Microsoft

Introducing Microsoft SQL Server 2016

Mission-Critical Applications, Deeper Insights, Hyperscale Cloud

Stacia Varga, Denny Cherry, Joseph D'Antoni

Introducing Microsoft SQL Server 2016

Mission-Critical Applications, Deeper Insights,Hyperscale Cloud

Stacia Varga, Denny Cherry, Joseph D'Antoni

Microsoft

PUBLISHED BY
Microsoft Press
A division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

ISBN: 978-1-5093-0195-9

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Support at mspinput@microsoft.com. Please tell us what you think of this book at http://aka.ms/tellpress.

This book is provided "as-is" and expresses the author's views and opinions. The views, opinions and information expressed in this book, including URL and other Internet website references, may change without notice.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

Acquisitions and Development Editor: Devon Musgrave
Project Editor: John Pierce
Editorial Production: Flyingspress
Cover: Twist Creative • Seattle

Visit us today at

microsoftpressstore.com

- **Hundreds of titles available** – Books, eBooks, and online resources from industry experts
- **Free U.S. shipping**
- **eBooks in multiple formats** – Read on your computer, tablet, mobile device, or e-reader
- **Print & eBook Best Value Packs**
- **eBook Deal of the Week** – Save up to 60% on featured titles
- **Newsletter and special offers** – Be the first to hear about new releases, specials, and more
- **Register your book** – Get additional benefits

Microsoft

Contents

Introduction

SQL Server 2016 is the latest addition to Microsoft's data platform, with a variety of new features and enhancements that deliver breakthrough performance, advanced security, and richer, integrated reporting and analytics capabilities. Built using the new rapid-release model, SQL Server 2016 incorporates many features introduced first in the cloud in Microsoft Azure SQL Database. Furthermore, SQL Server 2016 includes the capability to dynamically migrate historical data to the cloud.

Introducing Microsoft SQL Server 2016 leads you through the major changes in the data platform, whether you are using SQL Server technology on-premises or in the cloud, but it does not cover every new feature added to the platform. Instead, we explain key concepts and provide examples for the more significant features so that you can start experiencing their benefits firsthand.

Who should read this book

We wrote this book to help anyone with an interest in SQL Server 2016 understand its newest capabilities. It is particularly useful for readers who have some prior experience with the SQL Server platform, whether as a database administrator, an application developer, a business-intelligence professional, or a technical manager.

Assumptions

For the most part, this book expects that you have at least a minimal understanding of relational database concepts and understand how to write and execute Transact-SQL queries. In some parts of the book, we explain features that expand the functionality of SQL Server beyond traditional relational database concepts and assume that you have no experience in these areas. In parts of the book that discuss new features in SQL Server Analysis Services or SQL Server Reporting Services, we assume that you have some experience developing solutions or managing implementations that use these components.

This book might not be for you if…

This book might not be for you if you have never worked with SQL Server. In a book of this size, we cannot explain every feature in SQL Server or describe how it differs from other databases. It also

might not be for you if you require an in-depth explanation of specific features in SQL Server or if you need a comprehensive listing of all the changes in the product. Our purpose in this book is to acquaint you with the most significant features at a high-level so that you can determine how best to take advantage of these features.

Organization of this book

This book is divided into nine chapters, each of which focuses on a different benefit that you can experience in the SQL Server family.

- Chapter 1, "Faster queries," introduces enhancements to In-Memory OLTP, memory-optimized tables, and Analysis Services to make Transact-SQL and analytical queries run faster.
- In Chapter 2, "Better security," we describe the new security features that give you more options for protecting your data.
- Chapter 3, "Higher availability," explains the new options for building high-availability solutions.
- The focus in Chapter 4, "Improved database engine," is the significant database engine enhancements that enable you to better manage performance and storage.
- Chapter 5, "Broader data access," explains how to work with temporal tables, JSON, and PolyBase.
- In Chapter 6, "More analytics," we provide an overview of expanded analytical capabilities in Analysis Services tabular models and the new, scalable in-database analytics made possible by SQL Server R Services.
- Chapter 7, "Better reporting," highlights the improvements to the Reporting Services development environment and the user experience.
- In the last two chapters, we transition to new data platform features in the cloud by introducing the features that bring Azure SQL Database into greater parity with SQL Server on-premises in Chapter 8, "Improved SQL Database," and by describing how to get started with data warehousing in Azure in Chapter 9, "Introducing Azure SQL Data Warehouse."

Conventions and features in this book

This book presents information using conventions designed to make the information readable and easy to follow.

- Boxed elements with labels such as "Note" provide additional information.
- Text that you type (apart from code blocks) appears in bold.
- Transact-SQL code is provided to help you further understand concepts by using specific examples.

System requirements

You will need the following hardware and software to complete the examples in this book:

- SQL Server 2016 Developer Edition or higher with SQL Server Management Studio.
- Computer that has a 1.4 GHz or faster x64 processor (2 GHz recommended)
- 1 GB of memory (4 GB recommended)

- 6 GB of available hard-disk space
- Super-VGA 800 x 600 or higher-resolution display
- DVD-ROM drive (if installing SQL Server 2016 from DVD)
- Internet connection to download software as described in applicable chapters.

Depending on your Windows configuration, you might require Local Administrator rights to install or configure SQL Server 2016 and related products.

Prerelease software

To help you become familiar with SQL Server 2016 as soon as possible after its release, we wrote this book by using examples that worked with Community Technology Preview 3.2 or Release Candidate 3. Consequently, the final version might include new features, the user interface might change, or features that we discuss might change or disappear. Refer to the "What's New in SQL Server 2016" topic in Books Online for SQL Server at https://msdn.microsoft.com/en-us/library/bb500435.aspx for the most up-to-date list of changes to the product.

Acknowledgments

A book is a collaborative effort between a lot of people. We'd like to thank Devon Musgrave, our editor at Microsoft Press, for helping us through this project. In addition, we'd like to thank John Pierce for his work in ensuring our writing is clear. There are many other people behind the scenes in the Microsoft product groups that we thank for helping us understand how the new features work. You know who you are. We very much appreciate the work you do to make SQL Server better!

Errata, updates, & book support

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

http://aka.ms/IntroSQLserver2016/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 http://support.microsoft.com.

Free ebooks from Microsoft Press

From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

http://aka.ms/mspressfree

Check back often to see what is new!

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://aka.ms/tellpress

We know you're busy, so we've kept it short with just a few questions. Your answers go directly to the editors at Microsoft Press. (No personal information will be requested.) Thanks in advance for your input!

Stay in touch

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

CHAPTER 1

Faster queries

When users want data, they want it as fast as you can give it to them. Microsoft SQL Server 2016 includes several options for enabling faster queries. Memory-optimized tables now support even faster online transaction processing (OLTP) workloads, with better throughput as a result of new parallelized operations. For analytic workloads, you can take advantage of updateable, clustered columnstore indexes on memory-optimized tables to achieve queries that are up to one hundred times faster. Not only is the database engine better and faster in SQL Server 2016, but enhancements to the Analysis Services engine also deliver faster performance for both multidimensional and tabular models. The faster you can deliver data to your users, the faster they can use that data to make better decisions for your organization.

In-Memory OLTP enhancements

Introduced in SQL Server 2014, In-Memory OLTP helps speed up transactional workloads with high concurrency and too many latches by moving data from disk-based tables to memory-optimized tables and by natively compiling stored procedures. In-memory OLTP can also help improve the performance of data warehouse staging by using nondurable, memory-optimized tables as staging tables. Although there were many good reasons to use memory-optimized tables in the first release of In-Memory OLTP, several limitations restricted the number of use cases for which In-memory OLTP was suitable. In this section, we describe the many enhancements that make it easier to put memory-optimized tables to good use.

Note For a complete list of T-SQL constructs that are not supported by memory-optimized tables in SQL Server 2016, see "Transact-SQL Constructs Not Supported by In-Memory OLTP" at https://msdn.microsoft.com/en-us/library/dn246937.aspx.

Reviewing new features for memory-optimized tables

In SQL Server 2016, you can implement the following features in memory-optimized tables:

- FOREIGN KEY constraints between memory-optimized tables, as long as the foreign key references a primary key.
- CHECK constraints.
- UNIQUE constraints.
- Triggers (AFTER) for INSERT/UPDATE/DELETE operations, as long as you use WITH NATIVE_COMPILATION.
- Columns with large object (LOB) types—*varchar(max)*, *nvarchar(max)*, and *varbinary(max)*.
- Collation using any code page supported by SQL Server.

Indexes for memory-optimized tables now support the following features:

- UNIQUE indexes.
- Index keys with character columns using any SQL Server collation.
- NULLable index key columns.

Scaling memory-optimized tables

Memory-optimized tables in SQL Server 2016 scale much better than they did when they were introduced. Not only can you put more data into memory-optimized tables, but you can also experience better throughput to support bigger workloads. In the current release, you can store up to 2 terabytes (TB) of data in a memory-optimized table when you create a durable table by using the SCHEMA_AND_DATA option in the CREATE TABLE statement. Furthermore, you can create a memory-optimized table with a row size greater than 8,060 bytes, even without a LOB column, as shown in Example 1-1.

Example 1-1: Creating a memory-optimized table with a row size greater than 8,060 bytes

```
CREATE TABLE  ProductReviewHistory (
  ProductID INT IDENTITY PRIMARY KEY NONCLUSTERED,
  ProductEnglishDescription NVARCHAR(4000),
  ProductSpanishDescription NVARCHAR(4000),
  ProductReviewDate DATETIME,
  ProductEnglishReviewComments NVARCHAR(4000),
  ProductSpanishReviewComments NVARCHAR(4000),
) WITH (MEMORY_OPTIMIZED = ON);
```

Note Although SQL Server 2016 supports this new row size for memory-optimized tables, you should create as narrow a table as possible when your requirement is high-performance queries.

To further deliver improved performance for memory-optimized tables, SQL Server 2016 supports parallel scans of all indexes, including nonclustered and hash indexes. This capability is especially

beneficial when you have queries that scan large data sets. It also supports parallel plans when you are using interop access with memory-optimized tables.

SQL Server 2016 also added multithreaded capabilities to the following operations affecting memory-optimized tables:

- Persistence of changes to memory-optimized tables in checkpoint files.
- Log apply in a recovery operation.
- MERGE operation.

> **Note** Because of the change to the checkpoint operation, the *sys.dm_db_xtp_checkpoint_stats* and *sys.dm_db_xtp_checkpoint_files* dynamic management views (DMVs) are significantly different from the versions in SQL Server 2014. See "sys.dm_db_xtp_checkpoint_stats" at https://msdn.microsoft.com/en-us/library/dn133197.aspx and "sys.dm_db_xtp_checkpoint_files" at https://msdn.microsoft.com/en-us/library/dn133201.aspx for more information.

Introducing native client enhancements

You can now enable Multiple Active Result Sets (MARS) when connecting to a memory-optimized table or running a natively compiled stored procedure. This way, the database engine can begin fetching rows from a new result set before completely retrieving rows from an earlier request. To use MARS, you must enable it explicitly in the connection string, like this:

```
Data Source=<Your Server>; Initial Catalog=<Your Database>; Integrated Security=SSPI;
MultipleActiveResultSets=True
```

For the most part, you can use MARS with memory-optimized tables just as you do with disk-based tables, but there are a few differences:

- If two statements attempt to modify the same row, a write-write conflict occurs and the new operation fails.
- Because each statement runs under SNAPSHOT isolation and is in a batch-scoped transaction, the result of an operation performed by one statement is not visible to another statement. However, a rollback of one batch-scoped transaction does affect other transactions in the same batch.
- A user transaction cannot perform Data Definition Language (DDL) operations.

When you use a MARS-enabled connection, you can also execute natively compiled stored procedures. If a stored procedure contains a SELECT statement, that statement can yield execution to another statement; otherwise, the stored procedure runs completely without yielding to other statements.

By using the BEGIN TRANSACTION statement in a T-SQL statement, you can begin a new user transaction within the current user transaction. You can also create a save point in a transaction by using the SAVE TRANSACTION statement or by making a call to *transaction.Save(<save point name>)*. However, you cannot use these features in a natively compiled stored procedure.

Exploring T-SQL enhancements

New enhancements to the query surface area in SQL Server 2016 make it easier to implement memory-optimized tables. In particular, the ALTER TABLE statement supports more functionality so that you can make changes to existing memory-optimized tables without needing to first drop a table

and then re-create it with the changes. For better performance, you can now natively compile objects besides stored procedures and support more T-SQL constructs in your code.

ALTER TABLE statement

Beginning in SQL Server 2016, you can use the ALTER TABLE statement to change the table definition and indexes on memory-optimized tables, as shown in Example 1-2. You can add or drop columns, constraints, or indexes or change the bucket count of an index.

> **Note** You cannot use CREATE INDEX, DROP INDEX, or ALTER INDEX statements with memory-optimized tables. The only way to add, remove, or change indexes for these tables is to use the ALTER TABLE statement.

Example 1-2: Using the ALTER TABLE statement with a memory-optimized table

```
-- Add a column and an index
ALTER TABLE dbo.ShoppingCart
  ADD Quantity INT NULL,
      INDEX ix_CreatedDate(CreatedDate);-- Alter an index by changing the bucket count
ALTER TABLE dbo.ShoppingCart ALTER INDEX ix_UserId REBUILD WITH ( BUCKET_COUNT = 2000000);
-- Drop an index
ALTER TABLE dbo.ShoppingCart DROP INDEX ix_CreatedDate;
```

In most cases, the ALTER TABLE statement runs in parallel and writes only metadata changes to the transaction log as a performance optimization. However, the following single-threaded operations continue to require writing the entire altered table to the transaction log:

- Adding or altering a column with a LOB data type.
- Adding or dropping a columnstore index.
- Any operation affecting an off-row column—a column with a data type of *char*, *nchar*, *varchar*, *nvarchar*, *binary*, or *varbinary* that does not fit in the 8,060-byte row—except if the operation lengthens a column that is already off-row.

Natively compiled modules

Compiling tables and stored procedures as native code was introduced in SQL Server 2014 to enable more efficient query execution and data access for memory-optimized tables. New to SQL Server 2016 is the ability to natively compile other objects, such as scalar user-defined functions (UDFs), inline table-valued functions (TVFs), and triggers. In addition to the support of new objects, the query surface area for native compilation is expanded and applies not only to natively compiled stored procedures but to all natively compiled modules. You can also alter a natively compiled module rather than drop and re-create it when a change to its functionality is necessary.

Much like you do when creating a natively compiled stored procedure, you must include the NATIVE_COMPILATION and SCHEMABINDING options in the WITH clause when you create a natively compiled scalar UDF, as shown in Example 1-3. Another new capability in SQL Server 2016 is support for LOBs as an argument to built-in string functions and LOB as a return type, as shown in the example.

Example 1-3: Using LOBs in a natively compiled scalar UDF

```
CREATE FUNCTION dbo.TRIM_CONCAT (@string1 NVARCHAR(MAX), @string2 NVARCHAR(MAX))
RETURNS NVARCHAR(MAX)
WITH NATIVE_COMPILATION, SCHEMABINDING
AS
BEGIN ATOMIC WITH
  (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = N'English')
  RETURN RTRIM(@string1) + LTRIM(@string2);
END;
```

Note More details about working with natively compiled scalar UDFs is available at "Scalar Defined Functions for In-Memory OLTP," https://msdn.microsoft.com/en-us/library/dn935012.aspx.

SQL Server 2016 also supports the following capabilities for natively compiled modules:

- Use the OUTPUT clause with the INSERT, UPDATE, and DELETE statements in a natively compiled stored procedure.
- Omit the EXECUTE AS clause when you want EXECUTE AS CALLER as the execution context of a natively compiled module. You include an EXECUTE AS clause in a module only if you need to enforce a different execution context.
- Use any SQL Server collation in an expression.
- Use any of the built-in math and security functions.
- Use SELECT DISTINCT in the SELECT clause.
- Use OUTER JOIN, EXISTS, IN, and scalar subqueries in the FROM clause.
- Use OR or NOT in the WHERE clause.
- Combine result sets by using the UNION or UNION ALL operations.
- Configure nested execution of natively compiled modules by using the EXECUTE statement.

Managing memory-optimized tables

Statistics for memory-optimized tables are updated automatically as long as database compatibility is set to 130. Statistics are updated by data sampling rather than by a full scan.

If you created a memory-optimized table prior to SQL Server 2016 CTP 3.3 or in SQL Server 2014, you must run the script shown in Example 1-4 once to add the automatic-update functionality.

Example 1-4: Adding automatic update of statistics to a memory-optimized table

```
-- Enable automatic update on the database first, if necessary
 ALTER DATABASE CURRENT SET AUTO_UPDATE_STATISTICS ON;
 GO
-- Set the compatibility level, if necessary
ALTER DATABASE CURRENT SET COMPATIBILITY_LEVEL = 130;
GO
--
DECLARE @sql NVARCHAR(MAX) = N'';
SELECT
```

```
  @sql += N'UPDATE STATISTICS '
  + quotename(schema_name(t.schema_id))
  + N'.'
  + quotename(t.name)
  + ';' + CHAR(13) + CHAR(10)
FROM sys.tables AS t
  WHERE t.is_memory_optimized = 1;
EXECUTE sp_executesql @sql;
```

When the statistics change on a table, any natively compiled modules might not run as efficiently afterward. The automatic update does not recompile the natively compiled modules. Therefore, you should periodically use *sp_recompile* to reoptimize the natively compiled modules.

Planning data migration to memory-optimized tables

SQL Server 2016 includes several tools for helping you plan a migration of data in existing tables to In-Memory OLTP. Reports and tasks in SQL Server Management Studio (SSMS) help you evaluate workloads and create checklists of migration steps. You can also use SQL Server PowerShell or Windows PowerShell to evaluate candidate objects for migration.

In SSMS, you can now run the Transaction Performance Analysis Report directly on the production database. You do not need to configure a management data warehouse or data collectors as you do for SQL Server 2014. To run the report, right-click a database in Object Explorer, point to Reports, point to Standard Reports, and then select Transaction Performance Analysis Overview. You then choose Tables Analysis or Stored Procedure Analysis. If you choose the Table Analysis option, a report with a scatterplot chart showing the relative performance gain and amount of migration work required is displayed, as shown in Figure 1-1. When you click a table in the chart, a new report displays query-access characteristics and contention statistics for the selected table.

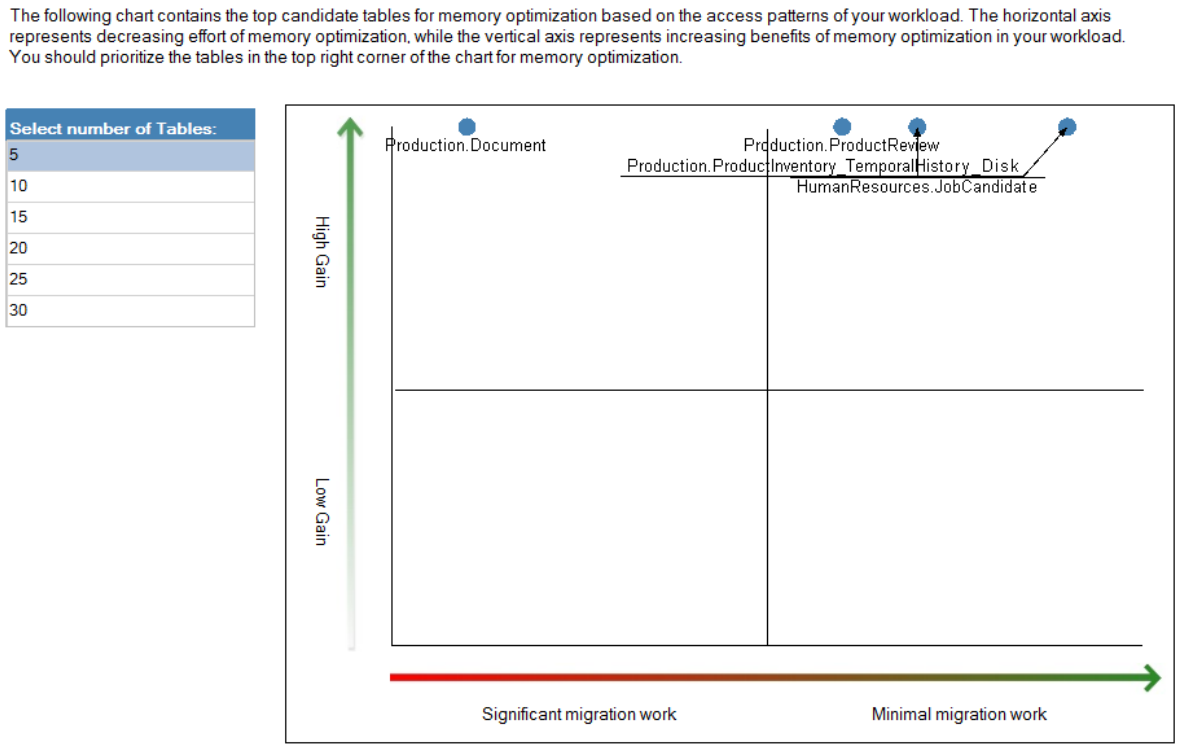


Figure 1-1: Transaction Performance Analysis Overview.

A new feature in SSMS is the ability to generate migration checklists. To do this, right-click a database in Object Explorer, point to Tasks, and then select Generate In-Memory OLTP Migration Checklists. This step launches a wizard that displays a welcome page. On the second page of the wizard, you specify a location in which to save the checklist and whether to generate a checklist for all tables and stored procedures in the database or for a specific list that you define. After you make this selection, the next page of the wizard includes a Script PowerShell Commands button and a Finish button. If you select the Script PowerShell Commands button, a text file opens to display the following command:

```
Save-SqlMigrationReport -Server '<Server Instance Name>' -Database 'AdventureWorks' -FolderPath
'C:\Users\<User>\Documents\<Path>'
```

When you click the Finish button, the wizard begins to generate a separate checklist for each table and stored procedure specified in the wizard. The status of each checklist is displayed in the table so that you can easily see whether any failed. After the wizard completes the checklists, you can find them as HTML files in the Stored Procedures, Tables, or User Defined Functions folders in the output path that you configured in the wizard. An example of a migration checklist for a table is shown in Figure 1-2.

Memory optimization checklist for [AdventureWorks].[JobCandidate]

Report Date/Time:4/18/16 1:09 PM

Description	Validation Result
The following unsupported data types are defined on this table:	Failed: More information
- Resume: 'Xml' is not supported.	
No computed columns are defined on this table.	Succeeded
No sparse columns are defined for this table.	Succeeded
No identity columns with unsupported seed and increment are defined for this table.	Succeeded
Supported foreign key relationships are defined on this table but the table cannot be migrated through the memory-optimization wizard. To migrate this table as well as the other tables involved in the FOREIGN KEY references, first remove the FOREIGN KEYs, then migrate the tables using the memory-optimization wizard, and finally add the FOREIGN KEY references to the migrated memory-optimized tables.	Failed: More information
- FK_JobCandidate_Employee_BusinessEntityID: Foreign Key on this table (referencing HumanResources.Employee)	
No unsupported constraints are defined on this table.	Succeeded
No unsupported indexes are defined on this table.	Succeeded
No unsupported triggers are defined on this table.	Succeeded
Post migration row size does not exceed the row size limit of memory-optimized tables.	Succeeded
Table is not partitioned or replicated.	Succeeded

Figure 1-2: Checklist for migrating a table to In-memory OLTP.

As an alternative to using SSMS to generate the checklists, you can use SQL Server PowerShell. In Object Explorer, right-click a database, and then click Start PowerShell. In the SQL Server PowerShell command window, enter the following command:

```
Save-SqlMigrationReport -FolderPath 'C:\Users\<User>\Documents\<Path>'
```

Notice that this is similar to the command generated by the migration checklist wizard, but it omits the server and database arguments because the SQL Server PowerShell command window connects directly to the correct database when you open it.

If you prefer, you can use a Windows PowerShell command window (open it with administrator permissions), in which case you must enter and execute the commands shown in Example 1-5. The second command produces a migration checklist for all tables and stored procedures in the database at the location you specify in the *FolderPath* argument. The object that you reference in the third command must be either a table or a stored procedure. Use this command when you want to generate a checklist for one object at a time.

Example 1-5: Generating migration checklists by using Windows PowerShell commands

```
> [System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMO')
> Save-SqlMigrationReport -Server "<Server Instance Name>" -Database "<Database Name>" -
FolderPath "<Path1>"
> Save-SqlMigrationReport -Server "<Server Instance Name>" -Database "<Database Name>" -Object
<Object Name> -FolderPath "<Path 2>"
```

In-memory analytics

Whereas OLTP workloads typically involve frequent, concurrent inserts and updates, analytic workloads are characterized by read operations and aggregations of much larger datasets. Traditionally, these workloads have been implemented in separate environments to avoid the resource contention between OLTP writes and analytic reads. Furthermore, analytic workloads are often moved to other storage structures, such as SQL Server Analysis Services (SSAS) multidimensional cubes or tabular models, to take advantage of special features useful for analytics. However, moving data to another structure introduces latency in the data and requires periodic processes to run to refresh the data. SQL Server 2016 adds more options for performing real-time analytics on rapidly changing in-memory data by adding support for columnstore indexes on memory-optimized OLTP tables.

Note If you still need SSAS to support other requirements (such as consolidation with other sources, centralization of business logic, or translations, to name a few), you can configure a cube partition for a multidimensional model to use relational online analytical processing (ROLAP) mode or configure a tabular model to use DirectQuery mode. Both ROLAP mode and DirectQuery mode can take advantage of data compression and faster query response times so that you have less data overall to manage, fewer processes to monitor, and no data to move between environments.

When you create an updateable columnstore index on a memory-optimized table, the database engine creates a separate copy of the data that your analytics workload uses while your OLTP workloads continue to run on the table data. That way, resource contention between the two workloads is significantly reduced when they run simultaneously. As changes occur in a table, the database engine automatically manages changes to the index to ensure that it remains current with the underlying table.

Reviewing columnstore index enhancements

An enhancement to clustered columnstore indexes in SQL Server 2016 is the ability to have a nonclustered B-tree index on a table that also has a clustered columnstore index. This capability is useful for workloads that have a significant percentage of singleton lookups but are mostly analytic in nature. While the columnstore index is extremely efficient for large scanning operations such as data aggregation, the B-tree structure is much more efficient for returning a few records.

In addition, this release adds other significant increases in performance and scalability to columnstore indexes. One of the biggest benefits is the improvement in batch-execution mode, a feature of columnstore indexes that allows results to be processed in batches of approximately 1,000 rows rather than row by row. Batch processing greatly reduces execution time and CPU utilization for common data warehouse queries such as aggregations. In SQL Server 2016, batch-execution mode is extended to single-threaded queries, sort operations (such as ORDER BY), and T-SQL window functions.

SQL Server 2016 also includes support for columnstore indexes in readable secondary databases in Always On Availability Groups. Furthermore, it includes support for read committed snapshot isolation (RCSI) and snapshot isolation (SI) modes. Consequently, you can design a scale-out data warehouse in which analytic workloads are spread across multiple nodes.

In previous versions of SQL Server, the best practice is to use an INT field as a predicate for your queries against columnstore tables because of the retrieval process used by the database optimizer to get rows from the internal dictionary supporting the columnstore index. SQL Server 2016 supports predicate pushdown of the *varchar* and *nvarchar* data types for columnstore indexes to improve performance for queries that might need to filter on a string value.

Getting started with in-memory analytics

The easiest way to get started with in-memory analytics is to add a clustered columnstore index to a disk-based table, as shown in Example 1-6. When you use the DROP_EXISTING = ON option in the CREATE COLUMNSTORE INDEX statement, the database engine drops an existing index before it creates a new one. If you omit this option and a columnstore index already exists, or if you include this option and there is no existing columnstore index, an error occurs. Notice that the syntax in this example does not include a column list because a clustered columnstore index uses all columns in the table.

Example 1-6: Creating a clustered columnstore index in a disk-based table

```
--Eliminate the DROP_EXISTING option if there is no existing columnstore index in the table
CREATE CLUSTERED COLUMNSTORE INDEX ix_cci_Product ON ProductBig
WITH (DROP_EXISTING=ON);
```

Using filtered columnstore indexes

A potential problem with running OLTP and analytic workloads on the same tables by using this approach is the overhead of maintaining the clustered columnstore index, which can have an adverse effect on OLTP workloads. To minimize this impact, you can instead create a nonclustered columnstore index with a filtered condition, as shown in Example 1-7. That way, the columnstore index contains only the *warm data*—the data that changes infrequently—and does not require updates to keep current with *hot data*—the frequently changing data in the table. The filter should restrict the index to warm-data rows to reduce the amount of updates required to keep the index up to date with the table. In addition, you should add a clustered index that includes the column in the columnstore index's filtered predicate. In this case, the clustered index contains every row in the table, including the hot data that is not covered by the columnstore index, as shown in Figure 1-3. An analytic query transparently combines both warm and hot data as needed to produce the correct result set.

Example 1-7: Creating a nonclustered columnstore index with a filtered condition

```
--Create the columnstore index with a filtered condition
CREATE NONCLUSTERED COLUMNSTORE INDEX ix_salesorderheaders_ncci
ON Sales.SalesOrderHeader (CustomerID, AccountNumber, SalesOrderID, SubTotal, ShipDate)
WHERE ShipDate < '2014-01-01';
--Create the clustered index on column with a filtered condition
--(but first drop existing clustered index, not shown)
CREATE CLUSTERED INDEX ix_salesorderheaders_ci ON Sales.SalesOrderHeader (ShipDate);
-- The following query returns rows from NCCI and hot rows not in NCCI
SELECT TOP 5 Year(ShipDate) as ShipYear, AccountNumber, SUM(SubTotal) as TotalSales
FROM Sales.SalesOrderHeader
WHERE ShipDate > '2013-10-01'
GROUP BY Year(ShipDate), AccountNumber;
```

Note You can create a nonclustered columnstore index for disk-based tables only. SQL Server 2016 does not support this type of index for memory-optimized tables.

If your OLTP workload performs many updates and deletes, the nonclustered columnstore index might become significantly fragmented and consequently less efficient. In this situation, you should consider using the COMPRESSION_DELAY option as described at "Real-time Operational Analytics: Compression Delay Option for Nonclustered Columnstore Index (NCCI)," https://blogs.msdn.microsoft.com/sqlserverstorageengine/2016/03/06/real-time-operational-analytics-compression-delay-option-for-nonclustered-columnstore-index-ncci/.

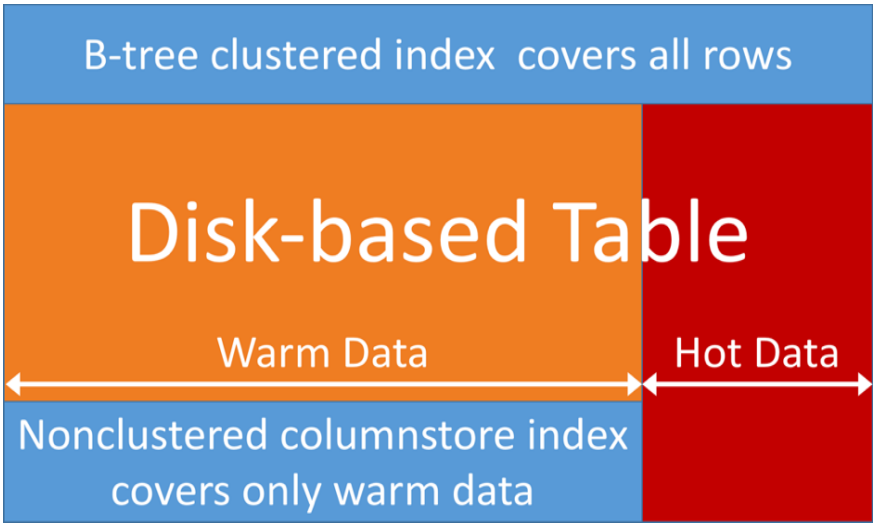


Figure 1-3: Indexing for hot and warm data in a disk-based table to support analytic queries.

If your OLTP table is an in-memory table, then columnstore technology benefits both the OLTP and analytic workloads. Use the INDEX…CLUSTERED COLUMNSTORE argument when you create the table, as shown in Example 1-8. When you use this argument, all columns in the table are stored as a clustered columnstore index.

Example 1-8: Creating an in-memory table with a columnstore index

```
CREATE TABLE UserSession (
  SessionID INT IDENTITY(1,1) NOT NULL PRIMARY KEY NONCLUSTERED,
  UserID INT NOT NULL,
  CreatedDate DATETIME2(7) NOT NULL,
  ShoppingCartId INT,
  INDEX ix_usersession_cci CLUSTERED COLUMNSTORE
)
  WITH (MEMORY_OPTIMIZED = ON);
```

Note Even when implementing a columnstore index, your analytics queries might require significant server resources such as CPU, I/O, or memory that adversely impact your OLTP workloads. In this case, another option to consider is using an AlwaysOn configuration to offload the analytics workload to a readable secondary.

Analysis Services enhancements

Analysis Services provides two different modes to support fast reporting and analytics: multidimensional and tabular. Multidimensional mode has long been a component in SQL Server, whereas tabular mode was introduced in SQL Server 2012. Before the introduction of columnar storage in the SQL Server database engine, the use of either of these Analysis Services modes was the best way to deliver superior query performance. Now you can choose whether to use in-memory OLTP or columnstore indexes in the database engine, one of the Analysis Services engines, or even a hybrid-mode in which you use ROLAP mode with a multidimensional model or DirectQuery mode for a tabular model to directly query the database engine instead of data stored by Analysis Services. In SQL Server 2016, both multidimensional mode and tabular mode benefit from enhancements that improve performance.

Understanding multidimensional performance improvements

The Analysis Services multidimensional engine in SQL Server 2016 includes an optimization for faster query performance. There is no user interface that makes this change noticeable. To quantify the benefits for your own workload, you should perform baseline testing of query performance in your current version of Analysis Services and compare the baseline results to testing in SQL Server 2016 Analysis Services.

First, the performance of queries that include an unnatural hierarchy is faster than it was in previous versions, although natural hierarchies remain faster yet. Nothing has changed with regard to the storage structure of an unnatural hierarchy. Instead, the engine is now able to execute the query in a more optimal fashion when it encounters a drill-down query such as the one shown in Example 1-9, which Excel generates when you explore a hierarchy. This optimization requires no configuration or redevelopment of models to improve query performance. You need only deploy your multidimensional model to a SQL Server 2016 Analysis Services multidimensional instance.

Example 1-9: Drilling down one level of a hierarchy

```
SELECT
NON EMPTY
Hierarchize(Drilldownmember({{Drilldownlevel({[Product].[Color-Size].[All]},,,
INCLUDE_CALC_MEMBERS)}},
{[Product].[Color-Size].[Color].&[White]},,,INCLUDE_CALC_MEMBERS))
DIMENSION PROPERTIES PARENT_UNIQUE_NAME,HIERARCHY_UNIQUE_NAME
ON COLUMNS
FROM [Adventure Works]
WHERE ([Measures].[Reseller Sales Amount])
CELL PROPERTIES VALUE, FORMAT_STRING, LANGUAGE, BACK_COLOR, FORE_COLOR, FONT_FLAGS
```

Natural versus unnatural hierarchies in a multidimensional database

A natural hierarchy describes the one-to-many relationship moving between levels from the top down, such as Year, Quarter, and Month, in which one year can have four quarters and one quarter can have twelve months. A specific month, such as January 2016, always has a single relationship to a quarter (Q1 2016) in the dimension and a single relationship to a year (2016). A natural hierarchy is always a better structure to design into a dimension because of the materialization of the data in the Analysis Services database. However, you must define attribute relationships for the natural hierarchy to materialize the data.

An unnatural hierarchy is one for which there is a more arbitrary relationship between levels. For example, consider a hierarchy with a Color level at the top and a Size level at the bottom. A color such as black can have sizes of L and M, but the color white can also have these sizes. You cannot define attribute relationships for this data because an error occurs during processing when the same size has multiple relationships. In earlier versions of Analysis Services, it is considered best practice to avoid the use of unnatural hierarchies or to "naturalize" the data by artificially enhancing it to enforce the one-to-many structure between levels and then defining attribute relationships.

Next, query processing is optimized for various other situations, including distinct count queries on ROLAP partitions. Now Analysis Services performs these queries faster by improving the processing time after the data is retrieved from SQL Server. Furthermore, if you add a columnstore index to the ROLAP partition source, you can experience significantly better performance gains on distinct count queries.

Last, a change in memory allocation request handling is available in SQL Server 2016. Prior to this version, you had two options: Windows Low-Fragmentation Heap (LFH) or a custom heap allocator for Analysis Services. In the last few versions, the default is set to LFH to improve performance for multiuser workloads. Ordinarily, LFH works well with small memory blocks, but some situations can compromise the efficiency of LFH for Analysis Services by fragmenting memory, which in turn reduces query performance. If you have queries that execute satisfactorily after the service restarts but slow down over time, a possible cause is fragmentation. In SQL Server 2016 Analysis Services, the default option is a new hybrid allocator that strikes a balance by using LFH for small allocations and the custom heap allocator for large allocations. You can still configure the msmdrv.ini file for Analysis Services to force LFH or the custom heap allocator if necessary.

Note Tabular mode uses the same memory settings as multidimensional mode, but it has its own msmdsrv.ini file in which these settings are defined. The tabular mode tends to require larger allocations than multidimensional mode, so the new hybrid allocator should rely more heavily on the custom heap allocator.

Understanding tabular performance improvements

Tabular-model performance has always been good because of its use of the same columnar technology as columnstore indexes and in-memory OLTP. However, it's even better in SQL Server 2016 because of DAX optimizations, storage-engine caching improvements, and changes to the way that T-SQL is generated for DirectQuery mode.

The first DAX optimization is a reduction in the number of queries sent to the storage engine from Power BI and Power BI Desktop. Prior to this change, a single chart could potentially send hundreds of queries to the storage engine. Now the Power BI client tools generate a single storage-engine query, whether the storage engine retrieves data from memory or from SQL Server in DirectQuery mode—as long as the query includes only simple measures. Furthermore, a single DAX query can return multiple result sets as intermediate results that Power BI can use across multiple requests.

The next optimization executes queries more efficiently in any client tool, including Microsoft Excel or SQL Server Reporting Services (SSRS) in addition to the Power BI client tools. This efficiency is a result of changes in the following areas that affect measure execution:

- **Variables** As we describe in more detail in Chapter 6, "More analytics," the use of variables in a DAX expression for measures allows you to reuse logic within the same expression, which can reduce overall execution time.
- **Conditional expressions** Rather than evaluating each branch of an IF or SWITCH conditional expression, a branch with a false condition no longer generates a storage-engine query.

- **Nonempty calculations** The number of scans necessary to retrieve a nonempty result set is reduced from many to one.
- **Multiple measures in same table** A single query now includes all measures from the same table.
- **Measure grouping** When a query requests a measure at different levels of granularity, such as Month, Quarter, Year, and Total, the query requests only the lowest-level measure value and then derives the higher-level values.
- **Joins** A storage-engine query now returns both dimension columns and measure values in the same query to eliminate redundant joins, and the order of joins now starts from the most restrictive intermediate table that correlates with the greatest number of other intermediate tables.
- **Multiple result sets** A storage-engine query now returns both dimension columns and measure values in the same query.
- **Countrows** This function is optimized to now use table heuristics.
- **Storage-engine caching** The storage engine uses one cache per database rather than one cache per server as it did in prior versions.

DirectQuery is the feature in tabular models that retrieves data from the data source instead of data that has been imported into memory. DirectQuery now generates a simpler and better performing query in SQL Server 2016 than it did in earlier versions. Specifically, the query translated to T-SQL eliminates tables that are unnecessary to the results and thereby reduces the number of queries sent to the database engine. This optimization applies not only to DAX queries (which have always been supported in DirectQuery mode), but also to MDX queries, which are newly supported in SQL Server 2016.

> **Note** Now that MDX queries are supported by DirectQuery, you can use Excel to connect to a DirectQuery-enabled tabular model.

If you are using Power BI or Power BI Desktop to connect to a DirectQuery-enabled tabular model, the translation to T-SQL is also improved in SQL Server 2016. The chattiness between the client tool and SQL Server has been reduced with no configuration changes required. To take advantage of this improvement, you need to restore your tabular database to a SQL Server 2016 Analysis Services tabular instance.

CHAPTER 2

Better security

SQL Server 2016 introduces three new principal security features—Always Encrypted, Row-Level Security, and dynamic data masking. While all these features are security related, each provides a different level of data protection within this latest version of the database platform. Throughout this chapter, we explore the uses of these features, how they work, and when they should be used to protect data in your SQL Server database.

Always Encrypted

Always Encrypted is a client-side encryption technology in which data is automatically encrypted not only when it is written but also when it is read by an approved application. Unlike Transparent Data Encryption, which encrypts the data on disk but allows the data to be read by any application that queries the data, Always Encrypted requires your client application to use an Always Encrypted–enabled driver to communicate with the database. By using this driver, the application securely transfers encrypted data to the database that can then be decrypted later only by an application that has access to the encryption key. Any other application querying the data can also retrieve the encrypted values, but that application cannot use the data without the encryption key, thereby rendering the data useless. Because of this encryption architecture, the SQL Server instance never sees the unencrypted version of the data.

Note At this time, the only Always Encrypted–enabled drivers are the .NET Framework Data Provider for SQL Server, which requires installation of .NET Framework version 4.6 on the client computer, and the JDBC 6.0 driver. In this chapter, we refer to both of these drivers as the ADO.NET driver for simplicity.

Getting started with Always Encrypted

Using Always Encrypted requires a small amount of preparation within the database storing the encrypted tables. While this can be done by using a wizard in SQL Server Management Studio, using T-SQL is a more repeatable process for production deployments, so this chapter will focus on the T-SQL configuration process. The preparation is a two-step process:

1. Create the column master key definition
2. Create the column encryption key

Column master key definition

The column master key is a certificate that is stored within a Windows certificate store, a third-party Hardware Security Module (HSM), or the Azure Key Vault. The application that is encrypting the data uses the column master key to protect the various column encryption keys that handle the encryption of the data within the columns of a database table.

Note Using an HSM, also known as an Enterprise Key Manager (EKM), requires the use of SQL Server Enterprise Edition. In this chapter, we describe the use of a self-signed certificate that you store in the Microsoft Certificate Store of the Windows operating system. While this is approach is not the optimal configuration, it demonstrates the concepts of Always Encrypted and is applicable to any edition of SQL Server.

You can create a column master key definition by using the graphical interface within SQL Server Management Studio (SSMS) or by using T-SQL. In SSMS, connect to the SQL Server 2016 database instance in which you want to use Always Encrypted to protect a database table. In Object Explorer, navigate first to the database, then to Security, and then expand the Always Encrypted Keys folder to display its two subfolders, as shown in Figure 2-1.

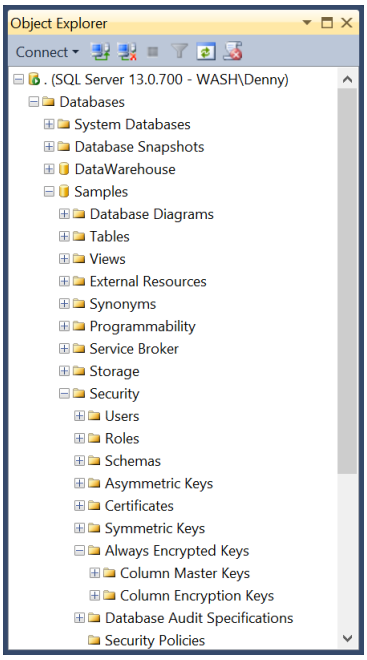


Figure 2-1: Always Encrypted Keys folder in SQL Server 2016 Object Explorer.

To create the column master key, right-click the Column Master Keys folder and select New Column Master Key. In the New Column Master Key dialog box, type a name for the column master key, specify whether to store the key in the current user's or local machine's certificate store or the Azure Key Vault, and then select a certificate in the list, as shown in Figure 2-2. If there are no certificates, or if you want to use a new self-signed certificate, click the Generate Certificate button, and then click OK. This step creates a self-signed certificate and loads it into the certificate store of the current user account running SSMS.

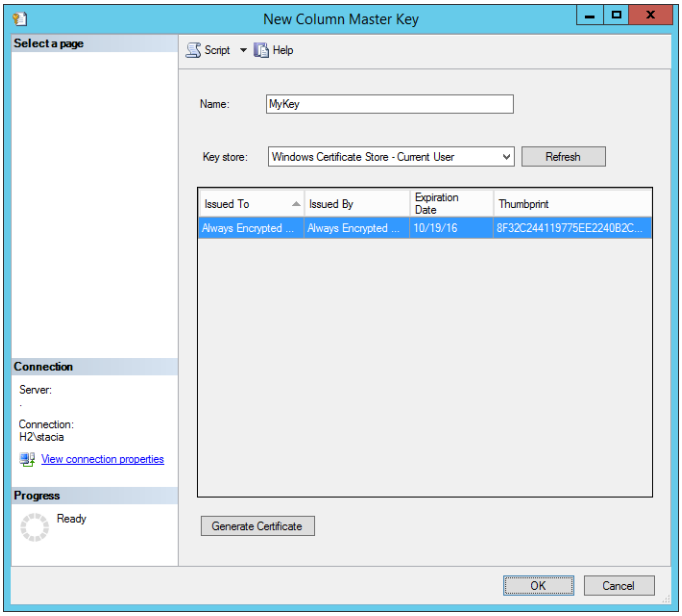


Figure 2-2: New Column Master Key dialog box.

Note You should perform these steps on a trusted machine, but not on the computer hosting your SQL Server instance. That way, the data remains protected in SQL Server even if the host computer is compromised.

After creating the certificate and configuring it as a column master key, you must then export and distribute it to all computers hosting clients requiring access to the data. If a client application is web-based, you must load the certificate on the web server. If it is an application installed on users' computers, then you must deploy the certificate to each user's computer individually.

You can find applicable instructions for exporting and importing certificates for your operating system at the following URLs:

- Exporting certificates
  - Windows 7 and Windows Server 2008 R2: https://technet.microsoft.com/en-us/library/cc730988.aspx.
  - Windows 8 and Windows Server 2012: https://technet.microsoft.com/en-us/library/hh848628(v=wps.620).aspx.
  - Windows 8.1 and Windows Server 2012 R2: https://technet.microsoft.com/en-us/library/hh848628(v=wps.630).aspx.
  - Windows 10 and Windows Server 2016: https://technet.microsoft.com/en-us/library/hh848628(v=wps.640).aspx.
- Importing certificates
  - Windows 7 and Windows Server 2008 R2: https://technet.microsoft.com/en-us/library/cc754489.aspx.
  - Windows 8 and Windows Server 2012: https://technet.microsoft.com/en-us/library/hh848630(v=wps.620).aspx.

- Windows 8.1 and Windows Server 2012 R2: https://technet.microsoft.com/en-us/library/hh848630(v=wps.630).aspx.
- Windows 10 and Windows Server 2016: https://technet.microsoft.com/en-us/library/hh848630(v=wps.640).aspx.

Certificate stores and special service accounts

When you import certificates into the certificate store on the computers with the application that encrypts and decrypts the data, you must import the certificates into either the machine certificate store or the certificate store of the domain account running the application.

As an alternative, you can create a column master key by using T-SQL. Although you might find that creating the key is easier using SSMS, T-SQL scripts provide you with a repeatable process that you can check into a source control system and keep safe in case you need to rebuild the server. Furthermore, because best practices for SQL Server 2016 discourage installation of SSMS on the server's console and Windows security best practices discourage certificate installation on unsecured systems such as users' desktops, the use of T-SQL scripts to create column master keys is recommended.

To create a column master key, use the CREATE COLUMN MASTER KEY statement, as shown in Example 2-1. This statement requires you to supply a name for the definition, such as MyKey, as shown in the example. You must also set the value for KEY_STORE_PROVIDER_NAME as MSSQL_CERTIFICATE_STORE. Last, you specify the path for the certificate in the certificate store as the KEY_PATH value. This value begins with CurrentUser when you use a certificate stored in the user account's certificate store or LocalMachine when using a certificate stored in the computer's certificate store. The rest of the value is a random-looking string of characters that represents the thumbprint of the selected certificate. This thumbprint is unique to each certificate.

Example 2-1: Creating a column master key

```
USE [Samples]
GO
CREATE COLUMN MASTER KEY MyKey
WITH
(
      KEY_STORE_PROVIDER_NAME = N'MSSQL_CERTIFICATE_STORE',
      KEY_PATH = N'CurrentUser/My/DE3A770F25EBD6071305B77FB198D1AE434E6014'
);
GO
```

Other key store providers?

You may be asking yourself what key-store providers are available besides the Microsoft SQL Server certificate store. You can choose from several other key-store providers. One option is MSSQL_CSP_PROVIDER, which allows you to use any HSM supporting Microsoft CryptoAPI. Another option is MSSQL_CNG_STORE, which allows you to use any HSM supporting Cryptography API: Next Generation. A third option is to specify AZURE_KEY_VAULT as the key-store provider, which requires you to download and install the Azure Key Vault key store provider on the machines accessing the protected data, which will be protected as described in "Using the Azure Key Vault Store Provider for Always Encrypted," at http://blogs.msdn.com/b/sqlsecurity/archive/2015/11/10/using-the-azure-key-vault-key-store-provider.aspx. Last,

you can use a custom provider, as described in "Creating Custom Key Store Providers for Always Encrypted (Azure Key Vault Example)," at http://blogs.msdn.com/b/sqlsecurity/archive/2015/09/25/creating-an-ad-hoc-always-encrypted-provider-using-azure-key-vault.aspx. Although this article provides an example using Azure Key Vault, you can apply the principles to the development of a custom provider.

Finding the certificate thumbprint

You can easily locate the thumbprint of the certificate in the certificate store by using the Certificate snap-in within the Microsoft Management Console (MMC). In MMC, on the File menu, select Add/Remove Snap-In. In the Add Or Remove Snap-ins dialog box, select Certificates in the Available Snap-ins list on the left, and click the Add button to move your selection to the right. The Certificates Snap-in dialog box prompts you to select a certificate store. Choose either My User Account or Computer Account, depending on which certificate store you are using. Click the Finish button, and then click OK. Expand the Certificates folder to locate your certificate in the Personal/Certificates subfolder, double-click the certificate, select the Details tab, and scroll to the bottom, where you can see the thumbprint that you use as the value for the CREATE COLUMN MASTER KEY DEFINITION statement.

Column encryption keys

After creating a column master key, you are ready to create the encryption keys for specific columns. The SQL Server 2016 ADO.NET driver uses column encryption keys to encrypt the data before sending it to the SQL Server and to decrypt the data after retrieving it from the SQL Server 2016 instance. As with the column master key, you can create column encryption keys by using T-SQL or SSMS. While the column master keys are easier to create by using T-SQL, column encryption keys are easier to create by using SSMS.

To create a column encryption key, use Object Explorer to connect to the database instance, navigate to the database, then to Security, and expand the Always Encrypted Keys folder. Right-click Column Encryption Keys, and then select New Column Encryption Key. In the New Column Encryption Key dialog box, type a name for the new encryption key, select a Column Master Key Definition in the drop-down list, as shown in Figure 2-3, and then click OK. You can now use the column encryption key in the definition of a new table.

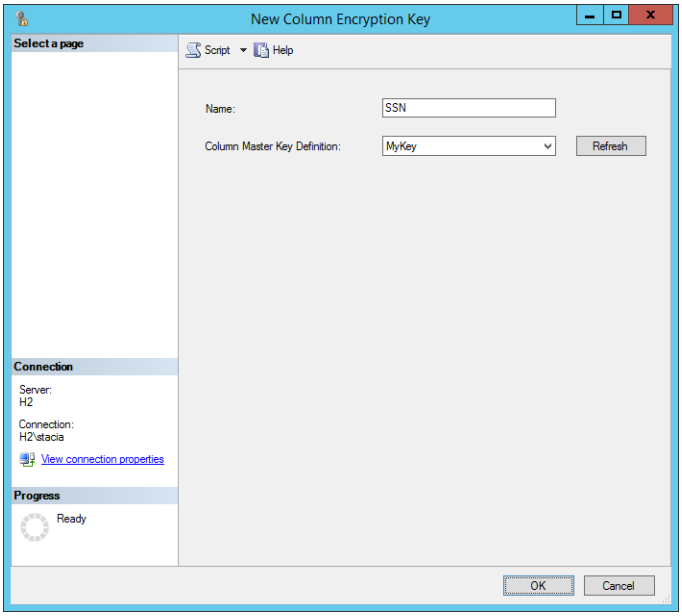


Figure 2-3: New Column Encryption Key dialog box.

To create a new column encryption key by using T-SQL, you use the CREATE COLUMN ENCRYPTION KEY statement as shown in Example 2-2.

Example 2-2: CREATE COLUMN ENCRYPTION KEY

```
USE [Samples]
GO
CREATE COLUMN ENCRYPTION KEY [MyColumnKey]
WITH VALUES
(
      COLUMN MASTER KEY DEFINITION = [MyKey],
      ALGORITHM = 'RSA_OAEP',
      ENCRYPTED_VALUE =
0x016E008000630075007200720065006E00740075007300650072002F006D0079002F00640065003300610037003700
0006600320035006500620064003600300037003100330030003500620037003700660062003100390038006400310061
0065003400330034006500360030003100340040D74119935C902E59F57A96C3E6F770826D247135FFFA759B5B013DF4DA
F7CFB760A5864DD8381B91924D067BE4F574B50DE7F0D53F278E1C003B5D192865B808C1590224F4A4BB463255101C36D
3089F46609B376D7B00FA9F9CEAF715398EECAB790AC6EC8BD18C17B3EB992CAE08FEA6A2F5A2BDDA4F5A700744E45861
F993A3C488127E5897B30892DD2734DD5D84F096882A393D5877C5A20E392888FE0357F46DB578AEB4C677CFFCE228127
6C4D12F3E5AC3BCCC09B78BB0E522D86F9B2CF989F14695B7CB95A478194ECBD175B5C7C1687B7589FD9145B2782CB0BB
AB6F7F5B0AC7F8C256EB0D3D87ABAE4F73137FA4AFA387B791B54AC503B53271D
);
GO
```

The CREATE COLUMN ENCRYPTION KEY statement accepts three parameters. The first parameter is COLUMN MASTER KEY DEFINITION, which corresponds to the column master key definition that you created in a previous step. The second parameter defines the encryption algorithm used to encrypt the value of the encryption key. In SQL Server 2016, the only supported parameter value at this time is RAS_OAEP. The third parameter is the value of the column encryption key after it has been encrypted by the column master key definition.

Note When creating column encryption keys, you should not use an unencrypted value as the ENCRYPTED_VALUE parameter of the CREATE COLUMN ENCRYPTION KEY statement. Otherwise, you compromise the benefits of Always Encrypted by making data vulnerable to attack.

The CREATE COLUMN ENCRYPTION KEY command accepts a minimum of one VALUE block, and a maximum of two VALUE blocks. Two VALUE blocks should be used when rotating encryption keys, either because a key has expired or because it has become compromised. Two keys should exist within the database long enough for all connected applications to download the new encryption keys from the database. Depending on the application design and client connectivity, this process may take minutes or months.

Generating new encrypted values

Given that the value is encrypted, how can new encrypted values be generated? The easiest way is to use SSMS to open the New Column Encryption Key dialog box shown in Figure 2-3, select the correct column master key definition, provide a name for the new encryption key, and then click the Script button at the top of the dialog box. This selection gives you the full CREATE COLUMN ENCRYPTION KEY statement, including a new random encrypted value. You can then add this new value as a second encryption key and thereby easily rotate the encryption keys.

Creating a table with encrypted values

After creating the column master key definition and column encryption keys, you can create the table to hold the encrypted values. Before you do this, you must decide what type of encryption to use, which columns to encrypt, and whether you can index these columns. With the Always Encrypted feature, you define column sizes normally, and SQL Server adjusts the storage size of the column based on the encryption settings. After you create your table, you might need to change your application to execute commands on this table using Always Encrypted. In this section, we describe the choices you have when creating your table and adapting your application.

Encryption types

Before creating a table to contain encrypted values, you must first make a choice about each column to be encrypted. First, will this column be used for looking up values or just returning those values? If the column is going to be used for lookups, the column must use a deterministic encryption type, which allows for equality operations. However, there are limitations on searching for data that has been encrypted by using the Always Encrypted feature. SQL Server 2016 supports only equality operations, which include equal to, not equal to, joins (which use equality), and using the value in the GROUP BY clause. Any search using LIKE is not supported. Additionally, sorting data that is encrypted using Always Encrypted must be done at the application level, as SQL Server will sort based on the encrypted value rather than the decrypted value.

If the column is not going to be used for locating records, then the column should use the randomized encryption type. This type of encryption is more secure, but it does not support searches, joins, or grouping operations.

CREATE TABLE statement for encrypted columns

When creating tables, you use the normal CREATE TABLE syntax with some additional parameters within the column definition, as shown in Example 2-3. Three parameters are used within the ENCRYPTED WITH syntax for the CREATE TABLE statement. The first of these is the ENCRYPTION_TYPE parameter, which accepts a value of RANDOMIZED or DETERMINISTIC. The second is the ALGORITHM

parameter, which only accepts a value of AEAD_AES_256_CBC_HMAC_SHA_256. The third parameter is the COLUMN_ENCRYPTION_KEY, which is the encryption key you use to encrypt the value.

Example 2-3: Creating a table using Always Encrypted

```
CREATE TABLE [dbo].[Customers](
 [CustomerId] [int] IDENTITY(1,1),
 [TaxId] [varchar](11) COLLATE Latin1_General_BIN2
 ENCRYPTED WITH (ENCRYPTION_TYPE = DETERMINISTIC,
 ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256',
 COLUMN_ENCRYPTION_KEY = MyColumnKey) NOT NULL,
 [FirstName] [nvarchar](50) NULL,
 [LastName] [nvarchar](50) NULL,
 [MiddleName] [nvarchar](50) NULL,
 [Address1] [nvarchar](50) NULL,
 [Address2] [nvarchar](50) NULL,
 [Address3] [nvarchar](50) NULL,
 [City] [nvarchar](50) NULL,
 [PostalCode] [nvarchar](10) NULL,
 [State] [char](2) NULL,
 [BirthDate] [date]
 ENCRYPTED WITH (ENCRYPTION_TYPE = RANDOMIZED,
 ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256',
 COLUMN_ENCRYPTION_KEY = MyColumnKey) NOT NULL
 PRIMARY KEY CLUSTERED ([CustomerId] ASC) ON [PRIMARY] );
 GO
```

The sample code shown in Example 2-3 creates two encrypted columns. The first encrypted column is the TaxId column, which is encrypted as a deterministic value because our application allows a search of customers based on their government-issued tax identification number. The second encrypted column is the BirthDate column, which is a randomized column because our application does not require the ability to search, join, or group by this column.

Indexing and Always Encrypted

Columns containing encrypted data can be used as key columns within indexes—provided that those columns are encrypted by using the DETERMINISTIC encryption type. Columns encrypted by using the RANDOMIZED encryption type return an error message when you try to create an index on those columns. Columns encrypted by using either encryption type can be used as INCLUDE columns within nonclustered indexes.

Because encrypted values can be indexes, no additional performance-tuning measures are required for values encrypted with Always Encrypted beyond the indexing and tuning that you normally perform. Additional network bandwidth and greater I/O are the only side effects that result from the increased size of the values being returned.

Application changes

The beauty of the Always Encrypted feature of SQL Server 2016 is that applications already using stored procedures, ORMs, or parameterized T-SQL commands should require no application changes to use Always Encrypted, unless nonequality operations are currently being used. Applications that build SQL statements as dynamic SQL within the application and execute those commands against the database directly need to be modified to use parameterization of their queries, a recommended security best practice for all applications, before they can take advantage of the Always Encrypted feature.

Another change required to make Always Encrypted work is the addition of a connection string attribute to the connection string of the application connecting to the database:

```
Column Encryption Setting=enabled
```

With this setting added to the connection string, the ADO.NET driver asks the SQL Server if the executing command includes any encrypted columns, and if so, which columns are encrypted. For high-load applications, the use of this setting may not be the best practice, especially if a large percentage of executing commands do not include encrypted values. Consequently, the .NET Framework provides a new method on the SqlConnection object called *SqlCommandColumnEncryptionSetting,* which has three possible values as shown in the following table.

Method value	Effective change
Disabled	There are no Always Encrypted columns or parameters to use for the queries that are executed by using this connection object.
Enabled	There are Always Encrypted columns and/or parameters in use for the queries that are executed by using this connection object.
ResultSet	There are no Always Encrypted parameters. However, executing queries using this connection object return columns encrypted by using Always Encrypted.

Note Be aware that the use of this method can potentially require a significant amount of change to your application code. An alternative approach is to refactor your application to use different connections.

For the best performance of SQL Server, it is wise to request only the metadata about Always Encrypted for those queries that use Always Encrypted. This means that in applications for which a large percentage of queries use Always Encrypted, the connection string should be enabled and the specific queries within the application should specify *SqlCommandColumnEncryptionSetting* as Disabled. For applications for which most queries are not using Always Encrypted values, the connection string should not be enabled, and *SqlCommandColumnEncryptionSetting* should be set for Enabled or ResultSet as needed for those queries that are using Always Encrypted columns. In most cases, applications are able to simply enable the connection string attribute, and application performance will remain unchanged while using the encrypted data.

Note While enabling the Always Encrypted setting has been designed to be an easy-to-implement solution for application data encryption, it is a very major change to application functionality. Like all major changes to application functionality, there should be rigorous testing of this feature in a testing environment, including load testing, before making this change in a production environment.

Migrating existing tables to Always Encrypted

In a production environment, there is no direct path to migrate an unencrypted table to a table that is protected by Always Encrypted. A multiphased approach to data migration is required to move data from the current table into the new table. The basic approach to move data from an existing table into an Always Encrypted table includes the following steps:

1. Build a new staging table.

2. Write a .NET application using ADO.NET to process the encryption of both existing and updated rows.
3. Run the .NET application built in the prior step.
4. Drop the existing table and rename the new table to use the old table name.
5. Change the application's connection string to include *Column Encryption Setting=enabled*.

> **Note** For nonproduction environments, you can use the Always Encrypted wizard or the Import/Export wizard in SSMS, which follow a process similar to the one we outline in this section.

Step 1: Build a new staging table

Because Always Encrypted does not support the conversion of an existing table into an Always Encrypted table, you must build a new table. The new table should have the same schema as the existing table. When you build the new table, the only changes you need to make are enabling the columns to be encrypted and specifying the collation as described in Example 2-3.

A large application is likely to require a large amount of time to encrypt and move the data, and it might not complete this process during a single maintenance window. In that case, it is helpful to make two additional schema changes. The first change is to add a column on the production table to track when a row is updated (if the table does not already have such a column). The second change is to add a trigger to the production table that fires on delete and removes any rows from the new table when the row is deleted from the production table. To reduce downtime when you move the table with the encrypted data into production, you should create any indexes existing on the production table on the new table before loading it with data.

Steps 2 and 3: Write a .NET application to encrypt the data and move it to the new table

Because of the design of Always Encrypted, data is encrypted only by applications using the ADO.NET driver with parameterized queries. This design prevents you from using SSMS to move data into the new table. Similarly, you cannot use an application to perform a simple query such as this:

```
INSERT INTO NewTable SELECT * FROM OldTable;
```

The rows must be brought from the database into a .NET application and then written back to the database using a parameterized query, one row at a time, for the data to be properly inserted as encrypted values in the database.

For small applications, this process can be completed quickly, within a single maintenance window. For larger applications, this processes may take several nights, which requires the application to be aware of data changes during the business day. After the application has processed the initial push of data from the source table to the new table, the application must run periodically to move over any changed rows to the new table until the cutover has been completed.

