330
 Service-Level Agreement (SLA) 460
 Service Master Key (SMK) 303, 306
 Service Packs (SPs) 604
 Service Principal Name (SPN) 69
 servicing model 604
 session_id column 387
 sessions 576, 585
 SET TRANSACTION ISOLATION LEVEL command 391
 setupadmin server role 277
 SetupCompatibilityCheck 4
 Setup.exe 135, 150
 sharding 121
 Shared Global Allocation Map (SGAM) 83
 SHOWPLAN permission 263
 Shrink Database task 627
 Simple Mail Transfer Protocol (SMTP) 608
 simple recovery model 464, 469, 487
 simultaneous multithreading (SMT) 49, 75
 single sign-on (SSO) 72, 222
 single-user mode 195
 sliding window partition strategy 375
 Slowly Changing Dimension (SCD) 399
 smalldatetime data type 336
 smallint data type 336
 smallmoney data type 336
 smart setup 146
 SMB 3.0 protocol 57
 SNAPSHOT isolation level 393
 Snapshot Isolation mode 186
 snapshot replication 473, 498

snippets 29
 soft-NUMA 50
 solid-state drives 53
 SORT_IN_TEMPDB option 565
 SOS_SCHEDULER_YIELD 582
 sparse columns 341, 352
 SPARSE keyword 352
 spatial data types 339
 spatial indexes 452
 spatial queries 340
 spatial reference ID (SRID) 340
 specialized data types 339
 special table types
 graph tables 362
 memory-optimized tables 357, 397
 PolyBase external tables 361
 system-versioned temporal tables 354
 split brain. *See* partitioning
 Split-Merge tool 121
 sp_sequence_get_range stored procedure 349
 sp_who2 command 387
 sp_who command 387
 SQL-authenticated logins 172
 SQLCMD 9
 SQL injection attacks 293
 SQL Server
 administering multiple 638
 auditing 319
 compared to Azure SQL Database 117
 databases, adding 169
 databases, moving and removing 189
 encryption in 301, 302
 failover cluster instance configuration 510
 installing and configuring features 164
 installing new instances 134
 maintaining 623
 Maintenance Plans 625
 managed backups 123
 minimizing installation footprint 128
 new servicing model 604
 post-installation server configuration 151
 pre-installation considerations 127
 server editions 131, 169
 timeouts 386
 upgrading 505
 volume usage and settings 127
 SQL Server Agent
 administering SQL Server Agent operators 618
 availability group environment 621
 Azure alternative to 216

- event forwarding 642
 - Job Activity Monitor 38
 - job, scheduling and monitoring 614
 - job history, configuring and viewing 615
 - job step security 612
 - jobs, configuring 612
 - notifying operators with alerts 39, 618
 - operators 40
 - overview of 37
 - securing permissions to interact with jobs 614
 - setting up 158
 - SQL Server Analysis Services**
 - Azure alternatives to 217
 - configuration and setup 168
 - installing 142
 - limiting memory usage by 154
 - SQL Server Authentication 243**
 - SQL Server Configuration Manager 11**
 - SQL Server Data Tools**
 - database deployment using 181
 - installing 137
 - tools included in 41
 - SQL Server Import And Export Wizard 42**
 - SQL Server Integration Services**
 - Azure alternatives to 217
 - benefits of 41
 - installing 143
 - moving logins by using 286
 - SQL Server Management Studio 21–41**
 - Activity Monitor tool 33
 - customizing menus and shortcuts 31
 - database creation using 180
 - download size 22
 - error logs 32
 - features of 23
 - filtering objects 27
 - installing 22, 137
 - IntelliSense tools 29
 - Maintenance Plans and 634
 - releases and versions 21
 - Server 478
 - Server Registration feature 24
 - snippets 29
 - SQLCMD mode 9
 - SQL Server Agent 37
 - upgrading 22
 - SQL Server memory manager 46**
 - SQL Server platform**
 - editions 131
 - performance and reliability monitoring tools 12
 - server editions 169
 - SQL Server Data Tools 41
 - SQL Server Management Studio 21
 - SQL Server Reporting Services 18
 - SQL Server setup 1–44
 - tools and services included with 7
 - SQL Server Profiler 13**
 - SQL Server Reporting Services**
 - Azure alternatives to 217
 - configuration and setup 165
 - installing 18, 137, 145
 - limiting memory usage by 155
 - Report Services Configuration Manager 20
 - SQL Server Setup**
 - automating 147
 - changing decisions after 134
 - Grant Perform Volume Maintenance Tasks feature 139
 - initiating 135
 - installing core features 142
 - logging setup 147
 - Mixed Mode authentication 141
 - smart setup 146
 - TempDB database 140
 - SQL Server Surface Area Configuration 157**
 - SQL Server Transaction Log Shipping 63**
 - sql_variant data type 345**
 - SSISDB Database**
 - configuration and setup 164
 - SSISDB Wizard 41
 - SSMS_IsInternetConnected 4**
 - Standard edition, appropriate use of 132**
 - Standard Storage 112**
 - statistics**
 - autocreate database statistics 184
 - index statistics 453
 - index usage statistics 445
 - updating index statistics 569, 624
 - STGeomFromText method 339**
 - STONITH 546**
 - STOPAT option 485**
 - STOPBEFOREMARK option 485**
 - storage. See data storage**
 - Storage-Area Network (SAN) 56, 128**
 - Storage Spaces 57**
 - stored procedures 267**
 - Stretch Database 122**
 - subnets 327**
 - subscribers 497**
 - Surface Area Configuration 157**
 - Surround With Snippets option 29**
 - swap file. See page file (Windows)**
 - sysadmin server role 274**
 - sys.dm_db_requests 387**
 - sys.dm_db_sessions 387**
 - sys.dm_exec_requests 576**
 - sys.dm_exec_sessions 576**
 - sys.dm_os_performance_counters 592**
 - sys.server_principals 242**
 - sys.sp_cdc_enable_db stored procedure 380**
 - system_health 586**
 - system-versioned temporal tables 354**
 - SYSTEM_VERSIONING option 357**
- ## T
- table design**
 - alphanumeric data types 334
 - binary data types 338
 - binary large objects (BLOBs) 367
 - capturing modifications to data 377
 - cascading 346
 - computed columns 352
 - constraints 346
 - data type selection 333
 - external tables 361, 457
 - graph tables 362
 - hierarchyid data type 339, 344
 - keys and relationships 345
 - memory-optimized tables 357, 397
 - numeric data types 334
 - numeric types 335
 - PolyBase external tables 361
 - referential integrity 346
 - rowversion data type 341
 - sequences 347
 - sparse columns 341, 352
 - spatial data types 339
 - specialized data types 339
 - special table types 354
 - sql_variant data type 345
 - string data and collation 335
 - system-versioned temporal tables 354, 381
 - table partitioning 370
 - temporal tables 381
 - Unicode support 335
 - uniqueidentifier data type 343
 - user-defined data types (UDTs) 350
 - user-defined types 350
 - XML data type 341
 - table partitioning**
 - defined 370
 - defining partitions 372