Step 4: Rename the table

Once all the data has been migrated, the existing table can be dropped or renamed so that it can be saved until testing has been completed. Then the new table can be renamed so that it now has the production table's name. Any indexes existing on the production table that do not exist on the new table should be created at this time, as well as any foreign keys that exist on the old table. Once testing is completed, if the old table is not deleted, any foreign keys using that table as a parent should be removed to prevent issues when rows are deleted.

Step 5: Update the application's connection string

Once the tables are changed, the application needs to know to use Always Encrypted. To do this, change the application's connection string to use the new *Column Encryption Setting=enabled* attribute or release a new version of the application that uses the *SqlCommandColumnEncryptionSetting* method on the connection object within the .NET code.

Using Always Encrypted in Microsoft Azure SQL Database

Always Encrypted is fully supported by the SQL Database platform, as we describe in Chapter 8, "Improved Azure SQL Database." You configure Always Encrypted for a SQL Database just as you do for an on-premises SQL Server 2016 deployment by using T-SQL commands. At the time of this writing, there are no enhancements in the Microsoft Azure portal for configuring Always Encrypted in SQL Database.

Row-Level Security

Row-Level Security (RLS) allows you to configure tables such that users see only the rows within the table to which you grant them access. This feature limits which rows are returned to the user, regardless of which application they are using, by automatically applying a predicate to the query. You can use a filter predicate to silently filter the rows that are accessible by the user when using INSERT, UPDATE, or DELETE statements. In addition, you can use the following block predicates to block the user from writing data: AFTER INSERT, AFTER UPDATE, BEFORE UPDATE and BEFORE DELETE. These block predicates return an error to the application indicating that the user is attempting to modify rows to which the user does not have access.

You implement RLS by creating an inline table function that identifies the rows accessible to users. The function you create can be as simple or complex as you need. Then you create a security policy to bind the inline table function to one or more tables.

Note Although you can create a complex RLS inline table function, bear in mind that complex queries are typically slow to execute. Besides ensuring that your function properly limits access to specific rows in a table, you should take care that it does so with minimal impact to application performance.

RLS is designed to simplify your application code by centralizing access logic within the database. It should be noted that, as with any RLS solution and workarounds, it is possible for users with the ability to execute arbitrary T-SQL commands to infer the existence of data that should be filtered, via side-channel attacks. Therefore, RLS is intended for scenarios where the queries that users can execute are controlled, such as through a middle-tier application.

Be aware that RLS impacts all users of a database, including members of the db_owner fixed database role. Members of this role have the ability to remove the RLS configuration from tables in the database. However, by doing so, all other users again have access to all rows in the table.

Note You can use branching logic in the inline table function for RLS when you need to allow members of the db_owner fixed database role to access all rows in the table.

Creating inline table functions

The method by which users connect to a database determines how you need to write the inline table function. In an application that connects users to the database with their individual Windows or SQL login, the function must directly match each user's login to a value within the table. On the other hand, in an application that uses a single SQL login for authentication, you must modify the application to set the session context to use a database value that sets the row-level filtering as we explain in more detail later in this section. Either way, when you create a row-level filtering inline table function, you must enable SCHEMABINDING and the function must return a column that contains a value of 1 (or any other valid value) when the user can view the row.

> **Note** You can implement RLS on existing tables without rebuilding the tables because the inline table function that handles the filtering is a separate object in the database, which you then bind to the table after you create the function. Consequently, you can quickly and easily implement RLS in existing applications without requiring significant downtime.

Application using one login per user

When your application logs into the database engine by using each user's Windows or SQL login, your inline table function needs only to compare the user's login against a table in the database to determine whether the user has access to the requested rows. As an example, let's say you have an Orders application for which you want to use RLS to restrict access to order information to the person entering the order. First, your application requires an Order table, such as the one shown in Example 2-4. When your application writes a row into this table, it must store the user's login in the SalesRep column.

Example 2-4: Creating an Orders table

```
CREATE TABLE Orders
    (
    OrderId int,
    SalesRep sysname
    );
```

Your next step is to create an inline table function like the one shown in Example 2-5. In this example, when a user queries the Orders table, the value of the SalesRep column passes into the @SalesRep parameter of the fn_Orders function. Then, row by row, the function compares the @SalesRep parameter value to the value returned by the USER_NAME() system function and returns a table containing only the rows for which it finds a match between the two values.

Example 2-5: Creating an inline table function to restrict access by user login

```
CREATE FUNCTION dbo.fn_Orders(@SalesRep AS sysname)
    RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN
SELECT 1 AS fn_Orders_result
WHERE @SalesRep = USER_NAME();
GO
```

Note The data type of the parameter in your inline table function must match the corresponding column data type in the table that you plan to secure with RLS, although it is not necessary for the parameter name to match the column name. However, managing your code is easier if you keep the names consistent.

Now let's consider what happens if your database contains related information in another table, such as the OrderDetails table shown in Example 2-6.

Example 2-6: Creating an OrderDetails table

```
CREATE TABLE OrderDetails
      (
      OrderId int,
      ProductId int,
       Qty int,
       Price numeric(8,2)
      );
GO
```

To apply the same security policy to this related table, you must implement additional filtering by creating another inline table-valued function, such as the one shown in Example 2-7. Notice that you continue to use the USER_NAME() system function to secure the table by a user-specific login. However, this time the inline table-valued function's parameter is @OrderId, which is used in conjunction with the SalesRep column.

Example 2-7: Creating an inline table function to restrict access by user login in a related table

```
CREATE FUNCTION dbo.fn_OrderDetails(@OrderId AS int)
    RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN
              SELECT 1 AS fn_Orders_result
              FROM Orders
              WHERE OrderId = @OrderId
                      AND SalesRep = USER_NAME();
GO
```

Application using one login for all users

When your application uses a single login for all users of the application, also known as an application account, you use similar logic as you do when the application passes user logins to the database. Let's continue with a similar example as the one in the previous section, but let's add some additional columns to the Orders table, as shown in Example 2-8. In this version of the Orders table, the SalesRep column has an *int* data type instead of the *sysname* data type in the earlier example.

Example 2-8: Creating a variation of the Orders table

```
CREATE TABLE Orders
    (
    OrderId int,
```

```
    SalesRep int,
    ProductId int,
    Qty int,
    Price numeric(8,2)
    );
GO
```

Additionally, the inline table function changes to reflect the single login, as shown in Example 2-9. Notice the parameter's data type is now *int* instead of *sysname* to match the column in the table shown in Example 2-8. In addition, the predicate in the function now uses the SESSION_CONTEXT system function and outputs the result as an *int* data type to match the input parameter's data type.

Example 2-9: Creating an inline table function for an application using a single login

```
CREATE FUNCTION dbo.fn_Orders(@SalesRep AS int)
    RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN
      SELECT 1 AS fn_Orders_result
      WHERE @SalesRep = CONVERT(SESSION_CONTEXT(N'UserId') AS int);
GO
```

You must also modify your application code to use the *sp_set_session_context* system stored procedure, which sets the value returned by the SESSION_CONTEXT system function, as shown in Example 2-10. This system stored procedure supports two parameters—the key name of the value to add and the value to store for this key. In this example, the key name is *UserID* and its value is set to the *UserId* of the application user, which the application passes into the stored procedure by using the *@UserId* input parameter. Applications can call *sp_set_session_context* in line within the stored procedures or directly at application startup when the connection is created.

Example 2-10: Using the *sp_set_session_context* system stored procedure

```
CREATE PROCEDURE GetOrder
      @OrderId int,
      @UserId int
AS
EXEC sp_set_session_context @key=N'UserId', @value=@UserId;
SELECT *
FROM Orders
WHERE OrderId = @OrderId;
GO
```

Creating security policies

After creating inline table-valued functions, you next bind them to the table that you want to secure. To do this, use the CREATE SECURITY POLICY command, as shown in Example 2-11. In the security policy, you can define a filter predicate by specifying the inline table-valued function name, the column name to pass to the function, and the table to which the policy applies.

Example 2-11: Creating a security policy

```
CREATE SECURITY POLICY dbo.OrderPolicy
    ADD FILTER PREDICATE dbo.fn_Orders(SalesRep) ON dbo.Orders
    WITH (STATE=ON);
```

You can specify multiple filter predicates in the security policy when you want to filter rows in different tables, as shown in Example 2-12.

Example 2-12: Creating one security policy for multiple tables

```
CREATE SECURITY POLICY dbo.OrderPolicy
    ADD FILTER PREDICATE dbo.fn_Orders(SalesRep) ON dbo.Orders,
    ADD FILTER PREDICATE dbo.fn_OrderHistory(OrderId) ON dbo.OrderHistory
    WITH (STATE = ON);
```

Using block predicates

When you use the filter predicate as shown in the examples in the preceding section, the security policy affects "get" operations only. Users are still able to insert rows that they cannot subsequently query. They can also update rows they can currently access and even change the rows to store values that block further access. You must decide whether your application should allow this behavior or should prevent users from inserting rows to which they do not have access. To do this, use a block predicate in addition to a filter predicate.

As shown in Example 2-13, you can use both filter and block predicates in a security policy. In this example, the security policy allows users to query for rows using the SELECT statement and returns only rows to which the user has access. A user can insert new rows into the table as long as the SalesRep value matches the user's login. Otherwise, the insert fails and returns an error to the user. Similarly, an update to the table succeeds as long as the user doesn't attempt to change the value of the SalesRep column. In that case, the update fails and returns an error to the user.

Example 2-13: Using block and filter predicates in a single security policy

```
CREATE SECURITY POLICY dbo.OrderPolicy
    ADD FILTER PREDICATE dbo.fn_Orders(SalesRep) ON dbo.Orders,
    ADD BLOCK PREDICATE dbo.fn_Orders(SalesRep) ON dbo.Orders AFTER INSERT,
    ADD BLOCK PREDICATE dbo.fn_Orders(SalesRep) ON dbo.Orders AFTER UPDATE
    WITH (STATE = ON);
```

Note You can use a filter predicate to prevent users from updating or deleting records they cannot read, but the filter is silent. By contrast, the block predicate always returns an error when performing these operations.

Using RLS in SQL Database

You can use RLS in SQL database by using the same T-SQL commands described in this chapter. At the time of this writing, you cannot use the Azure portal to implement RLS.

Dynamic data masking

When you have a database that contains sensitive data, you can use dynamic data masking to obfuscate a portion of the data unless you specifically authorize a user to view the unmasked data. To mask data, you can use one of the following four masking functions to control how users see the data returned by a query:

- **Default** Use this function to fully mask values by returning a value of XXXX (or fewer Xs if a column length is less than 4 characters) for string data types, 0 for numeric and binary data types, and 01.01.2000 00:00:00.0000000 for date and time data types.
- **Email** Use this function to partially mask email addresses like this: `aXXX@XXXX.com`. This pattern masks not only the email address but also the length of the email address.
- **Partial** Use this function to partially mask values by using a custom definition requiring three parameters as described in the following table:

Parameter	Description
Prefix	Number of starting characters to display, starting from the first character in the value.
Padding	Value to be displayed between the prefix and suffix characters.
Suffix	Number of ending characters to display, starting from the last character in the value.

- **Random** Use this function to fully mask numeric values by using a random value between a lower and upper boundary that you specify.

Random function may display unmasked data

The *Random()* data-masking function may on occasion display the actual value that is stored in the table. This behavior is the result of using a random value that could match the value to mask if it is within the specified range. You should consider whether the business rules of your application allow for this behavior before using this masking function. Whenever possible, use a range of values outside the possible range of values to mask to ensure that there is no possibility of an accidental data leak. While it is possible that the random value will return the actual value, there is no way of knowing that the displayed random value is in fact the actual value without knowing the actual value.

Dynamic data masking of a new table

To configure dynamic data masking for a new table, use the CREATE TABLE statement with the MASKED WITH argument, as shown in Example 2-14. In this example, the *default()* function masks the TaxId column for complete masking, and the *partial()* function masks the FirstName column by displaying its first three characters and its final character and replacing the remaining characters with xyz.

Example 2-14: Creating a table with two masked columns

```
CREATE TABLE [dbo].[Customer](
      [CustomerId] [int] IDENTITY(1,1) NOT NULL,
      [TaxId] [varchar](11) MASKED WITH (FUNCTION = 'default()'),
      [FirstName] [nvarchar](50) MASKED WITH (FUNCTION = 'partial(3, "xyz", 1)') NULL,
      [LastName] [nvarchar](50) NULL,
PRIMARY KEY CLUSTERED
(
      [CustomerId] ASC)
) ON [PRIMARY];
GO
```

Dynamic data masking of an existing table

Because dynamic data masking changes only the presentation of data returned by a query, there is no change to the underlying table structure. That means you can easily add dynamic data masking to a column in an existing table without rebuilding the table. To this, use the ALTER TABLE statement with the ALTER COLUMN and ADD MASKED arguments, as shown in Example 2-15.

Example 2-15: Adding dynamic data masking to an existing table

```
ALTER TABLE [dbo].[Customers]
ALTER COLUMN [LastName] ADD MASKED WITH (FUNCTION = 'default()');
```

Likewise, you can remove dynamic data masking quickly and easily without rebuilding a table or moving data because only metadata changes rather than the schema. You remove dynamic data masking from a column by using the ALTER TABLE statement with the ALTER COLUMN and DROP MASKED arguments, as shown in Example 2-16.

Example 2-16: Removing dynamic data masking from a table

```
ALTER TABLE [dbo].[Customers]
ALTER COLUMN [LastName] DROP MASKED;
```

Understanding dynamic data masking and permissions

When you use dynamic data masking, the permissions that you assign to users affect whether users see plain text values or masked values. Specifically, members of the db_owner fixed database role always see plain text values, whereas users who are not members of this role see masked data by default.

If you need to grant a user permission to see plain text data in a table, you must grant the new UNMASK permission at the database level. To do this, use the GRANT UNMASK statement in the database containing the masked values, as shown in Example 2-17.

Example 2-17: Granting the UNMASK permission

```
GRANT UNMASK TO MyUser;
```

Note It is not possible to grant table-level access to masked data. You can grant this privilege only at the database level. Consequently, you can mask either all masked data within the database for a user or none of the data.

To remove this permission, you use the REVOKE statement as shown in Example 2-18.

Example 2-18: Revoking the UNMASK permission

```
REVOKE UNMASK TO MyUser;
```

Figure 2-4 shows examples of query results when you apply dynamic data masking to a table. The first query shows default and email masking. The second result set shows the same queries executed after giving the user permissions to view masked data.

```
/*Query the table with masking enabled*/
execute as user='MyUser'

select *
from customer

revert
/*Allow the user to see unmasked values*/
GRANT UNMASK TO MyUser;

/*Query the table with masking disabled*/
execute as user='MyUser'

select *
from customer

revert

REVOKE UNMASK TO MyUser;
```

Results | Messages

CustomerId	TaxId	FirstName	LastName	MiddleName	Address1	Address2	Address3	City	PostalCode	State	BirthDate	Email
1	xxxx	xxxx	test	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	bXXX@XXXX.com
2	xxxx	xxxx	test	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	jXXX@XXXX.com

CustomerId	TaxId	FirstName	LastName	MiddleName	Address1	Address2	Address3	City	PostalCode	State	BirthDate	Email
1	111-11-1111	Bill	test	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	bgates@contoso.co
2	111-11-1112	Jonathon	test	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	jsmith@contoso.net

Figure 2-4: Query results for masked and unmasked values.

Data-masking permissions and configuration survive when you copy data from one object to another. For example, if you copy data from a user table to a temporary table, the data remains masked in the temporary table.

Masking encrypted values

Dynamic data masking does not work with encrypted values if you encrypt data in the application tier or by using the Always Encrypted feature. If you encrypt data before storing it in the SQL Server database engine, the engine cannot mask a value that it cannot decrypt. In this case, because data is already encrypted, there is no benefit or extra protection from applying dynamic data masking.

Using dynamic data masking in SQL Database

Dynamic data masking is also available for use in SQL Database. You can configure it by using T-SQL or by using the Microsoft Azure portal. In the Azure portal, navigate to the list of SQL Databases within SQL DB, and then select the database to view its properties. Next, in the Settings panel, select Dynamic Data Masking, as shown in Figure 2-5. In the Dynamic Data Masking window, a list of masking rules is displayed in addition to a list of columns for which data masking is recommended. You can enable data masking on those columns by clicking the Add Mask button to the right of the column name.

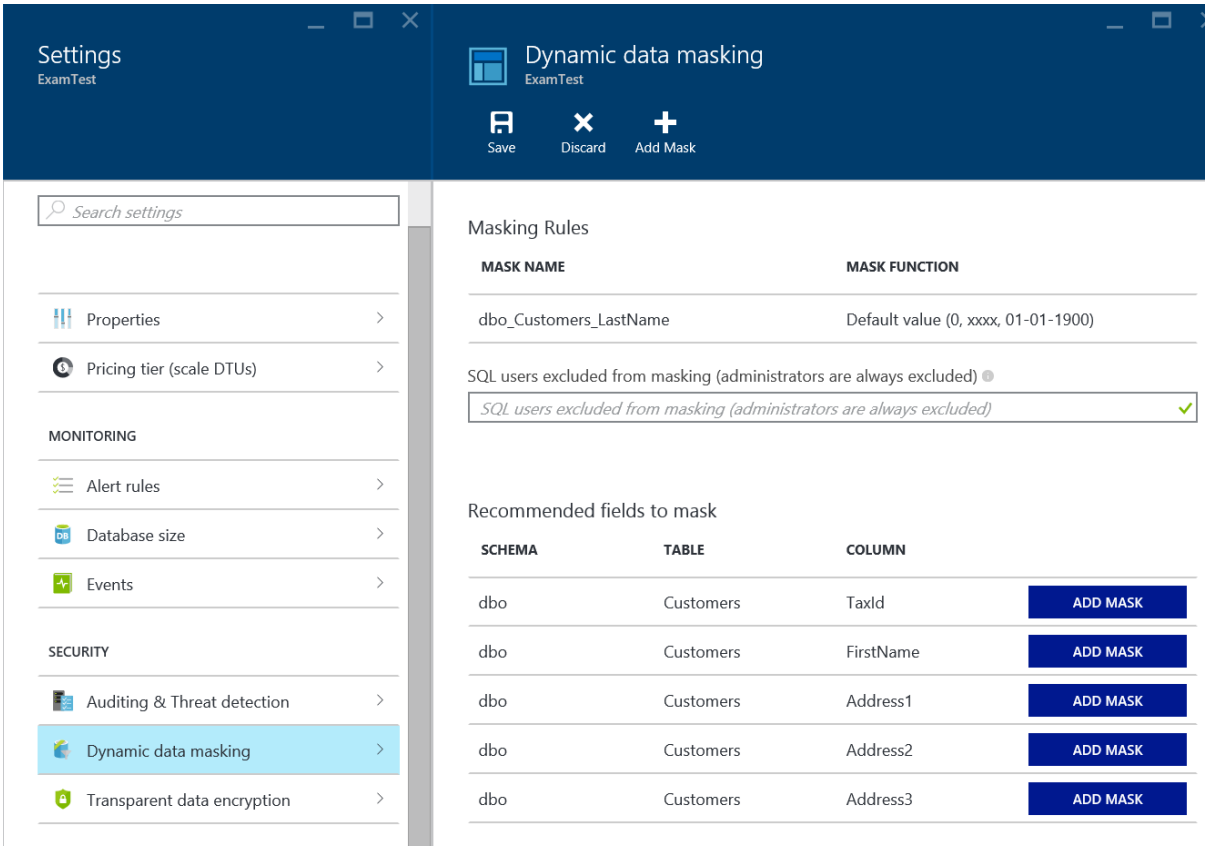


Figure 2-5: Configuring dynamic data masking for a SQL Database in the Azure portal.

After specifying the mask function to apply to selected columns, click the Save button at the top of the window to save the configuration changes to your SQL Database. After saving these changes, users can no longer see the unmasked data in the SQL Database tables unless they have the unmask privilege within the database.

Get the latest news from Microsoft Press sent to your inbox.

- New and upcoming books
- Special offers
- Free eBooks
- How-to articles

Sign up today at MicrosoftPressStore.com/Newsletters

Microsoft

CHAPTER 3

Higher availability

In a world that is always online, maintaining uptime and streamlining maintenance operations for your mission-critical applications are more important than ever. In SQL Server 2016, the capabilities of the AlwaysOn Availability Group feature continue to evolve from previous versions, enabling you to protect data more easily and flexibly and with greater throughput to support modern storage systems and CPUs. Furthermore, AlwaysOn Availability Groups and AlwaysOn Failover Cluster Instances now have higher security, reliability, and scalability. By running SQL Server 2016 on Windows Server 2016, you have more options for better managing clusters and storage. In this chapter, we introduce the new features that you can use to deploy more robust high-availability solutions.

AlwaysOn Availability Groups

First introduced in SQL Server 2012 Enterprise Edition, the AlwaysOn Availability Groups feature provides data protection by sending transactions from the transaction log on the primary replica to one or more secondary replicas, a process that is conceptually similar to database mirroring. In SQL Server 2014, the significant enhancement to availability groups was the increase in the number of supported secondary replicas from three to eight. SQL Server 2016 includes a number of new enhancements that we explain in this section:

- AlwaysOn Basic Availability Groups
- Support for group Managed Service Accounts (gMSAs)
- Database-level failover
- Distributed Transaction Coordinator (DTC) support
- Load balancing for readable secondary replicas
- Up to three automatic failover targets
- Improved log transport performance

New to availability groups?

If you are still using database mirroring, there are several reasons to transition your high-availability strategy to availability groups. Database mirroring is deprecated as of SQL Server 2012, for example, and basic availability groups are now included in SQL Server 2016 Standard Edition as a replacement. Also, if you are exploring options for high-availability/disaster-recovery (HA/DR) solutions but have never implemented availability groups, SQL Server 2016 provides several benefits to consider.

Whereas database mirroring occurs at the database level, using a single thread to perform the data replication, data is moved within availability groups by using a worker pool, which provides better throughput and reduces CPU overhead. When your application requires multiple databases, you can assign the databases to a single availability group to ensure that they all fail over at the same time. By contrast, the unit of failover with database mirroring is a single database. With database mirroring, you use a SQL Server witness instance to manage automatic failover, but with availability groups you rely on Windows Server Failover Clustering (WSFC) to arbitrate uptime and connections. Furthermore, clustering is a more robust solution than database mirroring because it provides additional levels of protection.

A key benefit of availability groups is the ability to scale out replicas that you can configure to support both high-availability and disaster-recovery requirements. For high-availability scenarios, you should locate two or three servers in the same geographic location, configured to use synchronous-commit mode and automatic failover. That said, automatic failover should be used only in low-latency scenarios because writes to the primary replica are not considered complete until they reach the transaction log on the secondary replica. For disaster-recovery scenarios in which the servers are more than 100 kilometers apart, asynchronous-commit mode is a better choice to minimize the performance impact on the primary replica.

Another benefit of availability groups is the ability for databases on a secondary replica to support online reads as well as database backups. This capability allows you to implement a scale-out architecture for reporting solutions by having multiple copies of secondary replicas in multiple geographies. You provide connectivity to the availability group by using a virtual IP address called the *listener*, which you configure to connect transparently to the primary replica or to a secondary replica for reading. Figure 3-1 is a diagram of an availability group with replicas in New York, Los Angeles, and Seattle and a listener to which clients connect.

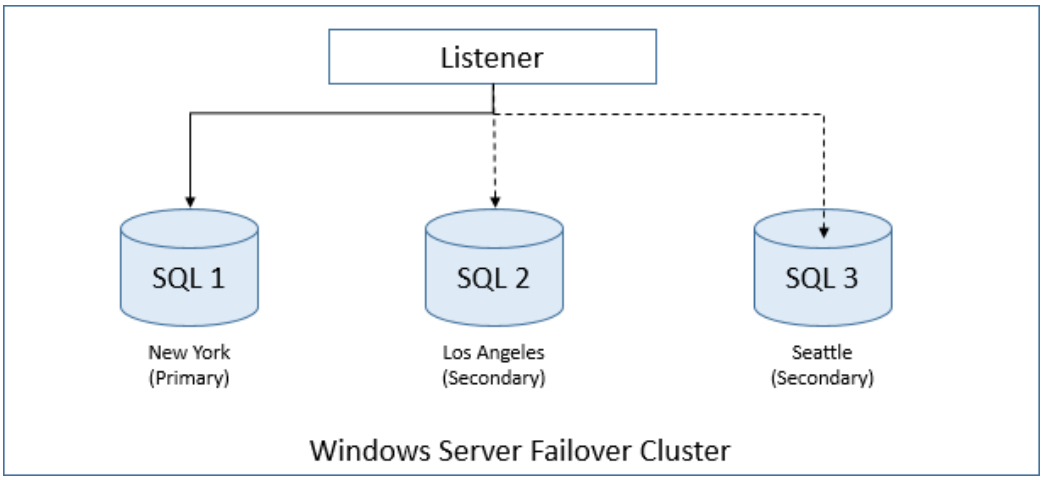


Figure 3-1: An AlwaysOn Availability Group with a primary replica and two secondary replicas.

Supporting disaster recovery with basic availability groups

You can now use basic availability groups in the Standard Edition to automatically fail over a single database. The use of basic availability groups is subject to the following limitations:

- Two replicas (one primary and one secondary)
- One availability database
- No read access on secondary replica
- No backups on secondary replica
- No availability group listener
- No support in an existing availability group to add or remove a replica
- No support for upgrading a basic availability group to an advanced availability group

Despite these limitations, with a basic availability group you get benefits similar to database mirroring in addition to other features. For each replica, you can choose either synchronous-commit or asynchronous-commit mode, which determines whether the primary replica waits for the secondary replica to confirm that it has hardened the log. Moreover, performance is better because basic availability groups use the same improved log transport operations that we describe later in this chapter.

The steps to configure basic availability groups are similar to those for regular availability groups, with some exceptions. You start by using the New Availability Group Wizard, which you launch in SQL Server Management Studio (SSMS) by right-clicking the AlwaysOn High Availability folder in Object Explorer. When you reach the Specify Replicas page, you click the Add Replica button to add the primary and secondary replicas, but then the button becomes unavailable, as shown in Figure 3-2. In addition, you cannot change the value for the Readable Secondary drop-down list, nor can you access the Backup Preferences or Listener tabs.

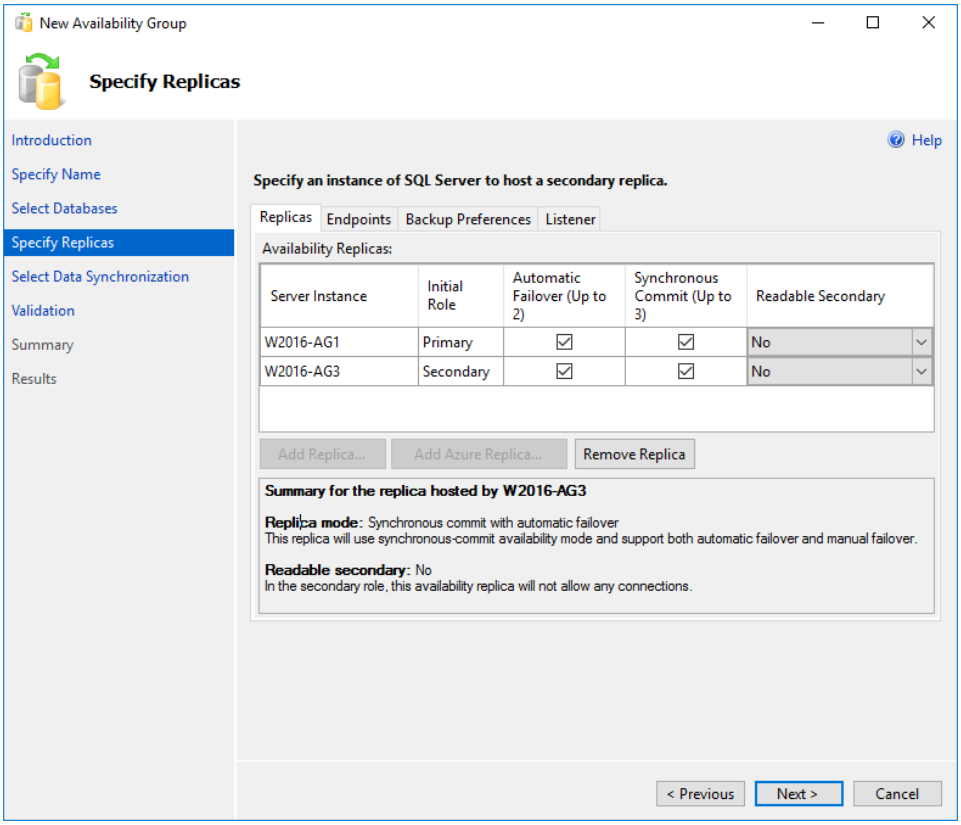


Figure 3-2: Configuring replicas for a basic availability group.

Note Although including an Azure replica in your disaster-recovery architecture is fully supported for basic availability groups, the New Availability Group Wizard does not allow you the option to add it. However, you can perform this step separately by using the Add Azure Replica Wizard, which is described in "Use the Add Azure Replica Wizard (SQL Server)" at https://msdn.microsoft.com/en-us/library/dn463980.aspx.

Using group Managed Service Accounts

To comply with regulatory auditing requirements, DBAs or system administrators in a large enterprise must frequently reset service account passwords across SQL Server instances. However, managing individual service account passwords involves a high degree of risk because downtime is likely to occur if anything goes wrong. To address this problem, Microsoft enhanced Windows Server 2012 so that you can more easily manage passwords for a service account in Active Directory by creating a single service account for your SQL Server instances and then delegating permissions to each of those servers. By default, Active Directory changes the password for a group Managed Service Account (gMSA) every thirty days, although you can adjust the password-change interval to satisfy your audit requirements.

In SQL Server 2012 and SQL Server 2014, you can implement this feature only in standalone configurations. In SQL Server 2016, you can now use gMSAs with both availability groups and failover clusters. If you are using Windows Server 2012 R2 as your operating system, you must install KB298082 to ensure that services can seamlessly log on after a password change. However, no patches are required if you install SQL Server 2016 on Windows Server 2016.

Triggering failover at the database level

Beginning in SQL Server 2012, AlwaysOn Availability Groups and AlwaysOn Failover Cluster Instances (FCIs) use the *sp_server_diagnostics* stored procedure to periodically monitor the health of a server. The default behavior is to fail over an availability group or an FCI when the health monitoring reveals any of the following conditions:

- The stored procedure returns an error condition.
- The SQL Service service is not running.
- The SQL Server instance is not responding.

However, in versions earlier than SQL Server 2016, this check does not account for database-level failures. Beginning in SQL Server 2016, you can enable Database Level Health Detection when you create an availability group, as shown in Figure 3-3. This way, any error that causes a database to be suspect or go offline also triggers a failover of the availability group.

Note The *FailureConditionLevel* property determines the conditions that trigger a failover. For normal operations, the default value is suitable. However, you can reduce or increase this property's value if necessary. To learn more, see "Configure FailureConditionLevel Property Settings" at https://msdn.microsoft.com/en-us/library/ff878667.aspx.

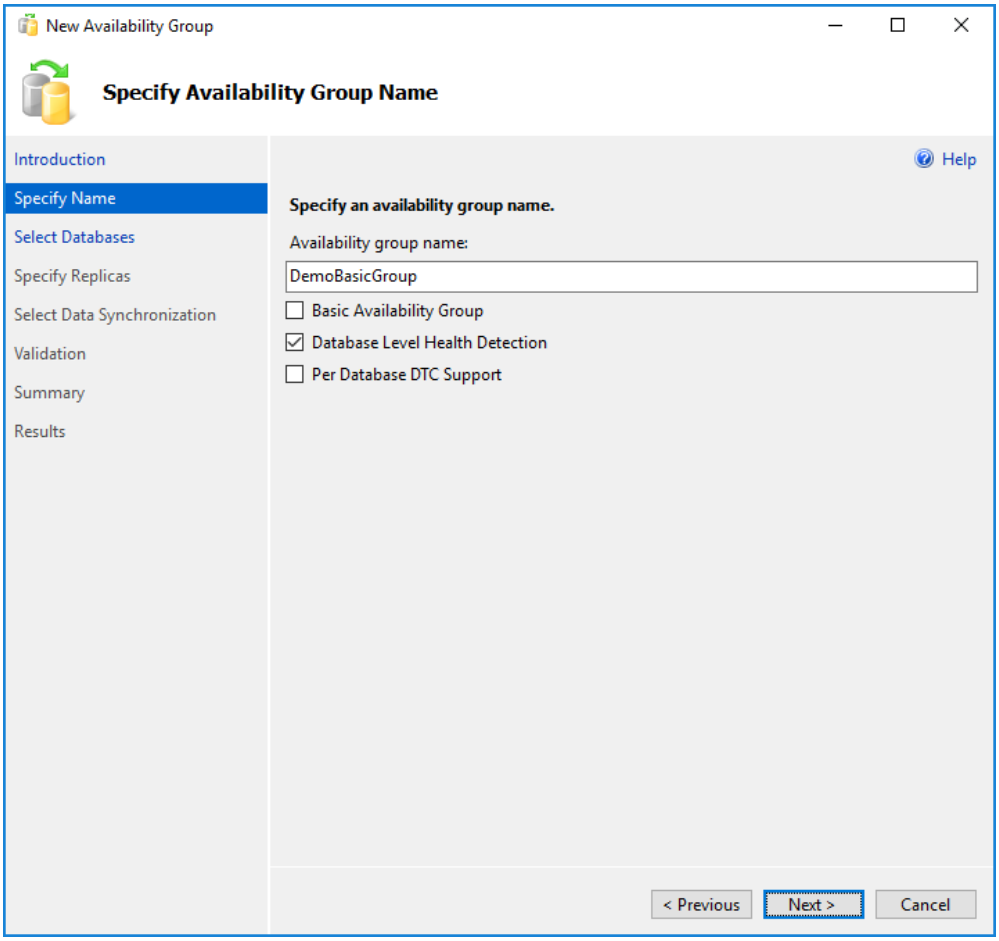


Figure 3-3: Creating a new availability group with database-level health detection.

Note Enabling Database Level Health Detection needs to be weighed carefully with the needs of your application and its intended behavior in response to a failover. If your application can support a database failover, Database Level Health Detection can enhance your total availability and uptime.

Automatic page repair

An important capability of availability groups is automatic page repair. If the primary replica cannot read a page, it requests a fresh copy of the page from a secondary. However, in the event that the primary replica cannot be repaired, such as when storage fails completely, you might be able to bring your secondary replica online as a primary replica.

Supporting distributed transactions

One of the features long supported in FCIs, but not in availability groups, is the use of the Distributed Transaction Coordinator (DTC). This feature is required if your application performs transactions that must be consistent across multiple instances. When running SQL Server 2016 on Windows Server 2016, you can now implement support for distributed transactions when you create a new availability group. To do this, you select the Per Database DTC check box (shown earlier in Figure 3-3) or by using the T-SQL command shown in Example 3-1. Note that you cannot add DTC support to an existing

availability group. By enabling DTC support, your application can distribute transactions between separate SQL Server instances or between SQL Server and another DTC-compliant server, such as Oracle or WebSphere.

Example 3-1: Creating an availability group with DTC support

```
CREATE AVAILABLITY GROUP [2016DEMO] WITH DTC_SUPPORT=PER_DB
```

Note Because each database in an availability group is synchronized independently while the cross-database transaction manager operates at the SQL Server instance level, an active cross-database transaction might be lost during an availability group failover. Consequently, cross-database transactions are not supported for databases hosted by one SQL Server or within the same availability group.

Scaling out read workloads

You can use availability groups for scale-out reads across multiple secondary copies of the availability group. In SQL Server 2016, as in the previous version, you can scale up to as many as eight secondary replicas, but the scale-out reads feature is not enabled by default. To support scale-out reads, you must configure read-only routing by using the T-SQL commands shown in Example 3-2. You must also create an availability group listener and direct connections to this listener. In addition, the connection string must include the *ApplicationIntent=ReadOnly* keyword.

Example 3-2: Configuring read-only routing

```
ALTER AVAILABILITY GROUP [AG1]
 MODIFY REPLICA ON
N'COMPUTER01' WITH
(SECONDARY_ROLE (ALLOW_CONNECTIONS = READ_ONLY));
ALTER AVAILABILITY GROUP [AG1]
 MODIFY REPLICA ON
N'COMPUTER01' WITH
(SECONDARY_ROLE (READ_ONLY_ROUTING_URL = N'TCP://COMPUTER01.contoso.com:1433'));

ALTER AVAILABILITY GROUP [AG1]
 MODIFY REPLICA ON
N'COMPUTER02' WITH
(SECONDARY_ROLE (ALLOW_CONNECTIONS = READ_ONLY));
ALTER AVAILABILITY GROUP [AG1]
 MODIFY REPLICA ON
N'COMPUTER02' WITH
(SECONDARY_ROLE (READ_ONLY_ROUTING_URL = N'TCP://COMPUTER02.contoso.com:1433'));

ALTER AVAILABILITY GROUP [AG1]
 MODIFY REPLICA ON
N'COMPUTER01' WITH
(PRIMARY_ROLE (READ_ONLY_ROUTING_LIST=('COMPUTER02','COMPUTER01')));

ALTER AVAILABILITY GROUP [AG1]
 MODIFY REPLICA ON
N'COMPUTER02' WITH
(PRIMARY_ROLE (READ_ONLY_ROUTING_LIST=('COMPUTER01','COMPUTER02')));
GO
```

Note You can also use Windows PowerShell to configure a read-only routing list, as described in "Configure Read-Only Routing for an Availability Group (SQL Server)," at https://msdn.microsoft.com/en-us/library/hh710054.aspx.

In SQL Server 2012 and SQL Server 2014, the read traffic is directed to the first available replica, without consideration for load balancing. An alternative solution requires the use of third-party hardware or software load balancers to route traffic equally across the secondary copies of the databases. In SQL Server 2016, you can now balance loads across replicas by using nested parentheses in the read-only routing list, as shown in Example 3-3. In this example, connection requests first try the load-balanced set containing Server1 and Server2. If neither replica in that set is available, the request continues by sequentially trying other replicas defined in the list, Server3 and Server4 in this example. Only one level of nested parentheses is supported at this time.

Example 3-3: Defining a load-balanced replica list

```
READ_ONLY_ROUTING_LIST = (('Server1','Server2'), 'Server3', 'Server4')
```

Defining automatic failover targets

In SQL Server 2012 and SQL Server 2014, you can define a maximum of two replicas running in an automatic failover set, but now SQL Server 2016 allows for a third replica to support a topology such as is shown in Figure 3-4. In this example, the replicas on Node01, Node02, and Node03 are configured as an automatic failover set. As long as data is synchronized between the primary replica and one of the secondary replicas, failover can take place in an automatic fashion with no data loss.

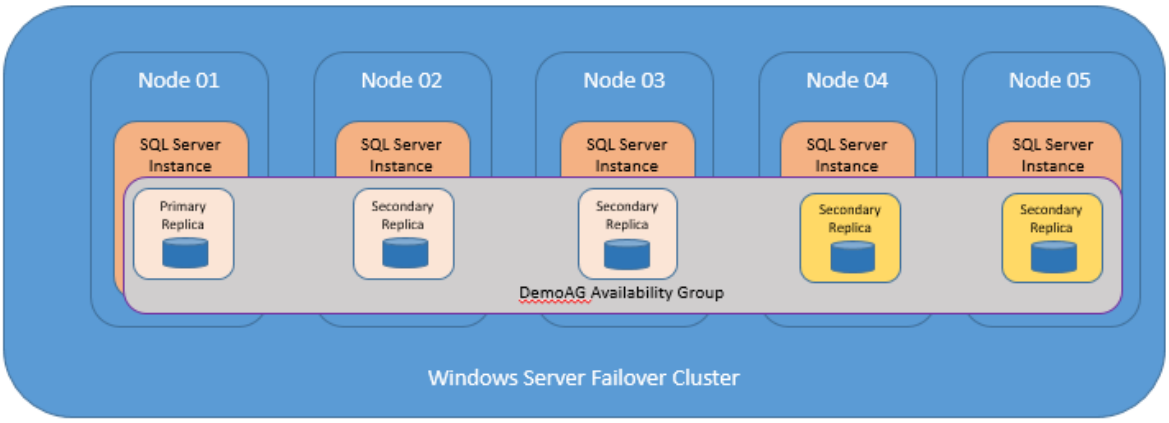


Figure 3-4: Availability Group topology with three automatic failover targets.

When configuring availability group failover, you can choose from among the following failover modes:

- **Automatic Failover** A failover that occurs automatically on the failure of the primary replica, which is supported only when both the primary replica and at least one secondary replica are configured with AUTOMATIC failover mode and the secondary replica is currently synchronized.
- **Planned Manual Failover (without data loss)** A failover that is typically initiated by an administrator for maintenance purposes. This requires synchronous-commit mode, and the databases must currently be synchronized.

- **Forced Failover (with possible data loss)** A failover that occurs when the availability group is configured with asynchronous-commit mode, or the databases in the availability group are not currently synchronized.

For automatic failover to work, you must configure all members of the availability group for synchronous-commit mode and for automatic failover. You typically configure automatic failover for high-availability scenarios, such as rolling updates to SQL Server. In this configuration, any uncommitted transactions that have not reached the secondary replica are rolled back in the event of failover, thereby providing near zero data loss.

Reviewing the improved log transport performance

When AlwaysOn Availability Groups were first introduced in SQL Server 2012, solid-state storage devices (SSDs) were far less prevalent in enterprise IT architectures than they are now. SSDs enable more throughput, which can be problematic on a standalone system and can overwhelm the ability to write to the secondary database. In prior versions of SQL Server, the underlying processes responsible for synchronizing data between replicas are shared among availability groups, database mirroring, Service Broker, and replication. In SQL Server 2016, these processes are completely rewritten, resulting in a streamlined protocol with better throughput and lower CPU utilization.

Although the process has been refactored, the sequence of internal operations for the log transport, shown in Figure 3-5, continues to include the following steps:

1. **Log flush** Log data is generated and flushed to disk on the primary replica in preparation for replication to the secondary replica. It then enters the send queue.
2. **Log capture** Logs for each database are captured on the primary replica, compressed, and sent to the corresponding queue on the secondary replica. This process runs continuously as long as database replicas are connecting. If this process is not able to scan and enqueue the messages quickly enough, the log send queue continues to grow.
3. **Send** The messages are removed from the queue and sent to each secondary replica across the network.
4. **Log receive/Log cache** Each secondary replica gets messages from the primary replica and then caches the messages.
5. **Log hardened** The log is flushed on the secondary replica, and then a notification is sent to the primary replica to acknowledge completion of the transaction.
6. **Redo pages** The flushed pages are retrieved from the redo queue and applied to the secondary replica.

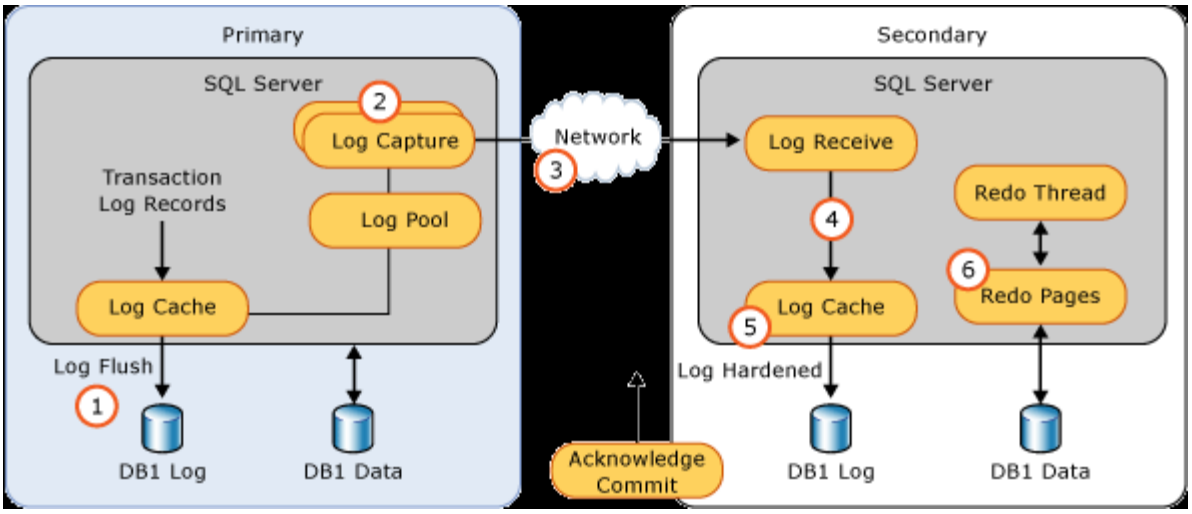


Figure 3-5: Log transport operations for AlwaysOn Availability Groups.

Bottlenecks can occur in this process during the log-capture step on the primary replica and the redo step on the secondary replica. In previous versions of SQL Server, both steps were single-threaded. Consequently, bottlenecks might occur during large index rebuilds on availability groups with high-speed storage and on local networks, because these single-threaded steps had trouble keeping up with the stream of log records. However, in SQL Server 2016 these steps can use multiple threads that run in parallel, resulting in significant performance improvements. Furthermore, the compression functions in the log-capture step have been replaced by a newer Windows compression function that delivers up to five times better performance. During testing with high-throughput storage devices, speeds up to 500 MB/s have been observed. Considering that this throughput is a compressed stream, the redo step is receiving 1 GB/s, which should support the busiest applications on the fastest storage.

Microsoft Azure high-availability/disaster-recovery licensing changes

Hybrid disaster-recovery scenarios are becoming increasingly popular. If you choose to implement hybrid disaster recovery, be sure to maintain symmetry between on-premises and cloud solutions. The license mobility benefit included with software assurance (SA) allows you to use a secondary copy of SQL Server in any high-availability or disaster-recovery scenario without purchasing another license for it. In the past, you could not use this benefit with the SQL Server images on Azure Virtual Machines. Now you can deploy a new SQL Server image on an Azure Virtual Machine without incurring charges as long as the secondary replica is not active. This means you can automate the scale-out of your high-availability/disaster-recovery solutions with a minimum of effort and cost.

Windows Server 2016 Technical Preview high-availability enhancements

Nearly every version of Windows Server since Windows Server 2008 R2 has had major enhancements to the operating system's failover clustering stack as a result of development investments in related technologies. First, Hyper-V, the virtualization platform in the operating system, uses the clustering stack for its high-availability and disaster-recovery scenarios. Microsoft Azure also uses this same functionality. Because SQL Server has failover clustering at the center of its high-availability/disaster-recovery technologies, it also takes advantage of the clustering features in the operating system. Sometimes these features are visible from the database tier, allowing you to make configuration

changes, but other features from the operating system, such as dynamic quorum, enhance SQL Server's uptime without requiring configuration. Windows Server 2016 Server Technical Preview includes the following features that enhance SQL Server's uptime:

- Workgroup clusters
- Cloud witness
- Storage Spaces Direct
- Site-awareness
- Troubleshooting enhancements to Windows Server Failover Clusters (WSFC)
- Cluster operating system rolling upgrade

Creating workgroup clusters

Earlier in this chapter, we explained how basic availability groups replace nearly all the functionality in database mirroring. The one advantage that database mirroring has over availability groups in prior versions of SQL Server is the ability to provide data protection across Active Directory (AD) domains or without an AD domain. Starting with Windows Server 2016 Technical Preview and SQL Server 2016, you can now create a workgroup cluster with nondomain servers as well as servers attached to different AD domains. However, there is no mechanism for using a file share witness. Instead, you must create a cloud witness, as we describe in the next section, or a shared disk.

Each server that you want to add to a workgroup cluster requires a primary DNS suffix in the full computer name, as shown in Figure 3-6. Add this suffix by clicking the More button in the Computer Name/Domain Changes dialog box, which you access from System Properties for the server.

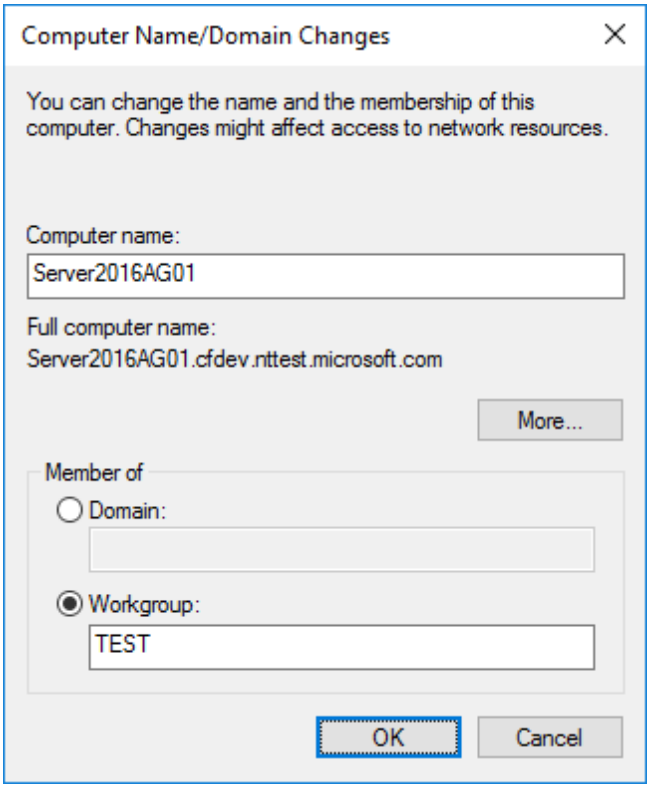


Figure 3-6: A server with a DNS suffix assigned to a workgroup.

Note In the current release of Windows Server 2016 Technical Preview, this feature is enabled by default. However, you must use PowerShell to create a workgroup cluster because the Failover Cluster Manager snap-in does not support this functionality. Furthermore, configuring a cross-domain cluster is more complex than configuring a workgroup cluster because Kerberos is required to make cross-domain authentication work correctly. You can learn more about configuration requirements for both scenarios by referring to "Workgroup and Multi-domain clusters in Windows Server 2016" at http://blogs.msdn.com/b/clustering/archive/2015/08/17/10635825.aspx.

Configuring a cloud witness

Maintaining quorum is the most important function of any clustering software. In SQL Server, the most frequently deployed model is Node And File Share Majority. One of the challenges when you are designing a disaster-recovery architecture is to decide where to place the file share witness. Microsoft's official recommendation is to place it in a third data center, assuming you have a primary/secondary availability group configuration. Many organizations already have two data centers, but fewer have a third data center. Windows Server 2016 Technical Preview introduces a new feature, the cloud witness, that address this problem.

You create a cloud witness by using the Cluster Quorum Wizard. Before you launch this wizard, you must have an active Azure subscription and a storage account. To launch the wizard, right-click the server in the Failover Cluster Manager, point to More Actions, select Configure Cluster Quorum Settings, select the Select The Quorum Witness option, and then select the Configure A Cloud Witness option, as shown in Figure 3-7.

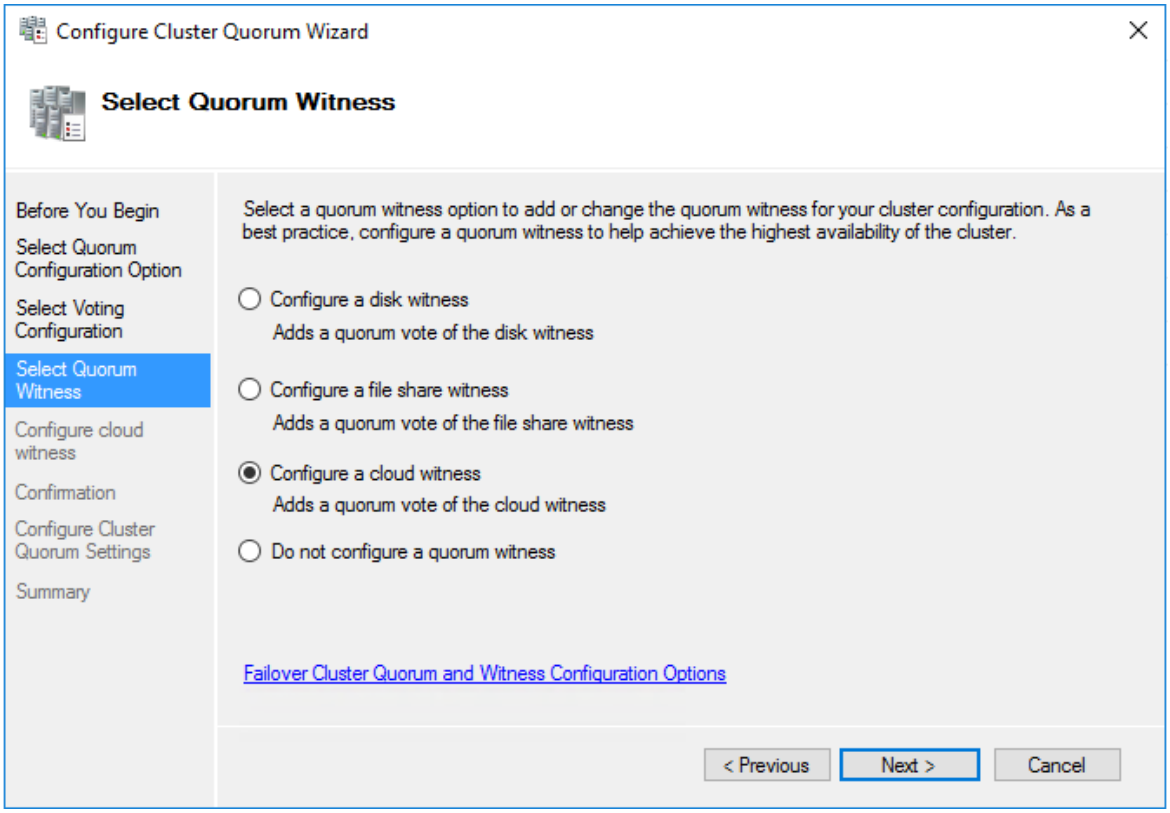


Figure 3-7: Creating a cloud witness for a cluster quorum.

On the next page of the wizard, provide the name of your Azure storage account, copy the storage account key from the Azure portal to the clipboard, and type the Azure service endpoint, as shown in Figure 3-8.

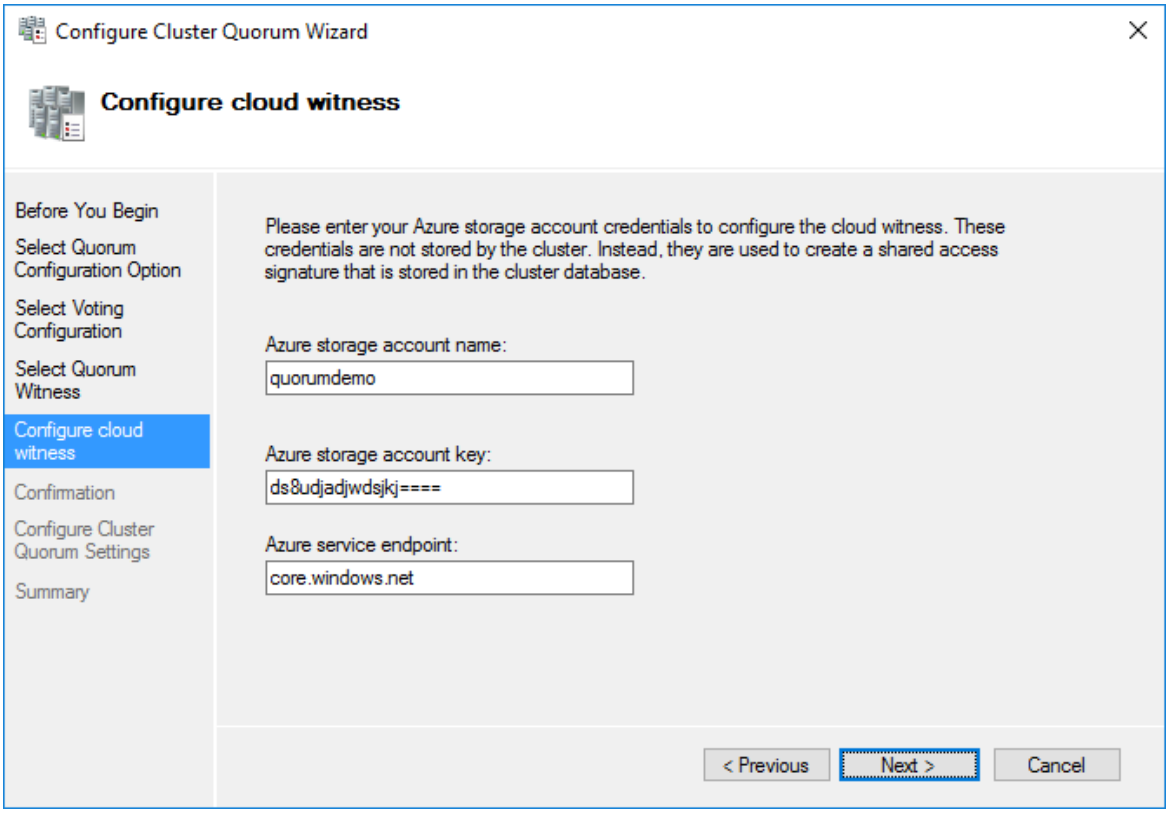


Figure 3-8: Addition of Azure credentials to cloud witness configuration.

When you successfully complete the wizard, the cloud witness is displayed in the Cluster Core Resources pane in the Failover Configuration Manager snap-in, as shown in Figure 3-9.

Figure 3-9: Successful addition of a cloud witness to cluster core resources.

Because you select the Azure region when you create a storage account, you have control over the placement of your cloud witness. All communications to and from Azure storage are encrypted by default. This new feature is suitable for most disaster-recovery architectures and is easy to configure at minimal costs.

Note More information on this topic is available in "Understanding Quorum Configurations in a Failover Cluster" at https://technet.microsoft.com/en-us/library/cc731739.aspx.

Using Storage Spaces Direct

Windows Server 2016 Technical Preview introduces the Storage Spaces Direct feature, which seamlessly integrates several existing Windows Server features to build a software-defined storage stack, as shown in Figure 3-10. These features include Scale-Out File Server, Clustered Shared Volume File Systems (CSVFS), and Failover Clustering.

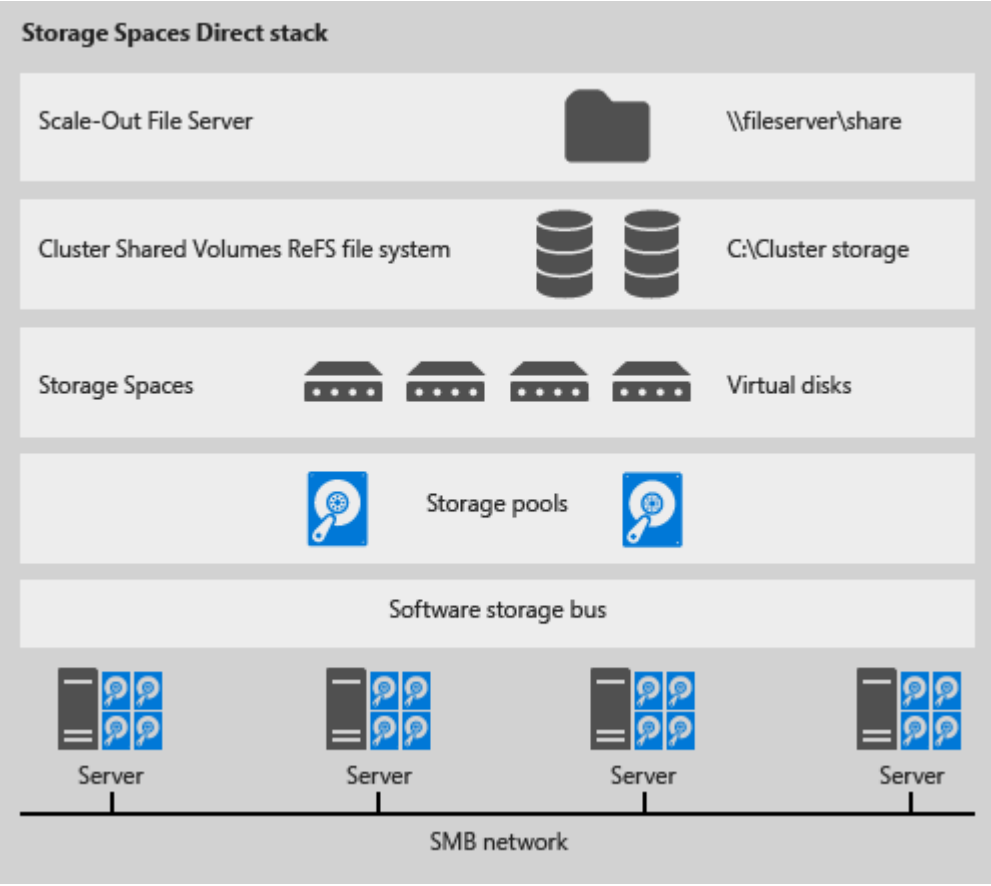


Figure 3-10: Storage options in Storage Spaces Direct.

The main use case for this feature is high-performance primary storage for Hyper-V virtual files. Additionally, storage tiers are built into the solution. If you require a higher tier of performance for TempDB volumes, you can configure Storage Spaces Direct accordingly. SQL Server can take full advantage of this feature set because the infrastructure supports AlwaysOn Availability Groups and AlwaysOn Failover Cluster Instances, thereby providing a much lower total cost of ownership compared with traditional enterprise storage solutions.

Note See "Storage Spaces Direct in Windows Server 2016 Technical Preview" at https://technet.microsoft.com/en-us/library/mt126109.aspx to learn more.

Introducing site-aware failover clusters

Windows Server 2016 Technical Preview also introduces site-aware clusters. As a consequence, you can now group nodes in stretched clusters based on their physical location. This capability enhances key clustering operations such as failover behavior, placement policies, heartbeat between nodes, and quorum behavior.

One of the key features of interest to SQL Server professionals is failover affinity, which allows availability groups to fail over within the same site before failing to a node in a different site. Additionally, you can now configure the threshold and site delay for heartbeating, which is the network ping that ensures the cluster can talk to all its nodes.

You can not only specify a site for a cluster node, you can also define a primary location, known as a *preferred site*, for your cluster. Dynamic quorum ensures that the preferred site stays online in the event of a failure by lowering the weights of the disaster-recovery site.

> **Note** Currently (in Windows Server 2016 TP4), the site-awareness functionality is only enabled through PowerShell and not through Failover Cluster Manager. More information is available in "Site-aware Failover Clusters in Windows Server 2016" at http://blogs.msdn.com/b/clustering/archive/2015/08/19/10636304.aspx.

Windows Server Failover Cluster logging

Troubleshooting complex cluster problems has always been challenging. One of the goals of WSFC logging in Windows Server 2016 is to simplify some of these challenges. First, the top of the cluster log now shows the UTC offset of the server and notes whether the cluster is using UTC or local time. The cluster log also dumps all cluster objects, such as networks, storage, or roles, into a comma-separated list with headers for easy review in tools such as Excel. In addition, there is a new logging model call *DiagnosticVerbose* that offers the ability to keep recent logs in verbose logging while maintaining a history in normal diagnostic mode. This compromise saves space but also provides verbose logging as needed.

> **Note** Additional information is available in "Windows Server 2016 Failover Cluster Troubleshooting Enhancements – Cluster Log" at http://blogs.msdn.com/b/clustering/archive/2015/05/15/10614930.aspx.

Performing rolling cluster operating system upgrades

In prior versions of SQL Server, if your SQL Server instance was running in any type of clustered environment and an operating system upgrade was required, you built a new cluster on the new operating system and then migrated the storage to the new cluster. Some DBAs use log shipping to bring the downtime to an absolute minimum, but this approach is complex and, more importantly, requires a second set of hardware. With rolling cluster operating system upgrades in Windows Server 2016, the process is more straightforward.

Specifically, SQL Server requires approximately five minutes of downtime in the rolling upgrade scenario illustrated in Figure 3-11. In general, the process drains one node at a time from the cluster, performs a clean install of Windows Server 2016, and then adds the node back into the cluster. Until the cluster functional level is raised in the final step of the upgrade process, you can continue to add new cluster nodes with Windows Server 2012 R2 and roll back the entire cluster to Windows Server 2012 R2.

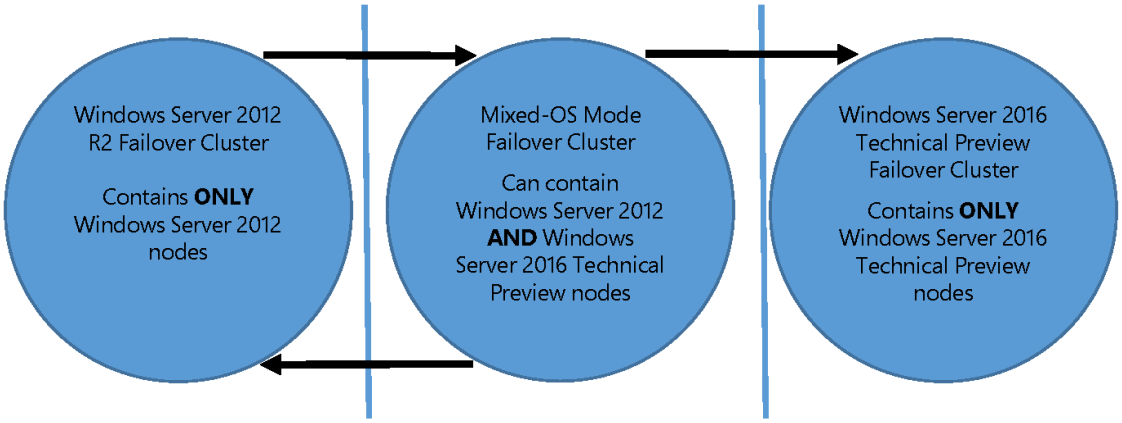


Figure 3-11: State transitions during a rolling operating system upgrade.

Note You use Failover Cluster Manager and PowerShell to manage the cluster upgrade. See "Cluster Operating System Rolling Upgrade" at https://technet.microsoft.com/en-us/library/dn850430.aspx to learn more.

CHAPTER 4

Improved database engine

In past releases of SQL Server, Microsoft has targeted specific areas for improvement. In SQL Server 2005, the storage engine was new. In SQL Server 2008, the emphasis was on server consolidation. Now, in SQL Server 2016, you can find enhanced functionality across the entire database engine. With Microsoft now managing more than one million SQL Server databases through its Database as a Service (DBaaS) offering—Microsoft Azure SQL Database—it is able to respond more quickly to opportunities to enhance the product and validate those enhancements comprehensively before adding features to the on-premises version of SQL Server. SQL Server 2016 is a beneficiary of this new development paradigm and includes many features that are already available in SQL Database. In this chapter, we explore a few of the key new features, which enable you to better manage growing data volumes and changing data systems, manage query performance, and reduce barriers to entry for hybrid cloud architectures.

TempDB enhancements

TempDB is one of the components for which performance is critical in SQL Server because the database engine uses it for temporary tables, query memory spills, index rebuilds, Service Broker, and a multitude of other internal functions. TempDB file behavior has been enhanced and automated in SQL Server 2016 to eliminate many performance problems related to the basic configuration of the

server. These changes allow administrators to focus their efforts on more pressing performance and data issues in their environments.

Configuring data files for TempDB

In earlier versions of SQL Server, the default configuration uses one data file for TempDB. This limitation sometimes results in page-latch contention, which has frequently been misdiagnosed by administrators as a storage input/output (I/O) problem for SQL Server. However, the pages for TempDB are typically in memory and therefore not contributing to I/O contention issues. Instead, three special types of pages are the cause of the page-latch contention issue: Global Allocation Map (GAM), Shared Global Allocation Map (SGAM), and Page Free Space (PFS). Each database file can contain many of these page types, which are responsible for identifying where to write incoming data in a physical data file. Whenever a process in SQL Server needs to use any of these files, a latch is taken. A latch is similar to a lock but is more lightweight. Latches are designed to be quickly turned on and just as quickly turned off when not needed. The problem with TempDB is that each data file has only one GAM, SGAM, and PFS page per four gigabytes of space, and a lot of processes are trying to access those pages, as shown in Figure 4-1. Subsequent requests begin to queue, and wait times for processes at the end of the queue increase from milliseconds to seconds.

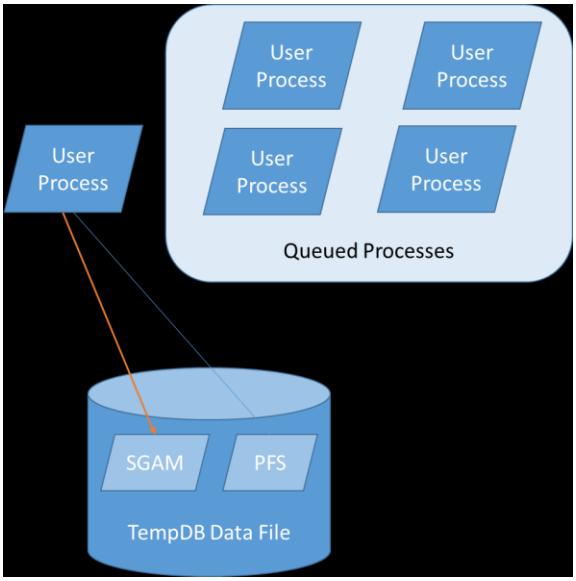


Figure 4-1: Contention in TempDB.

An easy way to remedy TempDB page-latch contention in SQL Server is to add more data files. In turn, SQL Server creates more of the three special types of pages and gives SQL Server more throughput to TempDB. Importantly, the files should all be the same size. SQL Server uses a proportional fill algorithm that tries to fill the largest files first, leading to hotspots and more latch contention. However, because the default setting creates only one file, many database administrators have not been aware of the solution. Even after learning about the need to create multiple files, there was often confusion about the correct number of files to configure, especially when factoring in virtual machines, hyperthreading, and cores versus CPU sockets.

In 2011, Microsoft released the following guidance for TempDB configuration:

As a general rule, if the number of logical processors is less than or equal to 8, use the same number of data files as logical processors. If the number of logical processors is greater than 8, use 8 data files and then if contention continues, increase the number of data files by multiples of 4 (up to the number of

logical processors) until the contention is reduced to acceptable levels or make changes to the workload/code.

> **Note** For more detail, see "Recommendations to reduce allocation contention in SQL Server tempdb database," at https://support.microsoft.com/en-us/kb/2154845.

Accordingly, in SQL Server 2016, this recommendation is built into the product setup. When you install SQL Server, the default configuration for TempDB now adapts to your environment, as shown in Figure 4-2. The setup wizard no longer creates a single file by default; instead, it assigns a default number of files based on the number of logical processors that it detects on the server, up to a maximum of 8. You can adjust the size of the files and the autogrowth rate if you like. Always monitor the growth of these files carefully, as performance is affected by file growth even when instant file initialization is enabled.

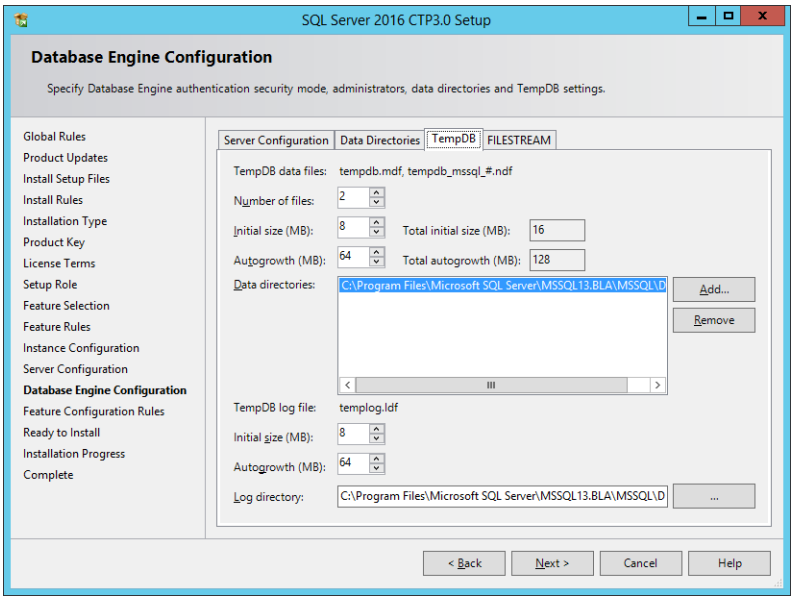


Figure 4-2: Configuring TempDB in SQL Server 2016.

> **Note** SQL Server defaults to a conservative setting of 8 megabytes (MB) for Initial Size and 64 MB for Autogrowth. A best practice is to start with an initial file size of 4,092 MB, with an autogrowth setting of 512 MB, as the initial file size is still small by most standards. Many DBAs dedicate a standard-size file system (typically 100–200 GB) to TempDB and allocate 90 percent of it to the data files. This sizing can reduce contention and also prevents any uncontrolled TempDB growth from impacting user databases.

Eliminating specific trace flags

Trace flags are commonly used by administrators to perform diagnostics or to change the behavior of SQL Server. With TempDB in earlier releases of SQL Server, administrators use trace flags 1117 and 1118 to improve performance. In SQL Server 2016, the effect achieved by enabling these two trace flags has been built into the database engine, rendering them unnecessary.

Trace flag 1117

Trace flag (TF) 1117 is related strictly to file groups and how data files grow within them. A file group is a logical container for one or more data files within a database. TF 1117 forces all data files in the same file group to grow at the same rate, which prevents one file from growing more than others,

leading to the hotspot issue described earlier in this chapter. Enabling this trace flag in earlier versions of SQL Server is a minor tradeoff in performance. For example, if you were using multiple data files in user databases, this trace flag affects them as well as TempDB's data files. Depending on your scenario, that could be problematic—an example would be if you had a file group that you did not want to grow as a single unit. Starting with SQL Server 2016, the behavior to grow all data files at the same rate is built into TempDB by default, which means you no longer need this trace flag.

Trace flag 1118

Administrators use trace flag 1118 to change page allocation from a GAM page. When you enable TF 1118, SQL Server allocates eight pages, or one extent, at a time to create a dedicated (or uniform) extent, in contrast to the default behavior to allocate a single page from a mixed extent. Unlike with TF 1117, there was no potential downside to enabling TF 1118—it is generally recommended for all SQL Server implementations in earlier releases. Starting with SQL Server 2016, all allocations of TempDB pages use uniform extent allocation, thus eliminating the need to use TF 1118.

Query Store

One of the most common scenarios you likely encounter is a user reporting that a query is suddenly running more slowly than in the past or that a long-running job that once took 3 hours is now taking 10. These performance degradations could be the result of changes in data causing out-of-date statistics or changes in execution parameters or be caused simply by reaching a tipping point in hardware capabilities. In previous versions of SQL Server, troubleshooting these issues requires you to gather data from the plan cache and parse it by using XML Query (xQuery), which can take considerable effort. Even then, you might not have all the information you need, unless you are actively running traces to baseline the user's environment.

The new Query Store feature in SQL Server 2016 simplifies identification of performance outliers, manages execution plan regression, and allows for easier upgrades between versions of SQL Server. It has two main goals—to simplify identification of performance issues and to simplify performance troubleshooting for queries caused by changes in execution plans. The query store also acts as a flight data recorder for the database, capturing query run-time statistics and providing a dashboard to sort queries by resource consumption. This vast collection of data serves not only as a resource for the automated functions of the query store, but also as a troubleshooting resource for the DBA.

This feature is one of the biggest enhancements to the SQL Server database engine since the introduction of dynamic management views (DMVs) into the database engine in SQL Server 2005. The query store gives unprecedented insight into the operations of a database. Whether you want to find the workloads in an instance, perform an in-depth analysis across executions of the same code, or fix a pesky parameter-sniffing problem, the query store offers a vast metastore of data, allowing you to quickly find performance issues.

Enabling Query Store

Query Store manages its metadata in the local database, but it is disabled by default. To enable it in SQL Server Management Studio (SSMS), open Object Explorer, connect to the database engine, navigate to the database for which you want to enable Query Store, right-click the database, select Properties, and then click Query Store in the Database Properties dialog box. You can change the Operation Mode (Requested) value from Off to Read Only or Read Write. By selecting Read Write, as shown in Figure 4-3, you enable Query Store to record the run-time information necessary to make better decisions about queries.

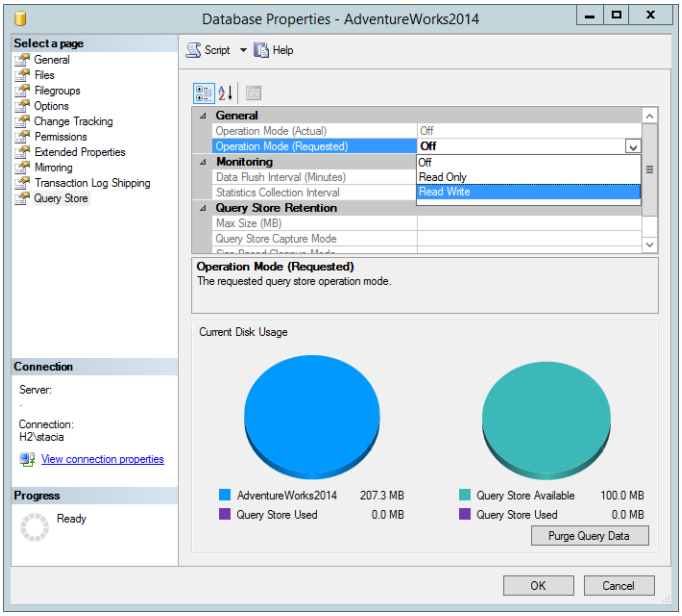


Figure 4-3: Enabling Query Store.

You can also use the T-SQL ALTER DATABASE command to enable Query Store, as shown in Example 4-1.

Example 4-1: Enabling Query Store

```
ALTER DATABASE AdventureWorks2014
SET QUERY_STORE = ON
    (
      OPERATION_MODE = READ_WRITE
    , CLEANUP_POLICY = ( STALE_QUERY_THRESHOLD_DAYS = 5 )
    , DATA_FLUSH_INTERVAL_SECONDS = 2000
    , MAX_STORAGE_SIZE_MB = 10
    , INTERVAL_LENGTH_MINUTES = 10
    );
```

Understanding Query Store components

The query store contains two stores: a plan store that persists the execution plans, and a run-time stats store that persists the statistics surrounding query execution, such as CPU, I/O, memory, and other metrics. SQL Server retains this data until the space allocated to Query Store is full. To reduce the impact on performance, SQL Server writes information to each of these stores asynchronously.

Note The default space allocation for Query Store is 100 MB.

You can use the following five catalog views, as shown in Figure 4-4, to return metadata and query execution history from the query store:

- **query_store_runtime_stats** Run-time execution statistics for queries.
- **query_store_runtime_stats_interval** Start and end times for the intervals over which run-time execution statistics are collected.

- **query_store_plan** Execution plan information for queries.
- **query_store_query** Query information and its overall aggregated run-time execution statistics.
- **query_store_query_text** Query text as entered by the user, including white space, hints, and comments.

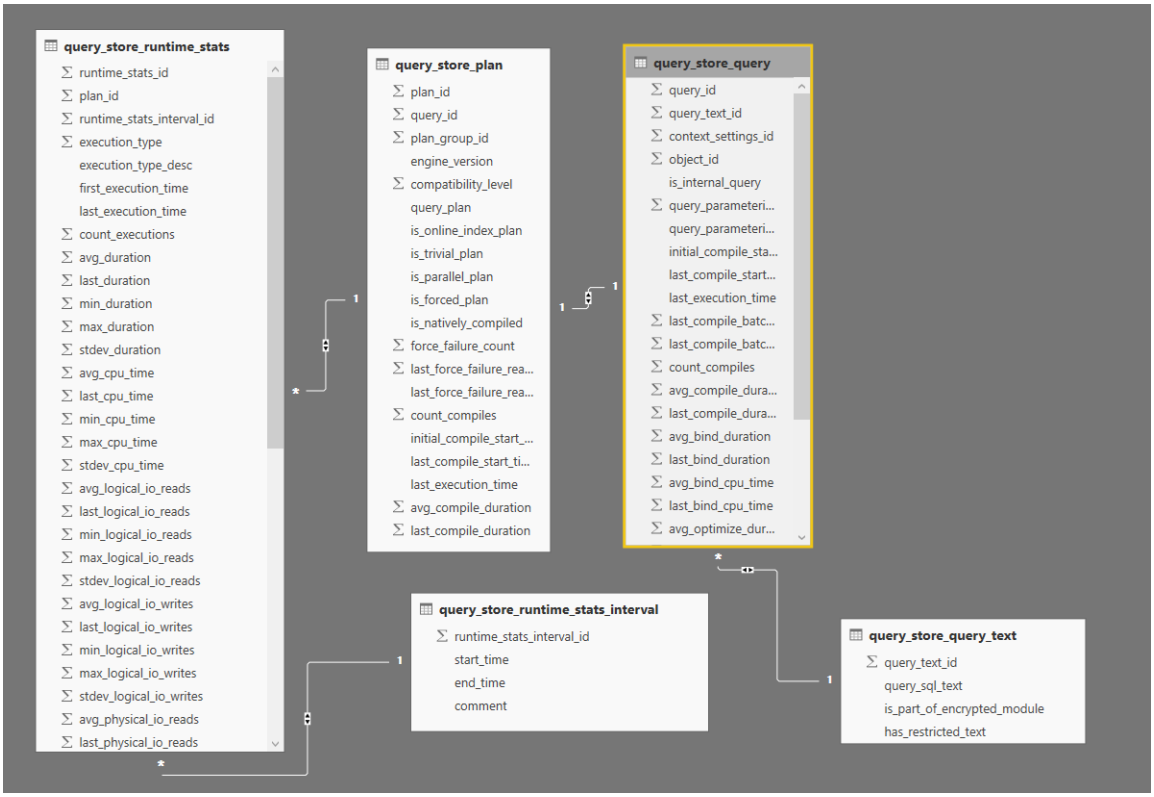


Figure 4-4: Query Store catalog views.

Reviewing information in the query store

The change in query execution plans over time can be a troubleshooting challenge unless you periodically mine the procedure cache to capture query plans. However, plans might be evicted from the cache as a server comes under memory pressure. If you use real-time querying, you have access only to the most recently cached plan. By using Query Store, as long as it is properly configured, you always have access to the information you need. One way to review this information is by using the dashboard views available in SSMS when you expand the Query Store folder for the database node, as shown in Figure 4-5. By taking advantage of this data, you can quickly isolate problems and be more productive in your tuning efforts.

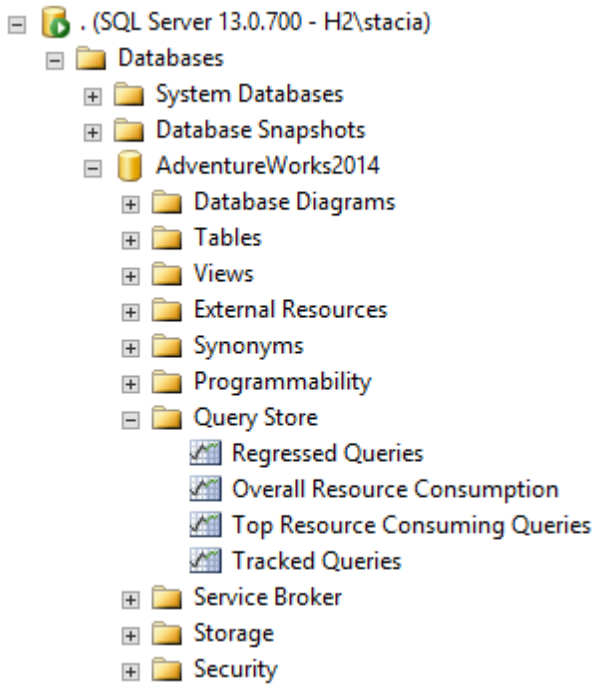


Figure 4-5: Query Store dashboards available in SSMS.

After enabling Query Store for a database, you have access to the following four dashboards:

- **Regressed Queries** Use this dashboard to review queries that might have regressed because of execution plan changes. The dashboard allows you to view the queries and their plans as well as to select queries based on statistics (total, average, minimum, maximum, and standard deviation) by query metric (duration, CPU time, memory consumption, logical reads, logical writes, and physical reads) for the top 25 regressed queries over the last hour.
- **Overall Resource Consumption** Use this dashboard to visualize overall resource consumption during the last month in four charts: duration, execution count, CPU time, and logical reads. You have the option to toggle between a chart view and a grid view of the query store data.
- **Top Resource Consuming Queries** Use this dashboard to review queries in the set of top 25 resource consumers during the last hour. You can filter the queries by using the same criteria available in the Regressed Queries dashboard.
- **Tracked Queries** Use this dashboard to monitor a specify query.

All the dashboards except Overall Resource Consumption allow you to view the execution plan for a query. In addition, you have the option to force an execution plan at the click of a button in the dashboard, which is one of the most powerful features of the query store. However, the plan must still exist in the query plan cache to use this feature.

You can customize Query Store dashboards to show more data or to use a different time interval. To do this, double-click a dashboard to open it, and then click the Configure button at the top of the dashboard to display and edit the configuration dialog box, as shown in Figure 4-6.

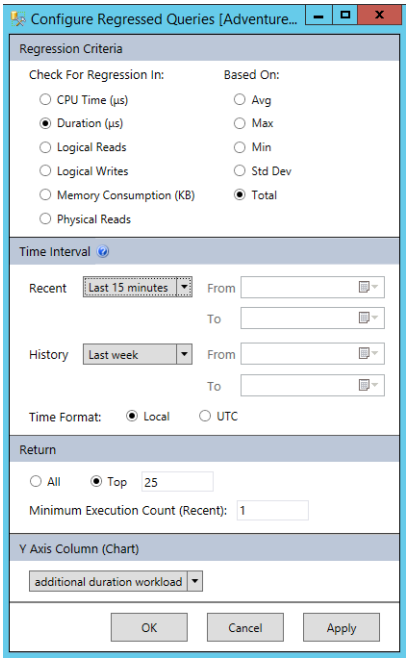


Figure 4-6: Configuring a Query Store dashboard.

Alternatively, you can query a DMV directly, which is a powerful approach for quickly isolating poorly performing queries. Example 4-2 shows a T-SQL statement to return the poorest performing queries over the last hour.

Example 4-2: Finding the poorest performing queries over the last hour

```
SELECT TOP 10 rs.avg_duration, qt.query_sql_text, q.query_id,
    qt.query_text_id, p.plan_id, GETUTCDATE() AS CurrentUTCTime,
    rs.last_execution_time
FROM sys.query_store_query_text AS qt
JOIN sys.query_store_query AS q
    ON qt.query_text_id = q.query_text_id
JOIN sys.query_store_plan AS p
    ON q.query_id = p.query_id
JOIN sys.query_store_runtime_stats AS rs
    ON p.plan_id = rs.plan_id
WHERE rs.last_execution_time > DATEADD(hour, -1, GETUTCDATE())
ORDER BY rs.avg_duration DESC;
```

Using Force Plan

The generation of an execution plan is CPU intensive. To reduce the workload on the database engine, SQL Server generates a plan once and stores it in a cache. Generally, caching the plan is good for database performance, but it can also lead to a situation known as *parameter sniffing*. The query optimizer uses parameter sniffing to minimize the number of recompiled queries. This situation occurs when a stored procedure is initially run with a given parameter against a table having a skewed number of values. You can use the query store's Force Plan option to address this problem.

To better understand parameter sniffing, consider an example in which you create a stored procedure like the one shown in Example 4-3.

Example 4-3: Understanding parameter sniffing

```
CREATE PROCEDURE sniff_demo
    @PARAMETER1 INT
AS
    UPDATE SNIFF_TABLE
    SET value=2
    WHERE ID=@PARAMETER1;
```

Now let's assume that you have a table such as the one shown here:

ID	Value
1	3
1	4
1	5
1	6
1	7
1	8
1	9
2	9

In this simple example of skewed values in a table, seven values have an ID of 1, and one value has an ID of 2. If you first run this procedure with a parameter value of 2, the execution plan generated by the database optimizer is likely to be less than optimal. Then, when you later execute the procedure with a parameter value of 1, SQL Server reuses the suboptimal plan.

Because skewed data might force your procedures into plans that are less than optimal for many queries, you have the opportunity to force the plan that is best optimized for all executions of a given stored procedure. While this approach might not offer the best performance for all values of a procedure's parameter, forcing a plan can give you more consistent overall performance and better performance on average. SQL Server honors plan forcing during recompilation for in-memory, natively compiled procedures, but the same is not true for disk-based modules.

You can also unforce a plan by using either the Query Store interface in SSMS or the *sp_query_store_unforce_plan* stored procedure. You might unforce a plan after your data changes significantly or when the underlying code changes enough to render the existing plan invalid.

Managing the query store

The query store is extremely helpful, but it does require some management. As we explained earlier in this chapter, the query store is not enabled by default. You must enable it on each user database individually. In addition, a best practice is to enable it on the model database.

Note At the time of this writing, Query Store is not currently included in the Database Properties dialog box in SSMS for the model database. To add it, you must enable Query Store by using the following code:

```
ALTER DATABASE MODEL SET QUERY_STORE=ON
```

After enabling the query store, you might need to change the space allocated to the query store from the default of 100 MB per database. If you have a busy database, this allocation might not be large

enough to manage execution plans and their related metadata. When this space fills up, the query store reverts to a read-only mode and no longer provides up-to-date execution statistics.

The size of your query store is also directly related to the statistics collection interval. The default for this value is 60 minutes, but you can adjust it to a higher frequency if you need more finely grained data. However, capturing data at a higher frequency requires more space for the query store.

Another setting to consider is size-based cleanup mode. By default, the query store converts to read-only mode when full. When you enable size-based cleanup, SQL Server flushes older queries and plans as new data comes in, thereby continually providing the latest data. Another option for space conservation is adjusting the capture mode of the query store from ALL to AUTO, which eliminates the capture of queries having insignificant compile and execution detail.

Tuning with the query store

After enabling the query store and collecting data over a baseline period, you now have a wealth of data and options to start troubleshooting performance issues. The query store allows you to spend more time troubleshooting problem queries and improving them, rather than on trying to find the proverbial needle in a haystack. A simple approach is to start troubleshooting queries on the basis of highest resource consumption. For example, you can look at queries consuming the most CPU and logical I/Os. After identifying poorly performing queries, you can then consider the following options:

- If multiple plans are associated with a query, identify the best-performing plan and use the Force Plan option to request it for future executions.
- If you observe a large gap between the estimated rows and the actual rows in a query, updating statistics might help performance.
- If query logic is problematic overall, work with your development team to optimize the query logic.

Stretch Database

One of the more common refrains in IT infrastructure organizations in recent years has been the high costs of storage. A combination of regulatory and business requirements for long-term data retention, as well as the presence of more data sources, means enterprises are managing ever-increasing volumes of data. While the price of storage has dropped, as anyone who owns enterprise storage knows, the total cost of ownership (TCO) for enterprise storage commonly used for databases is still very high. Redundant arrays of independent disks (RAID), support contracts, management software, geographical redundancy, and storage administrators all add to the high total cost of enterprise storage.

Another factor in the cost of storage is the lack of support for online data archiving in many third-party applications. To address this problem, a common approach is to use file groups and partitioning to move older data to slower disks. Although this approach can be effective, it also comes with high managerial overhead because it involves storage administrators in provisioning the storage and requires active management of partitions.

Perhaps more important than the TCO of enterprise storage is the impact of large databases and tables on overall administration and availability of the systems. As tables grow to millions and even billions of rows, index maintenance and performance tuning become significantly more complex. These large databases also affect availability service-level agreements as restore times can often exceed service-level agreements required by the business.

SQL Server 2016 introduces a new hybrid feature called Stretch Database that combines the power of Azure SQL Database with an on-premises SQL Server instance to provide nearly bottomless storage at

a significantly lower cost, plus enterprise-class security and near-zero management overhead. With Stretch Database, you can store cold, infrequently accessed data in Azure, usually with no changes to application code. All administration and security policies are still managed from the same local SQL Server database as before.

Understanding Stretch Database architecture

Enabling Stretch Database for a SQL Server 2016 table creates a new Stretch Database in Azure, an external data source in SQL Server, and a remote endpoint for the database, as shown in Figure 4-7. User logins query the stretch table in the local SQL Server database, and Stretch Database rewrites the query to run local and remote queries according to the locality of the data. Because only system processes can access the external data source and the remote endpoint, user queries cannot be issued directly against the remote database.

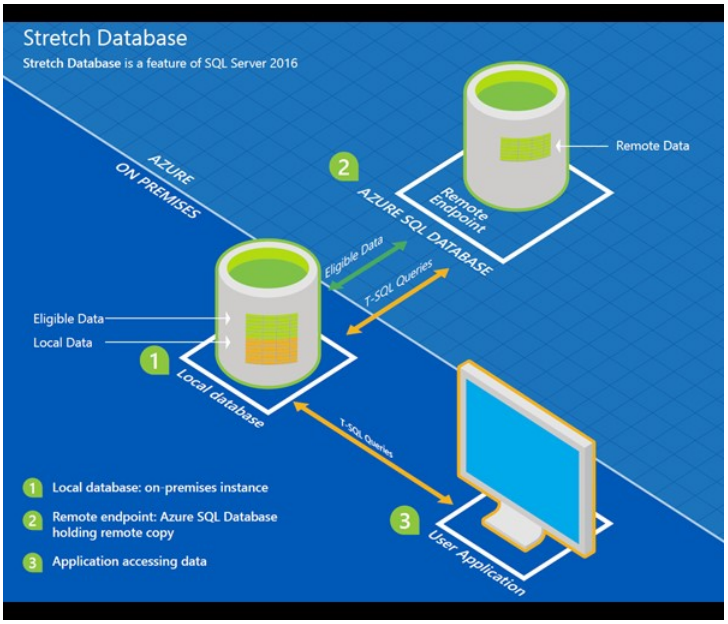


Figure 4-7: Stretch Database architecture.

Security and Stretch Database

One of the biggest concerns about cloud computing is the security of data leaving an organization's data center. In addition to the world-class physical security provided at Azure data centers, Stretch Database includes several additional security measures. If required, you have the option to enable Transparent Data Encryption to provide encryption at rest. All traffic into and out of the remote database is encrypted and certificate validation is mandatory. This ensures that data never leaves SQL Server in plain text and the target in Azure is always verified.

The external resource that references the Azure SQL Stretch Database can only be used by system processes and is not accessible by users. (See Figure 4-8.) Furthermore, it has no impact on the underlying security model of a stretch table.

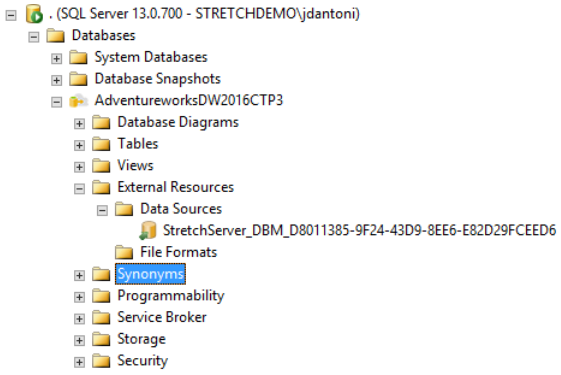


Figure 4-8: External resource for Stretch Database.

The security model in your on-premises database has a couple of components. The first requirement is to enable "remote data archive" for the instance. You will need to have either sysadmin or serveradmin permission. Once you have enabled this feature, you can configure databases for stretch, move data to your stretch database, and query data in your stretch database. To enable Stretch Database at the individual database level, you must have the CONTROL DATABASE permission. You will also need ALTER privileges on the tables you are looking to stretch.

As you would for a SQL Database that you provision manually, you must also create a firewall rule for the remote SQL Stretch Database database. That way, only safe IP addresses can connect to it. The creation of this firewall rule is part of the automation in the Stretch Database wizard if you enable your SQL Server database for stretch via SQL Server Management Studio.

Identifying tables for Stretch Database

Not all tables are ideal candidates for Stretch Database. In the current release, you cannot enable stretch for a table if it has any of the following characteristics:

- More than 1,023 columns
- Memory-optimized tables
- Replication
- Common language runtime (CLR) data types
- Check constraints
- Default constraints
- Computed columns

After eliminating tables with these characteristics from consideration, you have two options for identifying which of the remaining eligible tables in your environment are good candidates for stretching. First, you can use T-SQL to find large tables and then work with your application teams to determine the typical rate of change. A table with a high number of rows that are infrequently read is a good candidate. The other, more automated option is to use the Stretch Database Advisor, which is part of the SQL Server 2016 Upgrade Advisor. This advisor checks the current limitations for Stretch Database and then shows the best candidates for stretching based on benefits and costs, as shown in Figure 4-9.

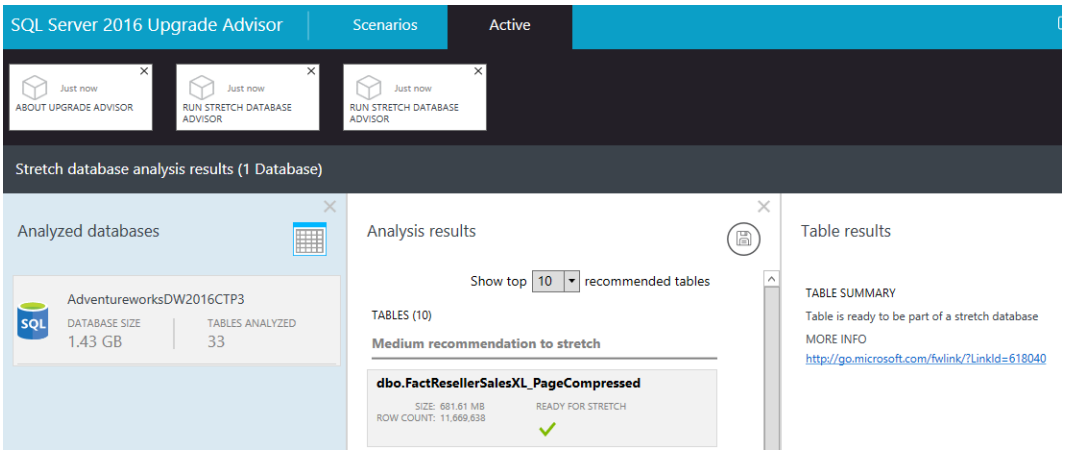


Figure 4-9: Analyzing candidates for Stretch Database in SQL Server 2016 Upgrade Advisor.

The best applications for Stretch Database are systems for which you are required to keep cold data for extended periods. By working with your application teams to understand which of your systems fit these scenarios, you can implement Stretch Database strategically to meet business requirements while reducing overall storage TCO and meeting business SLAs.

Configuring Stretch Database

Before you can configure Stretch Database in SQL Server, you must have an Azure account in place and change the REMOTE DATA ARCHIVE configuration option at the SQL Server instance level. To make this change, execute the command shown in Example 4-4.

Example 4-4: Changing the REMOTE DATA ARCHIVE configuration option

```
EXEC sp_configure 'remote data archive', '1';
GO
RECONFIGURE;
GO
```

You can then configure stretch, using the wizard that you launch by right-clicking the database in Object Explorer, pointing to Stretch, and clicking Enable. The wizard prompts you to supply a password for a database master key and select the table to stretch and then validates whether the table is eligible for stretch. Next, you sign in with your Azure credentials, select a subscription, and then select an Azure region. For performance reasons, choose the Azure region closest to your on-premises location.

Next, you have the option to create a new server or use an existing server. There is no impact on your existing SQL Databases if you choose to use an existing server. Your next step is to provide administrator credentials for the new SQL Database and to create a firewall rule allowing your on-premises databases to connect to SQL Database. When you click Finish on the last page of the wizard, the wizard provisions Stretch Database and begins migrating data to the new SQL Database.

Note As an alternative to using the wizard, you can perform the steps necessary to configure a database and a table for stretch by using T-SQL commands. For more information, see "Enable Stretch Database for a database" at https://msdn.microsoft.com/en-US/library/mt163698.aspx.

Monitoring Stretch Database

SQL Server 2016 includes a dashboard in SSMS to monitor Stretch Database. To view it, right-click the database name in Object Explorer, select Stretch Database, and then select Monitor to display the dashboard shown in Figure 4-10.

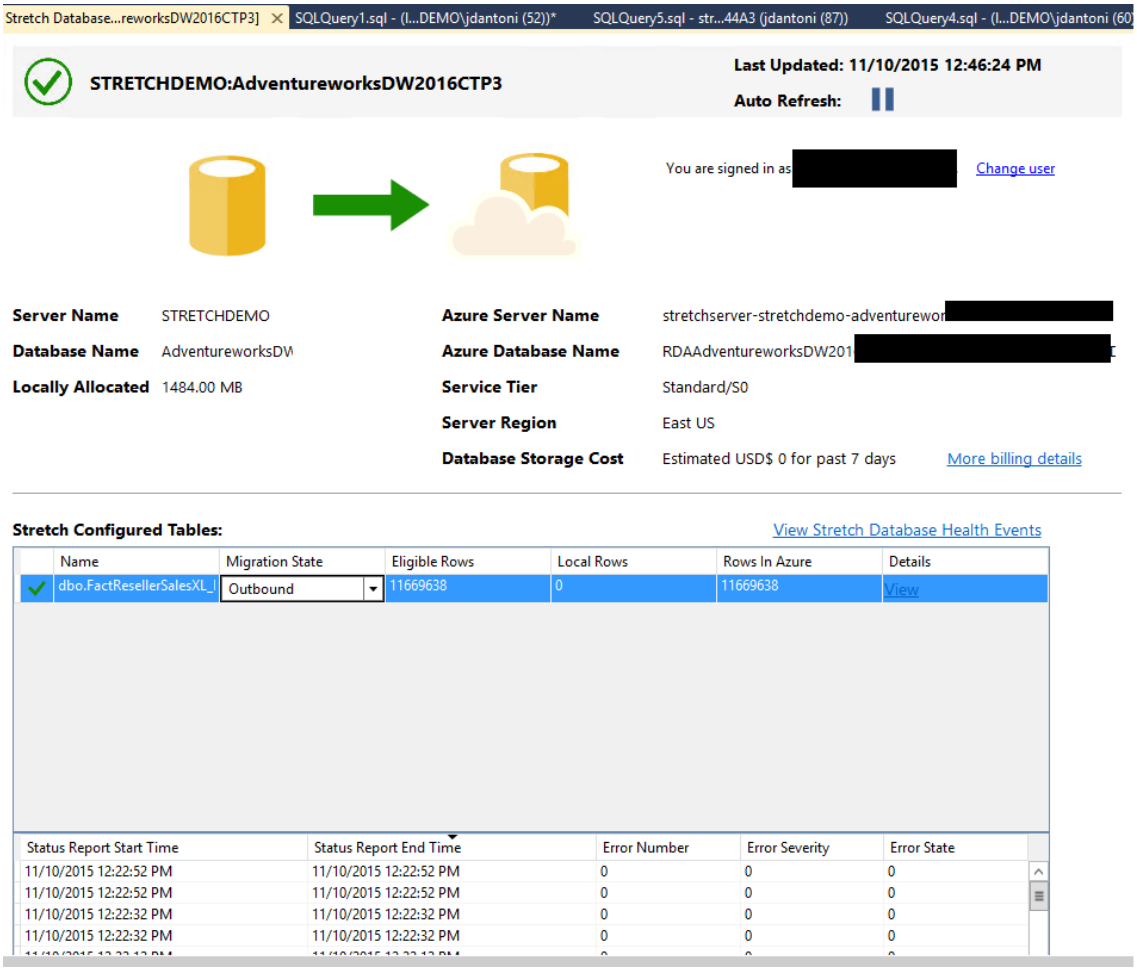


Figure 4-10: Monitoring Stretch Database in SSMS.

In this dashboard, you can see which tables are configured for stretch in addition to the number of rows eligible for stretch and the number of local rows. In Figure 4-10, all rows are stretched. You can also change the migration state of a table. The default state is Outbound, which means data is moving into Azure. However, you can pause the migration of the data.

Enabling Stretch Database also creates an Extended Events session called StretchDatabase_Health. You can view the extended events associated with this session by clicking the View Stretch Database Health Events link above the Stretch Configured Tables section of the dashboard. Additionally, you can explore two DMVs associated with Stretch Database: *sys.dm_db_rda_migration_status* and *sys.dm_db_rda_schema_update_status*.

Note Most common problems you encounter with Stretch Database are likely to be network or firewall related. As your first troubleshooting step, work with a network administrator to ensure that

you can reach your SQL Database over port 1433, which is a commonly blocked outbound port on many networks.

Another monitoring tool at your disposal is the new Remote Query operator in the execution plan for a stretch table, as shown in Figure 4-11. SQL Server 2016 also includes the Concatenation operator to merge the results of the on-premises data with the remote query results.

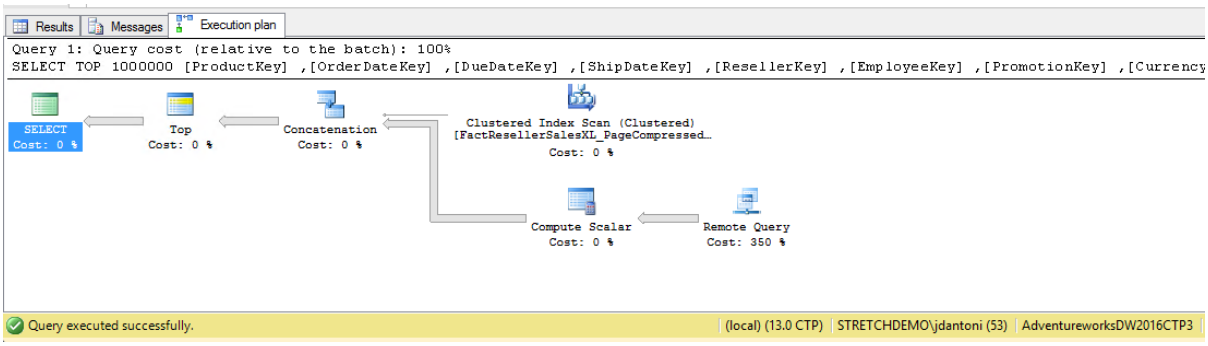


Figure 4-11: Reviewing the execution plan for a stretch table.

An important design pattern with Stretch Database is to ensure that your queries do not regularly retrieve unnecessary rows. Running poorly written queries against a stretch table can apply adverse performance. When troubleshooting performance issues on stretched tables, start your tuning effort as you would on a regular on-premises database. After eliminating issues related to your on-premises instance, examine the Azure portal to understand how the workload affects the stretch database.

If your remote query performance is still not sufficient, you have several options for tuning. First, ensure that your remote database is in the Azure data center nearest your on-premises data center to reduce latency. Next, monitor the Azure portal to observe the performance characteristics of the underlying Azure database. You might need to increase the service tier of the SQL Stretch Database. Last, work with your network administrator to guarantee quality of service between your site and your remote database.

Backup and recovery with Stretch Database

Backup and recovery of a stretch-enabled database does not include the SQL Stretch Database containing your remote tables. Nonetheless, your data remains protected because SQL Stretch Database leverages the built-in backup features of SQL Database. Accordingly, SQL Database is constantly making full and transaction log backups. The retention period for these backups is determined by the service tier of the database. However, when you back up your on-premises database, you are taking a shallow backup. In other words, your backup contains only the data that remains on-premises and does not include the migrated data.

To restore a database, follow these steps:

1. Restore your on-premises SQL Server database.
2. Create a master key for the stretch-enabled database.
3. Create a database-scoped credential for your SQL Database.
4. Run the restore procedure.

Now that you've read the book...

Tell us what you think!

Was it useful?
Did it teach you what you wanted to learn?
Was there room for improvement?

Let us know at http://aka.ms/tellpress

Your feedback goes directly to the staff at Microsoft Press, and we read every one of your responses. Thanks in advance!

CHAPTER 5

Broader data access

As the cost to store data continues to drop and the number of data formats commonly used by applications continues to change, you need the ability both to manage access to historical data relationally and to seamlessly integrate relational data with semistructured and unstructured data. SQL Server 2016 includes several new features that support this evolving environment by providing access to a broader variety of data. The introduction of temporal tables enables you to maintain historical data in the database, to transparently manage data changes, and to easily retrieve data values at a particular point in time. In addition, SQL Server allows you to import JavaScript Object Notation (JSON) data into relational storage, export relational data as JSON structures, and even to parse, aggregate, or filter JSON data. For scalable integration of relational data with semistructured data in Hadoop or Azure storage, you can take advantage of SQL Server PolyBase, which is no longer limited to the massively parallel computing environment that it was when introduced in SQL Server 2014.

Temporal data

A common challenge with data management is deciding how to handle changes to the data. At a minimum, you need an easy way to resolve an accidental change without resorting to a database restore. Sometimes you must be able to provide an audit trail to document how a row changed over time and who changed it. If you have a data warehouse, you might need to track historical changes

for slowly changing dimensions. Or you might need to perform a trend analysis to compare values for a category at different points in time or find the value of a business metric at a specific point in time.

To address these various needs for handling changes to data, SQL Server 2016 now supports temporal tables, which were introduced as a new standard in ANSI SQL 2011. In addition, Transact-SQL has been extended to support the creation of temporal tables and the querying of these tables relative to a specific point in time.

A temporal table allows you to find the state of data at any point in time. When you create a temporal table, the system actually creates two tables. One table is the current table (also known as the temporal table), and the other is the history table. The history table is created as a page-compressed table by default to reduce storage utilization. As data changes in the current table, the database engine stores a copy of the data as it was prior to the change in the history table.

The use of temporal tables has a few limitations. First, system versioning and the FileTable and FILESTREAM features are incompatible. Second, you cannot use CASCADE options when a temporal table is the referencing table in a foreign-key relationship. Last, you cannot use INSTEAD OF triggers on the current or history table, although you can use AFTER triggers on the current table.

Creating a rowstore temporal table

When you create a temporal table, you must include a primary key and *nonnullable period columns*, a pair of columns having a *datetime2* data type that you use as the start and end periods for which a row is valid. You should avoid using blob data types—*(n)varchar(max)*, *varbinary(max)*, *ntext*, and *image*—in a temporal table because of their size and the adverse performance impact that results. You must also specify one of the following conditions for the history table:

- **Anonymous** The system creates the history table with an automatically generated name in the same schema as the current table. This option is good for prototyping a solution because you do not need to concern yourself with naming conventions.
- **Default** The system creates the history table by using the schema and table names that you specify.
- **Existing** You specify an existing history table with a schema that conforms to the temporal table. This table cannot have a primary key defined.

Note When the system creates a history table for a partitioned current table, it creates the table on the default filegroup and does not configure partitioning for the history table.

Anonymous history table

To create a temporal table with an anonymous history table, you define the structure of your table, including a primary key and the period columns, as shown in Example 5-1. The period columns are always nonnullable even if you omit the NOT NULL constraint. Notice the PERIOD FOR SYSTEM_TIME argument that identifies the period columns. Last, the SYSTEM_VERSIONING = ON argument is necessary to enable the temporal feature.

Example 5-1: Creating a temporal table with an anonymous history table

```
CREATE TABLE [Production].[ProductInventory_Temporal_Auto](
   [ProductID] [int] NOT NULL,
   [LocationID] [smallint] NOT NULL,
   [Shelf] [nvarchar](10) NOT NULL,
   [Bin] [tinyint] NOT NULL,
   [Quantity] [smallint] NOT NULL,
```

```
   [ModifiedDate] [datetime] NOT NULL,
 CONSTRAINT [PK_ProductInventory_Temporal_Auto_ProductID_LocationID] PRIMARY KEY CLUSTERED
(
   [ProductID] ASC,
   [LocationID] ASC
), ValidFrom datetime2(7) GENERATED ALWAYS AS ROW START NOT NULL,
   ValidTo datetime2(7) GENERATED ALWAYS AS ROW END NOT NULL,
      PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON);
```

Note When you create a temporal table with an anonymous history table, you must have the CREATE TABLE permission in the database and ALTER PERMISSION on the schema for the new current and history tables.

When you execute the CREATE TABLE statement, the database engine creates the temporal table as you defined it. It also creates the history table as a rowstore table with page compression if possible and names it *MSSQL_TemporalHistoryFor_<current_temporal_table_object_id>_[suffix]*. The suffix appears in the name only if the first part of the table name is not unique. The database engine also adds a clustered index named *IX_<history table name>* that contains the period columns.

You can view the temporal table and its history table in Object Explorer in SQL Server Management Studio (SSMS), as shown in Figure 5-1. You can identify a temporal table by the *System-Versioned* label that is appended to the temporal table name. The *History* label is appended to the history table. The tables are identical, except that the history table does not include a primary key.

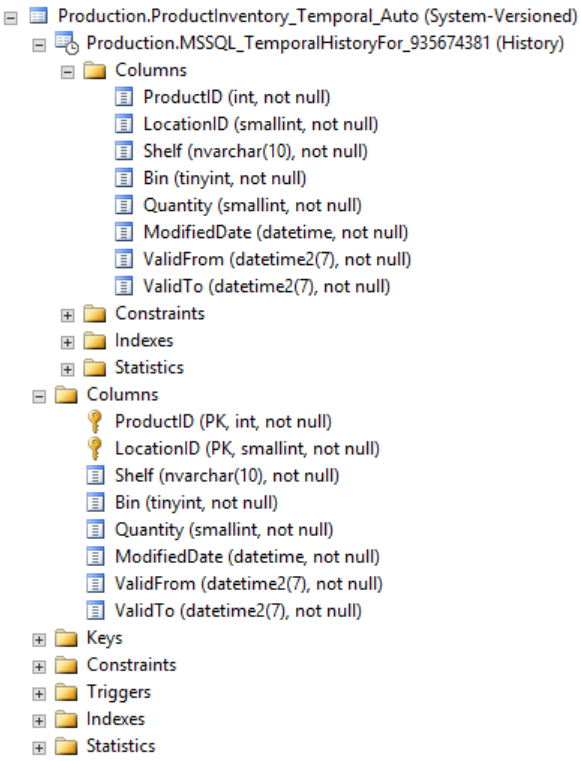


Figure 5-1: A temporal table and its history table displayed together in SSMS's Object Explorer.

Default history table

As an alternative, you can create a temporal table with a default history table by providing the schema and table name for the history table, as shown in Example 5-2. In this case, you must add the HISTORY_TABLE parameter to the WITH SYSTEM_VERSIONING = ON clause and include the schema explicitly with the history table's name. Optionally, you can use the HIDDEN clause when defining the period columns to exclude these columns by default in a `SELECT * FROM <table>` query. If the schema is missing or invalid or if the table already exists, the CREATE TABLE statement fails.

Example 5-2: Creating a temporal table with a default history table

```
CREATE TABLE [Production].[ProductInventory_Temporal](
   [ProductID] [int] NOT NULL,
   [LocationID] [smallint] NOT NULL,
   [Shelf] [nvarchar](10) NOT NULL,
   [Bin] [tinyint] NOT NULL,
   [Quantity] [smallint] NOT NULL,
   [ModifiedDate] [datetime] NOT NULL,
 CONSTRAINT [PK_ProductInventory_Temporal_ProductID_LocationID] PRIMARY KEY CLUSTERED
(
   [ProductID] ASC,
   [LocationID] ASC
), [ValidFrom] datetime2(7) GENERATED ALWAYS AS ROW START HIDDEN NOT NULL,
   [ValidTo] datetime2(7) GENERATED ALWAYS AS ROW END HIDDEN NOT NULL,
   PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = [Production].[ProductInventory_Temporal_History]));
```

Existing history table

When you prefer to manage the storage or indexes of the history table, you can create it yourself and then associate it with a new temporal table, as shown in Example 5-3. The history table must be consistent with the temporal table's schema by having the same column names, the same number of columns, and the same data types for each column. In addition, the columns must be in the same order in both tables. Furthermore, the history table is subject to a few limitations. It must be in the same database as the current table, and it cannot have a primary key, foreign key constraints, unique indexes, table or column constraints, or triggers. Also, you cannot use a history table to capture changes to data or for transactional or merge replication.

Note If the primary use case for a temporal table is auditing, create the history table with a rowstore table with a clustered index. If the temporal table must support analytical queries that aggregate data or apply windowing functions, add a clustered columnstore index.

Example 5-3: Creating a temporal table with an existing history table

```
-- Create history table
CREATE TABLE [Production].[ProductInventory_TemporalHistory](
   [ProductID] [int] NOT NULL,
   [LocationID] [smallint] NOT NULL,
   [Shelf] [nvarchar](10) NOT NULL,
   [Bin] [tinyint] NOT NULL,
   [Quantity] [smallint] NOT NULL,
   [ModifiedDate] [datetime] NOT NULL,
   [ValidFrom] datetime2(7) NOT NULL,
```

```
   [ValidTo] datetime2(7) NOT NULL
);
GO
-- Add indexes to history table
CREATE CLUSTERED COLUMNSTORE INDEX IX_ProductInventoryHistory
   ON [Production].[ProductInventory_TemporalHistory];
CREATE NONCLUSTERED INDEX IX_ProductInventoryHistory_ID_PERIOD_COLUMNS
   ON [Production].[ProductInventory_TemporalHistory] (ValidTo, ValidFrom, ProductID,
LocationID);
GO
-- Create temporal table
CREATE TABLE [Production].[ProductInventory_Temporal_WithExisting](
   [ProductID] [int] NOT NULL,
   [LocationID] [smallint] NOT NULL,
   [Shelf] [nvarchar](10) NOT NULL,
   [Bin] [tinyint] NOT NULL,
   [Quantity] [smallint] NOT NULL,
   [ModifiedDate] [datetime] NOT NULL,
 CONSTRAINT [PK_ProductInventory_Temporal_WE_ProductID_LocationID] PRIMARY KEY CLUSTERED
(
   [ProductID] ASC,
   [LocationID] ASC
), ValidFrom datetime2(7) GENERATED ALWAYS AS ROW START NOT NULL,
   ValidTo datetime2(7) GENERATED ALWAYS AS ROW END NOT NULL,
   PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = [Production].[ProductInventory_TemporalHistory]));
```

Maintaining consistency

By default, the database engine performs consistency checks when you create a temporal table or when you convert an existing table to temporal. These consistency checks include a check of the schema to ensure that the current and history tables have the correct structure and a check of the data to ensure that there are no overlapping records in these two tables and that temporal rules are enforced.

The schema check validates that the following conditions are true:

- The current and history tables have the same number of columns.
- The corresponding columns in each table have matching data types.
- The start- and end-period columns are nonnullable.
- A primary key constraint exists in the current table and does not exist in the history table.
- If the current table has hidden period columns, the history table must also hide the period columns.
- There is no identity column in the history table.
- There are no triggers in the history table.
- There are no foreign keys in the history table.
- There are no table or column constraints (aside from default column values) in the history table.
- The history table cannot be in a read-only filegroup.

- Change tracking or change data capture cannot be enabled for the history table.

Note If the history table is not empty, the data consistency check ensures that the end-period column value is greater than or equal to the start-period column value in each row.

Converting an existing table to temporal

If you have an existing table that you want to start using as a temporal table, you must add period columns and optionally add the HIDDEN flag on these columns, as shown in Example 5-4. That way, the change in the table structure does not affect existing applications. When you add the period columns, you must specify a default date and time for the start and end periods. For the start period, be sure to use a default date and time that is not in the future. The end period's default must be the maximum value for the *datetime2* data type. If your existing table already has period columns, include only the PERIOD FOR SYSTEM_TIME clause in the ALTER TABLE statement and reference the existing columns as parameters.

Important If you are not using Enterprise Edition, be aware that the addition of a nonnullable column to a table can be an expensive operation.

Example 5-4: Converting an existing table to temporal

```
-- Create history schema
CREATE SCHEMA History;
GO
-- Add period columns as hidden
ALTER TABLE [Production].[ProductInventory]
   ADD
      ValidFrom datetime2(0) GENERATED ALWAYS AS ROW START HIDDEN
           CONSTRAINT DF_ValidFrom DEFAULT CONVERT(datetime2 (0), '2008-03-31 00:00:00')
      , ValidTo datetime2(0) GENERATED ALWAYS AS ROW END HIDDEN
           CONSTRAINT DF_ValidTo DEFAULT CONVERT(datetime2 (0), '9999-12-31 23:59:59'),
      PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo);
GO
ALTER TABLE [Production].[ProductInventory]
   SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = [History].[ProductInventory_History]));
```

Understanding the effect of data changes

Let's consider a simple example in which insert, update, and delete operations change data in a temporal table. First, when you insert a row into a new temporal table, the database engine sets the start-period column's value to the transaction's start time in the Coordinated Universal Time (UTC) time zone and sets the end-period column's value to 9999-12-31, as shown in Figure 5-2. All rows inserted as part of the same transaction have the same start-period column value.

	ProductID	LocationID	Shelf	Bin	Quantity	ModifiedDate	ValidFrom	ValidTo
1	1	1	A	1	408	2016-02-29 15:43:56.080	2016-02-29 23:43:56.0815298	9999-12-31 23:59:59.9999999

Figure 5-2: A new row in a temporal table.

Note The start- and end-period columns are managed entirely by the database engine. It blocks any attempt to insert or update a value in these columns. Furthermore, you cannot truncate the

current table. If you need to clear data from the table, you must disable system versioning by using the ALTER TABLE statement with the WITH SYSTEM_VERSIONING = OFF argument.

When we update the row in the temporal table, the database engine performs the update, adjusts the start-period column's value, and inserts a new row into the history table, as shown in Figure 5-3, which displays the temporal table in the top result set and the history table in the bottom result set. The history table contains the row values as they were when the row was inserted into the temporal table, except that the end-period column value is set to the transaction time of the update operation. Note that this time matches the start-period column value for the corresponding row in the temporal table.

	ProductID	LocationID	Shelf	Bin	Quantity	ModifiedDate	ValidFrom	ValidTo
1	1	1	A	1	300	2016-02-29 16:10:29.203	2016-03-01 00:10:29.2055397	9999-12-31 23:59:59.9999999

	ProductID	LocationID	Shelf	Bin	Quantity	ModifiedDate	ValidFrom	ValidTo
1	1	1	A	1	408	2016-02-29 15:43:56.080	2016-02-29 23:43:56.0815298	2016-03-01 00:10:29.2055397

Figure 5-3: An update row in a temporal table and a corresponding row containing its original values in a history table.

Next, when you delete the row in the temporal table, the database engine removes it from the table and inserts a copy of it into the history table. The copy retains all the values from the temporal table except that the end-period time is set to the transaction time of the deletion, as shown in Figure 5-4.

	ProductID	LocationID	Shelf	Bin	Quantity	ModifiedDate	ValidFrom	ValidTo

	ProductID	LocationID	Shelf	Bin	Quantity	ModifiedDate	ValidFrom	ValidTo
1	1	1	A	1	408	2016-02-29 15:43:56.080	2016-02-29 23:43:56.0815298	2016-03-01 00:10:29.2055397
2	1	1	A	1	300	2016-02-29 16:10:29.203	2016-03-01 00:10:29.2055397	2016-03-01 00:32:22.8923131

Figure 5-4: An empty row in a temporal table and rows in a history table containing the original values of a row and its values at the time of its deletion in a history table.

Note When you perform a MERGE operation, the database engine modifies the temporal and history tables based on the specific INSERT, UPDATE, or DELETE operation affecting each row.

Using memory-optimized temporal tables

The velocity of change to your data determines how quickly history tables can grow. This aspect of the temporal feature can pose a challenge for memory-optimized tables because data can grow faster than the available memory. Furthermore, historical data is typically read less frequently than current data. For these reasons, having a memory-optimized history table is not recommended. However, you can create a durable memory-optimized temporal table and then store its history table on disk, as shown in Example 5-5. Consider adding a clustered columnstore index on the history table and a nonclustered index on the period columns (with the end-period column first) and the column or columns in the history table that correspond to the column (or columns) defined as the primary key in the current table.

Example 5-5: Creating a memory-optimized temporal table with a disk-based history table

```
-- Create history table
CREATE TABLE [Production].[ProductInventory_TemporalHistory_Disk](
   [ProductID] [int] NOT NULL,
   [LocationID] [smallint] NOT NULL,
```

```
   [Shelf] [nvarchar](10) NOT NULL,
   [Bin] [tinyint] NOT NULL,
   [Quantity] [smallint] NOT NULL,
   [ModifiedDate] [datetime] NOT NULL,
   [ValidFrom] datetime2(7) NOT NULL,
   [ValidTo] datetime2(7) NOT NULL
);
GO
-- Add indexes to history table
CREATE CLUSTERED COLUMNSTORE INDEX IX_ProductInventory_TemporalHistory_Disk
   ON [Production].[ProductInventory_TemporalHistory_Disk];
CREATE NONCLUSTERED INDEX IX_ProductInventory_TemporalHistory_Disk_ID_PERIOD_COLUMNS
   ON [Production].[ProductInventory_TemporalHistory_Disk] (ValidTo, ValidFrom, ProductID,
LocationID);
GO
-- Create temporal table
CREATE TABLE [Production].[ProductInventory_Temporal_InMemory](
   [ProductID] [int] NOT NULL,
   [LocationID] [smallint] NOT NULL,
   [Shelf] [nvarchar](10) NOT NULL,
   [Bin] [tinyint] NOT NULL,
   [Quantity] [smallint] NOT NULL,
   [ModifiedDate] [datetime] NOT NULL,
 CONSTRAINT [PK_ProductInventory_Temporal_IM_ProductID_LocationID] PRIMARY KEY NONCLUSTERED
(
   [ProductID] ASC,
   [LocationID] ASC
), ValidFrom datetime2(7) GENERATED ALWAYS AS ROW START HIDDEN NOT NULL,
   ValidTo datetime2(7) GENERATED ALWAYS AS ROW END HIDDEN NOT NULL,
   PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
)
WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA,
SYSTEM_VERSIONING = ON (HISTORY_TABLE = [Production].[ProductInventory_TemporalHistory_Disk]));
```

When you enable system versioning for a memory-optimized temporal table, the database engine creates an internal memory-optimized staging table to store changes and deletions as they occur in the current table. This table is not visible in Object Explorer in SSMS, but you can view metadata for this table in *sys.internal_tables*. The staging table name is *memory_optimized_history_table_<object id>*, where *<object id>* is the identifier of the current table.

Periodically, an asynchronous data-flush task transfers the data from the staging table to the history table. This occurs when the memory required by the staging table reaches 8 percent of the memory required by the current table. The goal is to limit the staging table's memory requirements to 10 percent of the current table's memory. In addition to the periodic flush, the contents of the staging table are moved to the history table when you add, drop, or alter columns in the current table or alter the table by setting SYSTEM_VERSIONING = OFF.

Note You can also force a flush of the staging table to the history table by using *sp_xtp_flush_temporal_history*. For more information on using this system stored procedure, see "sp_xtp_flush_temporal_history" at https://msdn.microsoft.com/en-us/library/mt591972.aspx.

If you perform massive deletions from the current table to reduce its footprint in memory, memory consumption is still high because the data persists in the staging table. Instead, delete rows in batches and invoke *sp_xtp_flush_temporal_history* to manage the impact on memory more effectively. Similarly, bulk updates can also take a toll on memory. Therefore, perform batch updates instead in combination with *sp_xtp_flush_temporal_history* for optimal results.