- horizontal 92, 371
- partition design guidelines 374
- sliding window partition strategy 375
- vertical partitioning 357, 377
- tail-of-the-log backups** 474
- targets** 585
- Target Server Memory** 600
- Target server (TSX)** 638
- TCP/IP protocol**
 - TCP/IP stack 297
 - turning on post-installation 158
- telemetry_xevent** 586
- TempDB**
 - buffer pool usage of 47
 - default settings for 140
 - locating files on VMs 116
 - managing 96
- temporal tables** 354, 381
- thin provisioning** 75
- ThreadHasAdminPrivilegeCheck** 4
- threat detection**. *See* auditing and threat detection
- Threat Detection feature** 318
- ticket-granting ticket (TGT)** 69
- time data type** 337
- time-outs** 386
- timestamp data type** 342
- time zones** 337
- tinyint data type** 336
- TORN_PAGE option** 480, 557
- TotalMemoryLimit** 154
- Total Server Memory** 599
- Trace Flag** 3226 163
- Trace Flag** 8002 107
- Trace Flags** 1118/1117 97
- transactional replication** 499
- transaction log**
 - backups 474
 - checkpoint system 88
 - delayed durability 85
 - file extension 85
 - file size and performance 91
 - incomplete transactions 86
 - log files required 85
 - Log Sequence Number (LSN) 86
 - log truncation 87
 - Minimum Recovery LSN (MinLSN) 89
 - MinLSN and active log 91
 - purpose of 85
 - recovering corrupt 560
 - recovery interval, setting 90
 - restarting with recovery 91

- space issues 88
- successful vs. unsuccessful transactions 85
- virtual log files (VLFs) 86
- Write-Ahead Logging (WAL) 85
- Transmission Control Protocol (TCP) port** 296
- transparent data encryption (TDE)** 174, 219, 303, 308
- Transport Control Protocol (TCP)** 58, 158, 207
- Transport Security Layer (TSL)** 58
- tree structures** 344
- Triple Data Encryption Standard (3DES)** 306
- troubleshooting**
 - error 1225 245
 - error 11732 350
 - error 41305 399
 - error 41325 399
 - using query store data in 416
- TRUNCATE TABLE command** 258
- Trustworthy setting** 174, 187
- T-SQL**
 - creating databases using 213
 - moving server permissions by using 288
 - moving server roles by using 288
 - moving SQL Server–authenticated logins by using 287
 - moving Windows–authenticated logins by using 287
 - T-SQL statements 632
- two-way synchronization** 378

U

- Unicode, table design and** 335
- uniform extents** 81
- unique constraints** 347
- uniqueidentifier data type** 343
- Universal Authentication** 243
- unsigned integers** 336
- updates** 152, 604
- UPDATE STATISTICS operation** 564
- Upgrade Advisor** 4, 136
- upgrading** 133, 198, 505
- USE PLAN query hint** 404
- user** 99
- user-defined data types (UDTs)** 350
- user-defined routes** 328
- user-defined types** 350
- users**. *See* logins and users

V

- VARBINARY(MAX) columns** 368
- varchar colum** 335
- Verify Backup Integrity** 632
- vertical partitioning** 357, 377
- VertiPaaSMemoryLimit** 155
- vi editor** 540
- VIEW DEFINITION permission** 263
- VIEW SERVER STATE permission** 263
- virtual CPU (vCPU)** 76
- virtual hard drives (VHDs)** 112
- virtual IP resource** 547
- Virtual Local-Area Network (VLAN)** 58
- virtual log files (VLFs)** 86
- virtual machines (VMs)**
 - Azure VMs, performance optimization 111
 - Azure VMs, sizing 115
 - benefits of 73
 - main players 73
 - purpose of 73
 - resource provisioning for 74
 - simultaneous multithreading and 49
- Virtual Network Name (VNN)** 62, 507
- virtual network service endpoints** 330
- Virtual Private Network (VPN)** 124
- VMware** 73
- Voice over IP** 299
- volumes**
 - defined 52
 - server volume alignment 127

W

- WAIT_AT_LOW_PRIORITY option** 566
- wait types** 554, 577
- WAIT_XTP_RECOVERY** 583
- Watch Live Data** 586
- wear-leveling** 53
- Web edit, appropriate use of** 132
- Windows authentication** 243
- Windows Management Instrumentation (WMI) alerts** 40
- Windows Server Failover Clustering** 61, 500, 507, 516, 519
- Windows Server Power Options** 159
- Windows Server Update Services** 146
- WITH CHANGE_TRACKING_CONTEXT clause** 380
- WITH RECOVERY option** 482
- WmiServiceStateCheck** 4