You can view the details of memory usage by querying the dynamic management view *sys.dm_db_xtp_memory_consumers*, as shown in Example 5-6. In this example, you identify the temporal table in the *sys.tables* table with a value of 1 in the *is_memory_optimized* column and a value

of 2 in the *temporal_type* column and locate its corresponding staging table in the *sys.internal_tables* table. Aggregate the values in the *allocated_bytes* and *used_bytes* columns of *sys.dm_db_xtp_memory_consumers* to view memory consumption for the specified memory-optimized table.

Example 5-6: Viewing memory usage for temporal and internal staging tables

```
WITH InMemoryTemporalTables
AS
(
   SELECT SCHEMA_NAME ( T1.schema_id ) AS TemporalTableSchema
     , T1.object_id AS TemporalTableObjectId
     , IT.object_id AS InternalTableObjectId
     , OBJECT_NAME ( IT.parent_object_id ) AS TemporalTableName
     , IT.Name AS InternalHistoryStagingName
   FROM sys.internal_tables IT
   JOIN sys.tables T1 ON IT.parent_object_id = T1.object_id
   WHERE T1.is_memory_optimized  = 1 AND T1.temporal_type = 2
)
, DetailedConsumption AS
(
   SELECT TemporalTableSchema
      , T.TemporalTableName
      , T.InternalHistoryStagingName
      , CASE
           WHEN C.object_id = T.TemporalTableObjectId
           THEN 'Temporal Table Consumption'
           ELSE 'Internal Table Consumption'
           END ConsumedBy
     , C.*
    FROM sys.dm_db_xtp_memory_consumers C
   JOIN InMemoryTemporalTables T
      ON C.object_id = T.TemporalTableObjectId OR C.object_id = T.InternalTableObjectId
)
SELECT TemporalTableSchema
   , TemporalTableName
   , sum ( allocated_bytes ) AS allocated_bytes
   , sum ( used_bytes ) AS used_bytes
FROM DetailedConsumption
WHERE TemporalTableSchema = 'Production'
    AND  TemporalTableName = 'ProductInventory_Temporal_InMemory'
GROUP BY TemporalTableSchema, TemporalTableName;
```

Querying temporal tables

A standard query against a temporal table returns rows from the current table only. When you need to retrieve historical data, you can add the FOR SYSTEM_TIME clause to your SELECT statement with one of the following subclauses to define the boundaries of the data to return, as shown in Example 5-7:

- **AS OF <date/time>** Returns the version of a row that was current at the specified date and time from either the current table or the history table. A row is considered current if the date/time parameter value in the subclause is equal to or later than the row's start period and less than its end period.
- **FROM <start date/time> TO <end date/time>** Returns any version of a row that was active in the specified range of dates from either the current table or the history table. A row is considered active if its end period is later than the start date/time parameter value and its start period is earlier than the end date/time parameter value.

- **BETWEEN <start date/time> AND <end date/time>** Returns any version of a row that was active in the specified range of dates from either the current table or the history table. It is similar to the FROM/TO subclause, except in this case a row is considered active if its start period is earlier than or equal to the end date/time parameter value. The end period must still be later than the start date/time parameter value.
- **CONTAINED IN (<start date/time>, <end date/time>)** Returns any version of a row that was active only during the specified range of dates from either the current table or the history table—that is, the start period is equal to or later than the start date/time parameter value and the end period is earlier than or equal to the end date/time parameter value.
- **ALL** Returns all row versions regardless of periods.

Example 5-7: Querying a temporal table

```
SELECT ProductID, LocationID, Shelf, Bin, Quantity, ModifiedDate, ValidFrom, ValidTo
FROM Production.ProductInventory_Temporal_Auto
FOR SYSTEM_TIME
BETWEEN '2016-02-29 00:00:00.0000000' AND '2016-03-01 00:00:00.0000000'
WHERE ProductID = 1 ORDER BY ValidFrom;
```

Note Because the period columns store UTC date/time values, be sure to adjust your local time to UTC time when providing a date/time parameter value in a temporal query.

Securing temporal tables

Temporal tables require layers of security in addition to those for standard tables. This security applies to CREATE TABLE, ALTER TABLE, and SELECT statements to control who can perform schema operations and who can access data.

To enable or disable system versioning on a temporal table, you must have the CREATE TABLE permission in the database and the ALTER permission on the schemas for the current and history tables. If you are using an existing history table when you convert a current table to temporal, you must have CONTROL permission on both the current table and the history table.

After you enable system versioning for a table, you cannot drop the table or use the ALTER TABLE statement to add or drop the period columns. However, you can use the ALTER TABLE statement for partitioning, but you can only switch a partition into the current table or out of the history table while system versioning is enabled. You can also use the ALTER TABLE statement to create an index or statistics or to rebuild the table.

If you grant a user SELECT permission on the current table only, the user can view data from that table but cannot access data from the history table by using a temporal query clause. If a user requires access to historical data, you must grant that user SELECT permission on both the current table and the history table.

Managing data retention

Data grows quickly as you store historical data, so you should consider how long you must keep the data available in the history table and how to move it out of that table when the retention period expires. To move data from the history table, you have the following options:

- Stretch Database
- Table partitioning
- Custom cleanup script

Stretch Database

If your primary requirement for historical data is to enable queries for recent data changes, but you occasionally need to provide access to older data, then you should consider transparently migrating historical data to SQL Database by using the Stretch Database feature that we introduce in Chapter 4, "Improved database engine." If you are using temporal tables strictly for auditing, you can stretch the entire history table and use the Stretch Wizard to set up Stretch Database. Otherwise, you can use an inline predicate function based on the value of the end-period column when you run a CREATE TABLE or ALTER TABLE statement to enable Stretch Database on your history table and migrate a subset of rows to SQL Database, as shown in Example 5-8.

Note The use of Stretch Database has no effect on how the database engine captures data changes or how it returns results for temporal queries from the history table. Although this is a good option for the history table, you cannot enable Stretch Database on a current table.

Example 5-8: Enabling Stretch Database for a history table

```
-- Create inline predicate function to filter historical data
CREATE FUNCTION dbo.fn_StretchBySystemEndTime(@systemEndTime datetime2)
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN SELECT 1 AS is_eligible
  WHERE @systemEndTime < CONVERT(datetime2, '2016-01-01T00:00:00', 101) ;
-- Create temporal table
ALTER TABLE [Production].[ProductInventory_TemporalHistory]
SET (
  REMOTE_DATA_ARCHIVE = ON
  (FILTER_PREDICATE = dbo.fn_StretchBySystemEndTime (ValidTo)
    , MIGRATION_STATE = OUTBOUND)
  );
```

Note If you prefer to use a sliding window and dynamically adjust the filter condition, create a scheduled script that uses the ALTER TABLE statement to disable Stretch Database temporarily, the ALTER FUNCTION statement to change the date used for filtering for the inline predicate function, and the ALTER TABLE statement to enable Stretch Database again. You can see an example of this script at "Manage Retention of Historical Data in System-Versioned Temporal Tables," https://msdn.microsoft.com/en-us/library/mt637341.aspx.

Table partitioning

If you are using Enterprise Edition, another option for managing the history table is to use table partitioning with a sliding window to easily move data out of the table when the data-retention period expires for a subset of data. You can keep system versioning on as you switch out the historical data and thereby avoid interruptions to normal workloads. Before you can use partition switching, the history table must have a clustered index on the end-period column.

Note You can also partition the current table and switch data into a partition with system versioning enabled. However, you cannot switch data out of the current table until you disable system versioning.

To start, you need to configure the initial partitioning configuration for the history table. Let's assume that you want historical data for up to three months in separate monthly partitions and that you enabled system versioning in January 2016. That means you need to create three partitions for the sliding window and one more empty partition to ensure a destination for new history rows added after the last month's partition range, as shown in Example 5-9.

Example 5-9: Creating partitions for a history table

```
-- Create partition function
  CREATE PARTITION FUNCTION [fn_Partition_ProductInventoryHistory_By_ValidTo] (datetime2(7))
    AS RANGE LEFT FOR VALUES
      (N'2016-01-31T23:59:59.999'
     , N'2016-02-29T23:59:59.999'
     , N'2016-03-31T23:59:59.999');
-- Create partition scheme
  CREATE PARTITION SCHEME [sch_Partition_ProductInventoryHistory_By_ValidTo]
    AS PARTITION [fn_Partition_ProductInventoryHistory_By_ValidTo]
    TO ([PRIMARY], [PRIMARY], [PRIMARY], [PRIMARY]);
-- Re-create index for alignment with partition schema
  CREATE CLUSTERED INDEX [ix_ProductInventoryHistory] ON [History].[ProductInventory_History]
    ([ValidFrom] ASC, [ValidTo] ASC)
    WITH
      (PAD_INDEX = OFF
      , STATISTICS_NORECOMPUTE = OFF
      , SORT_IN_TEMPDB = OFF
      , DROP_EXISTING = ON
      , ONLINE = OFF
      , ALLOW_ROW_LOCKS = ON
      , ALLOW_PAGE_LOCKS = ON
       , DATA_COMPRESSION = PAGE)
    ON [sch_Partition_ProductInventoryHistory_By_ValidTo] ([ValidTo]);
```

When the three partitions are full, you start the recurring partition maintenance task and schedule it to run once per month thereafter. Before you start, create a staging table with the same structure as the history table and add a clustered index on the period column dates and a constraint on the end-period column for the last day of the partition, as shown in Example 5-10.

Example 5-10: Creating a staging table for partition switching

```
-- Create staging table
CREATE TABLE [History].[Staging_ProductInventory_History] (
   [ProductID] [int] NOT NULL,
   [LocationID] [smallint] NOT NULL,
   [Shelf] [nvarchar](10) NOT NULL,
   [Bin] [tinyint] NOT NULL,
   [Quantity] [smallint] NOT NULL,
   [ModifiedDate] [datetime] NOT NULL,
   [ValidFrom] datetime2(7) NOT NULL,
   [ValidTo] datetime2(7) NOT NULL)
-- Create clustered index on same filegroup as partition to switch
CREATE CLUSTERED INDEX [IX_Staging_ProductInventory_History]
  ON [History].[Staging_ProductInventory_History]
  ( [ValidTo] ASC, [ValidFrom] ASC )
```

```
  WITH (
    PAD_INDEX = OFF
    , SORT_IN_TEMPDB = OFF
    , DROP_EXISTING = OFF
    , ONLINE = OFF
    , ALLOW_ROW_LOCKS = ON
    , ALLOW_PAGE_LOCKS = ON)
  ON PRIMARY;
-- Create constraints to match the partition to switch
ALTER TABLE [History].[Staging_ProductInventory_History] WITH CHECK
  ADD CONSTRAINT [CHK_Staging_ProductInventory_History_Partition_1]
    CHECK ( [ValidTo] <= N'2016-01-31T23:59:59.999');
ALTER TABLE [History].[Staging_ProductInventory_History]
  CHECK CONSTRAINT [CHK_Staging_ProductInventory_History_Partition_1];
```

Then create a dynamic script for the recurring partition maintenance task, which consists of the following steps:

1. Truncate the staging table, and then switch a partition between the history table and the staging table, like this:

   ```
   ALTER TABLE [History].[ProductInventory_History] SWITCH PARTITION 1 TO
   [History].[Staging_ProductInventory_History];
   ```

2. Merge partition 1 (now empty) with partition 2, like this:

   ```
   ALTER PARTITION FUNCTION [fn_Partition_ProductInventoryHistory_By_ValidTo]()
   MERGE RANGE (N'2016-02-29T23:59:59.999');
   ```

3. Create a new empty partition, like this:

   ```
   ALTER PARTITION SCHEME [sch_Partition_ProductInventoryHistory_By_ValidTo] NEXT
   USED;

   ALTER PARTITION FUNCTION [fn_Partition_ProductInventoryHistory_By_ValidTo]()
   SPLIT RANGE (N'2016-04-30T23:59:59.999');
   ```

4. Optionally archive the data in the staging table.

Custom cleanup script

A third option is to use a custom cleanup script to delete data from the history table. This approach requires you to first disable system versioning on the current table by using the SYSTEM_VERSIONING = OFF argument in the ALTER TABLE statement. To reduce the impact of running the cleanup script, you should run it at a time when workloads are light and modify the script shown in Example 5-11 as needed to keep the number of rows to delete in a single transaction below 10,000.

Example 5-11: Creating a cleanup script for a history table

```
CREATE PROCEDURE sp_CleanupHistoryData
  @temporalTableSchema sysname,
  @temporalTableName sysname,
  @cleanupOlderThanDate datetime2
AS
  DECLARE @disableVersioningScript nvarchar(max) = '';
  DECLARE @deleteHistoryDataScript nvarchar(max) = ''
  DECLARE @enableVersioningScript nvarchar(max) = ''
  DECLARE @historyTableName sysname
  DECLARE @historyTableSchema sysname
```

```
  DECLARE @periodColumnName sysname
-- Discover history table and end of period column for that table
EXECUTE sp_executesql
  N'SELECT @hst_tbl_nm = t2.name, @hst_sch_nm = s.name, @period_col_nm = c.name
    FROM sys.tables t1
    JOIN sys.tables t2 on t1.history_table_id = t2.object_id
    JOIN sys.schemas s on t2.schema_id = s.schema_id
    JOIN sys.periods p on p.object_id = t1.object_id
    JOIN sys.columns c on p.end_column_id = c.column_id and c.object_id = t1.object_id
    WHERE
      t1.name = @tblName and s.name = @schName',
      N'@tblName sysname,
      @schName sysname,
      @hst_tbl_nm sysname OUTPUT,
      @hst_sch_nm sysname OUTPUT,
      @period_col_nm sysname OUTPUT',
      @tblName = @temporalTableName,
      @schName = @temporalTableSchema,
      @hst_tbl_nm = @historyTableName OUTPUT,
      @hst_sch_nm = @historyTableSchema OUTPUT,
      @period_col_nm = @periodColumnName OUTPUT
IF @historyTableName IS NULL OR @historyTableSchema IS NULL OR @periodColumnName IS NULL
    THROW 50010, 'History table cannot be found. Either specified table is not system-versioned
    temporal or you have provided incorrect argument values.', 1

SET @disableVersioningScript =  @disableVersioningScript + 'ALTER TABLE [' +
    @temporalTableSchema + '].[' + @temporalTableName + '] SET (SYSTEM_VERSIONING = OFF)'
SET @deleteHistoryDataScript =  @deleteHistoryDataScript + ' DELETE FROM  [' +
    @historyTableSchema + '].[' + @historyTableName + '] WHERE [' +
    @periodColumnName + '] < ' + '''' + convert(varchar(128), @cleanupOlderThanDate, 126) +  ''''
SET @enableVersioningScript =  @enableVersioningScript + ' ALTER TABLE [' +
    @temporalTableSchema + '].[' + @temporalTableName + ']
SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = [' + @historyTableSchema + '].[' +
    @historyTableName + '], DATA_CONSISTENCY_CHECK = OFF )); '
BEGIN TRAN
     EXEC (@disableVersioningScript);
     EXEC (@deleteHistoryDataScript);
     EXEC (@enableVersioningScript);
COMMIT;
```

Reviewing temporal metadata

Several metadata views and functions are available for accessing information about temporal tables. This information is useful for creating scripts to monitor the current state of your environment or to perform maintenance tasks.

You can find metadata about tables, columns, and period columns in three system views, as shown in the following table:

Metadata view	Column	Description
sys.tables	temporal_type	Identifies the type of table: 0 = not temporal 1 = history table 2 = current table
	temporal_type_desc	Provides a text description for the temporal type
	history_table_id	Provides the object_id for the history table associated with a current table when applicable
sys.columns	generated_always_type	Identifies the column type for period columns: 0 = not applicable

		1 = row start 2 = row end
	generated_always_type_desc	Provides a text description for the Generated Always type
sys.periods	period_type	Name of the period
	period_type_desc	Identifies the type of period: 1 = system-time period
	object_id	Identifies the table containing the period_type column
	start_column_id	Identifies the column containing the start date/time
	end_column_id	Identifies the column containing the end date/time

You can also identify temporal tables or period columns by using the system functions shown in the following table:

Metadata function	Property	Description
OBJECTPROPERTY	TableTemporalType	Identifies the type of table: 0 = not temporal 1 = history table 2 = current table
OBJECTPROPERTYEX	TableTemporalType	Identifies the type of table: 0 = not temporal 1 = history table 2 = current table
COLUMNPROPERTY	GeneratedAlwaysType	Identifies the column type for period columns: 0 = not applicable 1 = row start 2 = row end

JSON

JSON is data in text format. JSON is popular because its language independence and simple data structure make it easy to use in modern web and mobile applications. But despite its popularity, the simplicity of JSON means it is not suitable for many business applications that require traditional online transactional processing database design. Instead, organizations are increasingly using NoSQL databases such as Azure DocumentDB for applications characterized by read-heavy workloads and simplified, denormalized data schemas that are not subject to frequent updates.

With the addition of JSON support to SQL Server 2016, you can now support both relational and nonrelational data structures in your environment and choose the appropriate model for your data requirements and workloads. Using new T-SQL functions and operators, you can easily export relational data to JSON or parse and import JSON into relational storage, even if you are working with in-memory OLTP, columnstore, or temporal relational tables. You can also write queries that combine results of relational and JSON data. Working with JSON in SQL Server is much like working with XML, except JSON is not a data type.

Note You can find a variety of use cases for JSON in SQL Server at the SQL Server Database Engine blog by searching for articles tagged as JSON, http://blogs.msdn.com/b/sqlserverstorageengine/archive/tags/json/.

Getting acquainted with JSON structures

JSON consists of two basic structures: an object containing one or more properties (also known as name/value pairs), and an ordered array of values. An object is enclosed in braces and includes a name followed by a colon and the value associated with the name. The name must be a string enclosed in double quotation marks, and the value can be a string in double quotation marks, a number, a Boolean value (true or false), null, another object, or an array. Multiple name/value pairs in an object are separated by commas, like this:

```
{"ProductID":709,"Name":"Mountain Bike Socks, M"}
```

An array is enclosed in brackets and can contain comma-separated values represented as a string within double quotation marks, a number, a Boolean value, null, or another object, like this:

```
[{"ProductID":709,"Name":"Mountain Bike Socks, M"}, {"ProductID":937,"Name":"HL Mountain Pedal"}]
```

Note The official language specification for JSON is available in "The JSON Data Interchange Format," at http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

Exporting data to JSON

Let's say you have a web application that requests data from your SQL Server database, but it requires the data to be formatted as JSON. Rather than include code in the application to perform the necessary transformations, you can easily export data from SQL Server tables to JSON format by adding the new FOR JSON clause to your SELECT statement.

As part of the export process, the database engine converts SQL data types to the appropriate JSON data types, as shown in the following table:

Category	SQL data type	JSON data type
Character	char nchar varchar nvarchar	string
Numeric	int bigint float decimal numeric	number
Bit	bit	Boolean
Date and time	date datetime datetime2 time datetimeoffset	string
Binary	binary	BASE64-encoded string
CLR	CLR geometry	Not supported (returns error)

	geography	
Other	uniqueidentifier money	string

In addition, the database engine escapes special characters as shown in the following table:

Special character	Escape sequence
Double quotation marks (")	\"
Backslash (\)	\\
Forward slash (/)	\/
Backspace	\b
Form feed	\f
New line	\n
Carriage return	\r
Horizontal tab	\t

When you use the AUTO argument with the FOR JSON clause, as shown in Example 5-12, the database engine automatically converts data types and formats and escapes the results of a SELECT statement according to JSON syntax as an array of objects. Each row in the result set is an object in the array, and each row contains name/value pairs consisting of the column name and the cell value for that row.

Example 5-12: Creating JSON output from a single table with FOR JSON AUTO

```
SELECT TOP 3 p.ProductID, p.Name
FROM Production.Product p
FOR JSON AUTO;
/* Result
[{"ProductID":1,"Name":"Adjustable Race"},
{"ProductID":879,"Name":"All-Purpose Bike Stand"},
{"ProductID":712,"Name":"AWC Logo Cap"}]
*/
```

When you combine two tables in the SELECT statement and include the FOR JSON AUTO clause, the structure of the JSON output changes, as shown in Example 5-13. With FOR JSON AUTO, you control the formatting by controlling the order of the columns. The columns from the table listed first in the column list are a first-level object in the JSON output array. Columns for the second table are grouped together as a second-level object nested inside the first-level object. This second-level object's name is the table alias, and its value is an array of the name/column pairs for the second table's columns.

Example 5-13: Creating JSON output from two tables with FOR JSON AUTO

```
SELECT TOP 2 p.ProductID, p.Name, pr.ProductReviewID, pr.ReviewerName, pr.EmailAddress,
       pr.Rating, pr.ReviewDate
FROM Product.ProductReview pr
INNER JOIN Production.Product p ON pr.ProductID = p.ProductID
FOR JSON AUTO;
/* Result
[{"ProductID":709,"Name":"Mountain Bike Socks, M",
    "pr":[
```

```
            "EmailAddress":"john@fourthcoffee.com","Rating":5,
            "ReviewDate":"2013-09-18T00:00:00"}
         ]
 },
 {"ProductID":937,"Name":"HL Mountain Pedal",
    "pr":[
           {"ProductReviewID":2,"ReviewerName":"David",
           "EmailAddress":"david@graphicdesigninstitute.com","Rating":4,
           "ReviewDate":"2013-11-13T00:00:00"}
         ]
 }
]

*/
```

If you prefer to have control over the formatting and naming of the objects at each level, you can use the FOR JSON PATH clause, as shown in Example 5-14. In this case, you use dot syntax in the column alias to define the names for the nested objects. When you use dot syntax, you provide the object, a dot, and the column alias enclosed in brackets, single quotation marks, or double quotation marks. Optionally, you can include the ROOT option to add a root element to the output.

Note You can also use the ROOT option when you use the FOR JSON AUTO clause.

Example 5-14: Creating JSON output from two tables with FOR JSON PATH

```
SELECT TOP 2 p.ProductID AS [Product.ID], p.Name AS [Product.Name],
             pr.ProductReviewID AS [Review.ID], pr.ReviewerName AS [Review.ReviewerName],
             pr.EmailAddress AS [Review.Email], pr.Rating AS [Review.Rating],
             pr.ReviewDate AS [Review.Date]
FROM Product.ProductReview pr
INNER JOIN Production.Product p ON pr.ProductID = p.ProductID
FOR JSON PATH, ROOT('Product Reviews');
/* Result
{"Product Reviews":
  [

   {"Product":{"ID":709,"Name":"Mountain Bike Socks, M"},
    "Review":{"ID":1,"ReviewerName":"John Smith",
              "Email":"john@fourthcoffee.com","Rating":5,
              "Date":"2013-09-18T00:00:00"}
   },
   {"Product":{"ID":937,"Name":"HL Mountain Pedal"},
    "Review":{"ID":2,"ReviewerName":"David",
              "Email":"david@graphicdesigninstitute.com","Rating":4,
              "Date":"2013-11-13T00:00:00"}
   }
  ]
}*/
```

Whether using FOR JSON AUTO or FOR JSON PATH, you can also include the following options to modify the formatting of the JSON text:

- **INCLUDE_NULL_VALUES** Includes cells with NULL values in the results. By default, these cells are excluded from the output.
- **WITHOUT_ARRAY_WRAPPER** Removes the outermost brackets for the JSON output.

Importing JSON data

In some cases, you might choose to use a hybrid approach by storing JSON data in a relational table. This approach is useful when the data that you need to store has a simple structure and does not change frequently or has properties that are needed only for a few rows in a table. With this approach, you can choose to store some data in standard relational structures and other data as JSON text and thereby combine the structures to best meet your performance and application requirements.

One option is to import data from an existing relational source and convert it to JSON before loading it into a table, as shown in Example 5-15. The target column for the JSON data must have a *varchar* or *nvarchar* data type unless you prefer to compress the data first. In that case, you use a column with a *varbinary* data type instead. Optionally, the column can have a constraint using the ISJSON function that we describe later in this chapter.

Example 5-15: Importing data as JSON text into a table

```
-- Create a varchar/nvarchar column in the target table
ALTER TABLE Production.Product
 ADD Reviews NVARCHAR(MAX)
 CONSTRAINT [JSON format for Reviews] CHECK(ISJSON(Reviews)>0);
-- Update JSON column with data from another SQL table
UPDATE Production.Product
 SET Reviews =
   (SELECT
      ReviewerName as [Reviewer.Name], EmailAddress as [Reviewer.Email], Rating, ReviewDate
    FROM Production.ProductReview
    WHERE Production.ProductReview.ProductID = Production.Product.ProductID
    FOR JSON PATH, WITHOUT_ARRAY_WRAPPER);
```

An alternative is to insert JSON data as a constant value into a relational table, as shown in Example 5-16. In this case, you format the string as the value for a single column with a *varchar* or *nvarchar* data type.

Example 5-16: Importing JSON data as a constant into a table

```
CREATE TABLE Production.ProductAlternate (
  Name NVARCHAR(50) NOT NULL,
  Reviews NVARCHAR(MAX) NULL
)
INSERT INTO Production.ProductAlternate(Name, Reviews)
  VALUES('Cycling Socks, M',
  '[
    {"Reviewer":{"Name":"John Smith","E-mail":"john@fourthcoffee.com"},
      "Rating":3,"ReviewDate":"2016-01-20T00:00:00"},
    {"Reviewer":{"Name":"Jill","E-mail":"jill@margiestravel.com"},
      "Rating":4,"ReviewDate":"2016-02-10T00:00:00"
 ]')
```

Converting JSON data to a table structure

When you need to convert JSON to a relational format, you can use the new OPENJSON function. With this technique, you do not need to preprocess JSON data in the application layer before you import it into a table or use it in a T-SQL query.

When you use the OPENJSON function, you decide whether to call it with an explicit schema or a default schema. With an explicit schema, you define the structure of the table that you want OPENJSON to return. With a default schema, the function returns a key column containing property names, a value column containing the values for each property, and type column containing a value for the JSON data type, as shown in the following table:

Type column value	JSON data type
0	null
1	string
2	int
3	true/false
4	array
5	object

When you call the OPENJSON function with a default schema, as shown in Example 5-17, you pass in the JSON text (or a variable to which you assign JSON text) as an argument and omit the WITH clause. The result of this function is a table containing one row for each property of the object. This table contains three columns—Key, Value, and Type—as shown in Figure 5-5.

Example 5-17: Using the OPENJSON function to convert a single JSON object to a table

```
SELECT * FROM OPENJSON('{"Reviewer":{"Name":"John Smith",
"Email":"john@fourthcoffee.com"},"Rating":5,"ReviewDate":"2013-09-18T00:00:00"}')
```

	key	value	type
1	Reviewer	{"Name":"John Smith","Email":"john@fourthcoffee.com"}	5
2	Rating	5	2
3	ReviewDate	2013-09-18T00:00:00	1

Figure 5-5: Rows returned by the OPENJSON function with a single JSON object as its argument.

Note Your database compatibility level must be set to 130 to use the OPENJSON function. All other JSON functions work with lower compatibility levels.

You can also pass an array of JSON objects to the OPENJSON function, as shown in Example 5-18. When you do this, the resulting table contains one row for each element of the array, as shown in Figure 5-6.

Example 5-18: Using the OPENJSON function to convert a JSON object array to a table

```
SELECT * FROM OPENJSON(
 '[
 {"Reviewer":{"Name":"John Smith","E-mail":"john@fourthcoffee.com"},
  "Rating":3,"ReviewDate":"2016-01-20T00:00:00"},
 {"Reviewer":{"Name":"Jill","E-mail":"jill@margiestravel.com"},
  "Rating":4,"ReviewDate":"2016-02-10T00:00:00"}
 ]')
```

	key	value	type
1	0	{"Reviewer":{"Name":"John Smith","E-mail":"john@fourthcoffee.com"},"Rating":3,"ReviewDate":"2016-01-20T00:00:00"}	5
2	1	{"Reviewer":{"Name":"Jill","E-mail":"jill@margiestravel.com"},"Rating":4,"ReviewDate":"2016-02-10T00:00:00"}	5

Figure 5-6: Rows returned by the OPENJSON function with a JSON object array as its argument.

By including a WITH clause after the OPENJSON function, you define an explicit schema in which you define the output columns, the data types, and the path of source properties for each column, as shown in Example 5-19. You can add a base path as a second argument to the OPENJSON function. In this example, $."Product Reviews" represents the root path. Because this path name includes an embedded space, it must be enclosed in double quotation marks. Double quotation marks are also required if the name starts with a $ symbol. In the WITH clause, the source property path includes a $ symbol to represent the starting point from the root defined as an argument in the OPENJSON function, followed by the remaining property names in the path when you have nested objects. A dot separates each property name. Although this example returns every property in the JSON data as a column, as shown in Figure 5-7, you can be selective about the columns to return and specify only a subset of properties in the WITH clause.

Example 5-19: Using the OPENJSON function to convert a JSON object array to a table

```
DECLARE @json VARCHAR(MAX);
SET @json=
'{"Product Reviews":
  [
    {"Product":{"ID":709,"Name":"Mountain Bike Socks, M"},
       "Review":{"ID":1,"ReviewerName":"John Smith",
                 "Email":"john@fourthcoffee.com","Rating":5,
                 "Date":"2013-09-18T00:00:00"}
     },
     {"Product":{"ID":937,"Name":"HL Mountain Pedal"},
     "Review":{"ID":2,"ReviewerName":"David",
               "Email":"david@graphicdesigninstitute.com","Rating":4,
               "Date":"2013-11-13T00:00:00"}
     }
  ]
 }';
SELECT * FROM OPENJSON(@json, '$."Product Reviews"')
WITH (
      ProductID int '$.Product.ID',
      ProductName varchar(100) '$.Product.Name',
      ReviewID int '$.Review.ID',
      ReviewerName varchar(100) '$.Review.ReviewerName',
      ReviewerEmail varchar(100) '$.Review.Email',
      Rating int '$.Review.Rating',
      ReviewDate datetime '$.Review.Date'
);
```

	ProductID	ProductName	ReviewID	ReviewerName	ReviewerEmail	Rating	ReviewDate
1	709	Mountain Bike Socks, M	1	John Smith	john@fourthcoffee.com	5	2013-09-18 00:00:00.000
2	937	HL Mountain Pedal	2	David	david@graphicdesigninstitute.com	4	2013-11-13 00:00:00.000

Figure 5-7: Rows returned by the OPENJSON function with an explicit schema.

Note If the JSON data has two paths with the same name, all built-in JSON functions return the value for the first path.

When you store JSON data in a relational table, you use the CROSS APPLY operator with the OPENJSON function to join the table rows with the corresponding JSON data, as shown in Example 5-

20. The result of this operation is a table with the JSON data transformed into rows and columns, as shown in Figure 5-8.

Example 5-20: Using the OPENJSON function with the CROSS APPLY operator

```
SELECT ProductID, Name, ReviewID, ReviewerName, ReviewerEmail, Rating, ReviewDate
FROM Production.Product
CROSS APPLY OPENJSON(Reviews)
WITH (
  ReviewID int '$.ReviewID',
  ReviewerName varchar(100) '$.Reviewer.Name',
  ReviewerEmail varchar(100) '$.Reviewer.Email',
  Rating int '$.Rating',
  ReviewDate datetime '$.ReviewDate'
)
 WHERE Reviews IS NOT NULL
```

	ProductID	Name	ReviewID	ReviewerName	ReviewerEmail	Rating	ReviewDate
1	709	Mountain Bike Socks, M	1	John Smith	john@fourthcoffee.com	5	2013-09-18 00:00:00.000
2	798	Road-550-W Yellow, 40	4	Laura Norman	laura@treyresearch.net	5	2013-11-15 00:00:00.000
3	937	HL Mountain Pedal	2	David	david@graphicdesigninstitute.com	4	2013-11-13 00:00:00.000
4	937	HL Mountain Pedal	3	Jill	jill@margiestravel.com	2	2013-11-15 00:00:00.000

Figure 5-8: Rows returned by the CROSS APPLY operator and OPENJSON function.

Working with a list of comma-separated values in T-SQL

Applications commonly generate lists of comma-separated values in the user interface, which you later need to separate by using any of a variety of techniques, such as a comma-splitting function. As long as database compatibility is set correctly, a simple approach to working with a list of comma-separated values is to use the list as an argument in the OPENJSON function. To avoid producing an error, be sure to enclose the list in brackets. Because the list is not executed by the OPENJSON function, there is no risk of SQL injection. Furthermore, the function understands the difference between commas used to separate items in the list and commas that are included as part of a string value.

Note You can find information about OPENJSON syntax at "OPENJSON (Transact-SQL),"at https://msdn.microsoft.com/en-us/library/dn921885.aspx, and more examples of working with OPENJSON in "Use OPENJSON with the Default Schema (SQL Server),"at https://msdn.microsoft.com/en-us/library/dn921891.aspx. For information about importing entire JSON files into SQL Server, see "Importing JSON files in SQL Server" at http://blogs.msdn.com/b/sqlserverstorageengine/archive/2015/10/07/importing-json-files-into-sql-server-using-openrowset-bulk.aspx.

Using other built-in JSON functions

SQL Server 2016 has additional built-in support for JSON with the following functions, which you can use in the SELECT, WHERE, or GROUP BY clause or in a constraint definition:

- **ISJSON** Returns 1 if the string argument supplied is a valid JSON structure, returns 0 if it is not, and returns null if the string argument is null. This function is demonstrated in Example 5-21.

Example 5-21: Using the ISJSON function

```
SELECT ProductID, Name, ProductNumber, Reviews
FROM Production.Product
WHERE ISJSON(Reviews) > 0;
```

- **JSON_VALUE** Extracts a scalar value with an *nvarchar(4000)* data type from a JSON string. This function takes two arguments, as shown in Example 5-22. The first argument is either the name of a variable to which you have assigned a JSON string or the name of a column containing JSON. The second argument is the path of the property to extract. If the JSON is invalid or the second argument is an object and not a property, the function returns an error. It also returns an error if the result is larger than 4,000 characters in strict mode.

Example 5-22: Using the JSON_VALUE function

```
SELECT ProductID, Name, ProductNumber, JSON_VALUE(Reviews, '$.Reviewer.Email') ReviewerEmail
FROM Production.Product
WHERE Reviews IS NOT NULL
```

Note If it is possible for a value to be greater than 4,000 characters, use the OPENJSON function instead.

- **JSON_QUERY** Extracts an object or array with an *nvarchar(max)* data type from a JSON string, as shown in Example 5-23. As with the JSON_VALUE function, the JSON_QUERY's first argument is a variable or column containing JSON, and the second argument is the property path. This function returns an error if the JSON is invalid or the property value is not an object. In this example, using `'$.Reviewer.Name'` as the second argument results in an error because the value is scalar, whereas `'$.Reviewer'` is an array containing the Name and Email properties and therefore valid as an argument to the JSON_QUERY function.

Example 5-23: Using the JSON_QUERY function

```
DECLARE @json NVARCHAR(1000);
SET @json = N'{"ReviewID":1,
               "Reviewer":{"Name":"John Smith",
                           "Email":"john@fourthcoffee.com"},
               "Rating":5,
               "ReviewDate":"2013-09-18T00:00:00"}';
SELECT JSON_QUERY(@json, '$.Reviewer') Reviewer;
```

- **JSON_MODIFY** Updates a property value in a JSON string, as shown in Example 5-24, and returns the updated string. Its first argument is the JSON string or reference, its second argument is the property path, and its third argument is the new property value. With this function, you can modify only one property at a time, but you can nest function calls if you need to make multiple changes as shown in the example.

Example 5-24: Using the JSON_MODIFY function

```
UPDATE Production.Product
SET Reviews = JSON_MODIFY(Reviews, '$.Rating', 4)
WHERE ProductID = 709
```

Note You can also use the JSON_MODIFY function to rename or delete a property or to append a new value to an array. For more information, see "JSON_MODIFY (Transact-SQL)" at https://msdn.microsoft.com/en-us/library/dn921892.aspx.

Lax mode versus strict mode

When you use any of the built-in JSON functions, you can use lax mode or strict mode. By default, a function runs in lax mode, which means it returns a null value if it encounters an error. For example, a function returns null if you request a property such as `$.Reviewer.Address` and the property does not exist. In strict mode, the function returns an error. You declare the mode explicitly by including the `lax` or `strict` keyword at the beginning of the path expression, like this: `lax$.Reviewer.Address` or `strict$.Reviewer.Address`.

Indexing JSON data

If you store JSON in a relational table, you might need to filter or sort by a property in the JSON data. For better performance, you might need an index. However, you cannot directly reference a JSON property in an index. Instead, you create a computed column and then create an index for that column, as shown in Example 5-25. The expression you use in the computed column must match the expression that queries use. When a query executes with this expression, the database engine recognizes the equivalent computed column and applies its index when possible. Because this new column is computed only when you build or rebuild the index, you do not incur any additional storage costs for the computed column.

Example 5-25: Index a JSON property as a computed column

```
ALTER TABLE Production.Product
  ADD vReviewer AS JSON_VALUE(Reviews, '$.Reviewer.Name');
CREATE INDEX idx_Production_Product_Reviewer_Name
  ON Production.Product(vReviewer);
```

Note If queries often return other columns from the same table, you can also use the INCLUDE argument when you create an index in order to add additional columns to the index as a further optimization. To learn more about indexing JSON data, see "Index JSON Data" at https://msdn.microsoft.com/en-us/library/mt612798.aspx.

PolyBase

PolyBase was introduced in SQL Server 2014 as an interface exclusively for Microsoft Analytics Platform System (APS; formerly known as Parallel Data Warehouse), with which you could access data stored in Hadoop Distributed File System (HDFS) by using SQL syntax in queries. In SQL Server 2016, you can now use PolyBase to query data in Hadoop or Azure Blob Storage and combine the results with relational data stored in SQL Server. To achieve optimal performance, PolyBase can dynamically create columnstore tables, parallelize data extraction from Hadoop and Azure sources, or push computations on Hadoop-based data to Hadoop clusters as necessary. After you install the PolyBase service and configure PolyBase data objects, your users and applications can access data from nonrelational sources without any special knowledge about Hadoop or blob storage.

Installing PolyBase

You can install only one instance of PolyBase on a single server, which must also have a SQL Server instance installed because the PolyBase installation process adds the following three databases: DWConfiguration, DWDiagnostics, and DWQueue. The installation process also adds the PolyBase engine service and PolyBase data movement service to the server.

Before you can install PolyBase, your computer must meet the following requirements:

- Installed software: Microsoft .NET Framework 4.5 and Oracle Java SE RunTime Environment (JRE) version 7.51 or higher (64-bit)
- Minimum memory: 4 GB
- Minimum hard-disk space: 2 GB
- TCP/IP connectivity enabled

To install PolyBase by using the SQL Server Installation Wizard, select PolyBase Query Service For External Data on the Feature Selection page. Then, on the Server Configuration page, you must configure the SQL Server PolyBase engine service and the SQL Server PolyBase data movement service to run under the same account. (If you create a PolyBase scale-out group, you must use the same service account across all instances.) Next, on the PolyBase Configuration page, you specify whether your SQL Server instance is a standalone PolyBase instance or part of a PolyBase scale-out group. As we describe later in this chapter, when you configure a PolyBase scale-out group, you specify whether the current instance is a head node or a compute node. Last, you define a range with a minimum of six ports to allocate to PolyBase.

Note The scale-out installation process automatically opens the firewall to allow incoming connections to the database engine, the PolyBase engine, the PolyBase data movement service, and SQL Browser. It also configures the firewall to allow incoming connections from other nodes in the scale-out group. If the Firewall service is not running during PolyBase installation, this step fails. In that case, you can start the Firewall service and enable the firewall rules described at "PolyBase Installation," https://msdn.microsoft.com/en-us/library/mt652312.aspx.

You can also install PolyBase from a command-prompt window by using a script such as the one shown in Example 5-26, using the following parameters specific to PolyBase:

- **/FEATURES** Identifies the features to install. If the database engine is already installed, specify PolyBase only. Otherwise, you must also include SQLEngine as a value.
- **/PBSCALEOUT** Specifies whether the instance to be installed is part of a scale-out group, with TRUE or FALSE as valid values.

- **/PBPORTRANGE** Defines a range of at least six ports to dedicate to PolyBase.
- **/PBENGSVCACCOUNT** Configures the service account for the PolyBase engine. The default value is NT Authority\NETWORK SERVICE.
- **/PBENGSVCPASSWORD** Specifies the password to use with the PolyBase engine service account.
- **/PBENGSVCSTARTUPTYPE** Configures the startup mode for the PolyBase engine service: Automatic (default), Manual, or Disabled.
- **/PBDMSSVCACCOUNT** Configures the service account for the PolyBase data movement service. The default value is NT Authority\NETWORK SERVICE.
- **/PBDMSSVCPASSWORD** Specifies the password to use with the PolyBase data movement service account.
- **/PBDMSSVCSTARTUPTYPE** Configures the startup mode for the PolyBase data movement service: Automatic (default), Manual, or Disabled.

Example 5-26: Installing PolyBase from a command-prompt window

```
Setup.exe /Q /ACTION=INSTALL /IACCEPTSQLSERVERLICENSETERMS /FEATURES=SQLEngine,Polybase
/INSTANCENAME=MSSQLSERVER /SQLSYSADMINACCOUNTS="<fabric-domain>\Administrator"
/INSTANCEDIR="C:\Program Files\Microsoft SQL Server" /PBSCALEOUT=TRUE
/PBPORTRANGE=16450-16460 /SECURITYMODE=SQL /SAPWD="<StrongPassword>"
/PBENGSVCACCOUNT="<DomainName>\<UserName>" /PBENGSVCPASSWORD="<StrongPassword>"
/PBDMSSVCACCOUNT="<DomainName>\<UserName>" /PBDMSSVCPASSWORD="<StrongPassword>"
```

Next you need to configure your PolyBase instance for connectivity to either Hadoop or Azure Blob Storage, as shown in Example 5-27, using one of the following values for the `@configvalue` parameter:

- **0** Disable Hadoop connectivity
- **1** Hortonworks HDP 1.3 on Windows Server and blob storage
- **2** Hortonworks HDP 1.3 on Linux
- **3** Cloudera CDH 4.3 on Linux
- **4** Hortonworks HDP 2.0 on Windows Server and blob storage
- **5** Hortonworks HDP 2.0 on Linux
- **6** Cloudera 5.1 on Linux
- **7** Hortonworks 2.1 and 2.2 on Linux, Hortonworks 2.2 on Windows Server and blob storage

Example 5-27: Configuring an external data source for PolyBase

```
sp_configure @configname = 'hadoop connectivity', @configvalue = 7;
reconfigure;
```

After you run the *sp_configure* and *reconfigure* commands, you must reboot the server. The reboot will, of course, stop and start the SQL Server service, which also stops and restarts the dependent

PolyBase services, but a server reboot is necessary before you can create PolyBase data objects, as we describe later in this chapter.

If you are using Hadoop, you can optionally perform these additional steps:

1. Enable pushdown computations to your Hadoop cluster by copying the value of the *yarn.application.classpath* configuration key from the yarn-site.xml file in the Hadoop configuration directory and pasting it into the *yarn.application.classpath* property in the yarn_site.xml file in the \Program Files\Microsoft SQL Server\<SQL Server instance>\Binn\Polybase\Hadoop\conf directory.
2. Configure Kerberos authentication to your Hadoop cluster by copying configuration key values from Hadoop configuration files into the value properties of the corresponding files in the \Program Files\Microsoft SQL Server\<SQL Server instance>\Binn\Polybase\Hadoop\conf directory, as shown in the following table:

Configuration file	Configuration key	Description
core-site.xml	polybase.kerberos.kdchost	KDC host name
	polybase.kerberos.realm	Kerberos realm
	hadoop.security.authentication	Authentication configuration, such as kerberos
hdfs-site.xml	dfs.namenode.kerberos.principal	Name node security principal, such as hdfs/_HOST@YOUR-REALM.COM
mapred-site.xml	mapreduce.jobhistory.address	Job history server IPC host:port, such as 0.0.0.0:10020
	mapreduce.jobhistory.principal	Kerberos principal name for the job history server, such as mapred/_HOST@YOUR-REALM.COM
yarn-site.xml	yarn.resourcemanager.principal	Yarn resource manager principal name, such as yarn/_HOST@YOUR-REALM.COM

Scaling out with PolyBase

Because data sets can become quite large in Hadoop or blob storage, you can create a PolyBase scale-out group, as shown in Figure 5-9, to improve performance. A PolyBase scale-out group has one head node and one or more compute nodes. The head node consists of the SQL Server database engine, the PolyBase engine service, and the PolyBase data movement service, whereas each compute node consists of a database engine and data movement service. The head node receives the PolyBase queries, distributes the work involving external tables to the data movement service on the available compute nodes, receives the results from each compute node, finalizes the results in the database engine, and then returns the results to the requesting client. The data movement service on the head node and compute nodes is responsible for transferring data between the external data sources and SQL Server and between the SQL Server instances on the head and compute nodes.

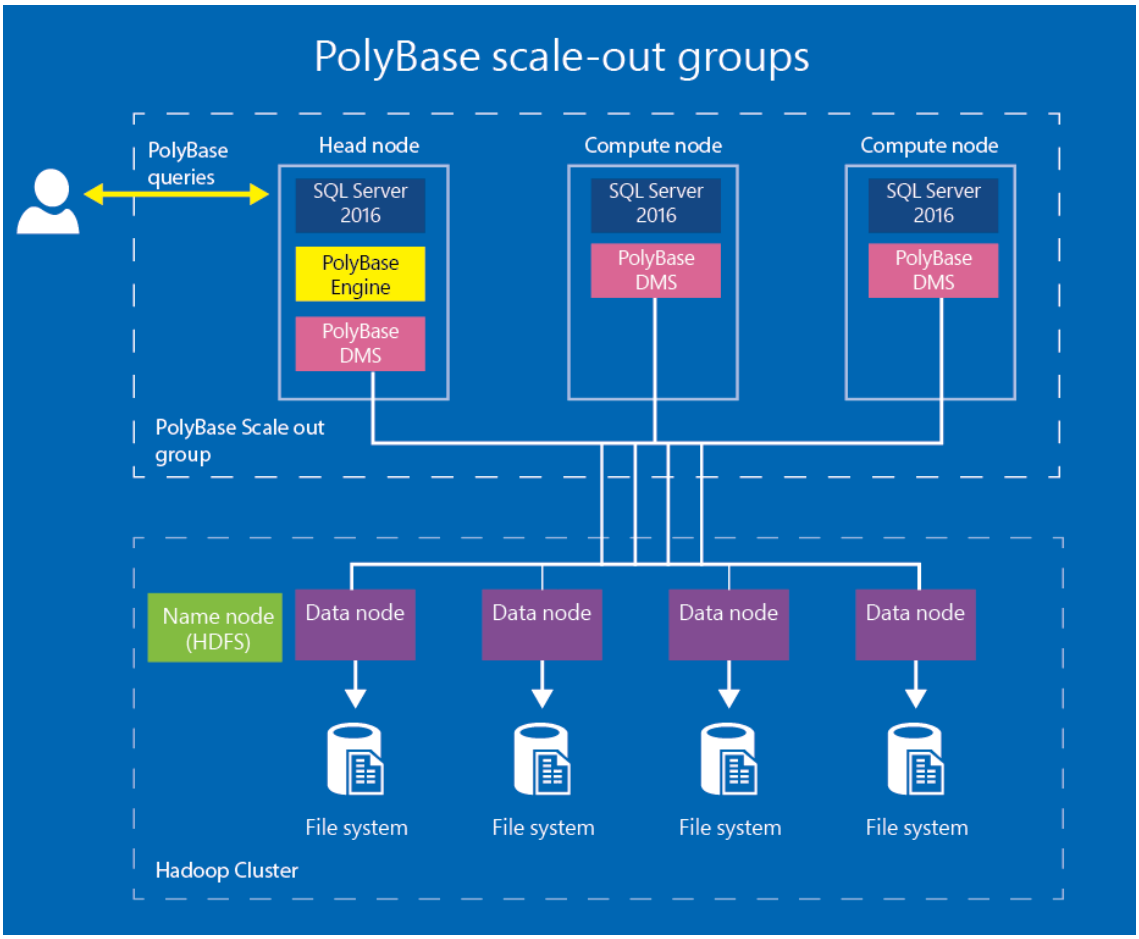


Figure 5-9: PolyBase scale-out groups.

After you install PolyBase on each SQL Server instance, the PolyBase instance is a head node by default. You must have one head node in the scale-out group and then use *sp_polybase_join_group* to transform the other instances to compute nodes, as shown in Example 5-28. The first parameter of this stored procedure is the computer name of the head node, the second parameter is the control channel port for the head node's data movement service, and the third parameter is the name of the SQL Server instance for the head node. The default port for the data movement service is 16450. After executing the stored procedure, restart the PolyBase engine and data movement services.

Example 5-28: Adding a compute node to a PolyBase scale-out group

```
-- Provide head node computer name, DMS port, and SQL Server instance as parameters
EXEC sp_polybase_join_group 'MySQLServer', 16450, 'MSSQLSERVER';
```

Note To remove a compute node from a scale-out group, use *sp_polybase_leave_group*. It does not take any arguments. You must restart the PolyBase engine and data movement services afterward.

You can confirm the configuration and status of compute nodes by using the following dynamic management views:

- ***sys.dm_exec_compute_nodes*** Returns the following columns: compute_node_id, type (HEAD or COMPUTE), name (server name and port number), and address (IP address).
- ***sys.dm_exec_compute_node_status*** Returns the status of all nodes in the group in the following columns: compute_node_id, process_id, process_name (Engine or DMS), allocated_memory, available_memory, process_cpu_usage, total_cpu_usage, and thread_count.
- ***sys.dm_exec_dms_services*** Reports the current status of the DMS service on each compute node.

Creating PolyBase data objects

To query data sources in Hadoop or blob storage, you need to create a set of T-SQL objects. First, if you are connecting to a Kerberos-secured Hadoop cluster or to blob storage, you need to create a database-scoped credential. Next, you need to create an external data source to define the location of your data and an external file format to describe the structure of your data. Then you can create external tables to define the table columns and associate the external table with an external data source and external file format.

Database-scoped credential

If you have data stored in blob storage or a Hadoop cluster that is configured to use Kerberos authentication, you must provide authentication information so that PolyBase can securely connect to the data source. To do this, you must create a master key and a database-scoped credential. The master key encrypts the database-scoped credential. When you create a database-scoped credential for blob storage, as shown in Example 5-29, replace the placeholder token for PASSWORD with a strong password, provide a unique name for the credential, and replace the placeholder token for SECRET with the storage account key for blob storage. To get this key, go to the storage account blade in the Azure Management Portal, click the Settings button, and then click Access Keys in the Settings blade. Click the Copy button to the right of one of the keys to store the key on the clipboard.

Example 5-29: Creating a database-scoped credential for blob storage

```
-- Create a master key if one does not already exist
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<StrongPassword>';
-- Create a database-scoped credential
CREATE DATABASE SCOPED CREDENTIAL AzureBlobStorageCredential
WITH
    IDENTITY = 'user',
    SECRET = '<azure_storage_account_key>';
```

Note You can use any string for IDENTITY when creating a database-scoped credential for blob storage. Authentication requires only the storage account key.

If you are connecting to a Kerberos-enabled Hadoop cluster, create the master key and database-scoped credential as shown in Example 5-30. Replace the placeholder token for PASSWORD with a strong password, provide a unique name for the credential, and replace the IDENTITY and SECRET placeholder tokens with the Kerberos user name and password, respectively.

Example 5-30: Creating a database-scoped credential for Kerberos-enabled Hadoop

```
-- Create a master key if one does not already exist
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<StrongPassword>';
-- Create a database-scoped credential
CREATE DATABASE SCOPED CREDENTIAL HadoopCredential
WITH IDENTITY = '<user>', SECRET = '<password>';
```

External data source

An external data source tells PolyBase where to find your data. If you are using blob storage, you specify the name of the blob container and the storage account in the LOCATION argument of the CREATE EXTERNAL DATA SOURCE statement, as shown in Example 5-31. The CREDENTIAL argument uses the database-scoped credential that you created for accessing blob storage.

Example 5-31: Creating an external data source for blob storage

```
CREATE EXTERNAL DATA SOURCE AzureBlobStorage with (
  TYPE = HADOOP,
  LOCATION ='wasbs://<datacontainer>@<azure_storage_account_name>.blob.core.windows.net/',
  CREDENTIAL = AzureBlobStorageCredential
);
```

To create an external data source for Hadoop storage, you specify the Uniform Resource Indicator (URI) for Hadoop in the LOCATION argument of the CREATE EXTERNAL DATA SOURCE statement, as shown in Example 5-32. For Hadoop storage, you can optionally include the RESOURCE_MANAGER_LOCATION argument and provide the address and port for the Hadoop Name Node. If you enabled pushdown computations for PolyBase, SQL Server's query optimizer can send data to the Name Node for preprocessing to reduce the volume of data transferred between SQL Server and Hadoop. If you omit this argument, pushdown computations are disabled.

Example 5-32: Creating an external data source for Hadoop storage

```
CREATE EXTERNAL DATA SOURCE HadoopStorage WITH (
  TYPE = HADOOP,
  LOCATION ='hdfs://10.10.10.10:8020',
  RESOURCE_MANAGER_LOCATION = '10.10.10.10:8032',
  CREDENTIAL = HadoopCredential
);
```

External file format

You must create an external file format to describe the general structure of your data by using the CREATE EXTERNAL FILE FORMAT statement applicable to your file type, as shown in Example 5-33. You define the structure of your data by using the following arguments:

- **FORMAT_TYPE** You can use one of the following options: PARQUET, ORC, RCFILE, or DELIMITEDTEXT. If you use RCFILE, you must also specify one of the following Hive Serializer and Deserializer (SerDe) methods: org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe or org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe.

- **FIELD_TERMINATOR** You use this argument as one of the format options for a delimited text file only. The default is a pipe character (|), but you can specify one or more characters if applicable. Enclose the field terminator in single quotation marks.
- **STRING_DELIMITER** This argument is another format option available only to a delimited text file. The default is an empty string, but you can specify one or more characters if applicable. Enclose the string delimiter in single quotation marks.
- **DATE_FORMAT** This argument specifies the date format to use for a delimited text file as a format option. If you do not supply one, PolyBase uses one of the following default formats:
  - **DateTime** 'yyyy-MM-dd HH:mm:ss'
  - **SmallDateTime** 'yyyy-MM-dd HH:mm'
  - **Date** 'yyyy-MM-dd'
  - **DateTime2** 'yyyy-MM-dd HH:mm:ss'
  - **DateTimeOffset** 'yyyy-MM-dd HH:mm:ss'
  - **Time** 'HH:mm:ss'

Note You can learn more about working with date formats in delimited text files at "CREATE EXTERNAL FILE FORMAT (Transact-SQL)," https://msdn.microsoft.com/en-us/library/dn935026.aspx.

- **USE_TYPE_DEFAULT** By default, PolyBase stores missing values in a delimited text file as NULL. You can set this argument as a format option to TRUE if you prefer that PolyBase replace missing values with one of the following default values: 0 for a numeric column, empty string for a string column, and 1900-01-01 for a date column.
- **DATA_COMPRESSION** By default, data is uncompressed. You can optionally add this argument to specify the compression method for the external data. You can use org.apache.hadoop.io.compress.DefaultCodec with the delimited text, RCFILE, and ORC file types; org.apache.hadoop.io.compress.GzipCodec with the delimited text and PARQUET file types; or org.apache.hadoop.io.compress.SnappyCodec with the ORC and PARQUET file types.

Example 5-33: Creating an external file format

```
-- PARQUET file
CREATE EXTERNAL FILE FORMAT MyPARQUETFileFormat
WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
);

-- ORC file
CREATE EXTERNAL FILE FORMAT MyORCFileFormat
WITH (
    FORMAT_TYPE = ORC,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.DefaultCodec'
);

-- RCFILE
CREATE EXTERNAL FILE FORMAT MyRCFILEFileFormat
WITH (
    FORMAT_TYPE = RCFILE,
    SERDE_METHOD = 'org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe',
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.DefaultCodec');
```

```
-- Delimited text file
CREATE EXTERNAL FILE FORMAT MyDelimitedTextFileFormat
WITH (
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS (
       FIELD_TERMINATOR = ',',
       USE_TYPE_DEFAULT = TRUE
),
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec'
);
```

External table

You use the CREATE EXTERNAL TABLE statement to describe the structure of your data in more detail by specifying column names and data types. The number of columns and data types must match your data or a mismatched row is rejected when you query the data. When you create an external table, SQL Server stores only the metadata and basic statistics about external data.

When your data is stored in blob storage, you use the WITH clause to provide the directory relative to the blob storage container that holds your data, reference the external data source, and specify the file format, as shown in Example 5-34.

Example 5-34: Creating an external table for file stored in blob storage

```
CREATE EXTERNAL TABLE [dbo].[CarSensorData] (
    [VIN] VARCHAR(50) NOT NULL,
    [Model] VARCHAR(50) NOT NULL,
    [TimeStamp] DATETIME NOT NULL,
    [EngineTemperature] int NOT NULL,
    [TirePressure] int NOT NULL
)
WITH (LOCATION='/rawcarevents/file_1.csv',
    DATA_SOURCE = AzureBlobStorage,
    FILE_FORMAT = MyDelimitedTextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
```

You can optionally include reject options. For REJECT_TYPE, you can specify VALUE to fail a PolyBase query when the number of rows rejected exceeds the number specified by the REJECT_VALUE argument. As an alternative, you can set REJECT_TYPE to PERCENTAGE to fail when the percentage of rows rejected exceeds the percentage specified by REJECT_VALUE. The percentage of rows depends on the current interval, which is set by the REJECT_SAMPLE_VALUE argument. PolyBase attempts to retrieve the number of rows that you set in this argument and calculates the percentage of failures based on this number. The percentage is recalculated when each set of rows is retrieved.

When your data is stored in Hadoop, the structure of the CREATE EXTERNAL TABLE statement is similar to the one you use for blob storage. You specify the data directory path from the root directory, reference the external data source, and specify the file format, as shown in Example 5-35.

Example 5-35: Creating an external table for a file stored in Hadoop

```
CREATE EXTERNAL TABLE [dbo].[HadoopCarSensorData] (
    [VIN] VARCHAR(50) NOT NULL,
    [Model] VARCHAR(50) NOT NULL,
```

```
    [TimeStamp] DATETIME NOT NULL,
    [EngineTemperature] int NOT NULL,
    [TirePressure] int NOT NULL
)
  WITH (LOCATION='/rawcarevents/car_sensor_data.tbl',
    DATA_SOURCE = HadoopStorage,
    FILE_FORMAT = MyDelimitedTextFileFormat,
    REJECT_TYPE = VALUE,
    REJECT_VALUE = 0
);
```

Note For more information about creating an external table, see "CREATE EXTERNAL TABLE (Transact-SQL)" at https://msdn.microsoft.com/en-us/library/dn935021.aspx.

Viewing PolyBase objects in SSMS

After you create the PolyBase objects, you can view them in SSMS in Object Explorer. Expand the database folder, expand Tables, and then expand External Tables to view your external tables, as shown in Figure 5-10. Your external data sources and external file formats are visible in the respective folders in the External Resources folder for your database.

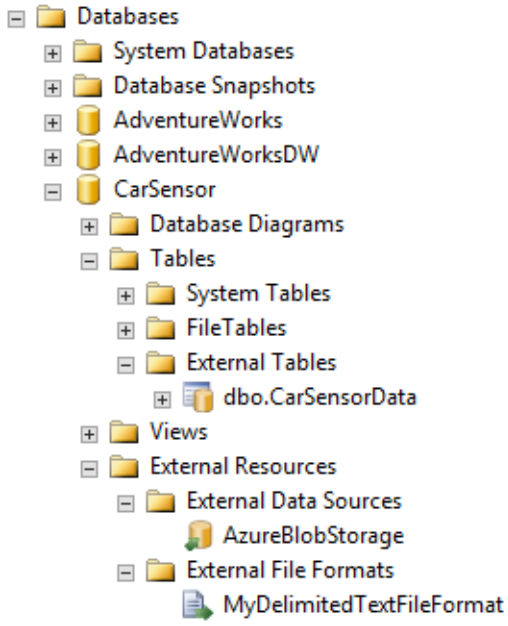


Figure 5-10: Viewing external resources in SSMS.

Using T-SQL with PolyBase objects

After you create external tables, your data is accessible by using standard T-SQL queries. Any application or reporting tool that can access SQL Server can use PolyBase by referencing an external table in the same way as for any relational table stored in SQL Server, without requiring special configuration in the tool or special knowledge from your users.

Not only can you write queries to retrieve data from external tables, you can also join the unstructured data with structured data stored in SQL Server, as shown in Example 5-36. If you are using data stored in Hadoop, you can add `OPTION (FORCE EXTERNALPUSHDOWN)` or `OPTION (DISABLE`

EXTERNALPUSHDOWN) to the end of your statement as a query hint to respectively force or disable pushdown computations.

Example 5-36: Joining a PolyBase external table with a SQL Server table

```
SELECT CustomerLastName, CustomerFirstName, CustomerEmail, cs.TirePressure
FROM Customer c
INNER JOIN CarSensorData cs ON c.VIN = cs.VIN
WHERE cs.TirePressure < 20
```

You can also export data from SQL Server to your external data source, but first you must configure SQL Server to allow this capability by using *sp_configure,* as shown in Example 5-37.

Example 5-37: Configure PolyBase export from SQL Server

```
sp_configure 'allow polybase export',1;
reconfigure;
```

Then use the INSERT INTO statement, as shown in Example 5-38, to export data. If the destination file or directory that you specified in the external table definition does not exist, your use of the INSERT INTO statement creates it.

Example 5-38: Exporting data from SQL Server to a PolyBase external table

```
INSERT INTO [dbo].[CarSensorDataArchive]
SELECT * FROM CarSensorData_local;
```

Troubleshooting with PolyBase system views and DMVs

You can use new DMVs in SQL Server 2016 to monitor and troubleshoot PolyBase. Use the following query to identify external tables in a database:

```
SELECT name, type from sys.tables WHERE is_external <> 0
```

To review the current set of PolyBase objects defined within a database, as shown in Figure 5-11, use the following views:

- ***sys.external_data_sources*** Displays information about external data sources, including data_source_id, name, location, type_desc, type, resource_manager_location, credential_id. Not shown in Figure 5-11 are the following columns related to elastic databases only: database_name and shard_map_name.
- ***sys.external_file_formats*** Displays information about external file formats in the following columns: file_format_id, name, format_type, field_terminator, string_delimiter, use_type_default, row_terminator, and encoding. Not shown in Figure 5-11 are the following columns: date_format, serde_method, and data_compression.
- ***sys.external_tables*** Displays information about external tables in the following columns: name, object_id, schema_id, parent_object_id, type, type_desc, data_source_id, file_format_id, location, reject_type, and reject_value. Not shown in Figure 5-11 are the following columns: principal_id, create_date, modify_date, is_ms_shipped, is_published, is_schema_published,

max_column_id_used, uses_ansi_nulls, reject_sample_value, distribution_type, distribution_desc, sharding_col_id, remote_schema_name, and remote_object_name.

	data_source_id	name	location	type_desc	type	resource_manager_location	credential_id
1	65536	AzureBlobStorage	wasbs://mycontainer@myblob.blob.core.windows.net	HADOOP	0	NULL	65537

	file_format_id	name	format_type	field_terminator	string_delimiter	use_type_default	row_terminator	encoding
1	65536	MyDelimitedTextFileFormat	DELIMITEDTEXT	,		1	\n	UTF8

	name	object_id	schema_id	parent_object_id	type	type_desc	data_source_id	file_format_id	location	reject_type	reject_value
1	CarSensorData	709577566	1	0	U	USER_TABLE	65536	65536	/rawcarevents/file_2.csv	VALUE	0
2	CarSensorDataArchive	741577680	1	0	U	USER_TABLE	65536	65536	/rawcarevents/file_3.csv	VALUE	0

Figure 5-11: Viewing metadata about external objects in SSMS.

For troubleshooting the distributed execution of PolyBase, use the following DMVs:

- ***sys.dm_exec_distributed_requests*** Sort the rows in this DMV by total elapsed time in descending order to find the execution_id for long-running queries. Include the CROSS APPLY operator to get the text value for sql_handle from *sys.dm_exec_sql_text*.
- ***sys.dm_exec_distributed_request_steps*** Filter this DMV by the execution_id found in *sys.dm_exec_distributed_requests* and sort the rows by total elapsed time in descending order to find the longest-running step for a query and note its step index. You can see the execution location—Head, Compute, or data movement service (DMS), the status, and the executed command, among other details.
- ***sys.dm_exec_distributed_sql_requests*** Filter this DMV by the execution_id and step_index from *sys.dm_exec_distributed_request_steps* to view the execution progress of a SQL operation (running in the Head or Compute location).
- ***sys.dm_exec_dms_workers*** Filter this DMV by the execution_id and step_index from *sys.dm_exec_distributed_request_steps* to view the execution progress of a DMS operation (running in the DMS location).
- ***sys.dm_exec_external_work*** Filter this DMV by the execution_id and step_index from *sys.dm_exec_distributed_request_steps* to view the execution progress of a DMS operation (running in the DMS location). This DMV helps you monitor operations that consume data from an external data source.
- ***sys.dm_exec_external_operations*** Filter this DMV by the execution_id and step_index from *sys.dm_exec_distributed_request_steps* to view the execution progress of map-reduce jobs pushed down to Hadoop when a PolyBase query runs against an external table.
- ***sys.dm_exec_compute_node_errors*** Use this DMV to review the errors that have occurred on a compute node.

CHAPTER 6

More analytics

Better and faster analytics capabilities have been built into SQL Server 2016. Enhancements to tabular models provide greater flexibility for the design of models, and an array of new tools helps you develop solutions more quickly and easily. As an option in SQL Server 2016, you can now use SQL Server R Services to build secure, advanced-analytics solutions at enterprise scale. By using R Services, you can explore data and build predictive models by using R functions in-database. You can then deploy these models for production use in applications and reporting tools.

Tabular enhancements

In general, tabular models are relatively easy to develop in SQL Server Analysis Services. You can build such a solution directly from a wide array of sources in their native state without having to create a set of tables as a star schema in a relational database. You can then see the results of your modeling within the design environment. However, there are some inherent limitations in the scalability and complexity of the solutions you can build. In the latest release of SQL Server, some of these limitations have been removed to better support enterprise requirements. In addition, enhancements to the modeling process make controlling the behavior and content of your model easier. In this section, we review the following enhancements that help you build better analytics solutions in SQL Server 2016:

- More data sources accessible in DirectQuery mode
- Choice of using all, some, or no data during modeling in DirectQuery mode
- Calculated tables
- Bidirectional cross-filtering
- Formula bar enhancements
- New Data Analysis Expressions (DAX) functions
- Using DAX variables

Accessing more data sources with DirectQuery

One of the benefits of using tabular models in Analysis Services is the ability to use data from a variety of data sources, both relational and nonrelational. Although prior versions of SQL Server support a quite extensive list of data sources, not all of those sources are available to use with DirectQuery, the feature in tabular models that retrieves data from the data source when a query is run instead of importing data into memory in advance of querying. Having live access to more data sources means that users can get answers to questions more quickly, and you have less administrative overhead to maintain in your analytic infrastructure.

In previous versions of SQL Server, you are limited to using SQL Server 2005 or later for a model in DirectQuery mode. In SQL Server 2016, the list of data sources supported for DirectQuery now includes the following:

- SQL Server 2008 or later
- Azure SQL Database
- Analytics Platform System (formerly Parallel Data Warehouse)
- Oracle 9i, 10g, 11g, and 12g
- Teradata V2R6, V2

When should you use DirectQuery?

Tabular models can compress and cache large volumes of data in memory for high-performance queries. DirectQuery might be a better option in some cases, but only if you are using a single data source. In general, you should use DirectQuery if any of the following situations apply: your users require real-time access to data, the volume of data is larger than the memory available to Analysis Services, or you prefer to rely on row-level security in the database engine.

Using DirectQuery can potentially have an adverse impact on query performance. If your source is SQL Server 2012 or later, you should consider implementing columnstore indexes so that DirectQuery can take advantage of query optimization provided by the database engine.

Even if you create a tabular model in in-memory mode, you can always switch to DirectQuery mode at any time. If you do this, any data previously stored in the cache is flushed, but the metadata is retained.

There are some drawbacks to using DirectQuery mode that you should consider before choosing it for your model. First, you cannot create calculated columns or calculated tables in the model, nor can you add a pasted table. An alternative is to use corresponding logic to create a derived column or a view in the underlying source. Second, because Analysis Services translates the DAX formulas and measures of your model into SQL statements, you might encounter errors or inconsistent behavior for some DAX functions that do not have a counterpart in SQL, such as time-intelligence functions or some statistical functions. In that case, you might be able to create a derived column in the source. You can see a list of functions that are not supported in DirectQuery mode at https://msdn.microsoft.com/en-us/library/hh213006.aspx#bkmk_NotSupportedFunc.

To learn more about DirectQuery mode in general, see https://msdn.microsoft.com/en-us/library/hh230898.aspx.

Modeling with a DirectQuery source

During the tabular modeling process, you import data from data sources into the design environment, unless your model is configured in DirectQuery mode. A new element in this process in SQL Server 2016 is that you can specify whether to create a model by using all the data (which is the only option

in earlier versions of SQL Server), no data (which is the new default), or a subset of data based on a query you supply.

To use the later two options, double-click the Model.bim file in Solution Explorer (if it is not already open in SQL Server Data Tools), and then click the file again to display its properties in the Properties window. If necessary, select SQL Server 2016 RTM (1200) in the Compatibility Level drop-down list. Then select On in the DirectQuery Mode drop-down list.

> **Important** If you upgrade the model from in-memory to DirectQuery mode, you cannot revert to a lower compatibility level.

To connect your model to the data source, you still use the Table Import Wizard, which you launch by selecting Import From Data Source from the Model menu. You select a relational database and then the tables for your model, but the data is no longer imported for DirectQuery-mode models. Instead, you work with an empty model that contains only metadata, such as column names and relationships, as shown in Figure 6-1. You can continue by configuring properties for columns, defining relationships, or working with the model in diagram view, just as you normally would if you import data instead. In DirectQuery mode, the data stays in the source until you generate a query in Excel or another presentation-layer application.

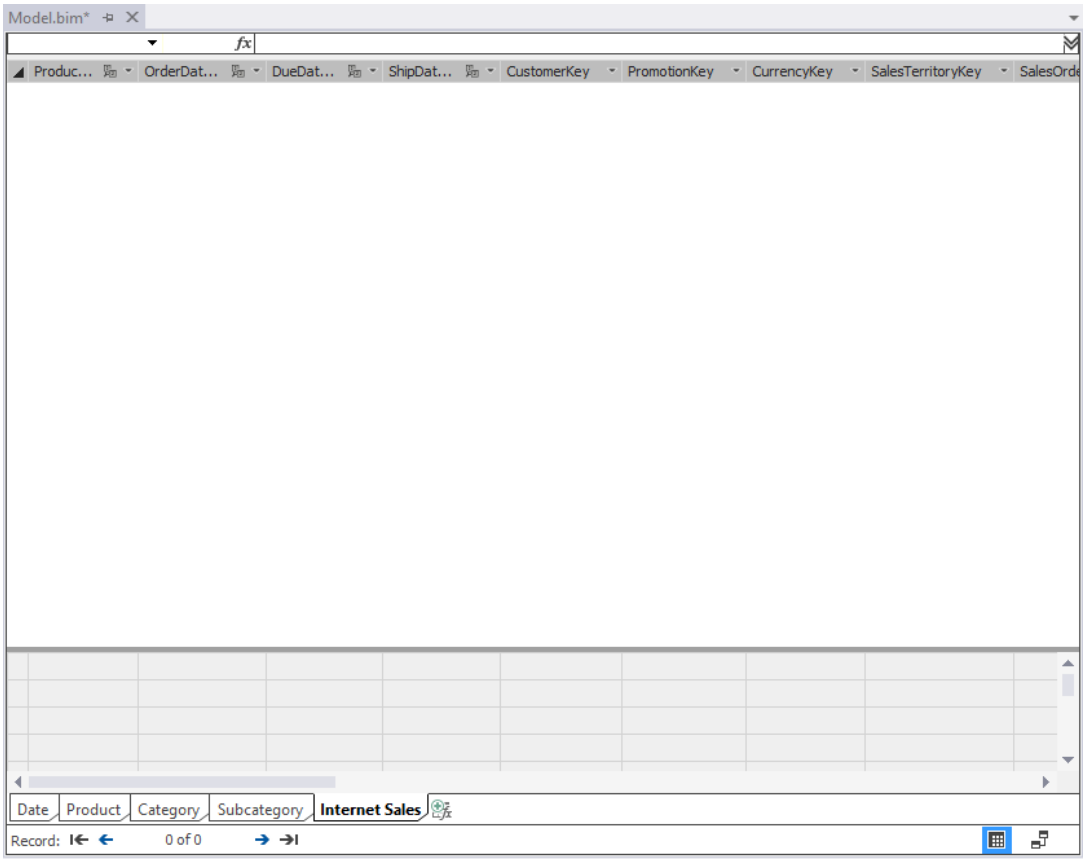


Figure 6-1: Working with a model in DirectQuery mode.

When you work without data in the model designer, the modeling process is likely to be faster because you no longer have to wait for calculations to be performed against the entire data set as you add or change measures. However, you might prefer to view sample data to help you better review the model during its design. To do this, select a table, and then select Partitions on the Table menu. Select the existing partition, which has the prefix DirectQuery, click the Copy button, and then select

the copy of the partition (which has the prefix Sample). Click the Edit SQL Query button to add a WHERE clause to the partition query, as shown in Figure 6-2, which returns a subset of the original partition query. When you close the Partition Manager and process the partition, the subset of rows defined by your query is displayed in the model designer.

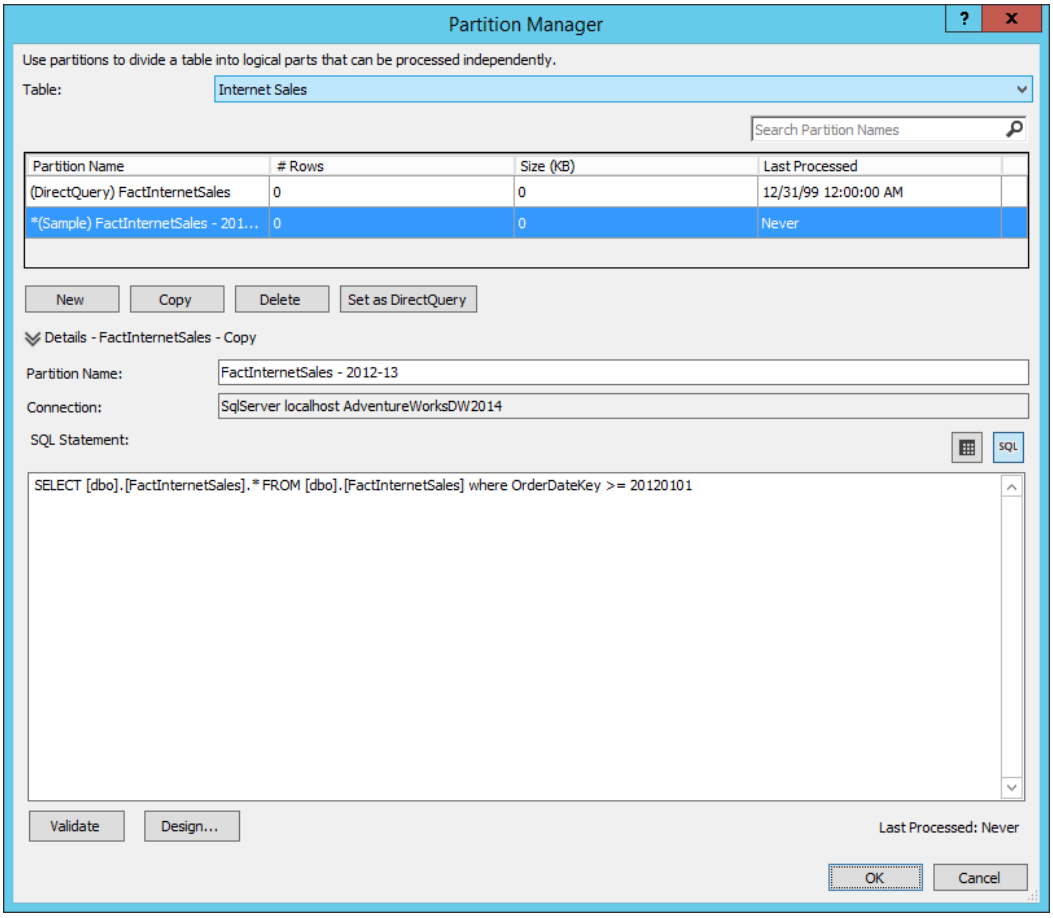


Figure 6-2: Configuring sample data for a partition in DirectQuery mode.

Note If you create multiple sample partitions, the model designer displays the combined results from all the queries.

Notice the Set As DirectQuery button below the list of partitions in Figure 6-2. This button appears only when you select a sample partition in the list. When you click this button, you reset the partition that Analysis Services uses to retrieve data for user queries. For this reason, there can be only one DirectQuery partition defined for a table at any time. If you select the DirectQuery button, the button's caption displays Set As Sample instead.

You can also use a sample data partition with the Analyze Table In Excel feature to test the model from the presentation layer. It's helpful to test the results of calculations in measures and to check relationships between tables even when you are using only a subset of data. To do this, select Analyze In Excel from the Model menu. When the model is in DirectQuery mode, the Analyze In Excel dialog box requires you to choose one of two options, Sample Data View or Full Data View, as shown in Figure 6-3.

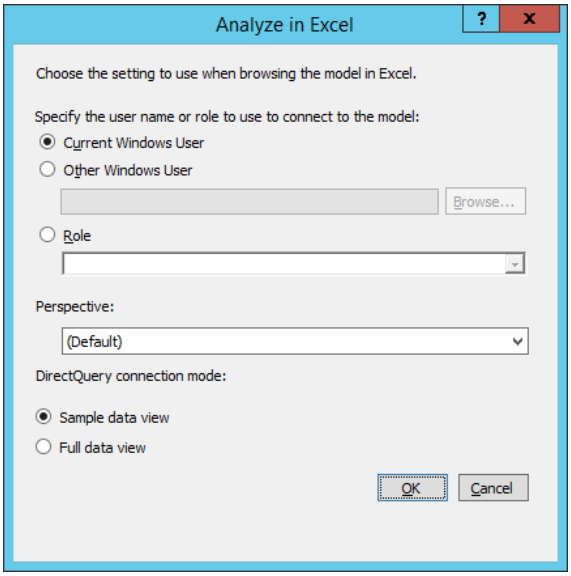


Figure 6-3: Using the sample data view in DirectQuery mode to analyze data in Excel.

If you create a sample partition for one table, you should create sample partitions for all tables. Otherwise, when you use the Analyze Table In Excel option, you see null values for columns in tables without a sample partition. For example, in Figure 6-4, CalendarYear is placed in rows but displays null values, and Total Sales is a measure added to the Internet Sales table for which sample data is defined.

	A	B
1	Row Labels	Total Sales
2		22239730.26
3	Grand Total	22239730.26

Figure 6-4: Viewing sample data for multiple tables in Excel in which a sample partition is defined for Internet Sales only.

Your sample partition can be a copy of the partition's data, if you prefer that. Simply make a copy of the DirectQuery partition and omit the addition of a WHERE clause. This approach is useful if your table is relatively small, and it also allows you to better confirm that the sample partitions defined for other tables are correct, as shown in Figure 6-5.

	A	B
1	Row Labels	Total Sales
2	2012	5842485.195
3	2013	16351550.34
4	2014	45694.72
5	Grand Total	22239730.26

Figure 6-5: Viewing sample data for multiple tables in Excel after adding a sample partition for the Date table.

Working with calculated tables

A new feature for tabular models in SQL Server 2016 is the *calculated table*—as long as you are working with a model that is not in DirectQuery mode. A calculated table is built by using a DAX expression. A model that includes calculated tables might require more memory and more processing time than a model without calculated tables, but it can be useful in situations for which you do not have an existing data warehouse with cleansed and transformed data. This technique is useful in the following situations:

- Creating small data sets to satisfy simple requirements without adding a lot of overhead to your technical infrastructure.
- Prototyping a solution before building a complete solution.
- Creating a simple date table by using the new CALENDAR() or CALENDARAUTO() functions.
- Separating a role-playing dimension into multiple tables for simpler modeling.

When you create a calculated table, you can choose to use only a few columns from a source table, combine columns from multiple tables, or apply complex expressions to filter and transform existing data into a new table. As a simple example, the FactInternetSales table in the AdventureWorksDW sample database contains the following three columns: OrderDateKey, ShipDateKey, and OrderDateKey. These three columns have a foreign-key relationship to a single role-playing dimension, DimDate. Instead of activating a relationship to change the context of a query from Order Date to Ship Date or Due Date, as required in earlier versions, you can now create one calculated table for ShipDate and another for DueDate.

To create a calculated table, you must set your model to compatibility level 1200. Select New Calculated Table from the Table menu or click the Create A New Table Calculated From A DAX Formula tab at the bottom of the model designer, as shown in Figure 6-6. The designer displays a new empty table in the model. In the formula bar above the table, enter a DAX expression or query that returns a table.

Figure 6-6: Adding a new calculated table.

To continue with our example, you can use a simple expression such as `=Date` to copy an existing role-playing dimension table, Date (renamed from DimDate). The model evaluates the expression and displays the results. You can then rename the calculated table, add relationships, add calculated columns, and perform any other activity that you normally would with a regular table. The tab at the bottom of the model designer includes an icon that identifies the table as a calculated table, as shown in Figure 6-7.

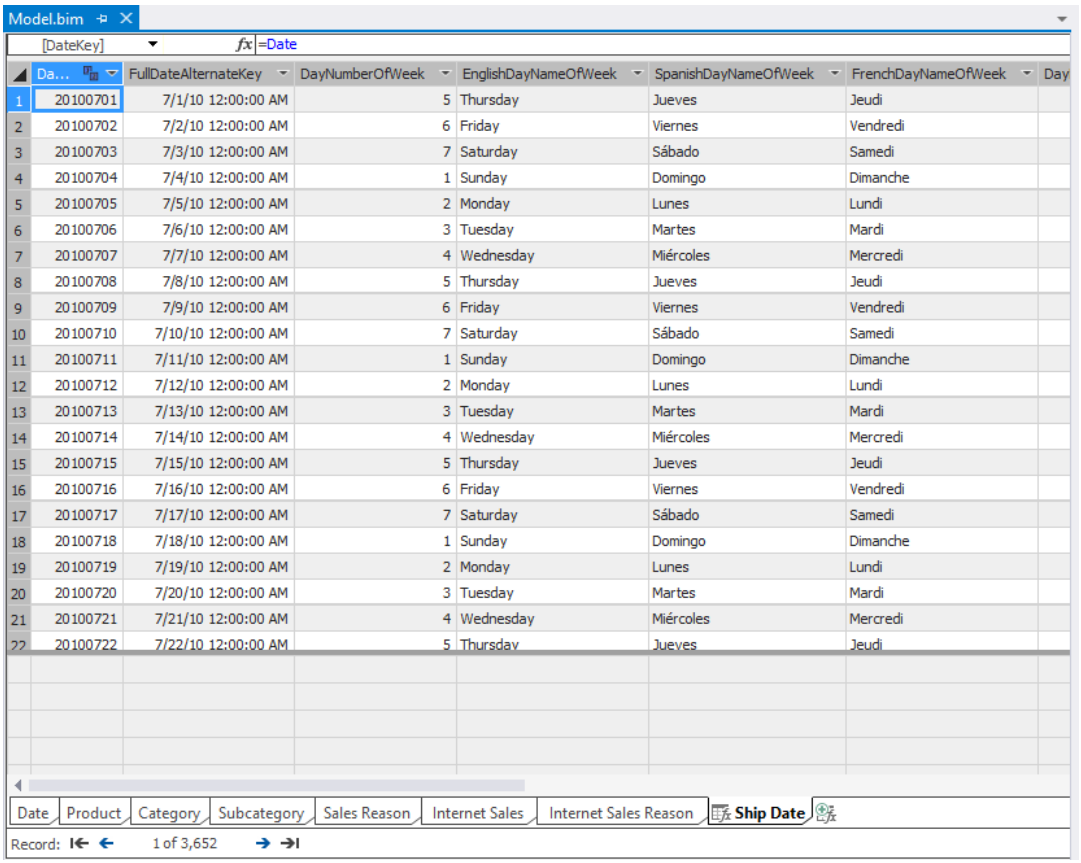


Figure 6-7: Viewing a calculated table in the model designer.

Note Some interesting uses for calculated tables are described at http://www.sqlbi.com/articles/transition-matrix-using-calculated-tables/ and https://pbidax.wordpress.com/2015/09/27/use-calculated-table-to-figure-out-monthly-subscriber-numbers/. Although these articles describe uses for calculated tables in Power BI, you can now use the same approach in tabular models.

Bidirectional cross-filtering

Another feature new to tabular models is bidirectional cross-filtering. This rather complex-sounding name allows cross-filtering across a table relationship in two directions rather than one direction, which has always been a feature in tabular models. This means that you no longer need to create complex DAX expressions to produce specific results, as long as you set your model to compatibility level 1200.

To better understand the implications of this new feature, we'll start by reviewing one-directional cross-filtering. A one-directional cross-filter applies in a one-to-many relationship such as that between a dimension table and a fact table. Typically, the fact table contains a column with a relationship to a corresponding column in the dimension table. In a tabular model based on AdventureWorksDW, FactInternetSales (which you can rename as Internet Sales) has the ProductKey column, and you can use this column to define a relationship in a single direction from the DimProduct dimension table (renamed as Product) to the fact table. If you import these tables into the model at the same time, the foreign-key relationship defined in the tables is automatically detected and configured for you. Otherwise, you can manually create the relationship.

When you use a PivotTable to query the model, you see how the dimension labels in the rows become the filter context to the value from the fact table, which appears on the same row, as shown in Figure 6-8. In this example, each row is a product line from the Product table. To derive the aggregate value in the Sales Count column in each row, the Analysis Services engine filters the table for the current product line value and computes the count aggregate for sales having that value. Because of the one-directional filter in the relationship, each entry for Sales Count is not only an aggregate value but also a filtered value based on the current row's dimension value. If no relationship existed between the two tables, the Sales Count value would display the total aggregated value of 60,398 in each row because no filter would be applicable.

Row Labels	Sales Count
M	16,898
R	15,552
S	23,358
T	4,590
Grand Total	**60,398**

Figure 6-8: Viewing the effect of a one-directional filter between Product and Internet Sales.

Although you can create measures in any table in a tabular model, the behavior you see in a PivotTable might not produce the results you want if you add a measure to a dimension table. Let's say that you add a distinct count of products to the Product table and then add Calendar Year to your query. In this case, a one-directional relationship exists between Date and Internet Sales, which can be combined with the one-directional relationship between Product and Internet Sales to compute Sales Count by year and by product line, as shown in Figure 6-9. However, because the relationship between Product and Internet Sales is one-directional from Product to Internet Sales, terminating at Internet Sales, there is no relationship chain that goes from Date to Internet Sales to Product that provides the filter context necessary to compute the distinct count measure. Consequently, the distinct count by product line, which is in the same table and thereby provides filter context, repeats across all years for each product line.

Row Labels	Sales Count	Distinct Product Count
M	**16,898**	**112**
2011	390	112
2012	1,064	112
2013	14,584	112
2014	860	112
R	**15,552**	**162**
2011	1,788	162
2012	2,156	162
2013	11,101	162
2014	507	162
S	**23,358**	**54**
2011		54
2012		54
2013	22,011	54
2014	1,347	54
T	**4,590**	**52**
2011		52
2012		52
2013	4,389	52
2014	201	52
Grand Total	**60,398**	**380**

Figure 6-9: Viewing the effect of a one-directional filter between Product and Internet Sales and Date and Internet Sales on measures in the Product table.

You can override this behavior by changing the relationship between Product and Internet Sales to a bidirectional relationship. To do this, select Manage Relationships from the Table menu, double-click

the relationship between these two tables, and select To Both Tables in the Filter Direction drop-down list, as shown in Figure 6-10.

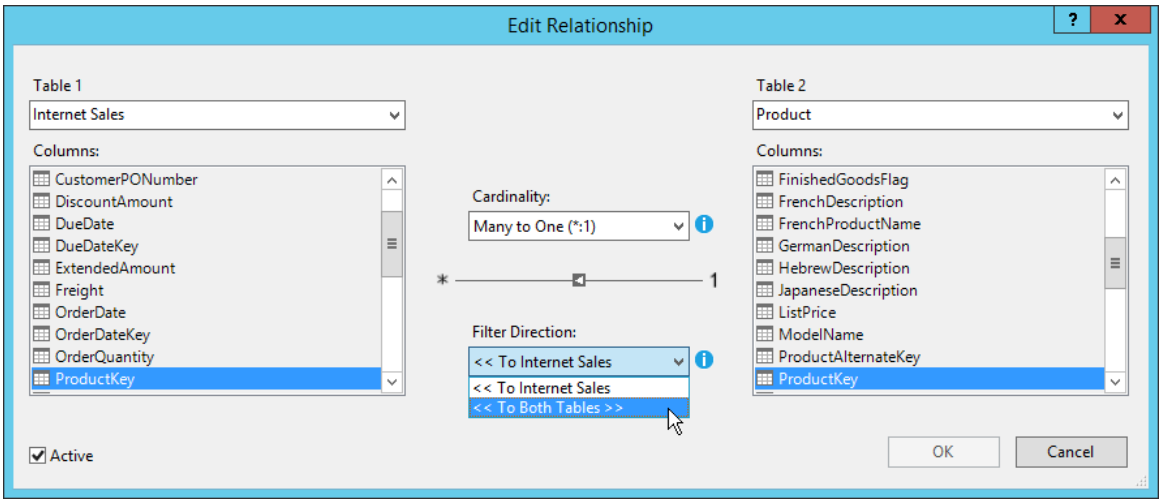


Figure 6-10: Setting a bidirectional cross-filter in a table relationship.

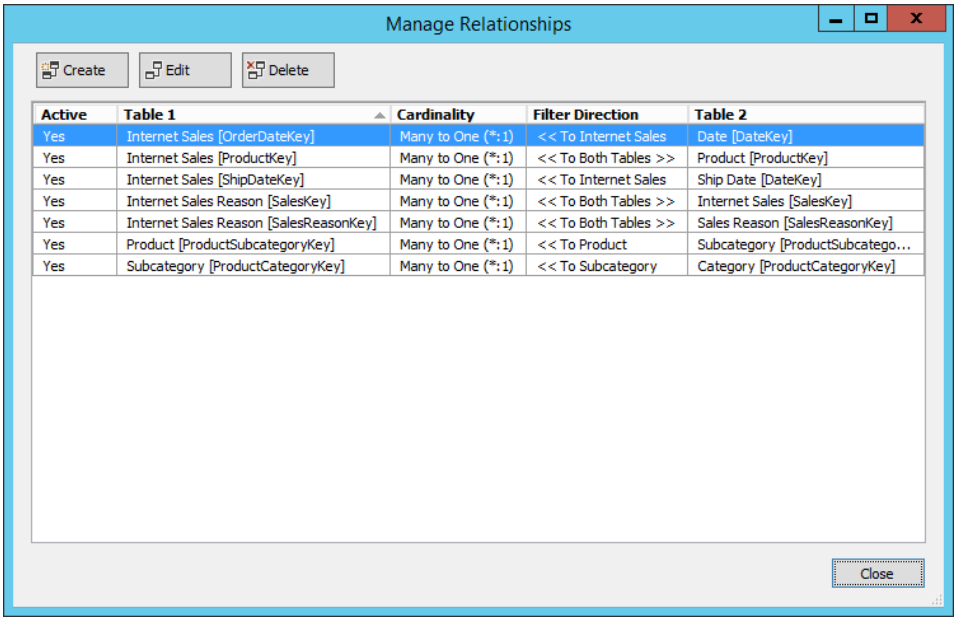
With this change, the filter context of the current year applies to both the Internet Sales table (as it did previously) to correctly aggregate Sales Count, and to the Product table to correctly aggregate Distinct Product Count, as shown in Figure 6-11.

	A	B	C
1			
2			
3	**Row Labels**	**Sales Count**	**Distinct Product Count**
4	**M**	**16,898**	**44**
5	2011	390	8
6	2012	1,064	10
7	2013	14,584	36
8	2014	860	26
9	**R**	**15,552**	**67**
10	2011	1,788	17
11	2012	2,156	32
12	2013	11,101	35
13	2014	507	25
14	**S**	**23,358**	**23**
15	2013	22,011	23
16	2014	1,347	23
17	**T**	**4,590**	**24**
18	2013	4,389	24
19	2014	201	17
20	**Grand Total**	**60,398**	**158**

Figure 6-11: Viewing the effect of a bidirectional filter between Product and Internet Sales and a one-directional filter between Date and Internet Sales on measures in the Product table.

Another problem that bidirectional cross-filtering can solve is the modeling of a many-to-many relationship, which in earlier versions of SQL Server required you to create complex DAX expressions. An example of a many-to-many relationship in the AdventureWorksDW database is the one in which the Internet Sales table stores individual sales orders by line number. The DimSalesReason table stores the reasons a customer indicated for making the purchase, such as price or quality. Because a customer could choose zero, one, or more reasons for any item sold, a factless fact table called FactInternetSalesReason is in the database to relate the sales reasons by sales order and line number. You can add this table to a tabular model and easily aggregate values in the Internet Sales table by sales reason after making a few adjustments to the model.

Because the structure of the two fact tables in this example does not allow you to define a relationship between them, you must add a calculated column called Sales Key (or another unique name that you prefer) to each of them to uniquely identify the combination of sales order and line number. To do this, you use a DAX expression similar to this: `=[SalesOrderNumber]&"-"&[SalesOrderLineNumber]` in each fact table. You can then create a relationship between the two tables using this common column and set the direction to To Both Tables, as shown in Figure 6-12.



Active	Table 1	Cardinality	Filter Direction	Table 2
Yes	Internet Sales [OrderDateKey]	Many to One (*:1)	<< To Internet Sales	Date [DateKey]
Yes	Internet Sales [ProductKey]	Many to One (*:1)	<< To Both Tables >>	Product [ProductKey]
Yes	Internet Sales [ShipDateKey]	Many to One (*:1)	<< To Internet Sales	Ship Date [DateKey]
Yes	Internet Sales Reason [SalesKey]	Many to One (*:1)	<< To Both Tables >>	Internet Sales [SalesKey]
Yes	Internet Sales Reason [SalesReasonKey]	Many to One (*:1)	<< To Both Tables >>	Sales Reason [SalesReasonKey]
Yes	Product [ProductSubcategoryKey]	Many to One (*:1)	<< To Product	Subcategory [ProductSubcatego...
Yes	Subcategory [ProductCategoryKey]	Many to One (*:1)	<< To Subcategory	Category [ProductCategoryKey]

Figure 6-12: Defining a many-to-many relationship.

Then, when you create a PivotTable to review sales counts by sales reason, the many-to-many relationship is correctly evaluated, as shown in Figure 6-13, even though there is no direct relationship between the Sales Reason table and Internet Sales. The grand total continues to correctly reflect the count of sales, which is less than the sum of the individual rows in the PivotTable. This is expected behavior for a many-to-many relationship because of the inclusion of the same sale with multiple sales reasons.

	A	B
1		
2		
3	**Row Labels**	**Sales Count**
4	Manufacturer	1,818
5	On Promotion	7,390
6	Other	3,653
7	Price	47,733
8	Quality	1,551
9	Review	1,640
10	Television Advertisement	730
11	**Grand Total**	**60,398**

Figure 6-13: Viewing the effect of a many-to-many relationship in a PivotTable.

Important Although you might be tempted to configure bidirectional filtering on all relationships to address all possible situations, it is possible for this configuration to overfilter results unexpectedly. Therefore, you should test the behavior of each filter direction change to ensure that you get the results you want.

By default, a relationship between two tables is one-directional unless you change this behavior at the model or environment level. At the model level, open the model properties, and then choose Both Directions in the Default Filter Direction drop-down list. This change applies only to your current model; any new models you create will default to single direction. If you prefer to change the default for all new models, select Options from the Tools menu, expand Analysis Services Tabular Designers in the navigation pane on the left, select New Project Settings, and then select Both Directions in the Default Filter Direction drop-down list, as shown in Figure 6-14.

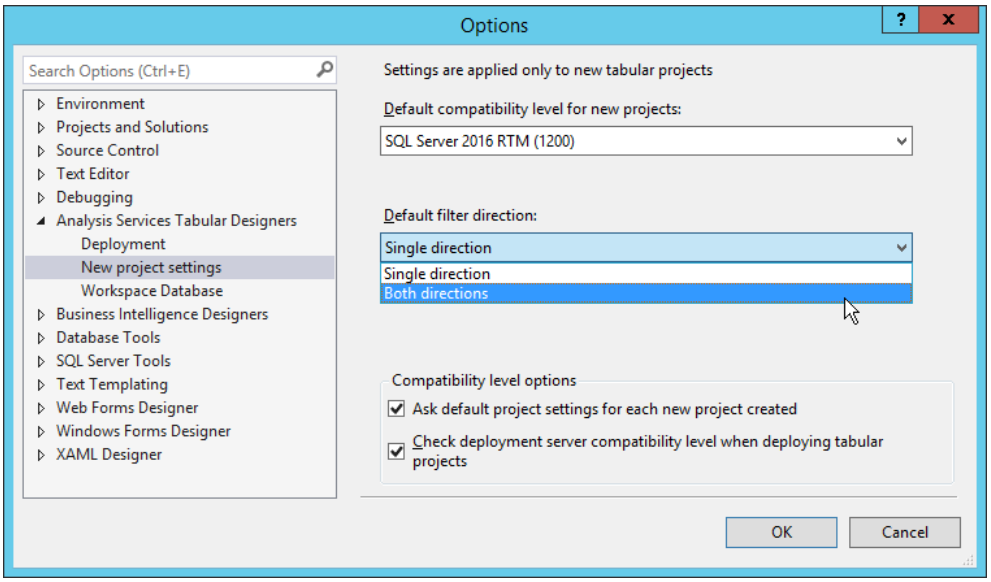


Figure 6-14: Setting the default filter direction for new projects.

Writing formulas

The user interface for the formula bar in the model designer has been improved by the addition of the following changes, which help you write and review DAX formulas more easily:

- **Syntax coloring** Functions are now displayed in a blue font, variables in a cyan font, and string constants in a red font to distinguish these expression elements more easily from fields and other elements.
- **IntelliSense** Errors are now identified by a wavy red underscore, and typing a few characters displays a function, table, or column name that begins with matching characters.
- **Formatting** You can persist tabs and multiple lines by pressing Alt+Enter in your expression to improve legibility. You can also include a comment line by typing **//** as a prefix to your comment.
- **Formula fixup** In a model set to compatibility level 1200, the model designer automatically updates measures that reference a renamed column or table.
- **Incomplete formula preservation** In a model set to compatibility level 1200, you can enter an incomplete formula, save and close the model, and then return to your work at a later time.

Introducing new DAX functions

The current release of SQL Server 2016 includes many new functions, which are described in the following table:

Function type	Function	Description
Date and time	CALENDAR()	Returns a table with one column named Date, containing a date range based on start and end dates that you provide as arguments.
	CALENDARAUTO()	Returns a table with one column named Date, containing a date range based on minimum and maximum dates present in the model's tables.
	DATEDIFF()	Returns the count of intervals (such as DAY or WEEK) between the start and end dates provided as arguments.
Filter	ADDMISSINGITEMS()	Includes rows with missing values in the result set.
	SUBSTITUTEWITHINDEX()	Returns a table containing the results of a left semi-join between two tables, replacing common columns in these tables with a single index column.
Information	ISONORAFTER()	Returns a Boolean value for each row to indicate whether two values (such as a column value and a constant value) are the same.
Math and Trig	ACOS()	Returns the arccosine, or inverse cosine, of a number.
	ACOSH()	Returns the inverse hyperbolic cosine of a number.
	ASIN()	Returns the arcsine, or inverse sine, of a number.
	ASINH()	Returns the inverse hyperbolic sine of a number.
	ATAN()	Returns the arctangent, or inverse tangent, of a number.
	ATANH()	Returns the inverse hyperbolic tangent of a number.
	COMBIN()	Returns the total number of possible groups of a number for a specified number of items.
	COMBINA()	Returns the number of combinations with repetitions of a number for a specified number of items.
	COS()	Returns the cosine of a number.
	COSH()	Returns the hyperbolic cosine of a number.
	DEGREES()	Converts radians into degrees.
	EVEN()	Rounds a number up to the nearest even integer.
	EXP()	Returns a decimal number for *e* raised to the power of a given number.
	GCD()	Returns the greatest common divisor between two specified numbers as an integer without a remainder.
	ISO.CEILING()	Rounds a number up to the nearest integer or to the nearest multiple of significance.
	LCM()	Returns the least common multiple between two specified integers.

Function type	Function	Description
	MROUND()	Returns a number rounded to the nearest multiple of a specified number.
	ODD()	Rounds a number up to the nearest odd integer.
	PI()	Returns the value of pi with 15-digit accuracy.
	PRODUCT()	Returns the product of values in a column.
	PRODUCTX()	Returns the product of an expression evaluated for each row in a table.
	QUOTIENT()	Performs division on the numerator and denominator provided as arguments and returns the integer portion of the result.
	RADIANS()	Converts degrees to radians.
	SIN()	Returns the sine of a number.
	SINH()	Returns the hyperbolic sine of a number.
	SQRTPI()	Returns the square root of pi multiplied by a specified number.
	TAN()	Returns the tangent of a number.
	TANH()	Returns the hyperbolic tangent of a number.
	XIRR()	Returns the internal rate of return.
	XNPV()	Returns the net present value.
Statistical	BETA.DIST()	Returns the beta distribution of a sample.
	BETA.INV()	Returns the inverse of the beta cumulative probability density function.
	CHISQ.INV()	Returns the inverse of the left-tailed probability of the chi-squared distribution.
	CHISQ.INV.RT()	Returns the inverse of the right-tailed probability of the chi-squared distribution.
	CONFIDENCE.NORM()	Returns the confidence interval as a range of values.
	CONFIDENCE.T()	Returns the confidence interval for a population mean using a student's t distribution.
	EXPON.DIST()	Returns the exponential distribution.
	GEOMEAN()	Returns the geometric mean of numbers in a column.
	GEOMEANX()	Returns the geometric mean of an expression evaluated for each row in a column.
	MEDIAN()	Returns the median of numbers in a column.
	MEDIANX()	Returns the median of an expression evaluated for each row in a column.
	PERCENTILE.EXC()	Returns the *k*th percentile of values in a range, where *k* is in the range 0 to 1, exclusive.
	PERCENTILE.INC()	Returns the *k*th percentile of values in a range, where *k* is in the range 0 to 1, inclusive.
	PERCENTILEX.EXC()	Returns the *k*th percentile of an expression evaluated for each row in a column, where *k* is in the range 0 to 1, exclusive.

Function type	Function	Description
	PERCENTILEX.INC()	Returns the *k*th percentile of an expression evaluated for each row in a column, where *k* is in the range 0 to 1, inclusive.
Text	CONCATENATEX()	Concatenates the results of an expression evaluated for each row in a table.
Other	GROUPBY()	Returns a table with selected columns with the results of evaluating an expression for each seat of GroupBy values.
	INTERSECT()	Returns a table containing values in one table that are also in a second table.
	ISEMPTY()	Returns a Boolean value indicating whether a table is empty.
	NATURALINNERJOIN()	Returns a table after performing an inner join of two tables.
	NATURALLEFTOUTERJOIN()	Returns a table after performing a left outer join of two tables.
	SUMMARIZECOLUMNS()	Returns a table containing combinations of values from two tables for which the combination is nonblank.
	UNION()	Returns a table containing all rows from two tables.

Using variables in DAX

You can now include variables in a DAX expression to break up a complex expression into a series of easier-to-read expressions. Another benefit of using variables is the reusability of logic within the same expression, which might possibly improve query performance.

Here's an example that focuses on sales of products in categories other than bikes and finds the ratio of the sales of these products with a unit price less than $50 to all sales of these products. First, to create this measure without variables and without using intermediate measures, you would use the expression shown in Example 6-1.

Example 6-1: Creating a complex DAX expression without variables

```
Non Bikes Sales Under $50 % of Total:=
sumx(
     filter(values(Category[CategoryName]), Category[CategoryName]<> "Bikes"),
     calculate(sum([SalesAmount]),'Internet Sales'[UnitPrice]<50)
)
/
sumx(
     filter(values(Category[CategoryName]), Category[CategoryName]<> "Bikes"),
     calculate(sum([SalesAmount]))
)
```

To reproduce the same results by using an expression with variables, you can use the expression shown in Example 6-2. You can use as many variables as you like in the expression. Use the VAR keyword to introduce each variable, then use the RETURN keyword for the final expression to resolve

the expression and return to the Analysis Services engine. Notice that a variable can be a scalar variable or a table.

Example 6-2: Creating a complex DAX expression with variables

```
Non Bikes Sales Under $50 % of Total:=
// create a table for all categories except Bikes
var
   tNonBikes = filter(values(Category[CategoryName]), Category[CategoryName]<> "Bikes")
// get the total of sales for tNonBikes table where UnitPrice is less than 50
var
   NonBikeSalesUnder50 = sumx(tNonBikes,
                              calculate(sum([SalesAmount]),'Internet Sales'[UnitPrice]<50))
// get the total of all sales for tNonBikes table
var
   NonBikeAllSales = sumx(tNonBikes,
                              calculate(sum([SalesAmount])))
// divide the first total by the second total
return
   NonBikeSalesUnder50 / NonBikeAllSales
```

As an alternative, you could create intermediate measures for NonBikeSalesUnder50 and NonBikeAllSales and then divide the former by the latter to obtain the final result. That approach would be preferable if you were to require the results of the intermediate measures in other expressions because variables are limited in scope to a single expression. If these results are not required elsewhere, consolidating the logic into one expression and using variables helps you to more easily see the flow of the expression evaluation.

R integration

R is a popular open-source programming language used by data scientists, statisticians, and data analysts for advanced analytics, data exploration, and machine learning. Despite its popularity, the use of R in an enterprise environment can be challenging. Many tools for R operate in a single-threaded, memory-bound desktop environment, which puts constraints on the volume of data that you can analyze. In addition, moving sensitive data from a server environment to the desktop removes it from the security controls built into the database.

SQL Server R Services, the result of Microsoft's acquisition in 2015 of Revolution Analytics, resolves these challenges by integrating a unique R distribution into the SQL Server platform. You can execute R code directly in a SQL Server database when using R Services (In-Database) and reuse the code in another platform, such as Hadoop. In addition, the workload shifts from the desktop to the server and maintains the necessary levels of security for your data. In Enterprise Edition, R Services performs multithreaded, multicore, and parallelized multiprocessor computations at high speed. Using R Services, you can build intelligent, predictive applications that you can easily deploy to production.

Installing and configuring R Services

To use SQL Server R Services, you must install a collection of components to prepare a SQL Server instance to support the R distribution. In addition, each client workstation requires an installation of the R distribution and libraries specific to R Services. In this section, we describe how to prepare your environment to use R Services.

Server configuration

R Services is available in the Standard, Developer, and Enterprise editions of SQL Server 2016 or in Express Edition with Advanced Services. Only the Enterprise edition supports execution of R packages in a high-performance, parallel architecture. In the server environment, you install one of the following components from the SQL Server installation media:

- **R Services (In-Database)** A database-engine feature that configures the database service to use R jobs and installs extensions to support external scripts and processes. It also downloads Microsoft R Open (MRO), an open-source R distribution. This feature requires you to have a default or named instance of SQL Server 2016.
- **R Services (Standalone)** A standalone component that does not require a database-engine instance and is available only in the Enterprise edition of SQL Server 2016. It includes enhanced R packages and connectivity tools from Revolution Analytics and open-source R tools and base packages. Selection of this component also downloads and installs MRO.

After installation of R Services (In-Database), you must execute the following command to enable R Services and then restart the SQL Server service:

```
exec sp_configure 'external scripts enabled', 1
reconfigure with override
```

As part of the installation of R Services (In-Database), 20 new Windows user accounts are added to the server as members of a new Windows group, SQLRUserGroup. This group is used to run tasks using the security token of the new SQL Server Trusted Launchpad service. Using implied authentication, SQL Server activates one of these accounts when you send an R script from a remote client, maps it to your identity, and runs the script on your behalf. If you need to run R scripts from a remote client using Windows authentication instead of a SQL login, you must grant permissions to these accounts to log on to SQL Server on your behalf. To do this, perform the following steps:

1. In Object Explorer in SQL Server Management Studio (SSMS), expand Security, right-click Logins, and select New Login.
2. Click Search in the Login – New dialog box.
3. Click Object Types in the Select User, Service Account, or Group dialog box.
4. Select the Groups check box, clear all other check boxes, and then click OK.
5. In the Enter The Object Name To Select box, type SQLRUserGroup, and then click Check Names.
6. Select the SQLRUserGroup associated with the SQL Server instance, and then click OK twice.

For each user that needs to run R scripts on a SQL Server with R Services (In-Database) installed, you must grant the following permissions in each database in which the user needs to use R:

```
USE <database>
GO
GRANT EXECUTE ANY EXTERNAL SCRIPT TO <user>
```

Note The default memory-allocation settings might allow SQL Server to consume most of the available physical memory without leaving adequate memory for R Services (In-Database). Consider changing the maximum memory for SQL Server to 80 percent of available physical memory. Refer to "Server Memory Server Configuration Options" at https://msdn.microsoft.com/en-us/library/ms178067.aspx for more information about configuring server memory.

Client workstation

To prepare a client workstation to work with R Services, install the following components:

- **R Services (Standalone)** This component adds the R packages and connectivity tools necessary to develop R solutions and downloads and installs MRO.
- **An R integrated development environment (IDE)** You can use any R IDE that you prefer.

Getting started with R Services

Although you execute your R code and run computations on SQL Server, you develop and test by using an R IDE of your choice. In this section, we describe how to prepare your data for exploration with R functions, how to build and use a predictive model, and how to test the accuracy of your model.

> **Note** The examples in this chapter are derived from "Data Science End-to-End Walkthrough," available at https://msdn.microsoft.com/en-US/library/mt612857.aspx, which includes additional topics about working with R and provides a PowerShell script you can download and use to prepare the data set used in the examples. This data set contains public data about New York City taxi fares, passenger counts, pickup and drop-off locations, and whether a tip was given.
>
> You can also learn more about working with R Services in "Data Science Deep Dive: Using the RevoScaleR Packages" at https://msdn.microsoft.com/en-US/library/mt637368.aspx.

Compute context

Before you can execute R on your data, you must use the *RxSetComputeContext* function in the R IDE to set the compute context for functions in the RevoScaleR package in RRE to run on SQL Server. Although you can use a single line of code to run this command, you can assign values to variables in separate lines of code and then use the variables as arguments to the function, as shown in Example 6-3.

Example 6-3: Setting compute context to SQL Server

```
connStr <- "Driver=SQL Server; Server=<srv>; Database=NYCTaxi_Sample; Uid=<login>;
                  Pwd=<password>"
sqlShareDir <- paste("C:\\AllShare\\",Sys.getenv("USERNAME"),sep="")
sqlWait <- TRUE
sqlConsoleOutput <- FALSE
cc <- RxInSqlServer(connectionString = connStr, shareDir = sqlShareDir,
                    wait = sqlWait, consoleOutput = sqlConsoleOutput)
rxSetComputeContext(cc)
```

Creating variables and assigning values is simple to do in R. As shown in Example 6-3, you define a name for the variable and then use the assignment operator (<-) followed by the value to assign. The value can be a string, a Boolean value, or an array, to name only a few object types.

In Example 6-3, several variables store values for use as arguments in the *RxInSqlServer* function. This function is responsible for creating the connection to a SQL Server database and sharing objects between the server context and your local computer context. In this example, it takes the following arguments:

- **connectionString** An ODBC connection string for SQL Server. At the time of this writing, you must use a SQL login in the connection string.
- **shareDir** A temporary directory in which to store R objects shared between the local compute context and the server compute context.

- **wait** A Boolean value to control whether the job will be blocking or nonblocking. Use TRUE for blocking, which prevents you from running other R code until the job completes. Use FALSE for nonblocking, which allows you to run other R code while the job continues to execute.
- **consoleOutput** A Boolean value that controls whether the output of R execution on the SQL Server displays locally in the R console.

Another variable stores the result of the *RxInSQLServer* function and is passed as an argument to the *rxSetComputeContext* function. Now your subsequent RevoScaleR functions run on the server instance.

> **Note** The RevoScaleR package enables the scalable, high-performance, multicore analytic functions. In this chapter, we explore several functions in this package. Setting the compute context affects only the RevoScaleR functions. Open-source R functions continue to execute locally.

> **Important** At the time of this writing, the RevoScaleR package requires a SQL login with the necessary permissions to create tables and read data in a database.

Data source

To execute R commands against data, you define a *data source*. A data source is a subset of data from your database and can be a table, a view, or a SQL query. By creating a data source, you create only a reference to a result set. Data never leaves the database. Example 6-4 shows how to create a data source object in the R IDE by first assigning a T-SQL query string to a variable, passing the variable to the *RxSqlServerData* function, and storing the data source reference in another variable.

Example 6-4: Creating a data source

```
sampleDataQuery <- "select top 1000 tipped, fare_amount, passenger_count, trip_time_in_secs,
     trip_distance, pickup_datetime, dropoff_datetime, pickup_longitude, pickup_latitude,
     dropoff_longitude, dropoff_latitude from nyctaxi_sample"
inDataSource <- RxSqlServerData(sqlQuery = sampleDataQuery,  connectionString = connStr,
     colClasses = c(pickup_longitude = "numeric", pickup_latitude = "numeric",
          dropoff_longitude = "numeric", dropoff_latitude = "numeric"),
     stringsAsFactors=TRUE, rowsPerRead=500)
```

In this example, the *RxSqlServerData* function takes the following arguments:

- **sqlQuery** A string representing a valid SQL query.
- **connectionString** An ODBC connection string for SQL Server. At the time of this writing, you must use a SQL login in the connection string.
- **colClasses** A *character vector* that maps the column types between SQL Server and R. For the purposes of this section, a character vector is a string. In this case, the string must contain the names of columns in the query paired with one of the following allowable column types: logical, integer, float32, numeric, character, factor, int16, uint16, or date.
- **rowsPerRead** Number of rows read into a chunk. R Services processes chunks of data and aggregates the results. Use this argument to control the chunk size to manage memory usage. If this value is too high, processing can slow as the result of inadequate memory resources, although a value that is too low might also adversely affect processing.

Note You can replace the sqlQuery argument with the table argument if you prefer to reference an entire table. You cannot use both arguments together.

Tip The sample data in this section does not contain categorical data. When your data contains categories, such as age groups or geographic regions, you should consider including the stringsAsFactors argument with the *RxSqlServerData* function. This argument is a Boolean value that controls whether to convert strings to factors. A factor is an R object type that is used in statistical functions. Functions such as *rxSummary* return more complete results for factors as compared to strings.

Data exploration

After creating a data source, you can use statistical functions or create plots and graphic objects with which to explore your data in the R IDE. A good starting point is the *rxGetVarInfo* function to display basic information about the structure of your data source. To do this, use the following code in your R IDE:

```
rxGetVarInfo(data = inDataSource)
```

Executing this code returns the results shown in Figure 6-15. The *rxGetVarInfo* function returns metadata about the columns of your data, which are called *variables* when working in R. The metadata includes the name and data type for each variable.

```
> rxGetVarInfo(data = inDataSource)
Var 1: tipped, Type: integer
Var 2: fare_amount, Type: numeric
Var 3: passenger_count, Type: integer
Var 4: trip_time_in_secs, Type: numeric, Storage: int64
Var 5: trip_distance, Type: numeric
Var 6: pickup_datetime, Type: character
Var 7: dropoff_datetime, Type: character
Var 8: pickup_longitude, Type: numeric
Var 9: pickup_latitude, Type: numeric
Var 10: dropoff_longitude, Type: numeric
Var 11: dropoff_latitude, Type: numeric
```

Figure 6-15: Executing the *rxGetVarInfo* function in the R console.

Another common function to use for becoming familiar with your data is *rxSummary*. For a basic statistical summary of your data, as shown in Figure 6-16, use the following code:

```
rxSummary(~., data = inDataSource)
```

```
> rxSummary(~., data = inDataSource)
Call:
rxSummary(formula = ~., data = inDataSource)

Summary Statistics Results for: ~.
Data: inDataSource (RxSqlServerData Data Source)
Number of valid observations: 1000

 Name              Mean      StdDev      Min       Max       ValidObs MissingObs
 tipped              0.00000  0.00000000   0.00000   0.00000 1000     0
 fare_amount         5.00000  0.00000000   5.00000   5.00000 1000     0
 passenger_count     1.00000  0.00000000   1.00000   1.00000 1000     0
 trip_time_in_secs 254.53300 28.36595739 178.00000 339.00000 1000     0
 trip_distance       0.70000  0.00000000   0.70000   0.70000 1000     0
 pickup_longitude  -73.97776  0.01787219 -74.03557 -73.86389 1000     0
 pickup_latitude    40.75737  0.02091854  40.66460  40.86837 1000     0
 dropoff_longitude -73.97793  0.01795312 -74.03428 -73.86764 1000     0
 dropoff_latitude   40.75796  0.02081026  40.67056  40.87299 1000     0
```

Figure 6-16: Executing the *rxSummary* function in the R console.

You can also use graphical objects such as a histogram to explore your data. Figure 6-17 shows the results of the *rxHistogram* function, which plots the count of observations in your data for each distinct value of *fare_amount* under 50 (to ignore outliers) by using the following code:

```
rxHistogram(~(fare_amount), data = inDataSource, title = "Fare Amounts Under $50", endVal=50)
```

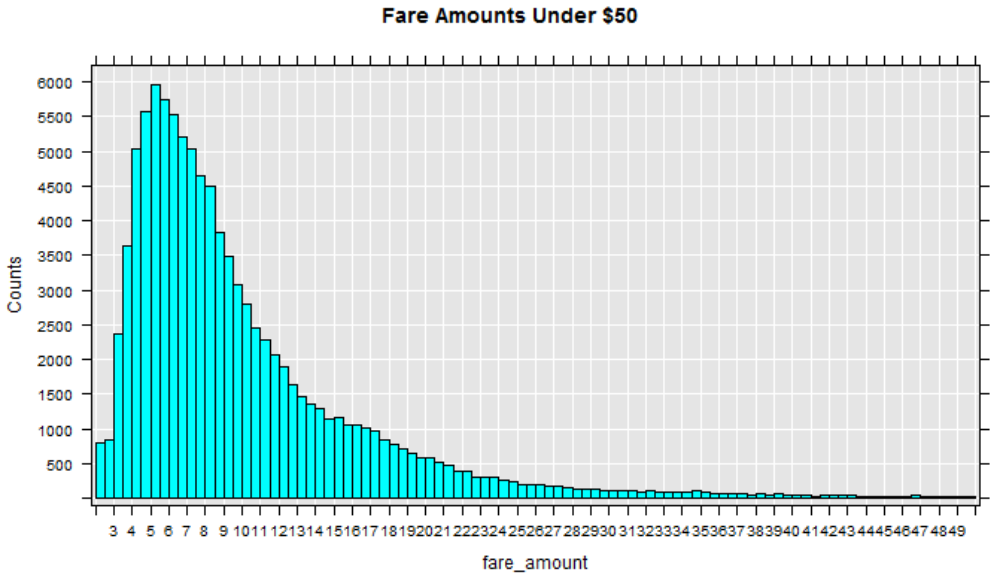


Figure 6-17: Viewing the plot created by executing the *rxHistogram* function.

Note To achieve the results shown in Figure 6-17, the code shown in Example 6-5 was modified to select the top 100,000 rows from the table and then executed again prior to executing the *rxHistogram* function. You can further fine-tune the appearance of the histogram by adding arguments to apply formatting to the axes and configure other style settings. For more information, refer to the function documentation at http://www.rdocumentation.org/packages/RevoScaleR/functions/rxHistogram.

Spatial data can be plotted on a map as another option for exploring data. Because a common security practice is to prevent SQL Server from accessing the Internet, you cannot perform the complete operation on the server. Instead, you use the local context to make a geocoding call to Google Maps to obtain a graphical layer for the map and send it to the server context to get the plot points for individual locations. To start the process, create a custom function to get the plot points, as shown in Example 6-5.

Example 6-5: Creating a custom function to plot spatial data

```
mapPlot <- function(inDataSource, googMap){
    library(ggmap)
    library(mapproj)
    ds <- rxImport(inDataSource)
    p <- ggmap(googMap)+
         geom_point(aes(x = pickup_longitude, y =pickup_latitude ), data=ds, alpha =.5,
         color="darkred", size = 1.5)
    return(list(myplot=p))
}
```

In this example, you use the *function* function to define a custom function and supply an argument list in parentheses. The arguments in this case are *inDataSource*, which is the *RxSqlServerData* object

created in an earlier example, and *googMap*, which you create in a later step. R allows you to reference this argument in the definition without testing for its existence because you are not yet attempting to execute the function.

The body of the function appears between the two braces. The first two lines use the *library* function to load the ggmap[1] and mapproj packages to ensure that the functions they provide are available and ready to use on your server. When you install MRO, a core set of packages and libraries is available for immediate use. A package is a collection of objects that can include code, data, or documentation that you use to extend base R, whereas a library is a directory containing packages. There are thousands of add-on packages contributed by the open-source community that you can download and use to build your analytical application.

Next, the *rxImport* function loads your data into server memory as a data frame called *ds*. This step prepares the data for use in open-source R functions because these functions cannot run in-database.

The variable *p* is a plot object consisting of two layers. The *ggmap* function produces a map from the object passed in as an argument, which we explain later in this section. The + operator adds another layer to the plot object. You can add as many layers as necessary by using this technique. The *geom_point* function creates a scatterplot of *pickup_longitude* on the x-axis and *pickup_latitude* on the y-axis from the in-memory data frame. The *alpha*, *color*, and *size* arguments set point transparency, point color, and point size, respectively. The final line of code assigns a tag, *myplot*, to the *p* variable and converts the object to a list data type, which is the return value of the custom function.

Next you execute the geocoding call to get the map, call the custom function to send the map and combine it with your plot points, and render the results in the local context, as shown in Example 6-6.

Example 6-6: Creating a custom function to plot spatial data

```
library(ggmap)
library(mapproj)
gc <- geocode("Manhattan", source = "google")
googMap <- get_googlemap(center = as.numeric(gc), zoom = 12, maptype = 'roadmap',
     color = 'color')
myplots <- rxExec(mapPlot, inDataSource, googMap, timesToRun = 1)
plot(myplots[[1]][["myplot"]]);
```

The first two lines are calls to load *ggmap* and *mapproj* again, but this time in the local context. Then the *geocode* function takes a street address or a place name as its first argument and sets the source to google as the second argument. The other possible source is *dsk,* which is the Data Science Toolkit (http://datasciencetoolkit.org), but this source tends to return results more slowly or timeout.

Next, the *get_googlemap* function uses the latitude and longitude coordinates stored in the *gc* variable to set the center of the map that it downloads from Google. The *zoom* argument takes an integer value ranging from 3 (for continent) to 21 (for building) to indicate the level of detail to display in the map. You can set the *color* argument to either color or black and white. The resulting map is stored in the *googMap* variable that you pass to the *mapPlot* function.

In the next line, the *rxExec* function executes the function specified as the first argument (the custom function *mapPlot*, in this case) on SQL Server, using the arguments passed as a list as subsequent arguments, until it encounters *rxExec* arguments such as *timesToRun*. The variable *myplots* stores the results of execution, which is a list containing one item called *myplot*.

---

[1] D. Kahle and H. Wickham, "ggmap: Spatial Visualization with ggplot2," *The R Journal* no. 5(1): 144–61, http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf.

Last, the *plot* function takes the first item in the *myplots* list and renders the object on the local computer. The result is a static map of Manhattan with multiple points representing pickup locations overlayed on the map, as shown in Figure 6-18.

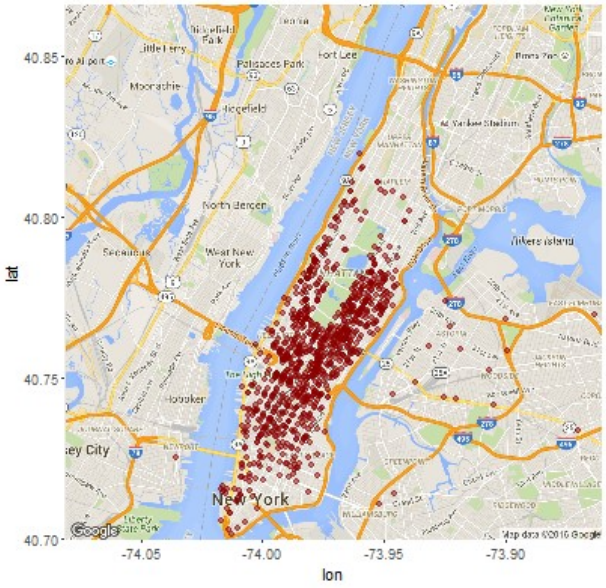


Figure 6-18: Viewing the map plot created by executing the custom *mapPlot* function.

> **Note** Before executing this code, the data source query was adjusted to return only 1,000 rows. It is important to note that the map is created locally and passed by a function that runs in the server context. The data is serialized back to the local computer where you view it in the Plot window in the R IDE.

Data transformation

Besides exploring data with R, you can also use R to transform data to enhance it for use in predictive modeling. However, when working with large data volumes, R transformations might not perform as optimally as similar transformations done by using a T-SQL function. You are not limited to these options, though. You might prefer to use T-SQL scripts or Integration Services to preprocess the data before using the data with R Services.

> **Note** You use the *rxDataStep* function in conjunction with custom functions to perform transformations by using the RevoScaleR package on the server. You can learn more about this function at http://www.rdocumentation.org/packages/RevoScaleR/functions/rxDataStep.

The taxi data currently includes coordinates for pickup and drop-off locations, which you can use to compute the linear distance. The database also includes a custom function, *fnCalculateDistance,* to use for this computation. To set up a new data source using a random sample that includes the computed distance, execute the code shown in Example 6-7 in your R IDE.

Example 6-7: Adding a data source with a feature computed in T-SQL

```
modelQuery = "SELECT tipped, fare_amount, passenger_count, trip_time_in_secs,trip_distance,
    pickup_datetime, dropoff_datetime,
    dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,  dropoff_latitude,
         dropoff_longitude) as direct_distance,
    pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
```

```
    FROM nyctaxi_sample
    tablesample (1 percent) repeatable (98052)"
modelDataSource = RxSqlServerData(sqlQuery = modelQuery,
                  colClasses = c(pickup_longitude = "numeric", pickup_latitude = "numeric",
                                 dropoff_longitude = "numeric", dropoff_latitude = "numeric",
                                 passenger_count  = "numeric", trip_distance  = "numeric",
                                 trip_time_in_secs  = "numeric", direct_distance  = "numeric"),
                  connectionString = connStr)
```

Predictive model creation

After preparing your data, you can create a model by using any of the functions available in the RevoScaleR package. The *RxLogit* function is a good choice for classification problems. It uses logistic regression to estimate the probability of a variable with two possible values. In the sample code shown in Example 6-8, the goal is to predict whether a tip was given. The *summary* function provides statistical information about the resulting model, as shown in Figure 6-19.