- worker threads 104
- working set 46
- WorkingSetMaximum 156
- WorkingSetMinimum 156
- workload groups 98
- workloads, protecting important 600
- World Wide Web (the web) 299
- Write-Ahead Logging (WAL) 85
- write-amplification 53
- write conflict error 399
- WRITELOG 583

X

- X.509 standard 302
- XE_FILE_TARGET_TVF 583
- XE_LIVE_TARGET_TVF 583
- XEvent Profiler 13
- XEvent Profiler tool 584
- XML data type 341, 453
- XML indexes 453

This page intentionally left blank

About the authors

William Assaf



William Assaf, MCSE, is a Microsoft SQL Server consultant and manager and blogs about SQL at sqltact.com. William has been a designer, database developer, and admin on application and data warehousing projects for private and public clients. He has helped write the last two generations of Microsoft SQL Server certification exams since 2012 and has been a Regional Mentor for PASS since 2015. William and fellow author Patrick Leblanc worked together on *SQL Server 2012 Step by Step* (Microsoft Press, 2015), having met at and together led the SQL Server User Group and SQLSaturday in Baton Rouge. William and his high school sweetheart enjoy travelling to speak at SQLSaturdays around the south, and hope to see to see you there, too.

Randolph West



Randolph West is a Data Platform MVP from Calgary, Alberta, Canada. He is coorganizer of the Calgary SQL Server User Group and Calgary SQLSaturday. He speaks at various conferences around the world, and acts on stage and screen. Randolph specializes in implementing best practices, performance tuning, disaster recovery, and cloud migrations, through his company Born SQL. You can read his blog at bornsql.ca.

Sven Aelterman



Sven Aelterman started with SQL Server when he first deployed version 2000 in a failover cluster scenario. Since then, he has worked as IT manager, principal consultant, and IT director. He currently serves the Trojans (students) of Troy University as a lecturer in information systems in the Sorrell College of Business and as director of IT for the College. In addition, he is cloud software architect for Sorrell Solutions, a business services nonprofit through which Trojans can gain real-world business and IT experience. In a fledgling attempt to give back to the community, he has spoken at many SQLSaturdays and code camps in the southeastern United States since 2005. He spoke about SSIS 2012 at Microsoft TechEd 2011. In 2012, he coauthored a book dedicated to SQL Server FILESTREAM. His involvement with Microsoft Azure resulted in the organization of two Global Azure Bootcamp events at Troy University. Sven blogs about a variety of Microsoft technologies at svenaelterman.wordpress.com and tweets and retweets about technology @svenaelterman.

Mindy Curnutt



Mindy Curnutt, an independent consultant, is 4X Microsoft Data Platform MVP and Idera ACE. She has been actively involved in the SQL Server Community for more than a decade, presenting at various User Group Meetings, SQLPASS Summits, as well as SQLSaturdays across North America. For two years, she was a team lead for the SQLPASS Summit Abstract Review Process and since 2015 has served as one of the three SQLPASS Summit program managers. She was a SME for a couple of the SQL 2012 and 2014 Microsoft SQL Server

Certification Exams and helped to author *SQL Server 2014 Step by Step*. Mindy currently serves on the board of directors for the North Texas SQL Server User's Group. She also serves as a mentor to others, helping to educate and promote scalable and sustainable SQL Server architecture and design. She is passionate about Data Security, Accessibility, Usability, Scalability and Performance. You can follow Mindy at her blog, mindycurnutt.com and on Twitter where she's known as @sqlgirl.

About the Foreword author

Patrick LeBlanc



Patrick LeBlanc is a data platform technical solution professional at Microsoft, working directly with customers on the business value of SQL Server. He coauthored *SharePoint 2010 Business Intelligence 24-Hour Trainer* (Wrox, 2011) and *Knight's Microsoft Business Intelligence 24-Hour Trainer* (Wrox, 2010), and founded www.sqllunch.com, a website devoted to teaching SQL Server technologies.