Example 6-8: Creating a logistic regression model

```
logitObj <- rxLogit(tipped ~ passenger_count + trip_distance + trip_time_in_secs +
                      direct_distance, data = modelDataSource)
summary(logitObj)
```

```
> summary(logitObj)
Call:
rxLogit(formula = tipped ~ passenger_count + trip_distance +
    trip_time_in_secs + direct_distance, data = modelDataSource)

Logistic Regression Results for: tipped ~ passenger_count + trip_distance +
    trip_time_in_secs + direct_distance
Data: modelDataSource (RxSqlServerData Data Source)
Dependent variable(s): tipped
Total independent variables: 5
Number of valid observations: 15635
-2*LogLikelihood: 21551.2358 (Residual deviance on 15630 degrees of freedom)

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)     -3.537e-02  3.372e-02  -1.049   0.2943
passenger_count -2.466e-02  1.164e-02  -2.119   0.0341 *
trip_distance    7.127e-03  7.589e-03   0.939   0.3477
trip_time_in_secs  2.169e-04  4.468e-05   4.854 1.21e-06 ***
direct_distance -9.206e-04  1.310e-03  -0.703   0.4821
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Condition number of final variance-covariance matrix: 8.4711
Number of iterations: 5
```

Figure 6-19: Viewing the summary of a logistic regression predictive model.

Note To learn more about the *rxLogit* function, see http://www.rdocumentation.org/packages/RevoScaleR/functions/rxLogit.

Model usage

After you build a predictive model, you can apply it to a data source to predict the dependent variable value, score the prediction, and store the results in a table by using the *rxPredict* function. You can see the code necessary to perform these steps in Example 6-9. In this example, you define a table without a schema in the *rxSqlServerData* function. The output from the *rxPredict* function returns the schema and creates the table, which means the SQL login associated with the *rxSqlServerData* function (as defined in the connection string) must have permissions to create a table, or the execution of the code fails.

Example 6-9: Predicting values

```
scoredOutput <- RxSqlServerData(
  connectionString = connStr,
  table = "taxiScoreOutput")
rxPredict(modelObject = logitObj, data = modelDataSource, outData = scoredOutput,
          predVarNames = "Score", type = "response", writeModelVars = TRUE, overwrite = TRUE)
```

Figure 6-20 shows the results of the output stored in the table. A value below 0.5 in the Score column indicates a tip is not likely.

	Score	tipped	passenger_count	trip_distance	trip_time_in_secs	direct_distance
1	0.499852466228027	0	1	0.7	254	0.6947578089
2	0.498152357793778	0	1	0.7	222	0.5427964547
3	0.499370343313086	0	1	0.7	244	0.4336593358
4	0.499616332756451	0	1	0.7	249	0.5427964547
5	0.499302131155759	0	1	0.7	243	0.4944477178
6	0.498454011207273	0	1	0.7	228	0.6456525887
7	0.501784526502719	0	1	0.7	289	0.5456760317
8	0.499508555912361	0	1	0.7	247	0.5399015196
9	0.500875565644517	0	1	0.7	273	0.7256508654
10	0.498202653378384	0	1	0.7	223	0.5598517959

Figure 6-20: Viewing the output of the *rxPredict* function in the taxiScoreOutput table in SSMS.

Note For simplicity in this example, the data used to train the model is also used to test the model. Typically, you partition the data, using one set to train the model and one set to test the model.

To learn more about the *rxPredict* function, see http://www.rdocumentation.org/packages/RevoScaleR/functions/rxPredict.

Model accuracy

After you create a model, you can use R functions to test its accuracy. ROCR is a useful package for testing the performance of classification models. Example 6-10 shows the code to install and load this library by using the *install.packages* and *library* functions.

Example 6-10: Testing a model's accuracy

```
if (!('ROCR' %in% rownames(installed.packages()))){
  install.packages('ROCR')
}
library(ROCR)
scoredOutput <- rxImport(scoredOutput)
pr <- prediction(scoredOutput$Score, scoredOutput$tipped)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```

To use the functions in ROCR, you must bring data from the server into your local environment by using the *rxImport* function. Next, you need to load the results of your predictions into a prediction object by using the *prediction* function, which takes the score from your model as its first argument and the predicted value as the second argument. Notice that the format of these arguments uses the name of the data source first, then a $ symbol, which is followed by the data source column.

The *performance* function takes the prediction object as the first argument and then you specify measures to return. In this case, *tpr* and *fpr* represent true positive rate and false positive rate and are

only two of many different types of performance metrics that the *performance* function returns. You can store the performance results in a variable that you can then plot, as shown in Figure 6-21.

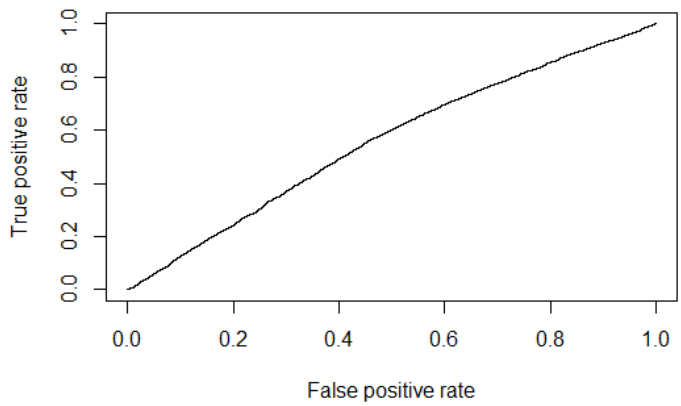


Figure 6-21: Viewing the plot of the prediction model performance.

Note You can see the other performance metrics accessible with the *performance* function at http://www.rdocumentation.org/packages/ROCR/functions/performance.

Using an R Model in SQL Server

After creating an R model, you can deploy it to SQL Server for use in applications and other tools. You can then invoke the model by calling the *sp_execute_external_script* stored procedure. You can call a model to score data in batch mode or to score data for an individual case. The sample database includes two stored procedures that allow you to perform each of these tasks.

Model deployment

This process requires you to serialize your model as a hexadecimal string that you send to the server and store in a varbinary(max) column in a database, as shown in Example 6-11. The *serialize* function produces the string, and the *paste* function ensures that the result is a single string. Then the RODBC package is installed to use the *odbcDriverConnect* function to open a connection to SQL Server. Next, the *paste* function concatenates the serialized string with a call to the *PersistModel* stored procedure to produce a query string that is passed into the *sqlQuery* function and executed. The *PersistModel* stored procedure is a custom stored procedure in the sample database that inserts a record into the *nyc_taxi_models* table.

Example 6-11: Deploying a model to SQL Server

```
modelbin <- serialize(logitObj, NULL)
modelbinstr=paste(modelbin, collapse="")
if (!('RODBC' %in% rownames(installed.packages()))){
  install.packages('RODBC')
}
library(RODBC)
conn <- odbcDriverConnect(connStr )
q<-paste("EXEC PersistModel @m='", modelbinstr,"'", sep="")
sqlQuery (conn, q)
```

Batch mode invocation of a model

The stored procedure in the sample database that invokes the model in batch mode is shown in Example 6-12. This stored procedure, *PredictTipBatchMode*, retrieves the stored model and stores it in a variable that becomes a parameter for the *sp_execute_external_script* stored procedure. You pass the data to score as a query string into *PredictTipBatchMode*. It becomes a data frame called *InputDataSet* used in the *rxPredict* function that *sp_execute_external_script* executes. The output of this stored procedure is a set of rows containing a score for each row in the input.

Example 6-12: Creating a stored procedure to invoke a model in batch mode

```
CREATE PROCEDURE [dbo].[PredictTipBatchMode] @inquery nvarchar(max)
AS
BEGIN
  DECLARE @lmodel2 varbinary(max) = (SELECT TOP 1 model
       FROM nyc_taxi_models);
  EXEC sp_execute_external_script @language = N'R',
                                  @script = N'
mod <- unserialize(as.raw(model));
print(summary(mod))
OutputDataSet<-rxPredict(modelObject = mod, data = InputDataSet, outData = NULL,
          predVarNames = "Score", type = "response", writeModelVars = FALSE, overwrite = TRUE);
str(OutputDataSet)
print(OutputDataSet)',
                                  @input_data_1 = @inquery,
                                  @params = N'@model varbinary(max)',
                                  @model = @lmodel2
  WITH RESULT SETS ((Score float));
END
```

Individual scoring mode invocation of a model

Rather than score a set of rows in batch mode, you can score a single case. Example 6-13 shows the *PredictTipSingleMode* stored procedure in the sample database, which illustrates this approach. It is similar to the previous example, except the *PredictTipSingleMode* stored procedure defines input parameters for each of the variables in your training data set. These parameters are then sent to a table-valued helper function in the sample database that computes the linear distance, and the result becomes the *InputDataSet* data frame. The output is a single value that represents the probability of a tip.

Example 6-13: Creating a stored procedure to invoke a model in single mode

```
CREATE PROCEDURE [dbo].[PredictTipSingleMode] @passenger_count int = 0,
@trip_distance float = 0,
@trip_time_in_secs int = 0,
@pickup_latitude float = 0,
@pickup_longitude float = 0,
@dropoff_latitude float = 0,
@dropoff_longitude float = 0
AS
BEGIN
  DECLARE @inquery nvarchar(max) = N'
  SELECT * FROM [dbo].[fnEngineerFeatures]( @passenger_count, @trip_distance, @trip_time_in_secs,
     @pickup_latitude, @pickup_longitude, @dropoff_latitude, @dropoff_longitude)'
  DECLARE @lmodel2 varbinary(max) = (SELECT TOP 1 model  FROM nyc_taxi_models);
  EXEC sp_execute_external_script @language = N'R',
                                  @script = N'
```

```
mod <- unserialize(as.raw(model));
print(summary(mod))
OutputDataSet<-rxPredict(modelObject = mod, data = InputDataSet, outData = NULL,
          predVarNames = "Score", type = "response", writeModelVars = FALSE, overwrite = TRUE);
str(OutputDataSet)
print(OutputDataSet)',
                                  @input_data_1 = @inquery,
                                  @params = N'@model varbinary(max), @passenger_count int,
@trip_distance float, @trip_time_in_secs int, @pickup_latitude float, @pickup_longitude float,
@dropoff_latitude float, @dropoff_longitude float',
                                  @model = @lmodel2,
                                  @passenger_count =@passenger_count ,
                                  @trip_distance=@trip_distance,
                                  @trip_time_in_secs=@trip_time_in_secs,
                                  @pickup_latitude=@pickup_latitude,
                                  @pickup_longitude=@pickup_longitude,
                                  @dropoff_latitude=@dropoff_latitude,
                                  @dropoff_longitude=@dropoff_longitude
  WITH RESULT SETS ((Score float));
END
```

CHAPTER 7

Better reporting

For report developers, Reporting Services in SQL Server 2016 has a more modern development environment, two new data visualizations, and improved parameter layout options. In addition, it includes a new development environment to support mobile reports. Users also benefit from a new web portal that supports modern web browsers and mobile access to reports. In this chapter, we'll explore these new features in detail.

Report content types

This release of Reporting Services includes both enhanced and new report content types:

- **Paginated reports** Paginated reports are the traditional content type for which Reporting Services is especially well suited. You use this content type when you need precise control over the layout, appearance, and behavior of each element in your report. Users can view a paginated report online, export it to another format, or receive it on a scheduled basis by subscribing to the report. A paginated report can consist of a single page or hundreds of pages, based on the data set associated with the report. The need for this type of report continues to persist in most organizations, as well as the other report content types that are now available in the Microsoft reporting platform.
- **Mobile reports** In early 2015, Microsoft acquired Datazen Software to make it easier to deploy reports to mobile devices, regardless of operating system and form factor. This content type is best when you need touch-responsive and easy-to-read reports that are displayed on smaller screens, communicate key metrics effectively at a glance, and support drill-through to view supporting details. In SQL Server 2016, users can view both paginated and mobile reports through the web portal interface of the on-premises report server.
- **Key performance indicators (KPIs)** A KPI is a simple type of report content that you can add to the report server to display metrics and trends at a glance. This content type uses colors to indicate progress toward a goal and an optional visualization to show how values trend over time.

Paginated report development enhancements

In this release of Reporting Services, the authoring tools for paginated reports work much like they did in previous releases, but with some enhancements. The first noticeable change is the overall

appearance of the authoring tools. In addition, these tools have been augmented by the addition of new visualizations and a new interface for working with parameters.

Introducing changes to paginated report authoring tools

As in prior versions of Reporting Services, there are two methods for authoring paginated reports:

- **Report Designer** A full-featured report development environment available as one of the business-intelligence templates installed in the new SQL Server Data Tools for Visual Studio 2015 (SSDT).
- **Report Builder** A standalone application that shares many common features with Report Designer.

Report Designer

Microsoft has released a new updated business-intelligence template in SSDT that you download from http://go.microsoft.com/fwlink/?LinkID=690931. This business-intelligence template includes an updated version of Report Designer that allows you to develop reports for multiple versions of Reporting Services. By default, you can develop reports for SQL Server 2016 Reporting Services or later, as shown in Figure 7-1, but you can change the TargetServerVersion property in the project's properties to target SQL Server 2008, SQL Server 2008 R2, SQL Server 2012, or SQL Server 2014. Report authors may continue to use SQL Server Data Tools for Business Intelligence in Visual Studio 2013 to develop reports for these earlier versions, but the new features specific to SQL Server 2016 that we discuss later in this chapter are not supported.

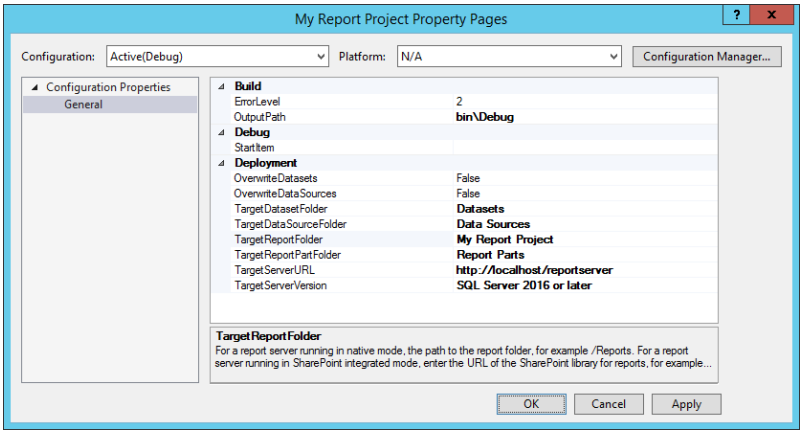


Figure 7-1: A new default value for the TargetServerVersion property in the project's properties.

Report Builder

Report Builder is an alternative report-development tool for power users and report developers who need only to create or edit one report at a time. You can start the ClickOnce version of the Report Builder by clicking the Report Builder button on the web portal toolbar on your report server at http://<servername>/reports. You can also download and install a standalone version of Report Builder from https://www.microsoft.com/en-us/download/details.aspx?id=51935 and then use the Windows Start menu to open it after installation. Previous versions of Report Builder use the light blue Office 2007 appearance, but the most recent version of Report Builder, shown in Figure 7-2, uses the same darker theme that appears in both Office 2016 and the Power BI Desktop application and continues to use a ribbon interface like Office applications.

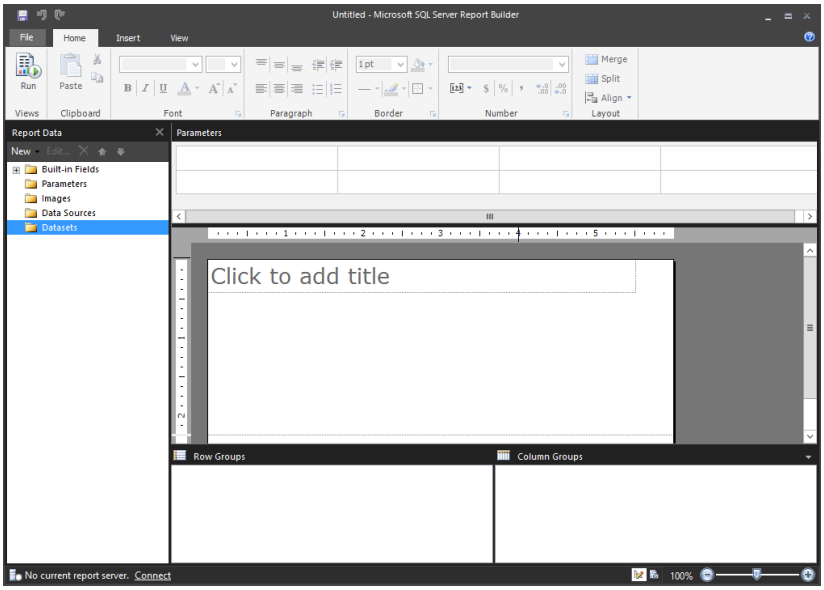


Figure 7-2: New Report Builder interface.

Exploring new data visualizations

All data visualizations included in prior versions of Reporting Services continue to be available, but the SQL Server 2016 version includes two new types of data visualizations:

- **Tree map** A tree map represents hierarchical categories as rectangles with relative sizes.
- **Sunburst** A sunburst chart is a hierarchical representation of data that uses circles for each level.

> **Note** Although the data visualizations from prior versions are still available, they now use a default palette named Pacific. This palette more closely resembles the default colors for data visualizations available in Microsoft Power BI.

Tree map

A tree map is useful to show how parts contribute to a whole. Each rectangle represents the sum of a value and is sized according to the percentage of its value relative to the total of values for all rectangles in the tree map. The rectangles are positioned within the tree map with the largest category in the upper-left corner of the parent rectangle and the smallest category in the lower-right corner. Each rectangle can contain another collection of rectangles that break down its values by another category that represents a lower level in a hierarchy.

As an example, in the tree map shown in Figure 7-3, the first level shows the United States as the largest category, followed by Canada, with the second largest category, and then progressively smaller rectangles are displayed for France, United Kingdom, Germany, and Australia. For each of these country categories, business type is the next lower level in the hierarchy, and rectangles for each distinct business type are displayed using the same pattern of largest to smallest from top left to bottom right within a country's rectangle. In this example, the largest business type in the United States is Value Added Reseller, followed by Warehouse, and then Specialty Bike Shop.

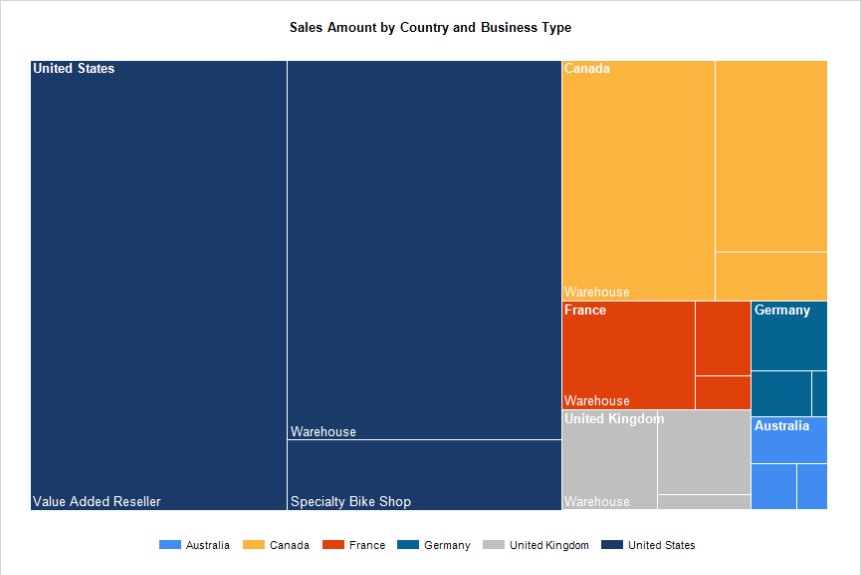


Figure 7-3: Tree map showing sales hierarchically by country and by business type.

To add a tree map to your report, you use the same technique as you do for any chart. Whether using Report Designer or Report Builder, you insert a chart into the report by choosing Chart from the toolbox or ribbon, and then select Tree Map in the Shape collection of chart types in the Select Chart Type dialog box, as shown in Figure 7-4.

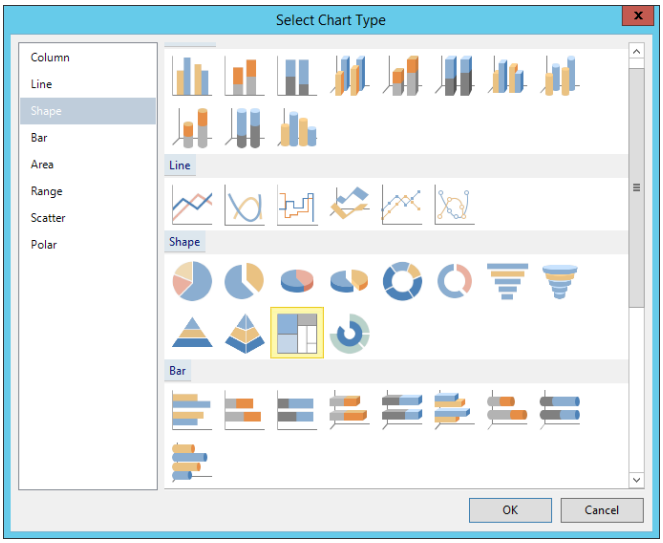


Figure 7-4: Selection of a tree map in the Select Chart Type dialog box.

To configure the chart, click anywhere on its surface to open the Chart Data pane. Then click the button with the plus symbol to add fields to the Values, Category Groups, or Series Groups areas, as shown in Figure 7-5. The value field determines the size of a rectangle for category groups and series groups. Each series group field is associated with a different color and becomes the outermost collection of rectangles. For example, with SalesTerritoryCountry as a series group, each country is identifiable by color in the tree map. Within each country's rectangle, each distinct value within a category group is represented by a separate rectangle. In this case, each country's rectangle contains three rectangles—Specialty Bike Shop, Value Added Reseller, and Warehouse. The proportion of an individual business type's sales amount value relative to a country's total sales determines the size of its rectangle.

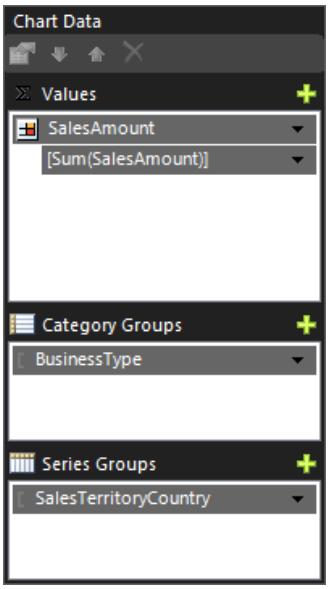


Figure 7-5: Configuring the Chart Data pane for a tree map.

To improve the legibility of a tree map, you should consider making the following changes to specific chart properties:

- **Size** You should increase the size of the chart because the default size, 3 inches wide by 2 inches high, is too small to view the data labels that are enabled by default. Click the chart object, but take care to click an element such as the Chart Title or a series in the chart, and then adjust the Size properties, Width and Height, in the Properties pane.
- **Legend** To maximize the space of the chart area allocated to the tree map, consider moving the legend above or below the chart. To do this, right-click the legend, select Legend Properties, and then select one of the Legend Position options to reposition the legend.
- **Data labels** Even after resizing the chart, you might find that the default 10 point font size used for the labels is too large to display labels in each rectangle or that the black font is difficult to read when the series color is dark. To reduce the size of the font and change its color to improve the visibility of the data labels, click the chart to display the Chart Data pane, click the field in the Values area, and then locate the Labels section in the Properties pane. When you expand this section, you can change font properties such as size and color as needed.

Note The size of rectangles in a tree map might continue to affect the visibility of the data labels even if you reduce the font size to 6 points. If the smaller label text cannot fit within the width of its rectangle, the label is not displayed.

- **Tooltip** One way to compensate for missing data labels in small rectangles, or to add more context to a tree map, is to add a tooltip, as shown in Figure 7-6. To do this, right-click a rectangle in the chart, select Series Properties, click the expression button next to the Tooltip box in the Series Properties dialog box, and type an expression such as this:

```
=Fields!BusinessType.Value + " : " + Format(Sum(Fields!SalesAmount.Value), "C0")
```

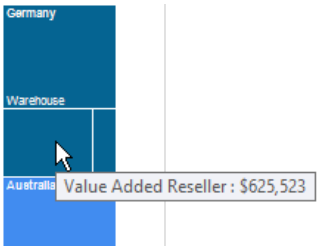


Figure 7-6: Tooltip displayed above a selected rectangle in a tree map.

You can add more than one field to the Category Groups or Series Groups areas of the Chart Data pane. However, the meaning of the chart is easier to discern if you add the second field only to the Series Groups area so that different colors help viewers distinguish values, as shown in Figure 7-7. If you add a second field to the Category Groups area, more rectangles are displayed in the tree map, but it's more difficult to interpret the hierarchical arrangement without extensive customization of the tree map's elements.

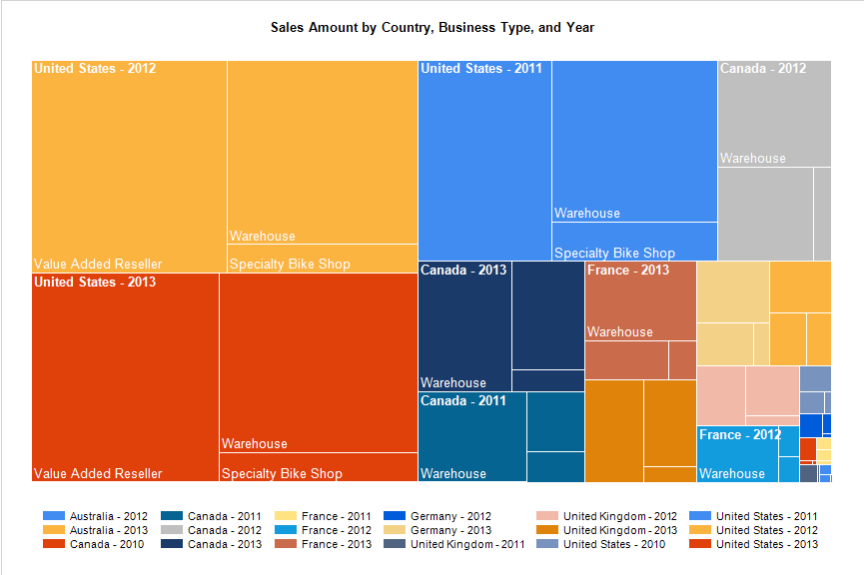


Figure 7-7: Tree map displaying two series groups.

Sunburst

A sunburst chart is a type of visualization that is a hybrid of a pie chart, using slices of a circle to represent the proportional value of a category to the total. However, a sunburst chart includes multiple circles to represent levels of hierarchical data. Color is the highest level of a hierarchy if a series group is added to the chart, but it is not required. If no series group is defined, the innermost circle becomes the highest level of the hierarchy. Each lower level moves farther from the center of the circle, with the outermost circle as the lowest level of detail. Within each type of grouping, color or circle, the slices are arranged in clockwise order, with the largest value appearing first and the smallest value appearing last in the slice.

As an example, in Figure 7-8, color is used to identify sales amount by year across all circles, with the largest color slice starting at the twelve o'clock position in the circle. At a glance, a viewer can easily see the relative contribution of each year to total sales and which year had the greatest number of sales. Next, the inner circle slices each color by country, again sorting the countries from largest to smallest in clockwise order. The outer circle further subdivides the countries by business type. In this example, some of the slices are too small for the labels to be displayed.

Figure 7-8: Example of a sunburst chart.

To produce a sunburst, you insert a chart into the report and select Sunburst from the Shape collection of chart types. Click the chart to open the Chart Data pane and use the button with the plus symbol to add fields to the Values, Category Groups, or Series Groups areas, as shown in Figure 7-9. The value field determines the size of a slice for category groups and series groups. Each series group field is associated with a different color and becomes the first division of the total value into proportional slices, although the inclusion of a series group is optional. Category groups then further subdivide values into slices, with the first category group in the list as the inner circle, and each subsequent category group added to the chart as another outer circle moving from the center.

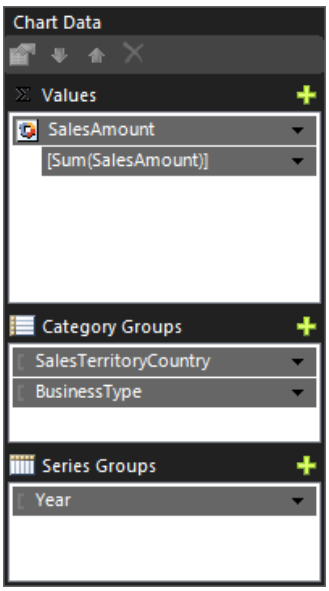


Figure 7-9: Chart Data pane configured for a sunburst chart.

As for a tree map, a sunburst chart's default properties are likely to produce a chart that is difficult to read. Therefore, you should consider modifying the following chart properties:

- **Size** The minimum recommended size for a sunburst chart is 5 inches wide. Click the chart object (but not an element such as the Chart) and then increase the Size properties, Width and Height, in the Properties pane.
- **Legend** More space is allocated to the sunburst chart when you move the legend above or below the chart. To do this, right-click the legend, select Legend Properties, and select one of the Legend Position options to reposition the legend.
- **Data labels** Reducing the label size and changing the font color are likely to improve legibility. To fix these properties, click the chart to display the Chart Data pane, click the field in the Values area, expand the Labels section in the Properties pane, and change the font size and color properties.

> **Note** Some sunburst slices can still be too small for some data labels even if you reduce the font size to 6 points.

- **Tooltip** To help users understand the values in a sunburst chart when data labels are missing from small slices, consider adding a tooltip by right-clicking a slice in the chart, selecting Series Properties, clicking the expression button next to the Tooltip box in the Series Properties dialog box, and then typing an expression such as this:

```
=Fields!BusinessType.Value + " : " + Fields!SalesTerritoryCountry.Value + " : " +
Format(Sum(Fields!SalesAmount.Value), "C0")
```

Managing parameter layout in paginated reports

In previous versions of Reporting Services, there was no option for configuring the layout of parameters unless you designed a custom interface to replace Report Manager for accessing reports. Now in both Report Designer and Report Builder, you can use a Parameters pane to control the relative position of parameters and to organize parameters into groups.

> **Note** In Report Builder, you can change the visibility of the Parameters pane by selecting or clearing the new Parameters check box on the View tab of the ribbon.

The new Parameters pane is a 4x2 grid that displays above the report design surface. To add a report parameter to the grid, you can continue to use the Report Data pane as you have in previous versions of Reporting Services. As an alternative, in the Parameters pane, right-click an empty cell and select Add Parameter, as shown in Figure 7-10, to open the Report Parameter Properties dialog box. Notice that the context menu that appears when you right-click a cell also includes commands to add or remove rows or columns, delete a parameter, or view a selected parameter's properties.

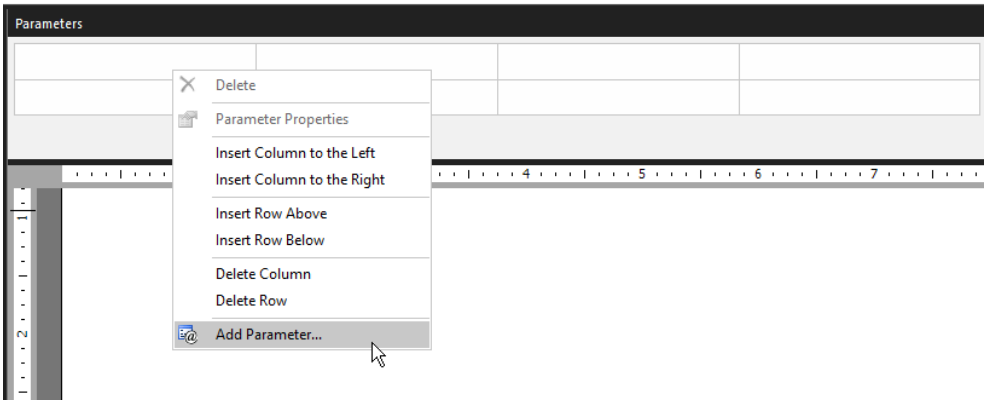


Figure 7-10: Adding a new parameter to a report by using the Parameters pane in Report Builder.

Note When you add a report parameter by using the Parameters pane, the parameter is added automatically to the Report Data pane. You can easily access a parameter's properties by double-clicking it in either location.

After adding a parameter, you can drag it to a new location. Consider using empty rows or columns to create groupings of parameters, as shown in Figure 7-11.

Figure 7-11: Using separate columns to group parameters in the Parameter pane.

Note If you design a report with cascading parameters, the sequence of parameters in the Report Data pane remains important. Cascading parameters are a set of at least two parameters in which a child parameter's available list of values is dependent on the user's selection of another parameter value, the parent parameter. The parent parameter must be displayed above the child parameter in the Report Data pane.

You cannot control the size of an unused parameter column, but the rendered report displays each column with enough separation to clearly distinguish groups, as shown Figure 7-12. You can create more separation between column groups by inserting another empty column in the Parameters pane.

Figure 7-12: Parameter groups in a rendered report.

Mobile report development

Mobile reports display data concisely for use on mobile devices. The acquisition of Datazen by Microsoft brings a suite of tools supporting the development of mobile reports into the Reporting Services platform. To create mobile reports, you use the SQL Server Mobile Report Publisher.

Note You can download the Mobile Report Publisher from https://www.microsoft.com/en-us/download/confirmation.aspx?id=50400 or from the Microsoft Store if you are using Windows 8 or Windows 10. You can also download it from the Reporting Services web portal.

Mobile reports enable you to create data mash-ups from a variety of data sources. You can use the same data sources and shared data sets published to the report server to connect data to mobile report elements such as gauges and charts, among others.

Introducing the mobile report designer

When you open Mobile Report Publisher, you see the Layout tab, which contains an empty design surface with 5 rows and 10 columns, as shown in Figure 7-13. You add various elements to the design surface to compose your report and can use slider controls above the design surface to resize the grid as needed.

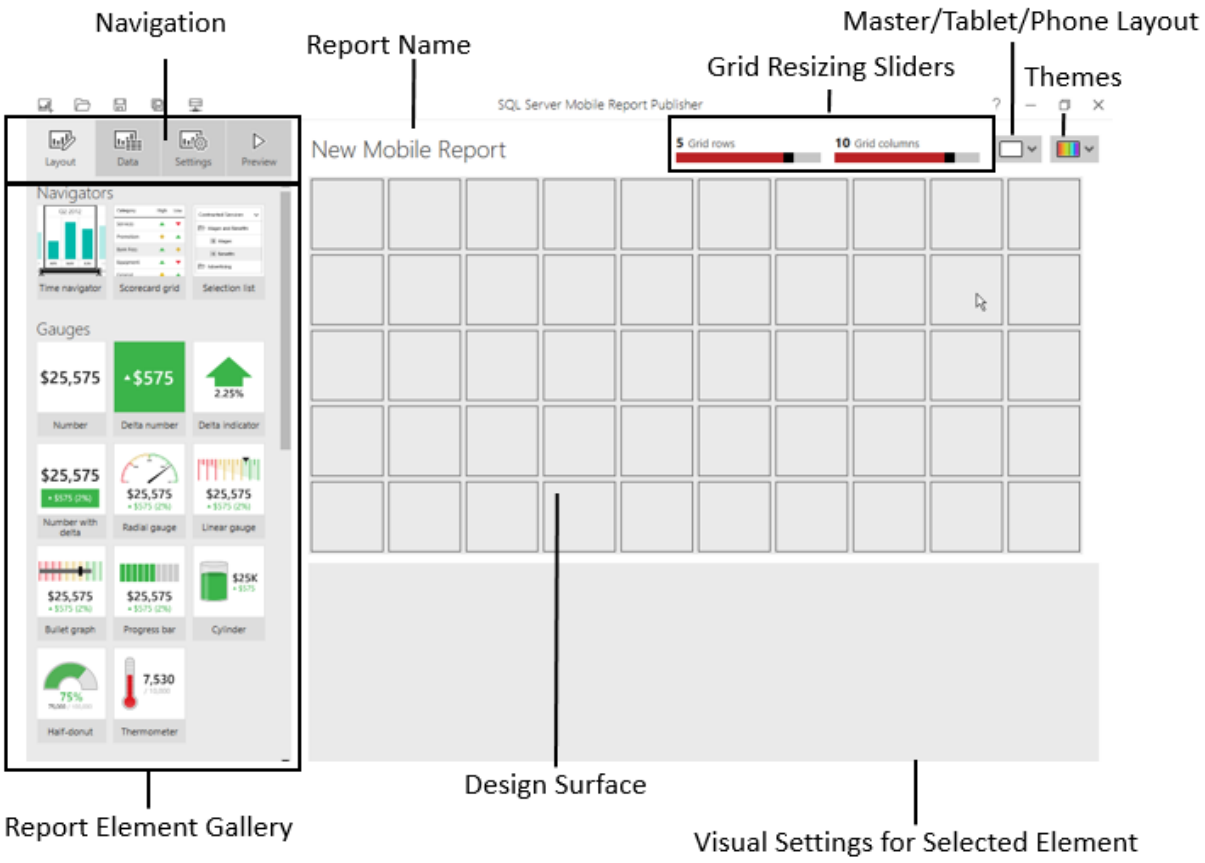


Figure 7-13: The mobile report layout view in Mobile Report Publisher.

In the layout view, you drag an element from the Report Element Gallery to one of the grid cells. Then you can configure the settings for the selected element in the Visual Properties panel that is displayed below the design surface. For example, when you add a time navigator element to the design surface, you can configure time levels, preset time ranges, the visualization type, and other properties, as shown in Figure 7-14.

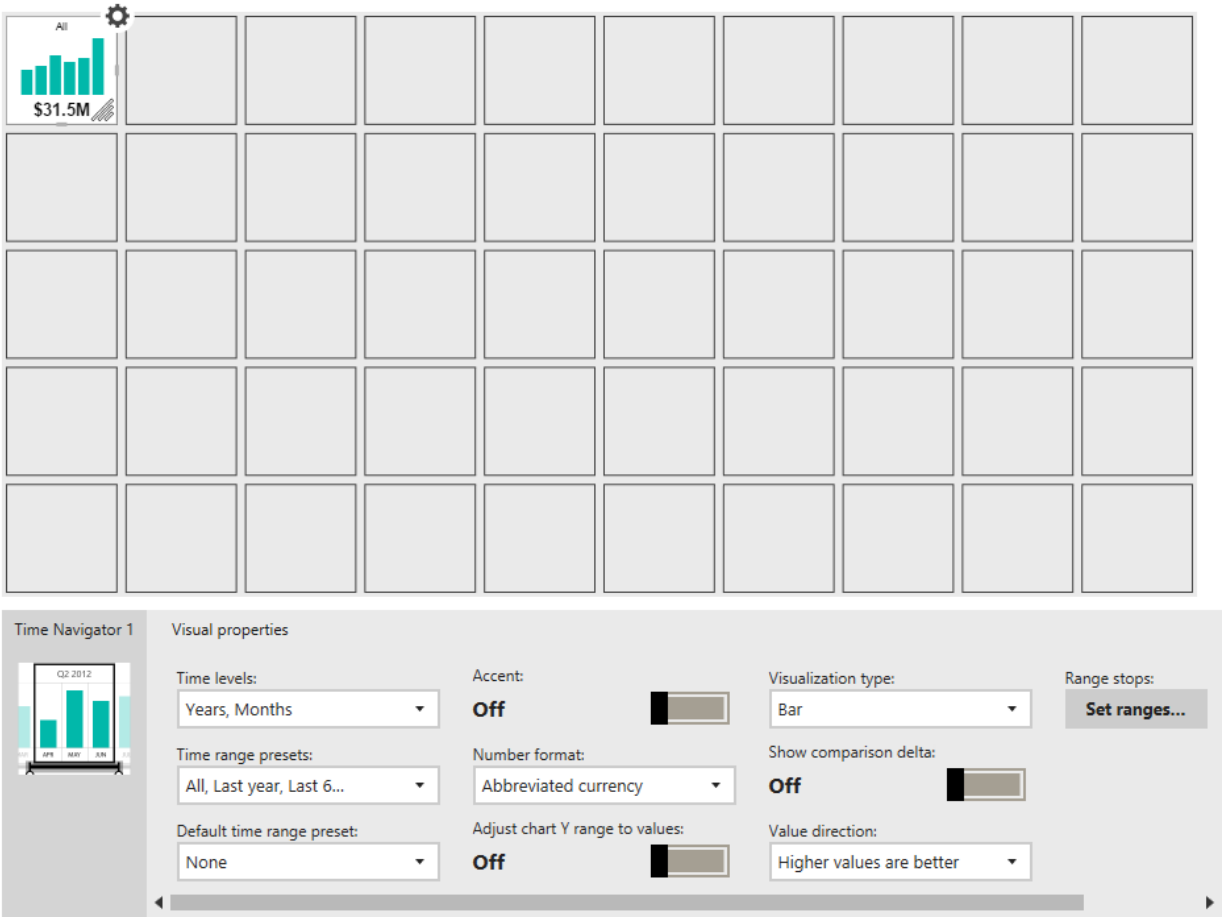


Figure 7-14: Working with visual properties for a mobile report element.

After adding an element to the mobile report design surface, you can drag the element's edge to resize it across multiple grid cells, as shown in Figure 7-15. When you create a mobile report, any element that you add to the designer displays simulated data. This feature allows you to prototype various layouts without needing to first structure your data.

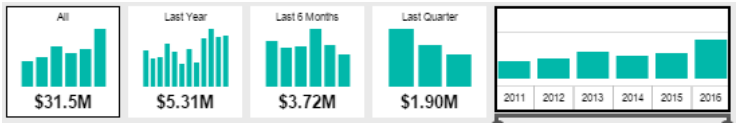


Figure 7-15: Resizing a mobile report element.

Connecting data to mobile report elements

At any time, you can click the Data button at the top left of the screen to switch to the mobile report's data view. In the data view, you can view the simulated data associated with a mobile report element or shared data sets on your report server, as shown in Figure 7-16. On the left, the Control Instances section displays thumbnails for each mobile report element added to the layout. As you select an element in this group, the Data Settings panel below the data grid displays the related settings, such as the data set currently connected to the element. In the data grid, a filter icon is displayed in the column header for filterable columns and a clock icon is displayed for a column with a *datetime* data type.

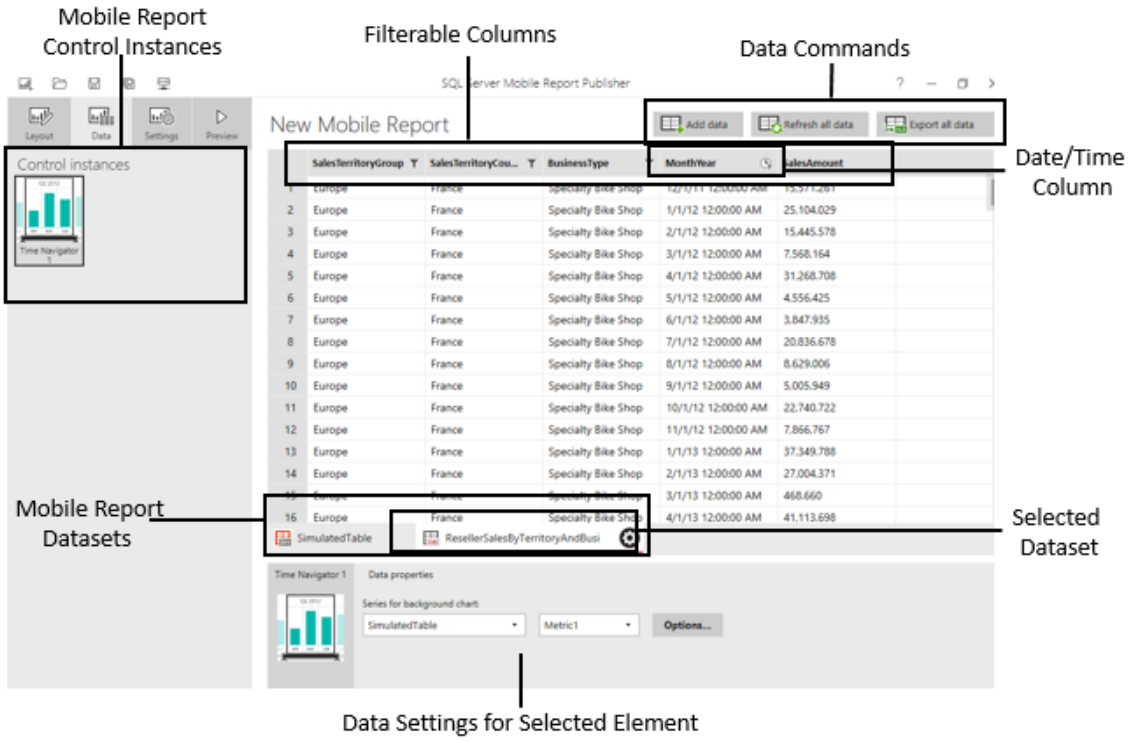


Figure 7-16: The mobile report designer's data view.

Note A column must be filterable to use it with mobile report elements that support filtering. Furthermore, a column must be identified as a *datetime* data type to use it in the time navigator or in any mobile report element that supports explicit time aggregation. There is no option to configure these properties explicitly in Mobile Report Publisher. The import process applies these properties automatically based on each column's data type. If these properties are not correct in the data set, you must correct the data types of the affected columns in the source data set or Excel file.

Working with simulated data

You can use the Export All Data button to export simulated data to an Excel file as a template for the structure of the data that you need to support your mobile report. You can then replace the data in the template and import the new Excel file with actual data, which you then connect to mobile report elements as described later in this chapter. After you connect all elements to actual data sources, the simulated data set is removed from the mobile report. The simulated data is never stored with the mobile report even if it is still in use by a mobile report element. Instead, it is generated at run time when you view the mobile report in layout or preview mode or open the data set.

To import data, click the Add Data button and choose one of the following options:

- **Local Excel** When you select a local Excel file (which can also be a CSV file), you must specify a worksheet in the file. In this case, Mobile Report Publisher imports the data from the file and stores it with your mobile report. You can click the Refresh All Data button in the designer data view to update data from the Excel file if the file's contents change later.

- **Report server** When you select the Report Server option, you select a specific report server and then navigate folders on the server containing data sources. Click a folder tile to display the data sets it contains and then select the data set tile to import its data into your mobile report.

Note The first time you select the Report Server option, the Connect To A Server prompt is displayed. In the Server Address box, type **<servername>/reports**, replacing the *<servername>* token with the name of the server hosting Reporting Services. Do not include the http:// prefix. If the report server is not configured to use Secure Socket Layer (SSL), clear the Use Secure Connection check box. If the data set uses a database login instead of Windows authentication, or if you want to use a different Windows account, clear the Use Current Windows Account check box and supply the applicable credentials.

Working with Excel Data

To import data from Excel, your file must have been created using Excel 2007 or later and must be saved with the .xlsx file extension. You can import only one worksheet at a time. Column headers are recommended but are not required. The import process will automatically assign one of the following data types to each column: *string*, *double* (numeric), *Boolean* (true/flase), or *datetime*. You should ensure that each column contains the same type of data in each row. Otherwise, the data-type assignment on import may not be correct. Numbers should not be formatted with currency or other symbols. Your data can contain formulas, but the import process resolves the formulas and imports only values.

Mobile Report Publisher automatically configures two properties for imported data when it detects a date/time column. You can override these properties by clicking the gear charm on the data view tab and then clicking the Dates charm. On the Set Date/Time Settings page, shown in Figure 7-17, you can select a specific column in the Date/Time Filter Column drop-down list (if there are multiple date/time columns available) instead of keeping the default value of Auto, which uses the first date/time column in the data view. You also have the option of choosing another level of granularity, such as Months or Weeks, in the Aggregation Time Unit drop-down list.

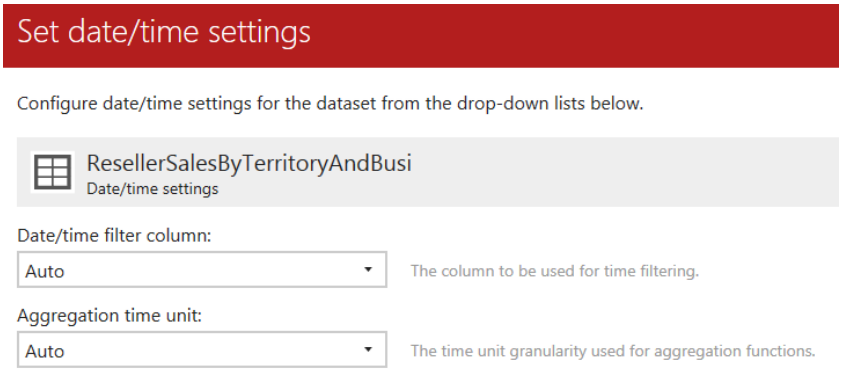


Figure 7-17: Configuring the Date/Time settings for a data view.

After importing data into a mobile report in Mobile Report Publisher, you can then connect each data set to mobile report elements. To do this, first select an element in the Control Instances pane to the left of the data view. Then, in the Data Properties panel below the data view, select the data view to assign (by name) from the applicable drop-down list for the selected element. As an example, use the Series For Background Chart drop-down list to assign a data set to a time navigator element, as shown in Figure 7-18. The options that are displayed in the Data Properties panel, including the name of the data view drop-down list, vary by mobile report element.

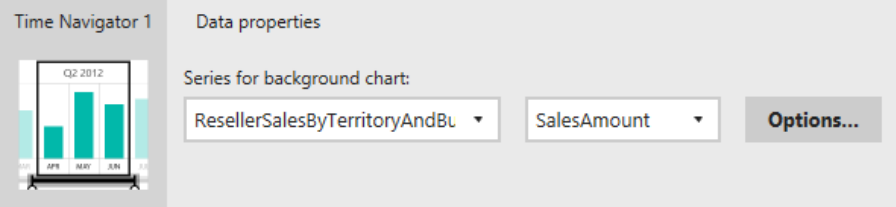


Figure 7-18: Assigning data properties to a time navigator mobile report element.

Some elements require you to connect two data sets, such as many of the gauge elements that compare one value (the Main Value) to another value (the Comparison Value). You can use the same data set for both values or use one data set for the Main Value and a separate data set for the Comparison Value.

Reviewing mobile report elements

The Mobile Report Element Gallery in Mobile Report Publisher contains a variety of element types:

- **Navigators** A navigator filters data by a time range or by one or more selected values.
- **Gauges** A gauge compares a value to a target, numerically, graphically, or both.
- **Charts** A chart uses visualizations to summarize a data view by using columns or shapes.
- **Maps** A map uses color to summarize data by geographical boundaries or superimposes bubbles representing aggregated data on geographical regions to enable comparisons by area.
- **Data Grids** A data grid is a table containing a fixed set of columns and a variable number of rows. It can display data only or, optionally, include delta values or indicators (or both) and a simple chart.

Navigators

You add navigators to a mobile report to enable filtering by a date or time column or by a column value. There are three types of navigators that you can use to filter elements in your mobile report:

- **Time navigator** You use a time navigator to specify a date range to use as a filter for any element in the mobile report that is connected to a data set containing a time-based column. The user can adjust the date range for the filter by using the slider control in the last panel of the time navigator, as shown in Figure 7-19. If there are multiple time-based columns in the data set, the navigator applies only to the first column by default, but you can override this by changing the data set's parameter settings as described in the previous section of this chapter.

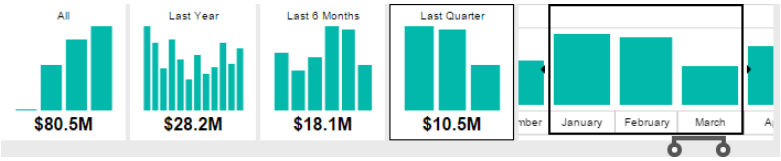


Figure 7-19: Time navigator displaying data as a column chart.

- **Selection list** You use a selection list navigator, shown in Figure 7-20, as another type of filter for the mobile report. This navigator can display a set of values as a list or as a hierarchical tree. You have the option to configure this list to include All in addition to values derived from a key column in a data set that you specify after connecting the selection list to a data set. You also specify which data sets to filter based on the user's selection. Another option is to configure the selection list to allow multiple selections.

Figure 7-20: Displaying the list of available values in a selection list navigator.

- **Scorecard grid** You use a scorecard grid navigator as a selection list that can include a gauge column to visualize progress toward goals in addition to the value columns from your data set, as shown in Figure 7-21. The gauge can be an indicator as shown in the figure, an arrow with conditional formatting based on defined delta value ranges, or a delta value with conditional formatting of either the background or foreground.

Employee Quotas

Category	Variance	Quota	Sales
Blythe, Michael	▼	11.2M	8.89M
Campbell, David	▼	4.03M	3.66M
Carson, Jillian	▼	12.2M	9.58M
Ito, Shu	▼	7.80M	6.18M
Jiang, Stephen	▼	1.28M	1.06M
Mensa-Annan, Tete	▼	2.43M	1.97M
Mitchell, Linda	▼	11.8M	9.48M
Pak, Jae	▼	10.5M	7.84M
Reiter, Tsvi	▼	8.54M	6.75M
Vargas, Garrett	▼	4.37M	3.33M
Abbas, Syed	▲	73K	151K
Tsoflias, Lynn	▼	1.69M	1.10M

Figure 7-21: Displaying a scorecard grid containing a delta indicator and value columns.

Filtering mobile report elements

After you add other types of elements to a mobile report, you can specify which navigator elements filter these other elements. To connect a navigator to a mobile report element, click the element in the Control Instances panel in the designer's data view, and then click the Options button in the Data Properties panel. In the dialog box that appears, select one or more navigators to use as filters for the selected element.

Gauges

There are two groups of gauges available as mobile report elements. One group consists primarily of numerical values, while the other group incorporates a graphical representation of a value in addition to the numerical value represented in the graph. All gauges allow you to specify a title, a subtitle, and a number format.

Most gauges allow you to choose how to represent the gauge indicator. You can display the delta value (that is, the difference between the actual value and the target), the delta value as a percentage of the target, the actual value as a percentage of the target value, or the delta value and its percentage of the target value. You also associate value ranges to conditionally format the gauge with red, amber, or green. These colors might be used as the background color for the entire gauge, the

background of the delta value or percentage, or a directional arrow. Furthermore, you specify whether high delta values are good (green) or bad (red). Apart from these settings and the ability to resize the elements or apply a theme to your mobile report, you have no additional control over the appearance of a gauge, such as font type or font size.

In addition, you can optionally define a drill-through target for any gauge. A drill-through target can be an existing mobile report to which you can pass a static value or a selected value to override default values in data sets. Alternatively, a drill-through target can be a custom URL that you can define to include parameter tokens as part of the query string.

In the group of numerical gauges, shown in Figure 7-22, you can choose from the following four options:

- **Number** This gauge displays a single aggregated value from the connected data set.
- **Delta number** You must define an actual and target value for this gauge, either from two columns in the same data set or from columns in two separate data sets. The default display is the delta value.
- **Delta indicator** You configure this gauge much like the delta number gauge. It uses a green arrow indicator pointing up to indicate a good value and a red arrow indicator pointing down to indicate a bad value. The default display is the delta's percentage of the target.
- **Number with delta** This gauge is a combination of the number, delta number, and delta indicator gauges. The default delta label is Value And Percentage From Target. The background for the delta uses the range stops to display red, amber, or green. You have the option to use this color for the entire gauge rather than for the delta number only.

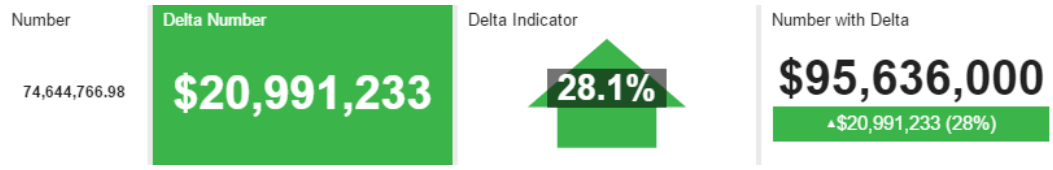


Figure 7-22: Displaying numerical gauges in a mobile report.

Note Mobile Report Publisher automatically aggregates the numerical value to display by using the Sum function. You can change this behavior by clicking the element in the Control Instances panel in the mobile report's data view. Then click the Options button in the Data Properties panel. In the dialog box that is displayed, select one of the other aggregate functions: Avg, Count, Min, Max, First, or Last.

In the group of graphical gauges shown in Figure 7-23, each displays the actual value and, optionally, a delta value and percentage as described for the numerical gauges. You can specify ranges to determine how to visualize values as red, yellow, or green objects or areas on a scale. This group contains the following seven options:

- **Radial gauge** This gauge uses a pointer on a scaled arc to represent the actual value's percentage of the target value.
- **Linear gauge** This gauge uses a linear scale with a pointer to represent the actual value's percentage of the target value.
- **Bullet graph** The bullet graph is similar to the linear gauge but replaces the pointer with a progress bar superimposed on the scale and a marker to indicate the actual value as a percentage of the target value.

- **Progress bar** The progress bar is similar to a data bar in paginated reports, in which the length of the bar represents the actual value as a percentage of the target value.
- **Cylinder** The cylinder shows the actual value as a height relative to the target value, but provides no context for the target. This visualization is more useful when you use multiple cylinder gauges to compare different actual values to one another.
- **Half donut** Like the progress bar, this gauge uses length to represent the actual value as a percentage of the target value, but uses an arc instead of a horizontal bar.
- **Thermometer** The thermometer is like cylinder but provides an outline to help you see the actual value as a percentage of the target value.

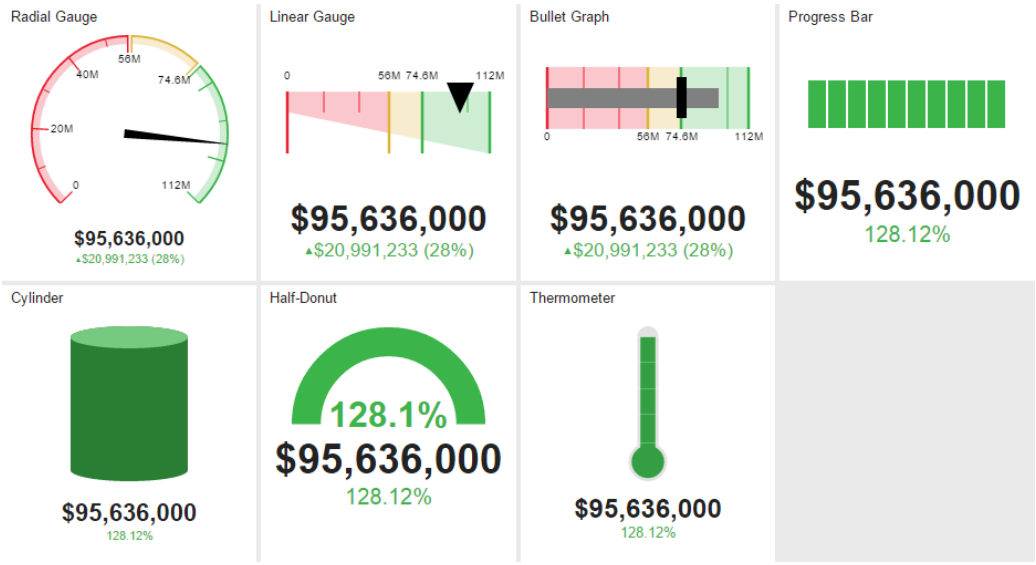


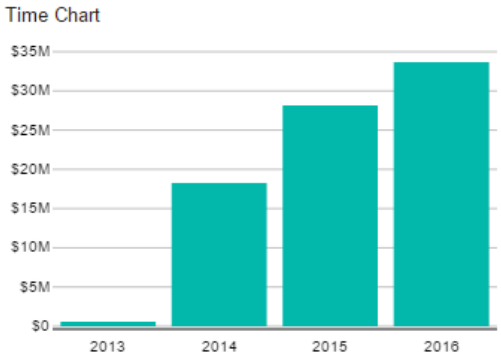
Figure 7-23: Displaying graphical gauges in a mobile report.

Charts

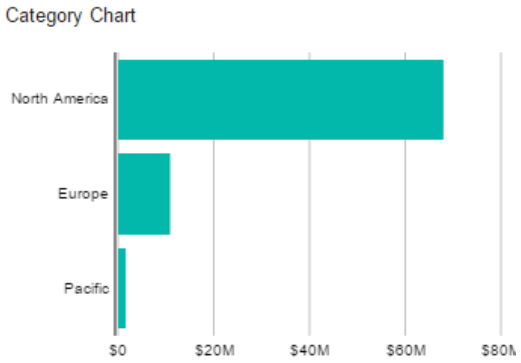
You can also add charts as another type of visualization when your mobile report requires more detail than gauges provide. But as with gauges, there are limited options for formatting a chart. Most chart elements allow you to set the number format for the axis labels and tooltips. Some chart elements allow you to switch a toggle to display a legend.

There are several different chart types that you can add to a mobile report:

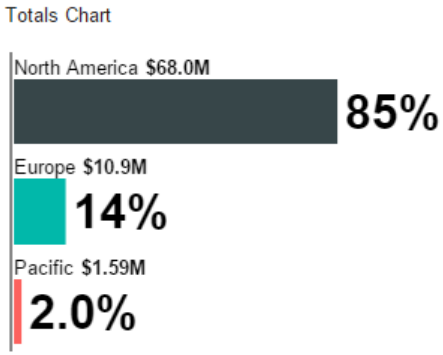
- **Time chart** This chart type uses a date/time column on the horizontal axis and plots a numeric value as a column chart series by default. You can change the time increment for the horizontal axis. You can change the visualization type to stacked bars, 100% stacked bars, line, stacked area, or stacked step area or keep the default side-by-side bar (column) visualization, as shown here:



- **Category chart** A category chart is similar to the time chart except that you designate a data set column to use instead of date/time along the horizontal axis. You also have the option to switch from vertical bars to horizontal bars and sort the data points in ascending or descending order, as shown here:

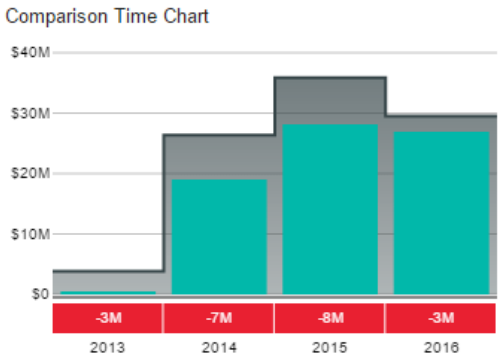


- **Totals chart** A totals chart displays data labels and, optionally, the percentage of total values for each bar. By default, it shows as separate data points the sum of each numerical column that you specify to include in the chart. That is, for each numerical column in the data set, the chart displays a single horizontal bar. However, you can change the data structure property from the default ByRows to ByColumns to create category groups based on the column with the fewest distinct values in the data set, as shown here:

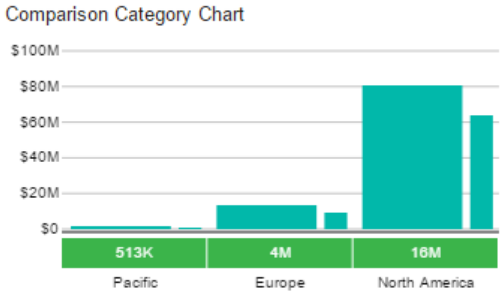


- **Comparison time chart** This chart type is similar to the time chart but includes a second data series, which can be another numerical column in the same data set or a separate data set that also includes a date-time column. The dashboard designer automatically aggregates both data series to the same level, which you can adjust from the default to a more granular level of detail, such as quarter or month. You can optionally configure the comparison series to use the same

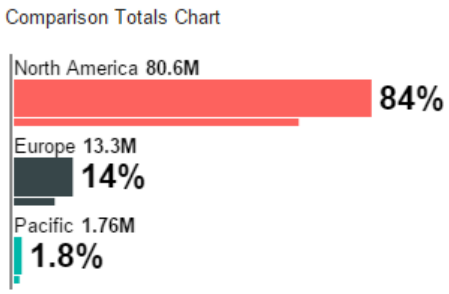
color as the main series. The main series always uses a vertical bar visualization, whereas the comparison series can use a line, thin bar, or step visualization, as shown here:



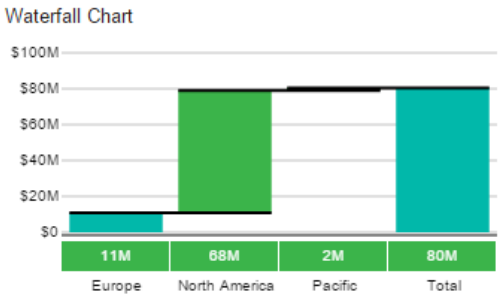
- **Comparison category chart** The comparison category chart is like the category chart, allowing you to define the chart orientation and apply sorting, but it also includes a comparison series. As with the comparison time chart, you can use not only a line or step visualization for the comparison series, you can also use a standard bar visualization. Another option is the thin bar visualization for the comparison values, as shown here:



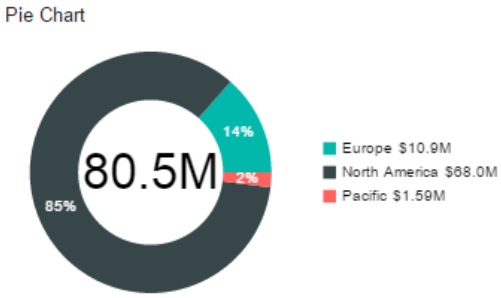
- **Comparison totals chart** This chart type is like the totals chart but uses a main series and a comparison series. The data label for the category applies to the common category for both series, while the numeric data label applies only to the main series value, as shown here:



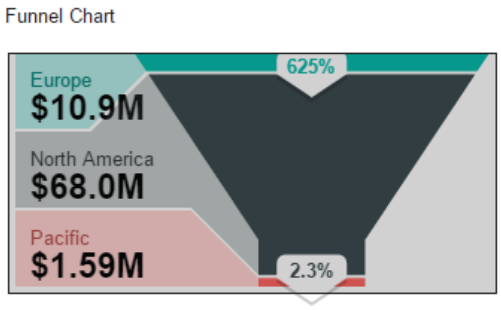
- **Waterfall chart** A waterfall chart is a visualization that shows how increases or decreases in a value accumulate to produce a total value. In the properties for this data element, you specify whether the plotted values are delta values or total values and whether green or red is associated with better values. The formatting is applied to each bar and to a corresponding box indicator along the horizontal axis, as shown here:



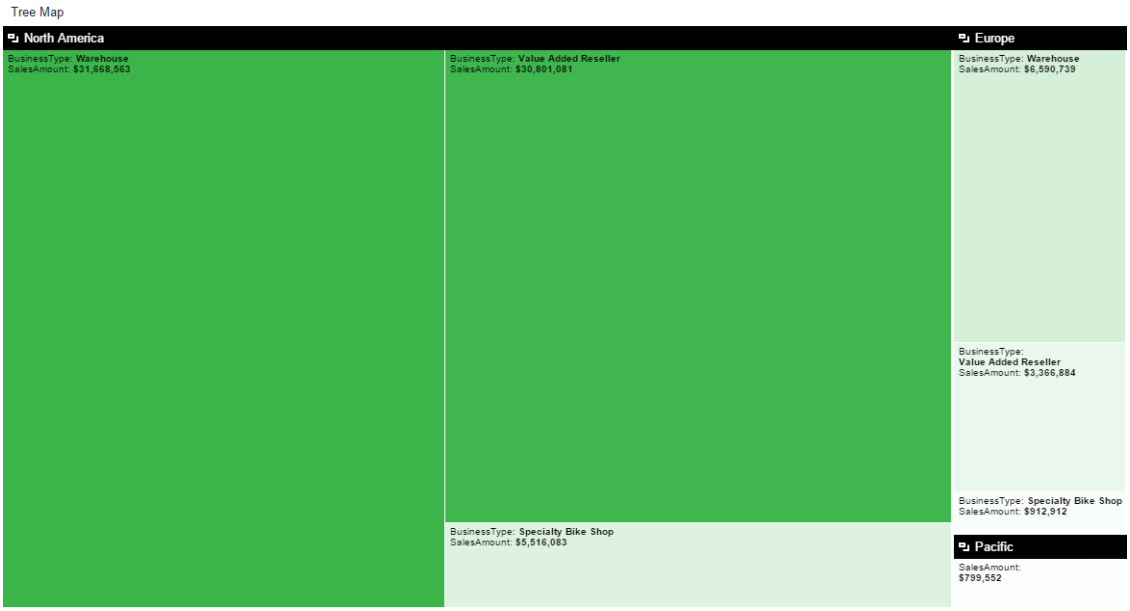
- **Pie chart** The pie chart displays categories as a percentage of the whole. If the categories are in separate columns in the data view, you select the ByColumns data structure in the layout view and select each category column in the data view. If your data uses row values for grouping, change the data structure to ByRows and select the column containing the category values in the data view. You can choose to visualize the data as a pie chart, a donut chart, or a donut with a total, as shown here:



- **Funnel chart** The funnel chart properties are similar to the pie chart properties, and the goal of the visualization to represent how parts relate to the whole is similar. The funnel chart uses color to segment a funnel shape and displays abbreviated values to the left of the funnel.



- **Tree map** A tree map chart in the mobile reports dashboard is similar to the tree map now available in paginated reports, which we describe earlier in this chapter. In the dashboard, you can control how color is applied. You can use a traditional heat map, a color groups heat map (by using one numerical value for rectangle size and another numerical value for comparison), or a hierarchical heat map with a maximum of two levels, as shown here:



Note Labels can be difficult to read on a tree map and cannot be configured to display a different size or color. However, a toggle that displays pop-ups, also known as tooltips, is available for this dashboard element and should be enabled to help viewers better understand the tree map.

Maps

For data associated with geographical locations, you have other visualization options. This map type does not require you to use spatial data, like you do when creating maps for paginated reports in Report Builder. Instead, your data view must contain a column that uses standard values for geographical locations, such as state names in the United States, or country or region names. You then choose the applicable map in the visual properties for the selected map element, such as USA or World Countries. You can choose from three different types of map visualizations, shown in Figure 7-24:

- **Gradient heat map** This map uses varying hues of the same color to represent data value ranges for the metric associated with the map. The map automatically includes a legend to aid interpretation of the value ranges by color.
- **Range stop heat map** This map uses a main value and a comparison value to calculate a delta between the two. You can then configure three value ranges based on the delta's percentage of the target to apply colors to the map based on higher or lower differences between the main and comparison values.
- **Bubble map** The bubble map uses relative sizes of bubbles to aid comparison of a specified metric across different geographical locations. You can specify whether the bubbles use the same or different colors, but no metric is associated with color.

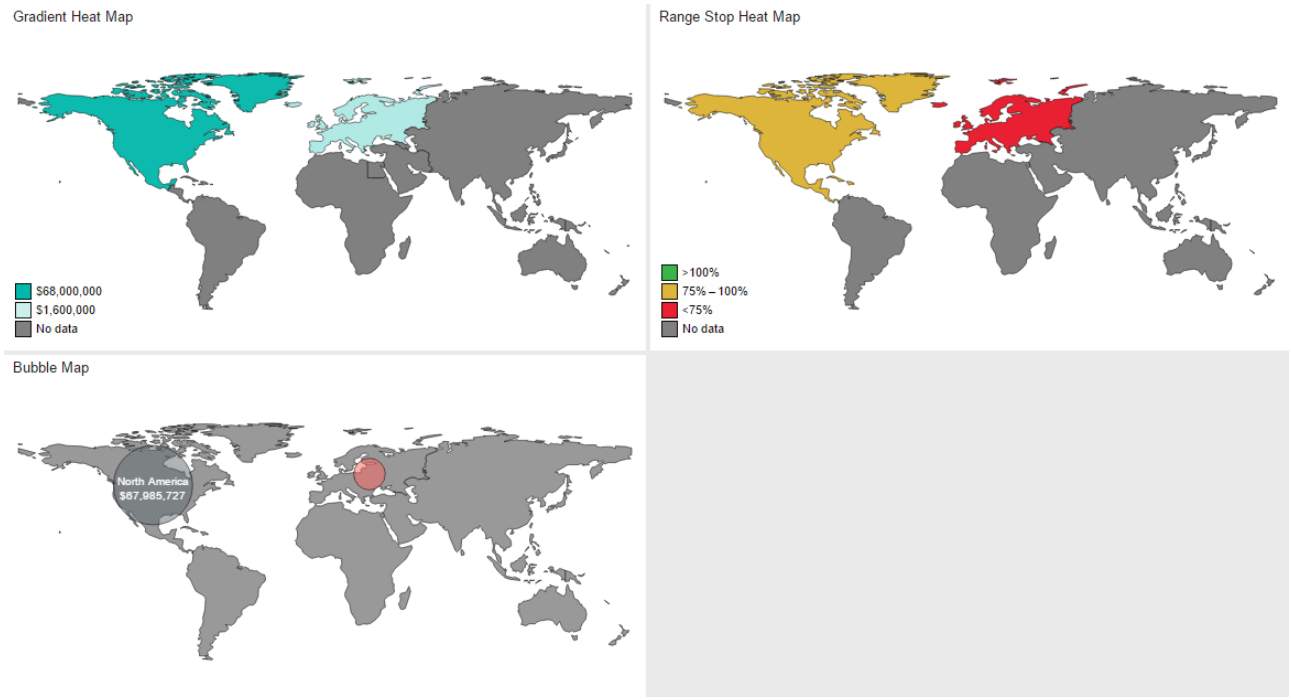


Figure 7-24: Displaying maps in a mobile report.

Data grids

When you need a tabular view of data in a mobile report, you can use one of the data grids. They all share common layout properties, such as whether to show or hide row numbers, associate a drill-through target, aggregate by time, or aggregate by another data-set column. In the data view properties for these elements, you select the columns to display in the data grid as well as the aggregation function to us, if applicable. The only formatting options you have are related to indicators, which are the same as the options described earlier in this chapter for gauges. You can, however, arrange the columns in any order.

You can choose from three types of data grids, shown in Figure 7-25:

- **Simple data grid** This data grid is the simplest of the three, as its name implies. It includes only a set of columns that you select from the data set. You can optionally specify a column to use for aggregation.
- **Indicator data grid** This data grid allows you to specify a set of columns, as in the simple data grid, and a gauge column to hold an indicator. When you configure the gauge column, you select a gauge type, define the value and comparison fields in your data set, and then specify the value direction as higher is good or lower is good.
- **Chart data grid** This data grid is like the indicator data grid, but with one additional type of column to hold a chart. In the data properties for this dashboard element, you select one data set for the standard and gauge columns and a separate data set for the chart visualization. The two data views must share a common column to enable the alignment of each chart to rows in the data grid. However, the column names do not need to match, When you configure the chart column, you specify the name of the column in each data set to use as the lookup. For the chart column, you can use a line, column, or area visualization.

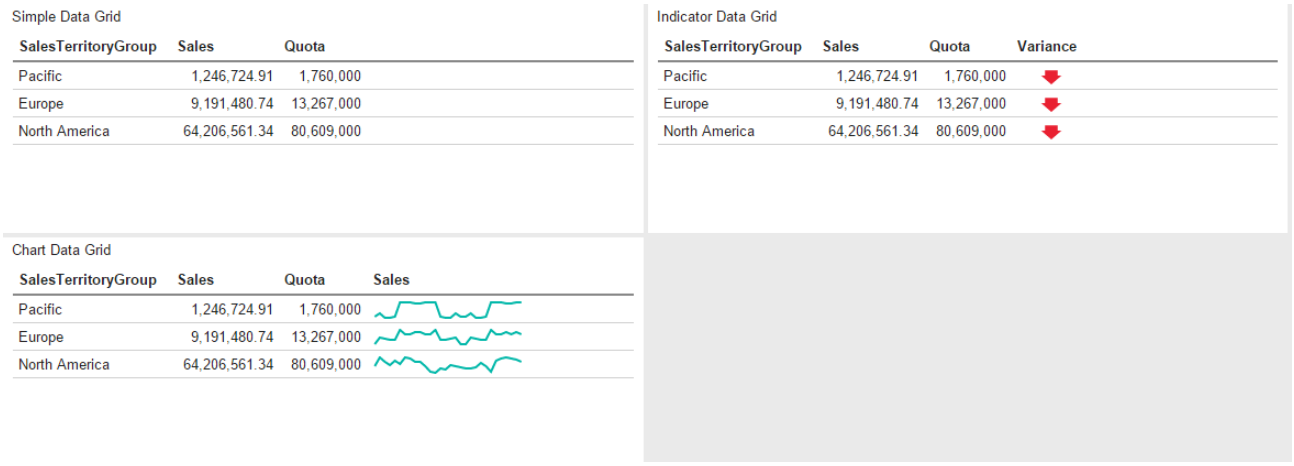


Figure 7-25: Displaying data grids in a mobile report.

Publishing mobile reports

When you are ready to share a mobile report, click the Save Mobile Report As button (above the Data button in the Navigation pane). You can choose to save locally or to the report server. If you click the Save To Server tile, you can provide a name for the mobile report, select a report server, and then click the Browse button to navigate through folders on the report server. After you click the folder in which to save the report, click the Choose Folder button, and then click the Save button.

KPI development

You use the Reporting Services web portal to create KPIs. From the main portal page at http://<yourserver>/reports, click New in the toolbar, and then click KPI. A new KPI screen is displayed, as shown in Figure 7-26.

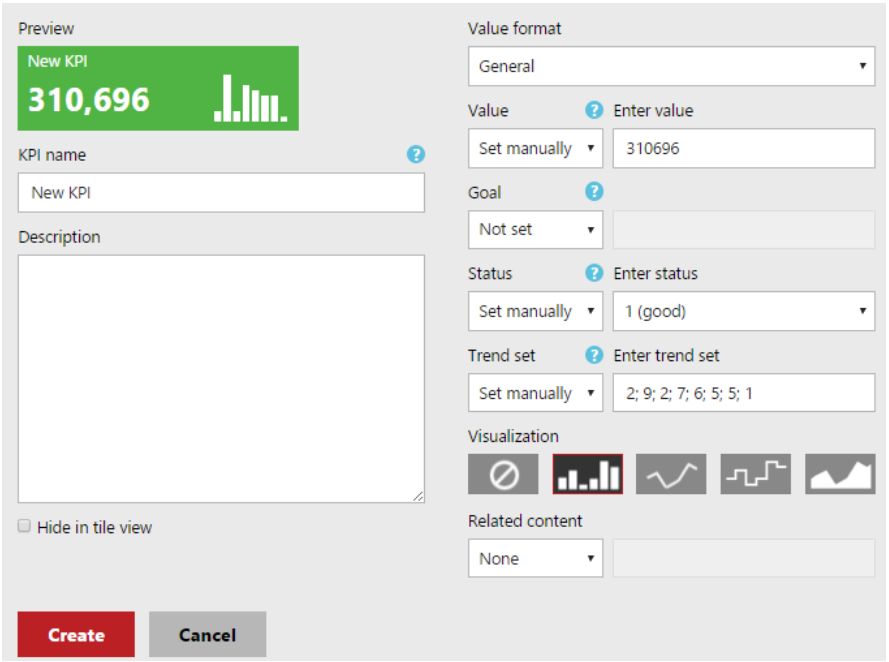


Figure 7-26: Creating a new KPI.

To configure a KPI, you specify up to four values: Value, the amount to monitor; Goal, the target amount to compare with Value; Status, a value to set the color of the background; and Trend, a set of values to visualize. For each of these values, you can set its value manually, associate it with a field in a shared data set on the report server, or leave its value empty. (If you choose to use a shared data set, remember that you can specify a cache refresh plan to update the KPI as frequently as necessary.) Last, you can choose to optionally include one of the following visualizations: column chart, line chart, step chart, or area chart.

Note Data sets for Value, Goal, and Status must return a single row of data. If you choose to use a query for Status, the query must return -1 for red, 0 for amber, and 1 for green. A query for Trend must return a sorted set of one or more values for use as data points in the visualization.

Report access enhancements

The user-facing side of Reporting Services also benefits from several enhancements in this release. First, browser rendering and broader support has been upgraded to accommodate modern web standards. Furthermore, the ActiveX control is no longer required to print from the web portal. Next, users can export reports directly to PowerPoint. Last, the process of working with subscriptions in the web portal has been improved with several new capabilities to streamline and simplify subscription management.

Accessing reports with modern browsers

When Reporting Services was initially added to the SQL Server platform, it was optimized for Internet Explorer 5. Since then, web standards have changed. As a result, modern browsers that are optimized for newer web standards such as HTML5 and CSS3 have emerged and grown in popularity. But however popular these browsers might be for users on a day-to-day basis, earlier versions of Reporting Services do not render reports consistently in these browsers at best or do not render them at all at worst. In SQL Server 2016, Reporting Services is redesigned with a new renderer that supports HTML5 and has no dependency on features specific to Internet Explorer, so users can have a consistent experience across supported browsers. The following table shows the browsers currently supported by the latest version of Reporting Services by operating system:

Browser	Windows 10	Windows 8 and 8.1	Windows 7	Windows Server 2012 and 2012 R2	Windows Server 2008 R2	Mac OS X 10.7-10.10	iOS 6-9 for iPad
Microsoft Edge	Yes	-	-	-		-	-
Microsoft Internet Explorer 10 and 11	Yes	Yes	Yes	Yes	Yes	-	-
Google Chrome	Yes	Yes	Yes	Yes	Yes	-	-
Mozilla Firefox	Yes	Yes	Yes	Yes	Yes	Yes	-
Apple Safari	-	-	-	-	-	Yes	Yes

Regardless of which browser you use, the first time you attempt to open a report, an error message is displayed if you have not configured the browser to run scripts. In response to the message, you can click to continue to view the report without scripts. In that case, the report renders in HTML, but the features supported by the report viewer are not displayed, such as the report toolbar and the document map.

> **Note** Enhancing the renderer to work across all browsers is a huge undertaking. Despite extensive testing, it is possible that a particular combination of report elements that worked well in an earlier version of Reporting Services no longer renders properly. If you find that a report does not render correctly with the new rendering engine, you can click the Switch To Compatibility Mode link on the right side of the report viewer toolbar to revert rendering to Reporting Services' prior style of rendering. You can also click the Send Feedback button next to this link if you continue to have a problem rendering a report. Clicking this link opens the SQL Server Reporting Services Forum on MSDN, where you can use the Ask A Question button to create a post describing the problem you are experiencing.

Not only is the rendering engine updated, but the Report Manager web application used for report access is no longer available. Instead, users access reports by using the new Reporting Services web portal, shown in Figure 7-27. The web portal includes a Favorites page on which you can view reports by type: KPIs, mobile reports, and paginated reports. You can switch to the Browse page to view reports and KPIs by navigating through folders.

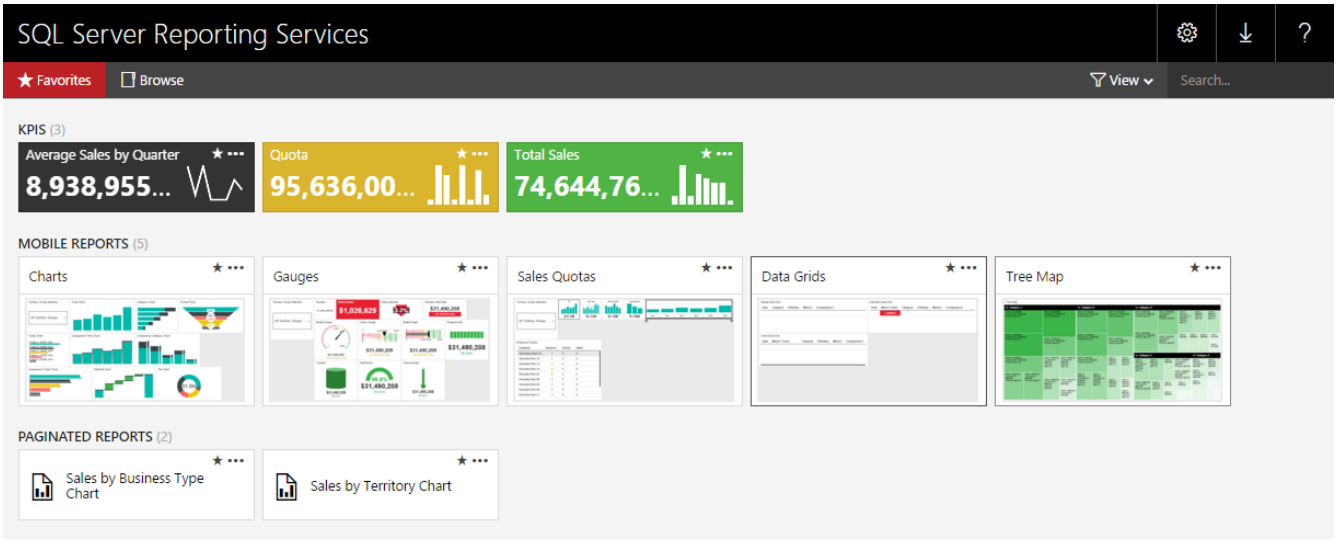


Figure 7-27: The home page of the new Reporting Services web portal displaying the Favorites tab.

Viewing reports on mobile devices

In addition to using the web portal to view mobile reports rendered as HTML5 pages in a web browser, you can also interact with these reports through a native user interface on the following major mobile platforms:

- **Windows 8 or later** On your tablets and touch-enabled devices, you can use semantic zoom while viewing reports. In addition, you can pin dashboards and KPIs to the Start screen.
- **iOS8 or later** You can access published dashboards and KPIs while online and review KPI summary data when offline.

Printing without ActiveX

Earlier versions of Reporting Services require users to install ActiveX to enable a control in Internet Explorer that allows them to print a paginated report from the browser. However, for security reasons, many enterprise users do not have the necessary permissions to install software on their computers, including ActiveX controls. Furthermore, many modern browsers do not support ActiveX. Consequently, in SQL Server 2016, Reporting Services provides a new solution by generating a printer-friendly PDF version of the report with the option to override the default page size.

When you click the printer icon in the report viewer toolbar, Reporting Services checks for the existence of the Acrobat PDF browser plug-in in Internet Explorer. If it does not exist, an error message prompts you to install the plug-in. However, if your browser does not have the plug-in, you are still able to print if you clear the error message. After you clear the error message, or if you are using a browser other than Internet Explorer, the Print dialog box is displayed, as shown in Figure 7-28. This dialog box allows you to adjust the paper size and page orientation by using the respective drop-down lists before printing your report.

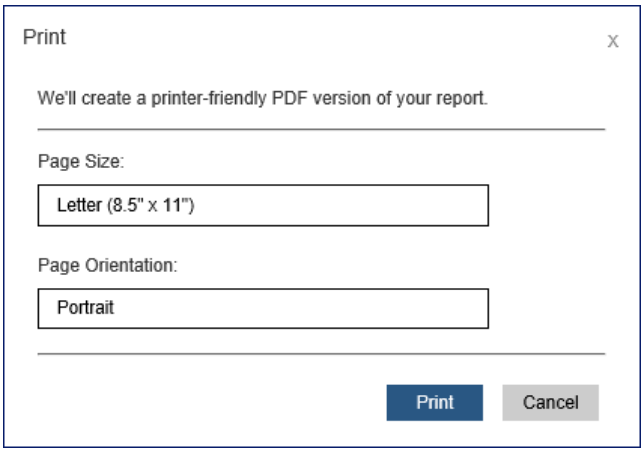


Figure 7-28: Print dialog box for browser without PDF control.

When you click the Print button in this dialog box in Internet Explorer, the operating system's Print dialog box displays more options for selecting which pages to print, the number of copies to print, and so on. If you choose to cancel at this point, the operating system's Print dialog box closes, and you then see another type of Print dialog box that displays a preview of the first page of your report, as shown in Figure 7-29. At the bottom of this dialog box is the Click Here To View The PDF Of Your Report link, which allows you to open your report in Acrobat Reader if it is installed on your computer. Otherwise, you can download the PDF to store it for later viewing once you have installed the necessary software.

Note When you use Edge as your browser and click the Print button in Reporting Services' Print dialog box, another tab opens in the browser and displays your report because Edge has a built-in PDF viewer.

In Chrome, when you click Print, a message appears and indicates that the report is being converted to PDF, and then Chrome's Print dialog box displays.

In Safari, a message indicates that your PDF file is ready and includes the link Click Here To View The PDF Of Your Report. When you click the link, the PDF file downloads and the Preview application opens to display your report.

Figure 7-29: Print dialog box with option to view the PDF of your report.

Just as in prior versions, report server administrators can control whether users see the print icon in the report viewer toolbar. However, the Enable Download For the ActiveX Client Print Control check box is no longer available for this purpose when configuring report server properties because this control is no longer supported. Instead, you change one of the advanced properties that controls the presence of the print icon. To do this, open SQL Server Management Studio by using Run As Administrator, connect to the Report Server, right-click the server node, select Properties, select the Advanced tab in the Server Properties dialog box, and change the EnableClientPrinting property from its default setting of True to False.

Exporting to PowerPoint

One of the many benefits of Reporting Services is the ability to export a report to a variety of different formats, such as Excel or Word. In the SQL Server 2016 release, the list of available options is expanded to include another popular Office application, PowerPoint. When you click the Export button in the report viewer toolbar, you now see PowerPoint listed as an option, as shown in Figure 7-30.

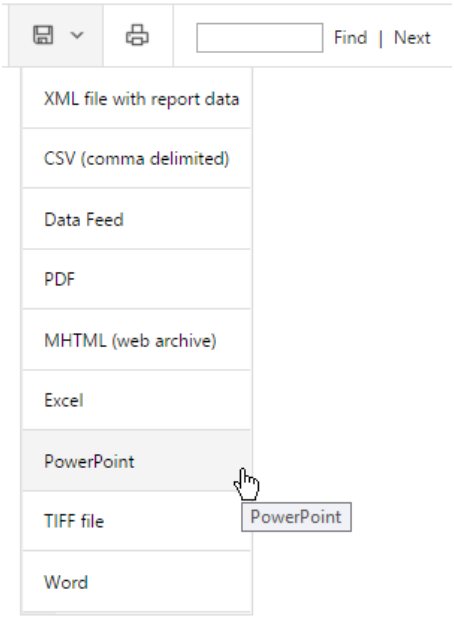


Figure 7-30: Choosing PowerPoint as an option for exporting a report.

Note You can also now use PowerPoint as a rendering format when configuring a subscription.

When you select the PowerPoint export option from the list, the PPTX file downloads to your computer. You then have the option to save it or, if you have PowerPoint installed on your computer, to open the file. In general, each page of your report becomes a separate slide in PowerPoint, as shown in Figure 7-31, although some report items might span multiple slides. Just as you must factor in the rendered page size during report development if you know that users plan to export to PDF or Word, you must ensure report items can fit on a single PowerPoint slide where possible. Otherwise, the Reporting Services rendering engine will divide the report item into two or more smaller pieces and allocate each piece to a separate slide, as shown in the third and fourth PowerPoint slides in Figure 7-31, which collectively represents the third page of a report when the page is rendered in HTML. Notice that objects from a report do not consume the entire vertical space within a PowerPoint slide.

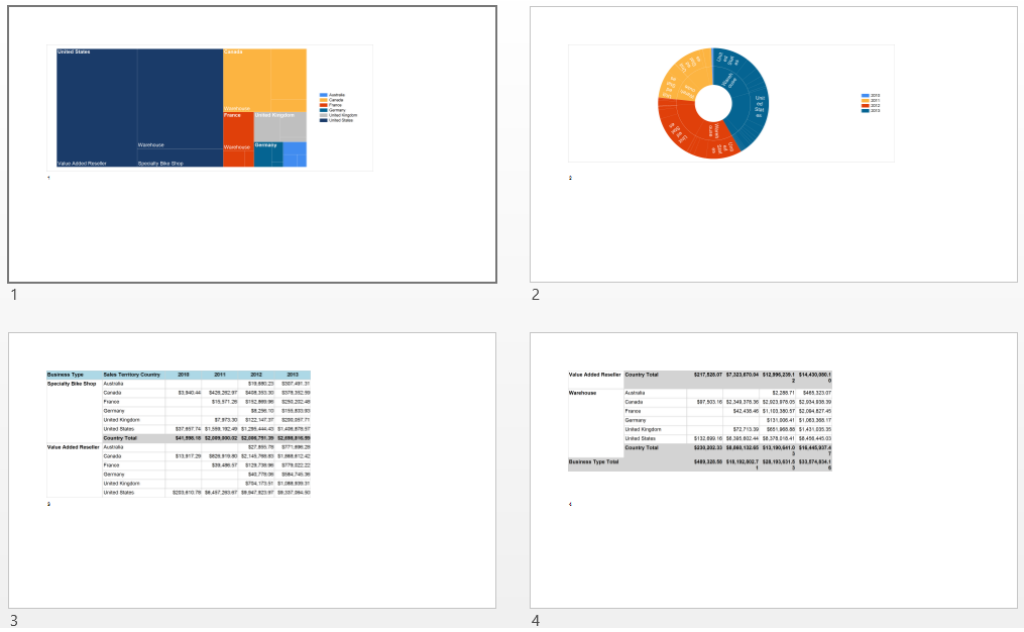


Figure 7-31: A report rendered as a PowerPoint file.

Note Although in an earlier section of this chapter we recommend placing legend items above or below a tree map or sunburst chart to maximize chart space, this recommendation is not applicable to reports that you plan to export to PowerPoint because the vertical space is more constrained.

If you click the Enable Editing button that appears when PowerPoint opens the file, you can interact with the objects added to the file. For example, you can edit freestanding text boxes containing static text such as a report title or page numbers from the page header or footer. Report items such as a chart or a matrix are added as picture objects and cannot be edited, although they can be resized and rearranged by moving them to a different location on the same slide or copying and pasting them to a different slide.

Pinning reports to Power BI

One of the ways that Reporting Services is integrating hybrid and on-premises reporting is a new feature that allows you to pin a report in the web portal to a Power BI dashboard. This capability has several requirements, however. You must be using Azure Active Directory (Azure AD), and the Power BI dashboard that you want to use must be part of an Azure AD managed tenant.

To enable this feature, your Windows login must be a member of the Azure AD managed tenant and also be the system administrator for both Reporting Services and the database hosting the Reporting Services databases. Using these administrative credentials, launch Reporting Services Configuration Manager, click the Power BI Integration tab, click the Register With Power BI button, and provide your Power BI login details.

Before you can pin a report to the dashboard, it must be configured to use stored credentials and SQL Server Agent must be running because Power BI uses a Reporting Services subscription to manage the scheduled refresh of the report. Furthermore, you can pin a report that contains only charts, gauges, or maps that are not nested inside other report items. To pin a report meeting these requirements, open the report in the web portal and click the Pin To Power BI Dashboard button in the web portal toolbar. A sign-in dialog box is displayed in which you must supply your Power BI login credentials. The first time you pin a report, another dialog box asks for permission to update your Power BI app. Next, items in your report that are eligible for pinning are displayed in the browser.

Click the item, select a dashboard, and then choose an hourly, daily, or weekly frequency for updating the report, as shown in Figure 7-32.

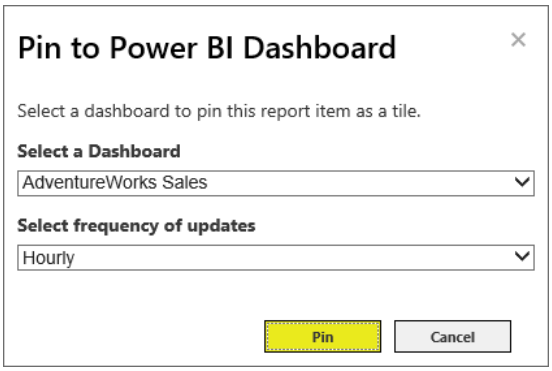


Figure 7-32: Selecting a dashboard for pinning a report.

A dialog box confirms the success or failure of the operation. If the pinning operation succeeds, you can click a link in the dialog box to open a web browser window and view your dashboard in Power BI. Your report shows as a tile in the dashboard, as shown in Figure 7-33, and will refresh periodically according to the schedule you set. When you click the report tile in the dashboard, a new browser window opens to display your report in the web portal from the report server from which it originated.

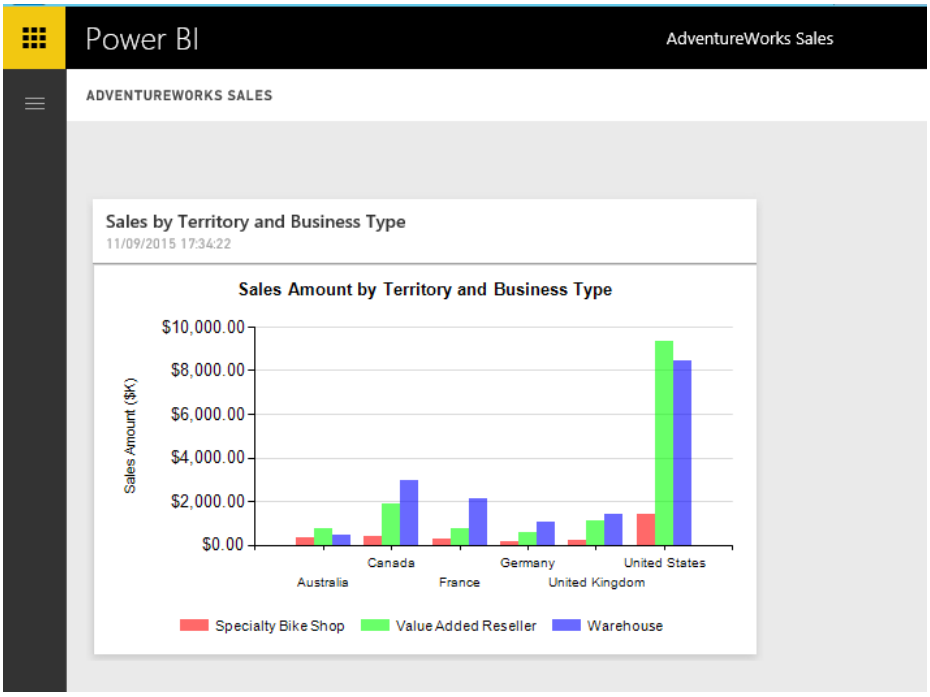


Figure 7-33: Displaying a Reporting Services report as a report tile in a Power BI dashboard.

Managing subscriptions

Subscription functionality does not change in SQL Server 2016 in general. You still configure subscriptions to deliver reports to named recipients or to a designated file share. However, there are a few new subscription-management features that we explore in this section:

- **Subscription description** You can include a subscription description when creating or changing a subscription, which makes it easier to identify a specific subscription when many exist for a single report.
- **Subscription owner change** After adding a subscription to the report server, you can easily change its owner.
- **Interface support for changing subscription status** Whether you have one or many subscriptions set up on the server, the web portal interface now includes Enable and Disable buttons to quickly change the status of subscriptions.
- **File share credentials** File share subscriptions have a new option to use administrator-defined credentials to add files to a file share.

Subscription description

The subscription definition page now includes a section in which you add a description, as shown in Figure 7-34, that is displayed when you create or edit a subscription. You can use this description to distinguish this subscription from others, which is helpful when you have several subscriptions associated with a single report. For example, use this column to describe recipients, the schedule, the delivery type, and other report delivery options so that you no longer have to edit the subscription to determine its settings.

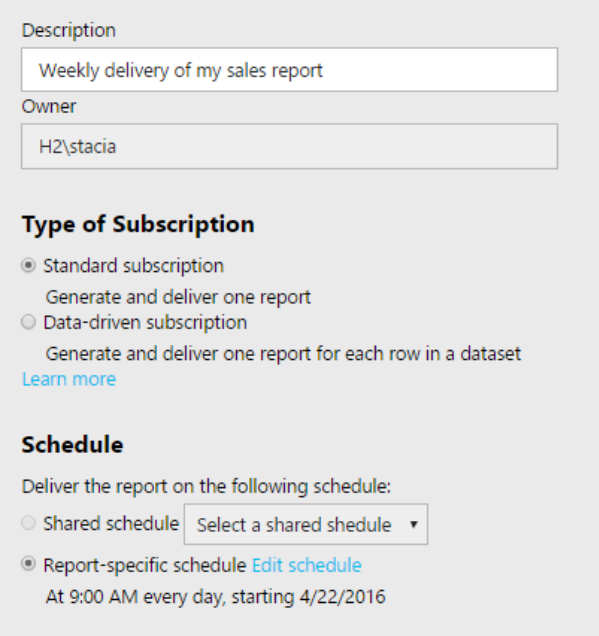


Figure 7-34: A portion of a subscription definition showing the new description.

When you add a description to a subscription, the description is displayed in the web portal on the Subscriptions page that you can access for a specific report or on the My Subscriptions page, where you can see all reports for which you have created subscriptions, as shown in Figure 7-35. You can sort subscriptions by the Description column by clicking the column header.

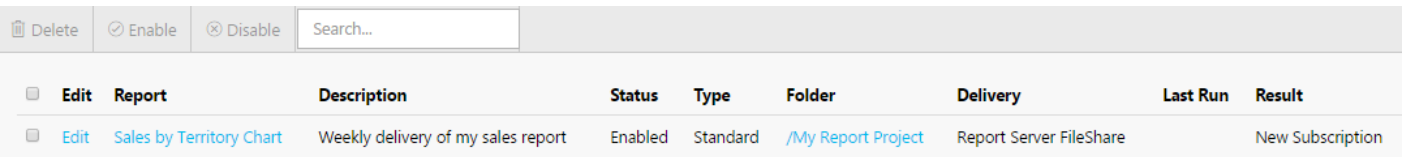


Figure 7-35: My Subscriptions page in the web portal with a new column for the subscription description.

Subscription owner change

By default, the user credentials are set as the owner of a subscription when a new subscription is created and cannot be changed during subscription creation. In prior versions of Reporting Services, a change of owner was possible only programmatically. Now you can edit a subscription in the web portal to change its owner. This feature is particularly helpful when users change roles in an organization. Both the current owner and the report server administrator have permissions to change the owner when editing the subscription in the web portal.

> **Note** This feature is available in both native and SharePoint-integrated modes.

Interface support for changing subscription status

In previous versions of Reporting Services, you can pause and resume a schedule to control when related subscriptions are active. Now there are an Enable and a Disable button in the web portal toolbar when you view subscriptions for an individual report or view the My Subscriptions page. This capability allows you more fine-grained control over the execution of specific subscriptions. When you disable a subscription, the Status column value changes to Disabled, as shown in Figure 7-36.

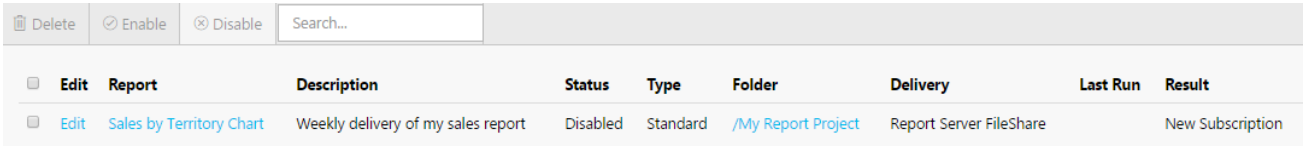


Figure 7-36: My Subscriptions page in the web portal displaying a disabled report.

> **Note** This feature is available in both native and SharePoint-integrated modes.

File share credentials

Rather than instructing users how to define credentials required to save a subscription to a file share, report server administrators can configure the report server to use a single domain user account that users can select when defining a file share subscription. To do this, open Reporting Services Configuration Manager and access the new Subscription Settings page. You enable this feature by selecting the Specify A File Share check box and adding a domain user account and password, as shown in Figure 7-37.

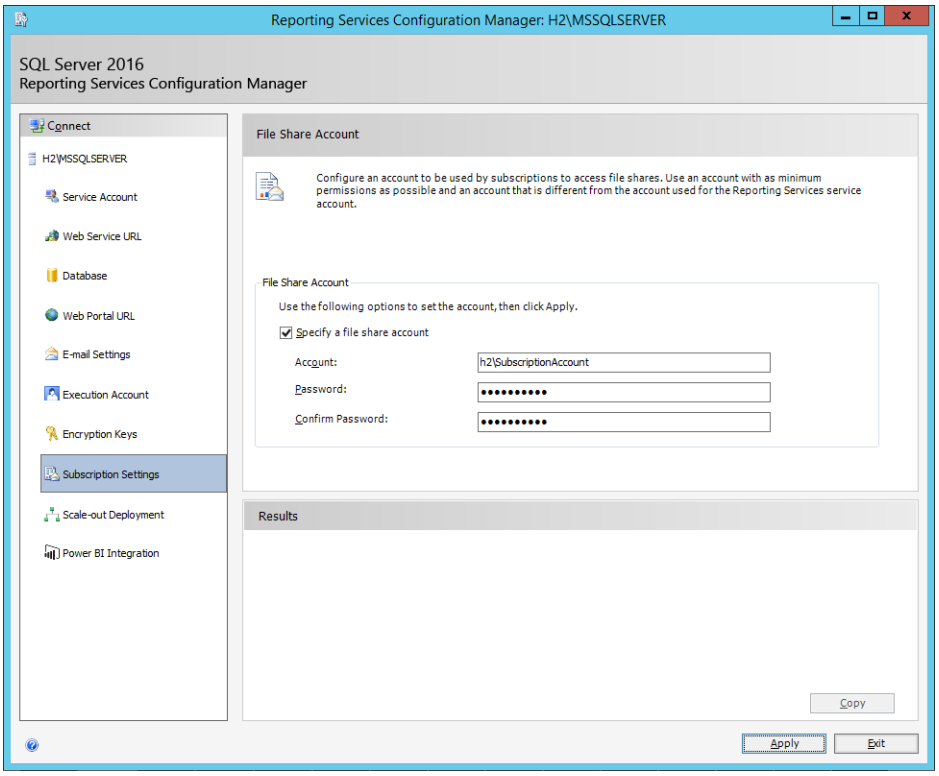


Figure 7-37: Subscription Settings page in Reporting Services Configuration Manager.

Note This feature is available only in native mode.

When this feature is enabled, the user can choose to associate the configured file share account with a subscription when setting the report delivery options for a file share subscription, as shown in Figure 7-38. Using this file share account is not required, however. The user can instead select Use The Following Windows User Credentials and supply the domain user name and password.

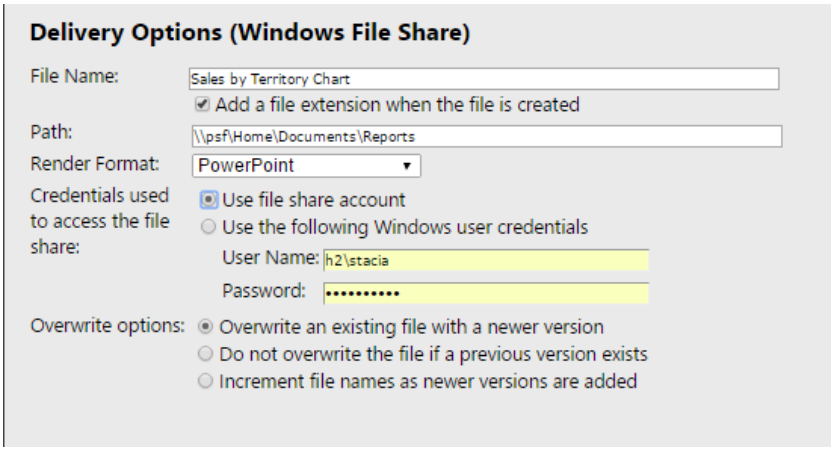


Figure 7-38: The Use File Share Account option when configuring a file share subscription.

CHAPTER 8

Improved Azure SQL Database

Microsoft Azure SQL Database was one of the first cloud services to offer a secure, robust, and flexible database platform to host applications of all types and sizes. When it was introduced, SQL Database had only a small subset of the features available in the SQL Server database engine. With the introduction of version V12 and features such as elastic database pools, SQL Database is now an enterprise-class platform-as-a-service (PaaS) offering. Furthermore, its rapid development cycle is beneficial to both SQL Database and its on-premises counterpart. By integrating new features into SQL Database ahead of SQL Server, the development team can take advantage of a full testing and telemetry cycle, at scale, that allows them to add features to both products much faster. In fact, several of the features in SQL Server 2016 described in earlier chapters, such as Always Encrypted and Row-Level Security, result from the rapid development cycle of SQL Database.

Introduction to SQL Database

Microsoft Azure SQL Database is one of many PaaS offerings available from Microsoft. It was introduced in March 2009 as a relational database-as-a-service called SQL Azure, but it had a limited feature set and data volume restrictions that were useful only for very specific types of small applications. Since then, SQL Database has evolved to attain greater parity with its on-premises predecessor, SQL Server. If you have yet to implement a cloud strategy for data management because of the initial limitations of SQL Database, now is a good time to become familiar with its latest capabilities and discover how best to start integrating it into your technical infrastructure. In this

section, we explore the evolution of SQL Database, explain how the service is priced, and highlight the elements of SQL Database that provide high availability and disaster-recovery benefits.

Reviewing the evolution of SQL Database

The initial version of SQL Database offered many of the same benefits available in other Azure cloud-based services—elasticity, scalability, high availability, fast provisioning, and easy administration. However, as compared to an on-premises instance of SQL Server, this early version of SQL Database had only a subset of features. For example, only simple data types were supported, and the database size was limited to 5 GB, which limited the type of applications that could use SQL Database to applications,such as mobile applications and websites. Also, there was no support for database backups, although it was possible to make a database extract as an alternative.

Although these aspects of SQL Database were challenges for enterprise-scale applications with complex data types and large data volumes, smaller-scale, new applications could be architected specifically to work well with SQL Databases's feature set. Furthermore, there were several benefits that database administrators and application developers could appreciate, such as the lower cost to deploy and maintain databases in the cloud, the use of established skills through the compatibility with existing management tools, and the elimination of administrative overhead required for on-premises hardware.

To address concerns related to data protection in the cloud, Microsoft not only included many security features available in on-premises SQL Server in the first release of SQL Database, but also enforced industry-leading best practices to physically secure the Azure environment and eliminate threats in its data centers. Furthermore, Azure security compliance is validated by third-party audits and certifications to ensure conformance with international and industry standards. The security features introduced in the initial release, such as a server-side firewall restricting access to specific IP addresses, SQL authentication, and encrypted connections, continue to be a part of SQL Database, but even more features have been added recently to provide stricter and more granular levels of security, as we describe later in the "SQL Database security" section of this chapter.

In addition to these security enhancements, SQL Database now has greater parity with on-premises SQL Server. This parity allows you to more easily port current on-premises applications to Azure and to better support enterprise and DevOps requirements in the cloud, thereby greatly expanding the number of potential use cases for SQL Database. Specifically, you can now take advantage of the following database features in the cloud:

- Parallel queries
- Table partitioning
- Online reindexing
- XML indexes
- CLR integration
- Change tracking for data changes
- Columnstore indexes
- T-SQL windowing functions
- Over 100 new DMVs

Understanding the pricing model

SQL Database is available as an Azure pay-as-you-go service, as it has been since its introduction. At that time, you set up your database by specifying either Web or Business edition, for which a flat per-month fee was assessed. Just as the feature set for SQL Database has evolved over time, the pricing model has evolved so that you have a wider set of options from which to choose. When you create a new SQL Database, you now select a service tier and performance level to allocate resources. Your monthly cost is calculated based on your choice of service tier and performance level, in addition to the number of hours of usage and the number of standard or active geo-replication secondary databases established, when applicable.

Note For more information about the factors influencing the monthly cost of SQL Database, see https://azure.microsoft.com/en-us/documentation/articles/sql-database-faq/.

Service tiers

SQL Database service tiers support different types of workloads and throughput levels. It currently includes the following service tiers:

- **Basic** Supports a database up to 2 GB for a small application with a light transactional workload and relational requirements. The benchmark transaction rate is 16,600 transactions per hour.
- **Standard** Supports a database up to 250 GB for a normal to midsize application requiring midlevel performance with built-in high availability. Performance is measured as transactions per minute, with a benchmark of 2,570 for the S2 performance level.
- **Premium** Supports a database up to 1 TB for a top-tier, mission-critical database requiring the highest level of performance and availability. Its benchmark transaction rate is 735 transactions per second at the P3 performance level.

Note Each tier is guaranteed basic availability. However, the Standard and Premium service tiers offer online secondary replicas in other Azure regions, with the Premium service tier offering active secondary replicas.

Performance levels

The unit of measurement for a performance level is a Database Throughput Unit (DTU). It represents a blend of CPU, memory, and read and write metrics related to the server resources allocated to your database. A database with 10 DTUs has twice the server resources of a database with 5 DTUs. Each performance level has a different range of DTUs available and different transactions rates, as shown in the following table.

Service tier	Performance level	DTUs	Benchmark transaction rate
Basic	N/A	5	16,600 transactions per hour
Standard	S0	10	521 transactions per minute
	S1	20	934 transactions per minute
	S2	50	2,570 transactions per minute

	S3	100	5,100 transactions per minute
Premier	P1	125	105 transactions per second
	P2	250	228 transactions per second
	P4	500	Not available
	P6/P3	1,000	735 transactions per second
	P11	1,750	Not available

Note The benchmark transaction rates are intended for general comparison only, based on a sample OLTP workload. You can learn more about the methodology for the benchmark development at "Azure SQL Database benchmark overview" at https://azure.microsoft.com/en-us/documentation/articles/sql-database-benchmark-overview/.

After you select an initial performance level, you should continuously monitor SQL Database and tune your application as necessary so that you can adjust to the smallest possible performance level—and thereby minimize your monthly cost—or scale up as needed for seasonal peaks. For example, if you manage a web retail application, you might experience a large spike in transactions between November and January. Rather than invest in expensive physical hardware to accommodate the peak months, you can minimize your operational expense by scaling a SQL Database up or by scaling it out during these months, and then scale it back in other months to accommodate normal operations.

You can monitor performance in the Azure Management Portal by selecting SQL Databases in the Resources list, and then selecting the database you want to monitor to view the resource utilization as a line chart and key metrics, as shown in Figure 8-1. When these metrics are below 40 percent utilization, you might consider moving to the next lower performance level. On the other hand, you might consider moving your database to the next higher performance level as these metrics approach 80 percent.

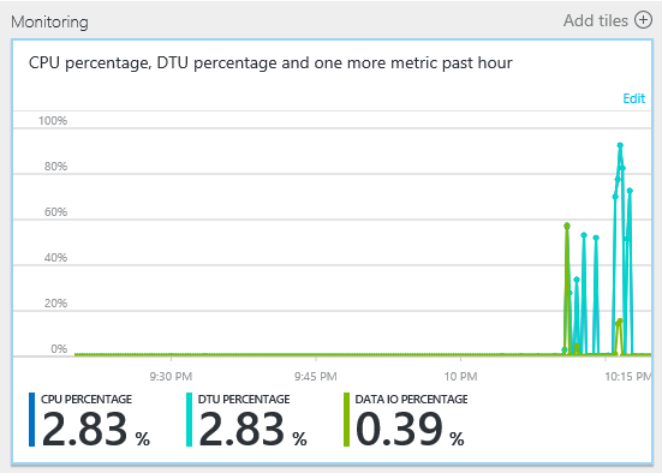


Figure 8-1: Monitoring a SQL Database in Azure Management Portal.

Note For deeper insight into your SQL Database performance, you can analyze dynamic management views (DMVs), Extended Events, or Query Performance Insight. Information about using these tools is available in "Azure SQL Database performance guidance for single databases" at https://azure.microsoft.com/en-us/documentation/articles/sql-database-performance-guidance/.

Protecting data with high availability and disaster recovery

SQL Database provides full-scale high availability and disaster recovery options that can support a variety of business needs. Regardless of service tier, every SQL Database is automatically protected with high availability in the event of hardware failure. When you create your SQL Database, you have at least three replicas of your database at any point in time, with one replica as a primary replica and the other two as secondary replicas. If the hardware hosting the primary replica fails, Azure automatically promotes one of the secondary replicas to primary and builds an additional secondary replica if necessary.

For the Premium service tier, active geo-replication adds an additional layer of protection. Much like the Always On Availability Groups feature in on-premises SQL Server, active geo-replication supports up to four copies of your SQL Database on servers in separate Azure regions for the highest level of availability.

Note Because these secondary copies are also readable and compatible with load balancing for read-only workloads, you can also use this feature as a scale-out option for your SQL Database.

Whereas building an infrastructure to support a multisite on-premises SQL Server infrastructure can be a time-consuming proposition that requires extensive network configuration and multiple installations, configuring geo-replication for SQL Database takes minutes. If your database is in the Premium service tier, you have the option to configure multiple databases. In just a few minutes and a few clicks, you can configure multiple readable secondary copies of your database in multiple Azure regions. To do this, browse to your SQL Database, click the All Settings link in the database blade, and then click Geo-Replication in the Settings blade. In the Geo-Replication blade, select a target region. You can use any region for a Premium database, but you must use the recommended region for a Standard database. In the Create Secondary blade, you specify whether the secondary type is readable or nonreadable for a Premium database. A Standard database secondary can only be configured as nonreadable. You can optionally add a secondary database to an elastic database pool on this blade. When you click Create to add the secondary, the process to seed the secondary begins and its status is shown in the Geo-Replication blade, as shown in Figure 8-2.

Figure 8-2: Configuring Geo-Replication for SQL Database in Azure Management Portal.

Azure also automatically performs backups of your SQL Database in any service tier. A full backup is performed once per week, a differential backup once per day, and a transaction log backup every five minutes. Each service tier has a different retention period for these backups—Basic is 7 days, Standard is 14 days, and Premium is 35 days. With any of your backups, you can easily perform a point-in-time data restore as a new database on the same server to prevent overwriting or losing data in the original database.

SQL Database security

The comprehensive Azure security model is the cornerstone of SQL Database security. Azure is built for resiliency with a trustworthy technology infrastructure and the Assume Breach strategy, combining built-in analytics and a comprehensive methodology to detect and respond to threats. Microsoft has explicit policies in place to control who, when, and how someone outside your organization can access your data. The processes and controls governed by these policies are regularly subjected to accredited third-party auditing.

Configuring security

SQL Database security is similar in practice to SQL Server security. Some features in SQL Database, such as firewall administration and auditing, correspond to features in SQL Server but are implemented differently within the Azure infrastructure.

Firewall administration

By default, the SQL Database firewall blocks access to all connections. After creating your SQL Database, you can use the Azure Management Portal to specify which IP addresses can connect to your database. In the SQL Database blade, click the server name to open the server blade, and then click the Show Firewall Settings link to open the Firewall Settings blade and define firewall rules at the server level, as shown in Figure 8-3. To define rules for each database individually, use the *sp_set_firewall_rule* stored procedure.

> **Note** If you use SQL Database to host data for a software-as-a-service (SaaS) application, you should implement firewall rules at the database level.
>
> You can also programmatically manage firewall settings with T-SQL, REST API, or Azure PowerShell. By using software-defined networking, you can fully automate your application deployment and quickly add new IP addresses. For more information, see "How to: Configure firewalls settings on SQL Database using TSQL" at https://azure.microsoft.com/en-us/documentation/articles/sql-database-configure-firewall-settings-tsql/, which includes links to the other methods.

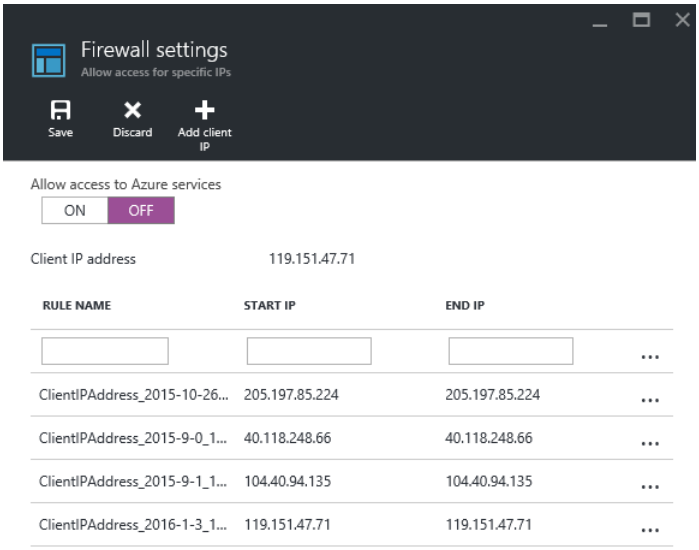


Figure 8-3: Configuring a firewall for SQL Database in Azure Management Portal.

Notice that you also have the option to select a check box on the Firewall Settings page in the portal to permit other Azure services in your subscription to access your database. Be aware that enabling this setting opens your database to all Azure services. As a security best practice, you should open your database only to the specific IP addresses that require access. (You can find a list of Azure Compute IP addresses at http://www.microsoft.com/en-us/download/details.aspx?id=41653.)

> **Note** SQL Database supports communication on TCP port 1433 only.

Authentication

SQL Database currently supports only SQL Server authentication. As one of the steps you perform when you create a SQL Database, you are prompted to create a login that becomes the server-level

principal account for your SQL Database server. This login is analogous to the system administrator (sa) login for an on-premises instance of SQL Server. It manages all server-level and database-level security and has permission to create other accounts that can manage logins and databases in SQL Database.

Server-level roles

Although you can use the server-level principal account to manage server-level security, you also have the option to use *sp_addrolemember* to assign logins to the following SQL Database security roles:

- ***loginmanager*** This role grants permission to create logins in SQL Database, much like you use the *securityadmin* role in on-premises SQL Server. Example 8-1 shows how to assign this role to a login.
- ***dbmanager*** This role grants permission to create databases by executing the CREATE DATABASE command in the master database. It is similar to the *dbcreator* role in on-premises SQL Server.

Example 8-1: Creating a SQL Database login and assigning a role

```
-- connect to the master database
CREATE LOGIN adminlogin WITH password='<loginpassword>';
CREATE USER adminuser FROM LOGIN adminlogin;
EXEC sp_addrolemember 'loginmanager', 'adminuser';
```

Important Be careful to assign these roles only to a limited set of administrators and not to general user accounts. Because these roles scope to the server, a user with permissions in the master database can access data in any SQL Database on the same server by using server-scoped DMVs.

Database-level roles

The built-in security roles at the database level are similar to on-premises SQL Server security roles. To implement database-level security, you can use the same fixed database roles, such as *db_datareader* or *db_datawriter*. Optionally, you can create custom roles for your application when you need explicit permissions for selected database objects.

Important As a security best practice, you should always use role-based security to manage database access.

Contained database users

SQL Database supports contained database users so that you can isolate a user account to a single database instead of creating a server login to which you must then grant database-level permissions. By setting up a contained database user, as shown in Example 8-2, you can move a database between servers and consolidate the user identity and permissions within the individual database.

Example 8-2: Creating a contained database user

```
-- connect to the database
CREATE USER userA WITH password='<userpassword>';
```

Note When a contained database user connects to the database, the connection string must include the database name to properly authenticate.

Auditing SQL Database

An important aspect of security management is auditing database activity. All service tiers of SQL Database include an auditing engine that tracks and writes events to an audit log in Azure storage. You can use auditing for regulatory compliance as well as for uncovering data anomalies or potential security issues. You can configure auditing to capture success or failure conditions for the following predefined events:

- Plain SQL
- Parameterized SQL
- Stored procedures
- Logins
- Transaction management

Auditing configuration

In the Azure Management Portal, navigate to the blade for the server hosting your SQL Database, click the All Settings link, click Auditing & Threat Detection, and set the Auditing switch to On. You can then click Storage Details to select an Azure storage account, specify the number of retention days, provide a table name, and set the storage access key as primary or secondary, as shown in Figure 8-4. In that case, all databases inherit the same audit settings. (As an alternative, you can configure audit policies for each SQL Database individually.) When you select Audited Events in the Auditing Settings blade, you can exclude specific success or failure events if you like. Click the Save As Default button at the top of the Auditing Settings blade to complete the setup process.

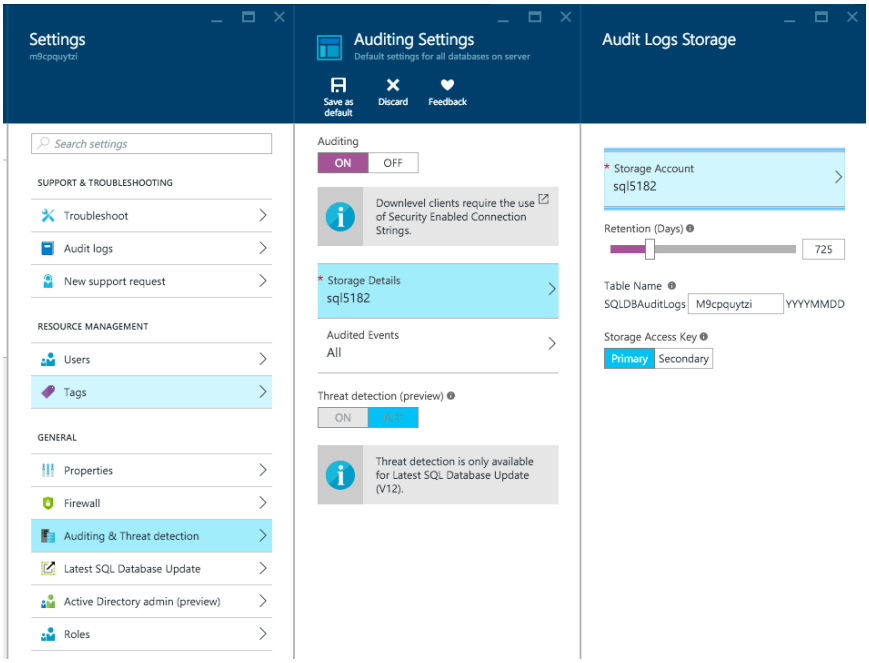


Figure 8-4:.Configuring auditing in the Azure Management Portal.

Important After you enable auditing, you must configure down-level clients to use a security-enabled connection string by changing the fully qualified domain name from *<servername>.database.windows.net* to *<servername>.database.secure.windows.net*, as described in "SQL Database – Downlevel clients support for Auditing" at https://azure.microsoft.com/en-us/documentation/articles/sql-database-auditing-and-dynamic-data-masking-downlevel-clients/.

Auditing data

After you enable auditing, you can view a summary of auditing data in a dashboard format in the Azure Management Portal. To do this, navigate to the Settings blade for your SQL Database, click Auditing & Threat Detection, and then click the Explore button in the Auditing & Threat Detection blade to open the Audit Records blade. The audit records are displayed as a table consisting of the following columns: Event Time, Application Name, Principal Name, Event Type, and Action Status. For more detail, you can click the Open In Excel button at the top of the Audit Records blade. The Excel workbook contains several predefined reports that analyze your database activity.

Important Another option is to use the Microsoft Power BI service to connect directly to your auditing logs, as described at "Monitoring your Azure SQL Database Auditing activity with Power BI," http://blogs.msdn.com/b/powerbi/archive/2015/05/14/monitor-your-azure-sql-database-auditing-activity-with-power-bi.aspx.

Encrypting data

Database encryption is becoming a more common security requirement for many organizations. SQL Database now includes the following features for data encryption:

- **Transparent Data Encryption (TDE)** TDE encrypts the underlying database files. No one having physical access to the files can read the data without also having the encryption key.
- **Cell-Level Encryption (CLE)** By using CLE, you can secure sensitive data, such as Social Security numbers, to prevent anyone from accessing that data without the decryption key.
- **Always Encrypted** You can use a set of client libraries to encrypt and decrypt data in SQL Database and protect your data end to end. The encryption and decryption keys remain under the control of your application.

Important Although each of these features strengthens the security of SQL Database, you must still employ security best practices when developing your application, including limiting access to the people or applications requiring data and enforcing the principle of least privilege in the database.

Transparent Data Encryption

TDE was introduced in SQL Server 2008 as a security measure for data at rest. Until SQL Server 2014, it was the only method available for natively encrypting database backups. TDE encrypts only the physical data files, transaction logs, and backups without directly encrypting data tables. That is, if a user has read permission to a database with TDE enabled, the user can query the database and access all the data without having an encryption key. If you move the encrypted files to another server, no one can open and view them on that server.

TDE for SQL Database uses the same technology built for on-premises SQL Server, but it has been enhanced to support Intel AES-NI hardware acceleration of encryption, which reduces the CPU/DTU overhead of enabling TDE. In addition, it is easier to configure. To enable TDE, open the blade for your SQL Database in the Azure Management Portal, click the All Settings link, and then click Transparent

Data Encryption in the Settings blade. Click the On button for Data Encryption in the Transparent Data Encryption blade, as shown in Figure 8-5.

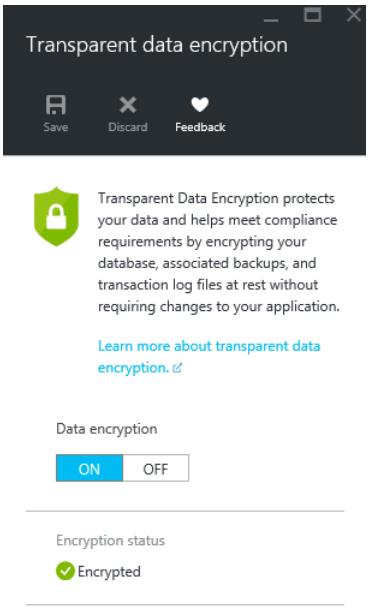


Figure 8-5: Enabling TDE in the Azure Management Portal.

Cell-Level Encryption

When you need to protect sensitive data such as personally identifiable information (PII) or passwords at a more granular level, you can use Cell-Level Encryption (CLE) for SQL Database. It is similar to CLE in on-premises SQL Server but differs in the following ways:

- The SQL Database service controls and manages the root as a certificate rather than by using an instance-specific Service Master Key (SMK).
- The Master Key (MK) does not require a password to simplify disaster recovery for cloud-only applications.

If you copy data into or out of SQL Database by using BACPAC files or the DACFx API, you might experience data loss for encrypted or signed data because there is no metadata to associate symmetric or asymmetric keys with the protected data. As another option, use certificates to encrypt your data and use the KEY_SOURCE and IDENTITY_VALUE fields so that the symmetric keys can easily be re-created.

> **Note** This limitation does not affect physical data-movement scenarios such as backup, database copy, or geo-replication.

Encrypted data in transit

By default, data is unencrypted in transit. Therefore, it is a common security practice to connect to on-premises instances of SQL Server by using connections secured with Secure Socket Layer (SSL) certificates. You do not need to manage certificates for SQL Database because it has a signed certificate issued by a certificate authority and its connections are automatically encrypted by using SSL for the Tabular Data Stream (TDS) transfer of data. However, to further increase security and to eliminate the possibility of man-in-the-middle attacks, set *Encrypt=True* and *TrustServerCertificate=False* in your database ADO.NET connection string.

Securing data at the row level

For managing multitenant databases, a common security requirement is to restrict users from accessing all rows in a table. In previous versions of SQL Database, this Row-Level Security (RLS) feature was available only by implementing extensive custom coding or application logic. In the current version of SQL Database, the restricting logic is in the database tier to protect access to specific rows regardless of the requesting tier. The filtering mechanism for RLS in SQL Database relies on a security predicate filter that is defined as an inline table function. You then create a security policy to enforce this function, just as we describe for RLS in on-premises SQL Server in Chapter 2, "Better security."

Dynamic data masking

Although CLE is a good option for obscuring PII and other sensitive data, you might need to allow some users to see a portion of the data, such as the last four digits of a customer's Social Security number for identification confirmation. Dynamic data masking is a feature in both on-premises SQL Server and SQL Database that we introduce in Chapter 2. Using dynamic data masking, you can keep data secure while enabling the visibility of a portion of the data.

You can configure dynamic data masking in SQL Database by using T-SQL or by using the Azure Management Portal. In the portal, navigate to your SQL Database, click All Settings, and then, in the Settings blade, select Dynamic Data Masking. In the Dynamic Data Masking blade, shown in Figure 8-7, a list of masking rules is displayed in addition to a list of columns for which data masking is recommended. You can enable data masking on those columns by clicking the Add Mask button to the right of the column name, or create a new mask by clicking the Add Mask button at the top of the blade.

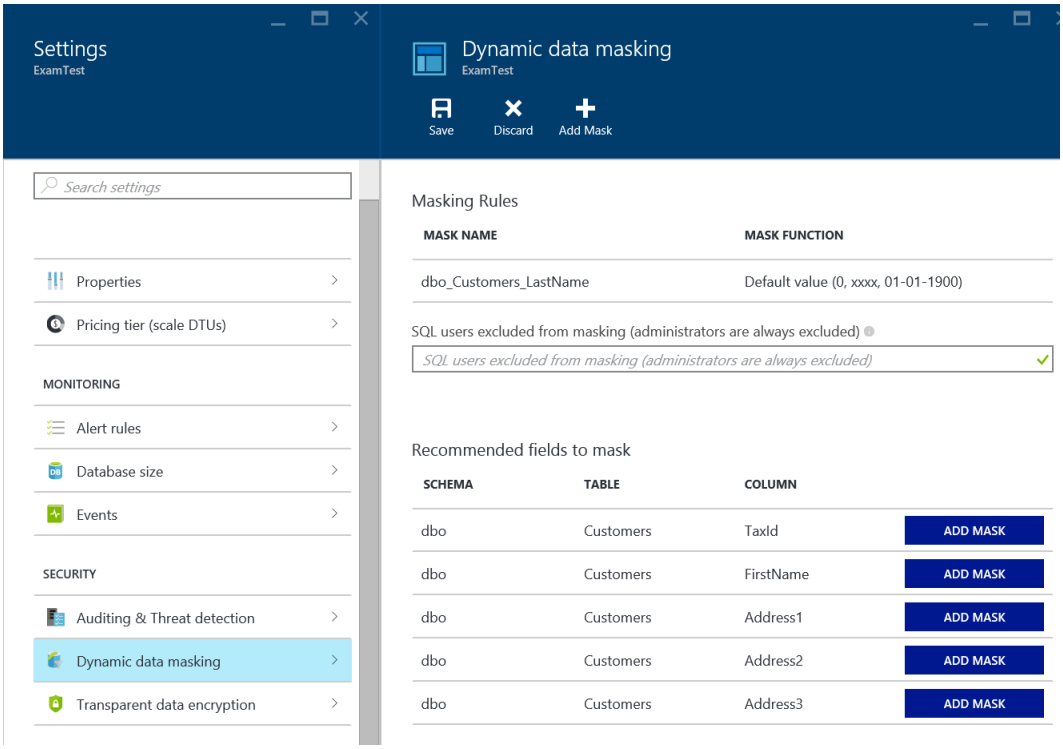


Figure 8-6: Configuring dynamic data masking for a SQL Database in the Azure Management Portal.

After specifying the mask function to apply to selected columns, click the Save button at the top of the blade to save the configuration changes to your SQL Database. After saving these changes, users

can no longer see the masked data in the SQL Database tables unless they have the unmask privilege within the database.

Developing secure applications

The development of secure applications requires many decisions, including how to best manage encryption keys and where to encrypt data in the application. TDE, CLE, and dynamic data masking are all useful features, but you must also consider how encrypting data affects the design of existing implementations in order to provide full protection for your data. To address this need, SQL Database now includes the Always Encrypted feature, which introduces a set of client libraries to allow operations on encrypted data transparently inside an application. This feature makes it easier to protect data by transparently encrypting the data in the client and keeping it encrypted throughout the rest of the application stack. Furthermore, your existing application requires only minimal changes because this level of security is provided by an ADO.NET client library. That way, you configure encryption in the application layer and Always Encrypted ensures your data is encrypted in all layers of the application, whether it is at rest or in motion. Implementation of Always Encrypted for SQL Database is the same as it is for on-premises SQL Server, as described in Chapter 2.

Elastic database features

Microsoft has introduced elastic database features into SQL Database to simplify the implementation and management of software-as-a-service (SaaS) solutions. To optimize and simplify the management of your application, use one or more of the following features:

- **Elastic scale** This feature allows you to grow and shrink the capacity of your database to accommodate different application requirements. One way to manage elasticity is to partition your data across a number of identically structured databases by using a technique called *sharding*. You use the elastic database tools to easily implement sharding in your database.
- **Elastic database pool** Rather than explicitly allocate DTUs to a SQL Database, you can use an elastic database pool to allocate a common pool of DTU resources to share across multiple databases. That way you can support multiple types of workloads on demand without monitoring your databases individually for changes in performance requirements that necessitate intervention.
- **Elastic database jobs** You use an elastic database job to execute a T-SQL script against all databases in an elastic database pool to simplify administration for repetitive tasks such as rebuilding indexes. SQL Database automatically scales your script and applies built-in retry logic when necessary.
- **Elastic query** When you need to combine data from multiple databases, you can create a single connection string and execute a single query. SQL Database then aggregates the data into one result set.

Managing elastic scale

Sharding is not a new concept, but it has traditionally been challenging to implement because it often requires custom code and adds complexity to the application layer. Elastic database tools are available to simplify creating and managing sharded applications in SQL Database by using an elastic database client library or the Split-Merge service. These tools are useful whether you distribute your database across multiple shards or implement one shard per end customer, as shown in Figure 8-7.

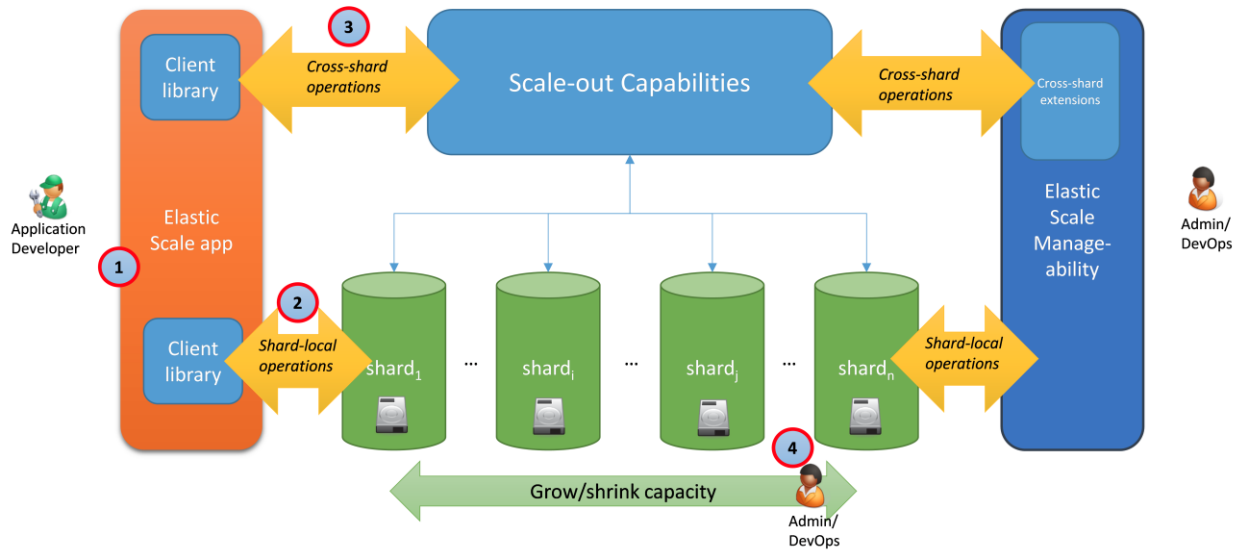


Figure 8-7: Using elastic database tools to manage sharding.

You should consider sharding your application if either of the following conditions apply:

- The size of application data exceeds the limitations of SQL Database.
- Different shards of the database must reside in different geographies for compliance, performance, or geopolitical reasons.

Note The elastic database client library requires Visual Studio 2012 (or higher), C#, and Nuget 2.7 (or higher). You can find information about acquiring and using the elastic database tools at "Deploy a split-merge service," https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-scale-configure-deploy-split-and-merge/#download-the-Split-Merge-packages, and at "Get started with Elastic Database tools," https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-scale-get-started/.

Elastic database client library

The elastic database client library is a Microsoft Visual Studio add-in that integrates into Entity Framework. You use this library to implement standard sharding patterns. In addition to enabling the data tier of your application to scale in and scale out, you can use the client library to manage metadata, connections, and queries for sharded data.

The elastic database client library supports the following features:

- **Shard map management** You use this feature to manage the shard metadata. After registering databases as shards, you define a shard map manager that directs applications requesting connections to the correct shard based on a *sharding key* or range of keys. A sharding key is a data element representing the shard to which the data is assigned, such as a customer ID number, which optimally is a single value to keep related transactions contained in a single database.
- **Data-dependent routing** This feature uses the data in a query to determine the connection to the correct shard so that you no longer need to define this connection in your application.
- **Multishard querying** This feature processes a query against separate shards in parallel and merges the results into a single result set using UNION ALL syntax. It also contains built-in retry logic.

- **Shard elasticity** This feature dynamically adjusts the allocated resources to accommodate the current workload, whether horizontal scaling of many shards or vertical scaling of individual shards is necessary. When your application no longer needs the additional resources, the database shrinks back to its normal state.

Split-Merge service

The Split-Merge service makes it easier to split and merge data across shards. You use it to manage scale-out and scale-in by adding and removing databases from your set of databases (known as your shard set) and redistributing the data afterward. A typical scenario is a single database supporting multiple tenants with widely varying workloads that begin to overwhelm the available resources. You can use Split-Merge to redistribute the data into new or less utilized shards when demand is higher, or you can merge the data again when it is possible to reduce the size of the shard set when demand is lower. This service includes shard map management and data-dependent routing.

Managing performance levels in elastic database pools

As we described earlier in this chapter, SQL Database performance is measured in DTUs. Choosing the appropriate service tier and performance level for a multitenant environment with widely varying performance requirements can be a challenge and costly if you overprovision a SQL Database. To address this challenge, you can use an elastic database pool to share a pool of DTUs across multiple SQL Databases. That way, each database gets the necessary resources on demand.

To configure an elastic database pool in the Azure Management Portal, navigate to the server blade and click the New Pool button at the top of the blade. In the Elastic Database Pool blade, assign a name, select a pricing tier (Basic, Standard, or Premium, similar to service tiers for SQL Database), and click Configure Pool to open a new blade in which you click the Add Database button to display a list of databases that you can add to the pool. The Configure Pool blade allows you to specify the maximum eDTUs (the DTUs for an elastic pool) and maximum storage capacity for the pool. Once you add a database to the pool, its resource usage adjusts dynamically to demand and supports bursting so that you no longer need to manage the performance level of that database individually.

You can monitor and manage the elastic database pool settings in the Azure Management Portal by navigating to the server blade and clicking the name of the elastic database pool. The elastic database pool's blade displays summary charts and metrics. In addition, you can view and adjust the elastic database pool's settings by clicking the Configure Pool button in the elastic database pool blade to open the Configure Performance blade, shown in Figure 8-8.

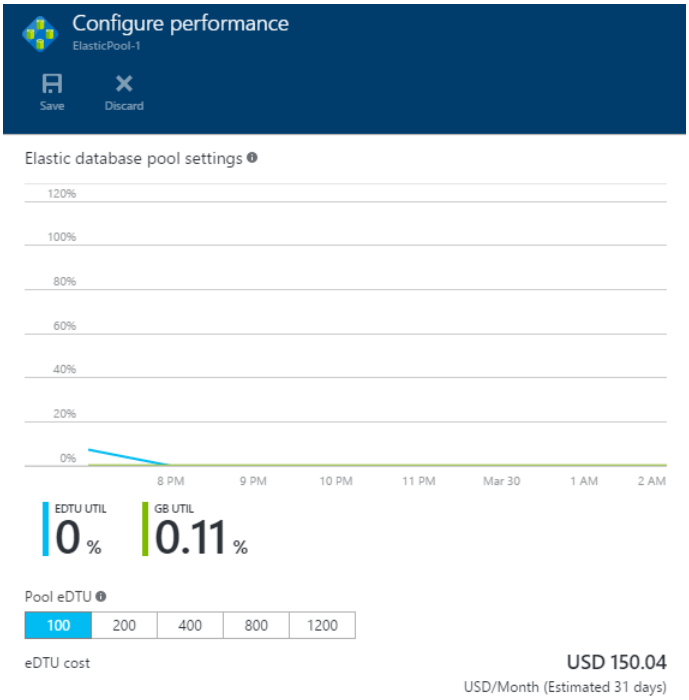


Figure 8-8: Elastic database pool settings in Azure Management Portal.

Note You can also configure an elastic database pool by using PowerShell or REST APIs. To learn more, see "SQL Database elastic database pool reference" at https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-pool-reference/.

Using elastic database jobs

Normally when you need to execute a T-SQL statement to perform an administrative task for multiple SQL Databases, you must log in to each database separately to execute the script. However, after you add databases to an elastic pool, you can execute that same script across all databases in the same elastic pool, in parallel and with built-in retry logic. You can use elastic database jobs to collect usage metrics from all databases, update reference data across databases, or implement schema changes, to name a few.

To get started, navigate to the elastic database pool shard in the Azure Management Portal. You must click the Preview Terms link to accept the terms and install the following Azure components: Cloud Service, SQL Database, Service Bus, and Storage. Then click Job Credentials to supply a user name and password that Azure uses to log in to a new SQL Database, known as the control database, used for storing metadata about the elastic database jobs. Once the components are installed, you can click Create Job at the top of the elastic database pool shard, supply the credentials for the control database, provide a name for the job, type in credentials to connect to the target database (with permissions to execute the script), and paste or type in the T-SQL script. After you click Save, you can click Run to execute the script.

The elastic database pool shard includes a Manage Jobs button to open a new shard in which you can see whether a job is running or has completed or failed. If you click a failed job, you can see its execution log.

Note Alternatively, you can use PowerShell APIs to create and manage jobs. More information is available in "Installing Elastic Database jobs overview" at https://azure.microsoft.com/en-

us/documentation/articles/sql-database-elastic-jobs-service-installation/ and "Create and manage a SQL Database elastic database jobs using PowerShell" at https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-jobs-powershell/.

Distributing queries with elastic query

An elastic query can execute a T-SQL query across multiple SQL Databases by using a single connection, without requiring those databases to coexist in a single elastic database pool. You can use elastic query with any Standard or Premium SQL Database, whether or not the database is in an elastic database pool.

Elastic query is particularly useful for returning the combined results from multiple databases to reporting or data-integration tools. Because Azure transparently handles the distributed query processing, users and applications see only one database, using any driver compatible with SQL Database, as shown in Figure 8-9.

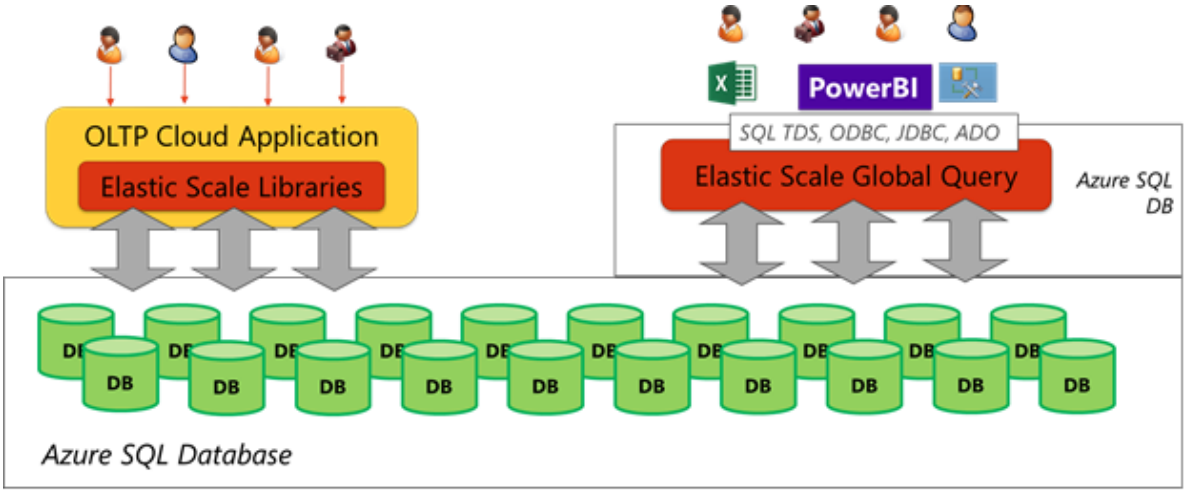


Figure 8-9: Executing an elastic query.

Note To use elastic query, you must create a database-scoped master key and credentials to connect to the shard map manager and the shards and then create an external data source and external tables. For more information, see "Getting started with cross-database queries (vertical partitioning)" at https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-query-getting-started-vertical/ and "Getting Started with elastic queries for sharding (horizontal partitioning)" at https://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-query-getting-started/.

CHAPTER 9

Introducing Azure SQL Data Warehouse

Cloud computing democratizes many technology infrastructure paradigms that once were available only to the largest of enterprises. In the past, options like multiple site disaster recovery or massively parallel computing data-warehouse appliances were beyond the reach of many smaller organizations. In the same way that Azure Infrastructure as a Service (IaaS) allows organizations to take advantage of multiple data centers without maintaining the requisite infrastructure themselves, Azure SQL Data Warehouse brings the power of Microsoft Analytics Platform System (APS) to the cloud and makes this massively parallel computing platform more accessible to organizations of all sizes. Whether you manage gigabytes or petabytes of data, or you need to provide data-warehouse access online at all times or just a few hours a week, SQL Data Warehouse can meet your data-warehousing needs.

Introduction to SQL Data Warehouse

SQL Data Warehouse adds cost-effective massively parallel processing (MPP) as a new capability available in Azure. It supports petabyte scale and can also scale up and down as your workloads

change. Whereas SQL Database is designed for OLTP systems and small organizational data marts, SQL Data Warehouse is built and optimized for systems with at least a terabyte of data, with the ability to scale out far beyond that. It uses the same performance optimizations built into APS that allow queries to scale out and take advantage of the parallel computation of complex, analytic queries.

There are four key components in the SQL Data Warehouse architecture, as shown in Figure 9-1:

- **Control node** You connect to SQL Data Warehouse through the control node, whether you are using business-intelligence, development, or data-loading tools. The control node is a SQL Database that coordinates the data movement and computational workloads in the system. The control node receives the queries and breaks them into units of work to be split across the compute nodes.
- **Compute nodes** The compute nodes are separate SQL Databases that provide the computing power for SQL Data Warehouse. The control node evenly distributes data across compute nodes when loading new data into the warehouse. For other operations, the control node assigns operations to each individual database based on the data it contains. Each compute node returns its partial results back to the control node, which assembles the final results.
- **Storage** All SQL Data Warehouse data is stored in Azure Blob Storage. This object-based storage service offers nearly limitless and transparent scale that allows the warehouse to scale storage operations separately from compute operations. An additional benefit is the ability to pause your data warehouse and persist the data in blob storage. That means you pay for computing resources only when they are in use. Furthermore, because blob storage is highly fault tolerant, it provides additional layers of protection for your data.
- **Data movement services** Data movement services (DMS) transparently move data between nodes as required to respond to user queries. Although this is a background service, you might notice execution plans containing DMS operations.

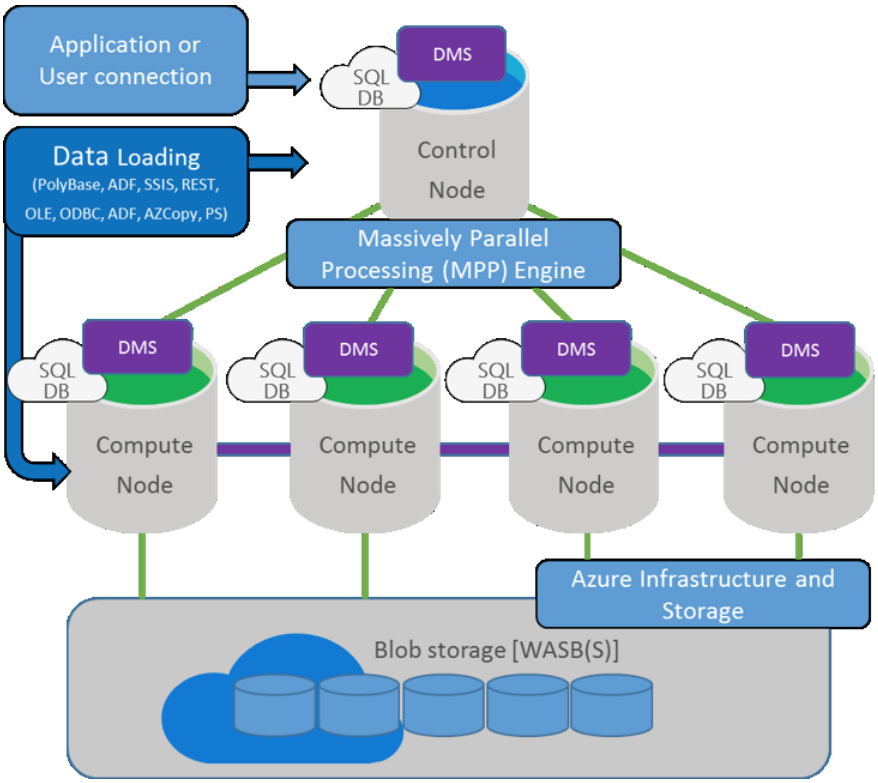


Figure 9-1: SQL Data Warehouse architecture.

SQL Data Warehouse includes many performance optimizations specifically designed for data-warehousing workloads. First, to support its MPP engine, it has a distributed query optimizer that parallelizes data-load and data-retrieval operations. To build an optimal query execution plan, the query optimizer relies on a complex set of statistics that cover the size and distribution of the data. Next, the DMS also uses algorithms and heuristics to move data efficiently among the compute nodes. Another optimization is the use of the clustered columnstore technology found in SQL Server that we describe in Chapter 1, "Faster queries." In a data-warehouse workload, columnstore indexes can compress data up to 5 times more than row-oriented storage and process queries up to 10 times faster. Analytic queries that are typical in data-warehouse workloads perform very well when you implement columnstore indexes.

SQL Data Warehouse also includes data protection. Because SQL Data Warehouse data is kept in blob storage, all data is automatically written to Azure Read-Access Geo-Redundant Storage (RA-GRS). This storage type replicates your data to a second region hundreds of miles away from the primary region, crossing Azure fault and upgrade domains and ensuring that your data is durable even in the event of a complete regional failure or disaster.

While infrastructure solutions are good, these do not protect against inadvertent user error that destroys data. SQL Data Warehouse has a built-in storage snapshot feature that takes a snapshot at least every eight hours and retains them for seven days for a discrete set of restore points. This service is built directly into SQL Data Warehouse and does not need to be configured.

Note For a current list of supported data types and table features that SQL Data Warehouse does not implement, refer to "Migrate your schema to SQL Data Warehouse" at https://azure.microsoft.com/en-us/documentation/articles/sql-data-warehouse-migrate-schema/.

To get started with SQL Data Warehouse, click New in the Azure Management Portal, click Data + Storage, and then click SQL Data Warehouse to open the blade shown in Figure 9-2. You must

provide a database name, set the performance level (which we describe in more detail in the "Scalability" section of this chapter), and choose the subscription, resource group, and server in which to host the data warehouse. You must also select one of the following options as a source:

- **Blank database** When you select this option, your database is empty and ready for you to create tables and other database objects. You can use an existing server in Azure or create a new one if you prefer. This server must be a V12 server.
- **Sample** This option creates a sample database on the AdventureWorksDW so that you can explore the features of SQL Data Warehouse before you build your own.
- **Backup** You can use this option to create a new database from an existing SQL Data Warehouse backup.

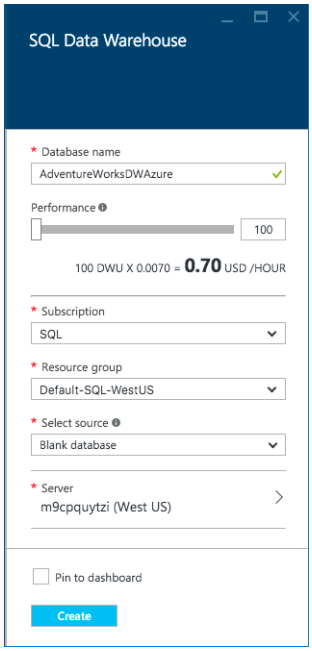


Figure 9-2: Creating a new SQL Data Warehouse.

Note After you create a SQL Data Warehouse, you cannot use SQL Server Management Studio to interact with it. Instead, use Microsoft SQL Server Data Tools 2015 (SSDT), which you can download from https://mdsn.microsoft.com/en-us/mt429383. In SSDT, open SQL Server Object Explorer from the View menu, click the Add SQL Server button, type in the name of the server hosting the SQL Data Warehouse, change the authentication to SQL Server Authentication, provide the server administrator's credentials, select your database, and click Connect.

Security

SQL Data Warehouse has a multilevel security model to support enterprise usage. The first layer of SQL Data Warehouse security is at the connection layer. Like SQL Database, you select IP addresses to allow connections to the data warehouse; a connection that is not listed is blocked from access.

Currently, SQL Data Warehouse supports SQL authentication only. Therefore, you enable user and application access by creating a SQL Server login with a strong password and a database user. You use database roles and permissions to allow access to data, exactly as you do in SQL Server.

You can use the Transparent Data Encryption feature in SQL Data Warehouse to encrypt data at rest, if necessary. You can implement this feature by switching the Data Encryption setting to On for your SQL Data Warehouse in the Azure Management Portal (as we describe in Chapter 8, "Improved SQL Database"). As an alternative, you can use the ALTER DATABASE SET ENCRYPTION ON command in T-SQL.

Auditing for SQL Data Warehouse is similar to the way SQL Database performs auditing, which we describe in Chapter 8. You have the option to enable or disable auditing as needed. Just as with SQL Database, you need an Azure storage account to store auditing events.

Scalability

The advantage of SQL Data Warehouse over a traditional on-premises data warehouse is the ability it gives you to scale the service quickly up or down as your workload demands change. If you experience rapid data or user growth in an on-premises environment, you typically undergo the expensive and time-consuming process of acquiring new hardware or justifying to an infrastructure team the need to add another 10 terabytes (TB) of storage for the data warehouse. With SQL Data Warehouse, you can easily change resource allocations in minutes.

Scaling storage and compute resources

SQL Data Warehouse decouples storage from compute resources by using nearly unlimited blob storage. This lets you effortlessly manage rapid data growth at a much lower cost than on-premises options. This separation of compute resources from the storage component also allows you to pause SQL Data Warehouse. During this time, you pay only for storage, which is a particularly good option for smaller organizations that do not require full-time report processing or for developers who need access to the data warehouse during normal business hours only.

Compute resources are also scalable. The unit of measurement for compute usage in SQL Data Warehouse is a Data Warehouse Unit (DWU), a metric for the underlying computing power required for your warehouse to ensure a standard performance level at any given time. By adjusting the DWUs up or down in increments of 100, you can scale your warehouse without managing the underlying hardware configuration. In the background, Microsoft runs performance-benchmark tests to determine how best to improve the hardware configuration and architecture of the service without impacting the performance of your workload.

Specifically, DWUs are a composite of three separate metrics that correlate highly with overall data-warehouse performance. Increasing the number of DWUs results in a linear increase in performance as measured by the following metrics:

- **Scan/Aggregation** This metric measures the time required to execute a standard data-warehouse query that scans a large number of rows and then performs a complex aggregation. It benchmarks the CPU and I/O (storage) capacity of the service.
- **Load** This metric benchmarks network and CPU performance by measuring the data-load process when using PolyBase to load a representative data set from blob storage into the data warehouse.
- **CREATE TABLE AS SELECT (CTAS)** CTAS is another benchmark of network and CPU performance. It measures the ability to create a copy of a table by reading data from storage, distributing it across the compute nodes, and writing it back to storage.

Deciding how to scale

With all of these factors in mind, how do you know how to size your initial data warehouse and when to scale up and scale down to match your workload and optimize costs? The concept of DWUs is designed to be as simple as possible for your performance needs. When you need faster results from your queries, increase the number of DWUs. When you need less power, decrease the number to reduce costs. As you plan your scaling strategy, consider the following scenarios:

- Data-warehouse usage patterns tend to correlate with business-activity patterns. For example, usage typically increases at the end of each week and month. During those times, you need more DWUs to service those users than at other times. You can automate elastic scaling by creating a script to increase the number of DWUs during peak usage and decrease them at other times. You can also pause your data warehouse at night, and thereby reduce overall compute charges, but continue to maintain data in storage until it is needed during business hours.
- During a period of heavy data-loading operations, you might increase DWUs so that users can access data more quickly.
- Each time you change the number of DWUs, run a few queries to baseline the new performance level. The process to scale up or down is easy and fast, which means you can experiment with different settings without committing to more than an hour of compute time with a specific setting.

> **Note** If your database is less than 1 TB in size, you might not see expected performance benefits from SQL Data Warehouse because most of its optimizations are designed for larger data sets. Microsoft recommends that you use SQL Database for data volumes that are less than 1 TB.

To scale your SQL Data Warehouse up or down, open the SQL Data Warehouse blade and click Scale. In the Scale blade, use the slider to adjust the number of DWUs, or type a number in the box to the right of the slider. You must use a value that is divisible by 100 or the performance level does not change in the blade. Click Save to permanently save the new performance level.

Data loads

Traditional data warehousing has two distinct usage patterns—large analytics queries and bulk loading of data from operational systems. The process to load data is commonly known as extract, transform, and load (ETL). To perform ETL processing for SQL Data Warehouse, you have the following options:

- Azure Data Factory
- PolyBase
- SQL Server Integration Services (SSIS)

> **Note** To use the first two options, you might need to use the AzCopy utility that you can download from https://azure.microsoft.com/en-us/documentation/articles/storage-use-azcopy/. You use this command-line utility to copy data from your on-premises environment to blob storage. If you need to load five or more terabytes into blob storage, you should instead use the Azure Import/Export service. When you use this service, you ship hard drives to Microsoft for loading directly into storage, which might be necessary for the initial loading of your data warehouse. You can learn more about this service at "Use the Microsoft Azure Import/Export Service to Transfer Data to Blob Storage," https://azure.microsoft.com/en-us/documentation/articles/storage-import-export-service/.

Using Azure Data Factory

Azure Data Factory (ADF) is a cloud-based data integration service that manages the movement and transformation of data elements between various Azure services on a scheduled basis. ADF pricing is based on the frequency of the activities that you run and whether an activity runs on-premises or in the cloud. In addition, when you use the Copy Activity, the cost depends on the location of the data source and the amount of time required to copy the data.

> **Note** You can find current information about ADF pricing at "Data Factory Pricing," https://azure.microsoft.com/en-us/pricing/details/data-factory/.

ADF includes the following components, shown in Figure 9-3:

- **Datasets** A dataset is any type of data that you reference in an activity, such as a table or a document.
- **Activities** An activity is an action that affects data, using zero or more datasets as input and creating one or more datasets as output. We describe the Copy and Stored Procedure activities later in this section, but there are several other activities available. For example, you can use the Hive or Piq activity to execute Hive or Pig queries on an HDInsight cluster or use the Batch Execution activity to invoke the Azure Machine Learning web service to apply a predictive model to a batch of data, to name a few.
- **Pipelines** A pipeline is a logical grouping of activities that collectively perform a task. You can manage and schedule this group of activities as a single item rather than work with each activity individually.
- **Linked services** A linked service is a connection to an external service such as a data store hosting a dataset or a compute resource required by an activity.

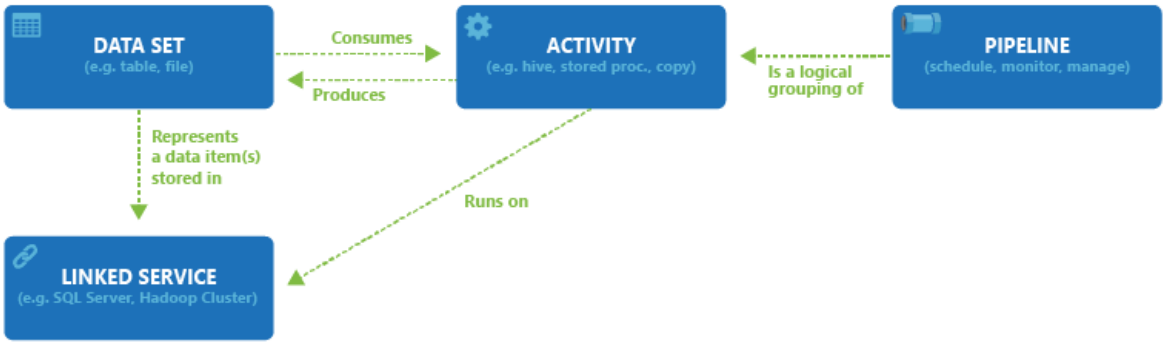


Figure 9-3: Components and relationships between components in ADF.

> **Note** To use Azure Data Factory with your SQL Data Warehouse, the firewall settings for the SQL Server instance must be configured to allow access to Azure services as described in Chapter 8.
>
> You can also use Visual Studio with the Azure Software Development Kit (SDK) to create objects for ADF as described at "Tutorial: Create a pipeline with Copy Activity using Visual Studio," https://azure.microsoft.com/en-us/documentation/articles/data-factory-get-started-using-vs/. Another option is to use PowerShell as described at "Tutorial: Create a pipeline with Copy Activity using Azure PowerShell," https://azure.microsoft.com/en-us/documentation/articles/data-factory-monitor-manage-using-powershell/.
>
> As an alternative to writing code, you can use the Copy Wizard to create the necessary linked services, datasets, and pipeline. Learn more at "Tutorial: Create a pipeline with Copy Activity using

Data Factory Coy Wizard," https://azure.microsoft.com/en-us/documentation/articles/data-factory-copy-data-wizard-tutorial/.

The Copy Activity is useful for moving data from a data source to a destination. This activity supports a wide variety of data stores. However, when you use the Copy Activity to move data into your SQL Data Warehouse, you are limited to the following data sources:

- Blob storage
- SQL Database
- On-premises SQL Server
- SQL Server on IaaS

Note You can find a complete list of supported data sources and destinations for the Copy Activity at "Data Movement Activities," https://azure.microsoft.com/en-us/documentation/articles/data-factory-data-movement-activities/.

As an example, let's say that you want to load data from an on-premises SQL Server instance into SQL Data Warehouse. In the Azure Management Portal, click New, click Data + Analytics, and then click Data Factory. In the New Data Factory blade, provide a name for the data factory; optionally change the subscription, resource group, and region; and then click Create. After you receive notification that the data factory is ready, open the Data Factory blade, and then click the Author And Deploy tile to open the Editor where you perform the remaining steps.

Next you must add a data gateway to connect Azure to your on-premises network. To do this, click the More Commands button in the toolbar, and then click New Data Gateway. In the Create blade, provide a name for the data gateway and click OK. You then have the option to perform an express setup by clicking the Install Directly On This Computer link or a manual setup by clicking the Download And Install Data Gateway link. If you choose the manual setup option, you must copy the key that is displayed in the Configure blade and use it when you configure the gateway, whereas the express setup configures the key automatically. An icon displays the gateway status in the notification area on the taskbar. You can proceed when the icon's ScreenTip displays "Data Management Gateway is running."

Note For installation considerations and prerequisites for using Data Management Gateway, refer to "Move data between on-premises sources and cloud with Data Management Gateway" at https://azure.microsoft.com/en-us/documentation/articles/data-factory-move-data-between-onprem-and-cloud/.

Next, you create a linked service for the source. Start by clicking the New Data Store button in the toolbar and selecting SQL Server. The JSON template for an on-premises SQL Server, as shown in Example 9-1, is displayed in the Drafts blade. Update the template with the server name, database name, and credentials. You must also have a gateway configured to allow ADF to connect to your on-premises server (as described at "Move data between on-premises sources and cloud with Data Management Gateway," https://azure.microsoft.com/en-us/documentation/articles/data-factory-move-data-between-onprem-and-cloud/*)* and supply the name of the gateway in the JSON template. Click Deploy on the toolbar to create the linked service.

Example 9-1: Creating a linked service to an on-premises SQL Server

```
{
  "name": "SqlServerLinkedService",
  "properties": {
```

```
      "type": "OnPremisesSqlServer",
      "description": "",
      "typeProperties": {
          "connectionString": "Data Source=<servername - required for credential
             encryption>;Initial Catalog=<databasename - required for credential
             encryption>;Integrated Security=False;User ID=<username>;Password=<password>;",
           "gatewayName": "<Name of the gateway that the Data Factory service should use to
              connect to the on-premises SQL Server database - required for credential
              encryption>",
           "userName": "<Specify user name if you are using Windows Authentication>",
           "password": "<Specify password for the user account>"
        }
    }
}
```

You need another linked service for the SQL Data Warehouse as the destination. Click New Data Store, and then select Azure SQL Data Warehouse to open the JSON template shown in Example 9-2. Update the template with the correct server name, database name, and credentials required to access the data warehouse, and then click Deploy to create the destination linked service.

Example 9-2: Creating a linked service to SQL Data Warehouse

```
{
  "name": "AzureSqlDWLinkedService",
  "properties": {
      "type": "AzureSqlDW",
      "description": "",
      "typeProperties": {
         "connectionString": "Data Source=tcp:<servername>.database.windows.net,1433;Initial
          Catalog=<databasename>;User ID=<username>@<servername>;Password=<password>;Integrated
          Security=False;Encrypt=True;Connect Timeout=30"
        }
    }
}
```

Your next step is to create datasets. You need one to represent the data to retrieve from a single table in the on-premises SQL Server and another to represent the data to insert into a corresponding table in your SQL Data Warehouse. If you need to load data from multiple tables, you need to create additional dataset pairs.

To continue, click the New Dataset button on the toolbar and select SQL Server Table. Then update the JSON template to include the correct linked service name and the table name to use as a source, as shown in Example 9-3. You must also specify a frequency of minute, hour, day, week, or month and an interval for the frequency. For example, if the frequency is Day and the interval is 1, ADF generates one *data slice* per day. A data slice is a scheduled instance of pipeline execution associated with a time period, and optionally with a defined date range from which to select data (as we describe later in this section). Set the *external* property to *true* to flag the dataset as an external table instead of one that is produced by ADF and deploy the template.

Example 9-3: Creating a dataset for an on-premises SQL Server table

```
{
    "name": "SQLServerDatasetInput",
    "properties": {
        "type": "SqlServerTable",
        "linkedServiceName": "SqlServerLinkedService",
```

```
        "structure": [],
        "typeProperties": {
            "tableName": "FactSurveyResponse"
        },
        "availability": {
            "frequency": "Hour",
            "interval": 1
        },
        "external": true
    }
}
```

Repeat the process for a SQL Data Warehouse table as shown in Example 9-4. The properties are similar in the templates for SQL Server and SQL Data Warehouse, but the type is *AzureSqlDWTable* for SQL Data Warehouse instead of *SQLServerTable,* as it is for SQL Server. Furthermore, you omit the *external* property for the output dataset because ADF generates it in this case.

Example 9-4: Creating a dataset for SQL Data Warehouse

```
{
    "name": "SQLDWOutput",
    "properties": {
        "type": "AzureSqlDWTable",
        "linkedServiceName": "AzureSqlDWLinkedService",
        "structure": [],
        "typeProperties": {
            "tableName": "FactSurveyResponse"
        },
        "availability": {
            "frequency": "Hour",
            "interval": 1
        }
    }
}
```

Now you can create a pipeline to manage the copy from the source to the destination. Click the New Pipeline button on the toolbar to display the JSON template. You can update the properties to reflect the name and description of the pipeline, add a Copy Activity, and define the start and end date/time for running the activity, as shown in Example 9-5. In this example, a query on the source dataset uses the ADF system variables *WindowStart* and *WindowEnd* to filter the data selection based on the start and end of the time interval for the current data slice.

Example 9-5: Creating a pipeline to copy data

```
{
  "name": "SQLServerToSQLDWPipeline",
  "properties": {
    "description": "Copy data from on-premises SQL Server to SQL Data Warehouse every hour",
    "activities": [
      {
        "type": "Copy",
        "name": "SQLtoSQLDW",
        "description": "Copy Activity",
        "typeProperties": {
          “source": {
            "type": "SqlSource",
            "sqlReaderQuery": "$$Text.Format('select * from FactSurveyResponse where Date >=
```

```
            \\'{0:yyyy-MM-dd HH:mm}\\' AND timestampcolumn < \\'{1:yyyy-MM-dd HH:mm}\\'',
            WindowStart, WindowEnd)"
          },
          "sink": {
            "type": "SqlSink"}
        },
        "inputs": [
          {
            "name": "SQLServerDatasetInput"
          }
        ],
        "outputs": [
          {
            "name": "SQLDWOutput"
           }
        ],
        "scheduler": {
          "frequency": "Hour",
          "interval": 1
        }
      }
    ],
    "start": "2016-04-01T00:00:00Z",
    "end": "2016-04-30T01:00:00Z"
  }
}
```

Note There are many ways to configure a Copy Activity depending on the source and destination and the whether you want to perform transformations. You can learn more at "Data Movement Activities," https://azure.microsoft.com/en-us/documentation/articles/data-factory-data-movement-activities/, and at "Transform and analyze using Azure Data Factory," https://azure.microsoft.com/en-us/documentation/articles/data-factory-data-transformation-activities/.

When you finish, you can close the Editor blade and the associated data factory blade to return to the data factory blade in which you see the Actions, Contents, and Operations tile collections. Click the Diagram tile in the Actions collection to view the pipeline and the related objects, as shown in Figure 9-4. When you right-click the pipeline, you can select Open Pipeline to see the activities in that pipeline and double-click an activity to open a new blade with summary information about the activity and a tile that is linked to the activity script and related datasets.

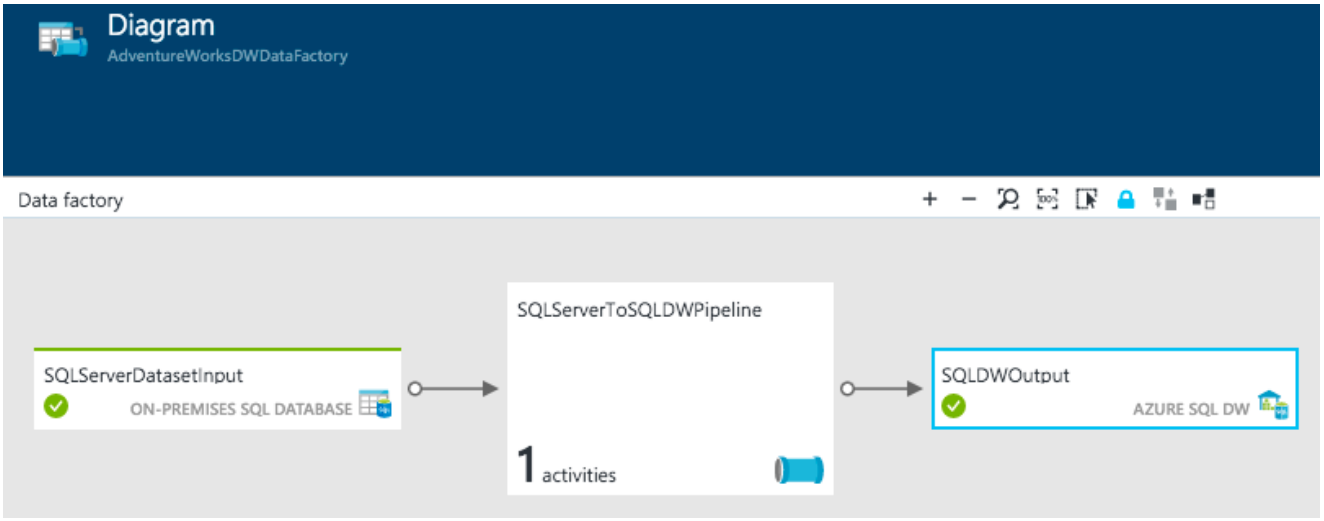


Figure 9-4: Reviewing a data factory's Diagram blade in the Azure Management Portal.

When you click a dataset tile, whether in the Diagram blade or in the data factory blade, you can see the status of pipeline execution for each data slice, as shown in Figure 9-5. For each time interval between the start and end date and time of the pipeline, Azure creates a data slice, and the scheduler executes the activity for the data slice having a date and time less than or equal to the current UTC time. If you scroll down the dataset blade, you can find a table that lists failed slices, if any. The Ready status indicates that ADF has successfully processed the data slice.

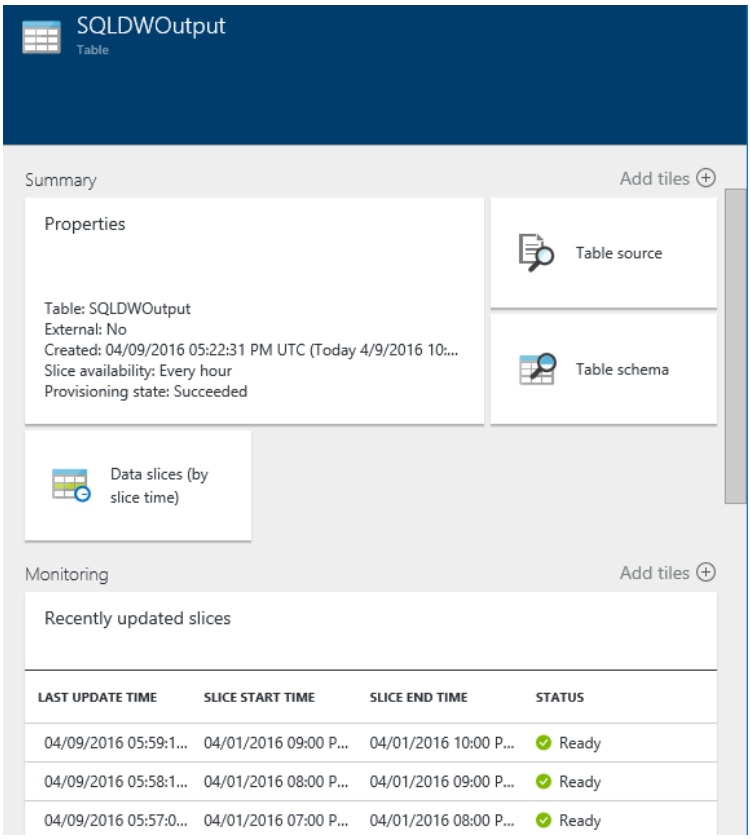


Figure 9-5: Reviewing a data factory's dataset blade in the Azure Management Portal.

Note You can also click the Monitor & Manage tile in the data factory blade to open a new browser tab containing a view of data factory assets and the status of activities, among other information. You can learn more about using this monitoring and management app at "Monitor and manage Azure Data Factory pipelines using new Monitoring and Management App," https://azure.microsoft.com/en-us/documentation/articles/data-factory-monitor-manage-app/.

You can also use ADF to execute stored procedures in your SQL Data Warehouse on a scheduled basis. To do this, you use the Stored Procedure activity in a pipeline. First, use the CREATE PROCEDURE statement in SQL Data Warehouse as you would in SQL Server. Then, in your data factory, define a linked service for your SQL Data Warehouse and an output dataset to hold the results of the stored procedure, which you can then use in a downstream process. At the time of this writing, an output dataset is required even if your stored procedure writes data directly into a table. In that case, create a dummy dataset to use as the output.

Next, create a pipeline to execute the stored procedure, as shown in Example 9-6. The optional *storedProcedureParameters* property contains a list of the stored procedure's parameter names and values. Note that the parameter name is case-sensitive and that you can use an ADF variable like *SliceStart* to pass the execution date and time to the stored procedure. You can also pass a static value by enclosing the value in double quotation marks.

Example 9-6: Creating a pipeline to execute a stored procedure

```
{
  "name": "SQLDWStoredProcPipeline",
  "properties": {
    "description": "Execute a stored procedure in SQL Data Warehouse every day",
    "activities": [
      {
        "type": "SqlServerStoredProcedure",
        "name": "SQLDWStoredProc",
        "description": "Stored procedure activity",
        "typeProperties": {
          "storedProcedureName": "sp_DoSomething",
          "storedProcedureParameters": {
            "DateTime": "$$Text.Format('{0:yyyy-MM-dd HH:mm:ss}', SliceStart)"
          }
        },
        "outputs": [
          {
            "name": "SQLDWStoredProcOutput"
          }
        ],
        "scheduler": {
          "frequency": "Day",
          "interval": 1
        }
      }
    ],
    "start": "2016-04-01T00:00:00Z",
    "end": "2016-04-15T01:00:00Z"
  }
}
```

Note To learn more about using ADF to execute a stored procedure in SQL Data Warehouse, see "SQL Server Stored Procedure Activity" at https://azure.microsoft.com/en-us/documentation/articles/data-factory-stored-proc-activity/.

Using PolyBase

Another option for loading data into SQL Data Warehouse is PolyBase, a SQL Server feature that we describe in Chapter 5, "Broader data access." You can use it to query text or CSV files or nonrelational data stored in Hadoop or blob storage and join the data queried to query results from your data warehouse. You can also use PolyBase to migrate data from one of these storage types into the data warehouse. To do this, PolyBase uses external tables that you define in the SQL Data Warehouse and then allows you to query these tables by using T-SQL.

Compared to other options, PolyBase is the fastest and most scalable of the loading methods. In addition, PolyBase automatically parallelizes the data-load process, thereby eliminating the need to explicitly create subsets of your data for multiple data loads that you run concurrently. Nonetheless, the use of PolyBase still requires some new development for your workloads as we describe later in this section. However, the load speed is fast because PolyBase bypasses the control node and loads data directly to the data nodes, as shown in Figure 9-6. The other load methods that we discuss in this chapter cannot access the data nodes directly and consequently run more slowly.

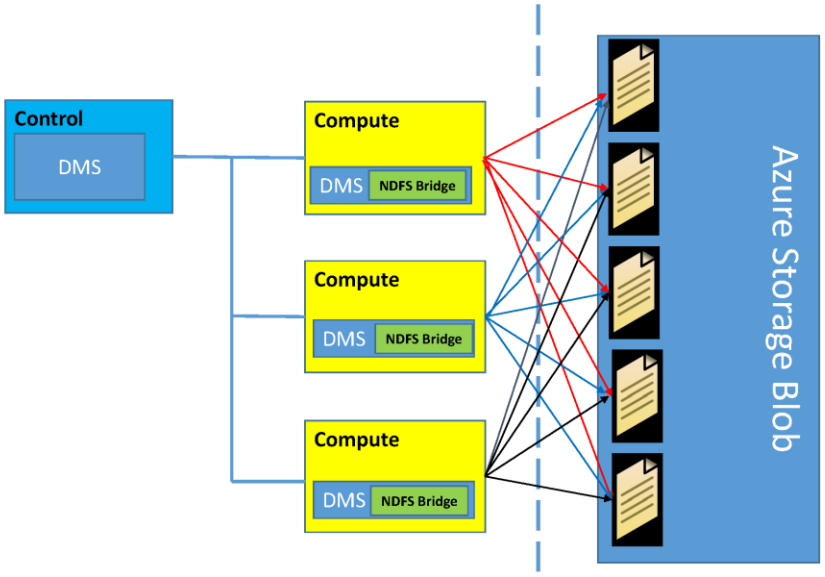


Figure 9-6: Parallel loading of SQL Data Warehouse from Azure Blob Storage.

As an example of using PolyBase for data loading, let's assume that you want to add new sales territories from blob storage into your data warehouse. The data is currently in a comma-delimited text file named **dimsalesterritory.txt,** which looks like this:

```
11,Netherlands,Netherlands,Europe
12,Denmark,Denmark,Europe
```

Navigate to the storage account blade into which you want to copy the text file, and then click the Blobs tile. Take note of the blob-service endpoint in the Essentials section of the Blob Service blade. Then, in the storage account blade, click the Settings button, and then click Access Keys in the

Settings blade. Click the Copy button to the right of one of the keys to store an access key on the clipboard.

Next, use the AzCopy utility to upload your text file. To do this, open a command-prompt window and go to the AzCopy installation directory like this:

```
cd /d "%ProgramFiles(x86)%\Microsoft SDKs\Azure\AzCopy"
```

Then, after replacing the placeholder tokens with the blob service endpoint and storage account key, run the following AzCopy command:

```
.\AzCopy.exe /Source:C:\Temp\ /Dest:<blob service endpoint URL>/datacontainer/dimsalesterritory/
/DestKey:<azure_storage_account_key> /Pattern:DimSalesTerritory.txt
```

Now open SSDT, connect to your SQL Data Warehouse in SQL Server Object Explorer, and then right-click the database and click New Query. Next, you must configure authentication as shown in Example 9-6 to create a master key and add a database-scoped credential to your SQL Data Warehouse that PolyBase uses to access the data in blob storage. In the CREATE DATABASE SCOPED CREDENTIAL statement, you can use any string value for IDENTITY because it is not used during authentication, but replace the placeholder token for SECRET with the same storage account key that you used to copy data into blob storage.

Example 9-6: Configuring authentication for access to blob storage

```
-- Create a master key if one does not already exist
CREATE MASTER KEY;
-- Create a database-scoped credential
CREATE DATABASE SCOPED CREDENTIAL AzureDWDataStorageCredential
WITH
    IDENTITY = 'user',
    SECRET = '<azure_storage_account_key>';
```

Then you need to create the following external objects in SQL Data Warehouse, as shown in Example 9-7:

- **External data source** Identifies where to find your blob storage. The LOCATION argument requires the name of the blob container and storage account into which you uploaded your data. The CREDENTIAL argument uses the database-scoped credential that you added to the SQL Data Warehouse earlier. For more information about creating an external data source and about using PolyBase to access a Hadoop source, see https://msdn.microsoft.com/en-us/library/dn935022.aspx.
- **External file format** Describes the structure of your data. The FORMAT_TYPE argument takes one of the following values: DELIMITEDTEXT, RCFILE, ORC, or PARQUET. For more information about creating an external file format and the various ways to use define formats, see https://msdn.microsoft.com/en-us/library/dn935026.aspx.
- **External table** Provides the table definition and location of data. Here you must include the column names and data types, provide the directory relative to the blob storage container that holds your data, reference the external data source, and specify the file format. For more information about creating an external table, see https://msdn.microsoft.com/en-us/library/dn935021.aspx.

Example 9-7: Creating external objects

```
-- Create an external data source
CREATE EXTERNAL DATA SOURCE AzureDWDataStorage
WITH (
    TYPE = HADOOP,
    LOCATION =
      'wasbs://datacontainer@<azure_storage_account_name>.blob.core.windows.net',
    CREDENTIAL = AzureDWDataStorageCredential
);
-- Create an external file format
CREATE EXTERNAL FILE FORMAT TextFile
WITH (
    FORMAT_TYPE = DelimitedText,
    FORMAT_OPTIONS (FIELD_TERMINATOR = ',')
);
-- Create an external table
CREATE EXTERNAL TABLE dbo.DimSalesTerritoryExternal (
    SalesTerritoryAlternateKey INT NOT NULL,
    SalesTerritoryRegion NVARCHAR(50) NOT NULL,
    SalesTerritoryGroup NVARCHAR(50) NOT NULL,
    SalesTerritoryCountry NVARCHAR(50) NOT NULL
)
WITH (
    LOCATION='/dimsalesterritory/',
    DATA_SOURCE=AzureDWDataStorage,
    FILE_FORMAT=TextFile
);
```

After creating the external resources, you can view them in SSDT in the SQL Server Object Explorer, as shown in Figure 9-7, and you can execute a query against the external table to read the data directly from the text file in blob storage.

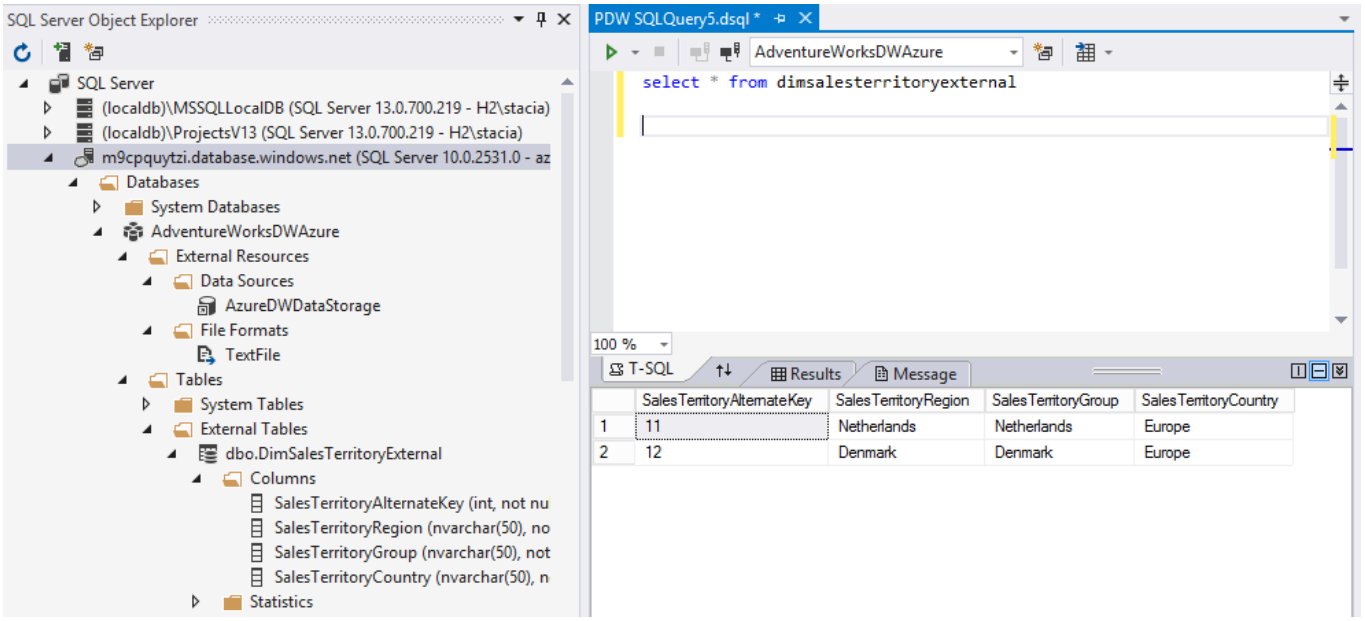


Figure 9-7: Viewing external resources in SSDT.

Although you can use PolyBase to query external tables, query performance is much better if you load the external table data into a SQL Data Warehouse internal table structure by using one of the methods shown in Example 9-8. If you are loading data into a new table, use the CREATE TABLE AS SELECT statement to create a table in SQL Data Warehouse from your external table. Otherwise, use

the INSERT INTO … SELECT FROM … statement. In both cases, you use the ROW_NUMBER function to generate a surrogate key since SQL Data Warehouse does not support identity columns.

Example 9-8: Loading a SQL Data Warehouse table from an external table

```
-- Option 1: Load a new table
CREATE TABLE dbo.DimSalesTerritory2
WITH
(
    CLUSTERED COLUMNSTORE INDEX,
    DISTRIBUTION = ROUND_ROBIN
)
AS
SELECT
  ROW_NUMBER() OVER(ORDER BY (SELECT 1)) AS SalesTerritoryKey,
  SalesTerritoryAlternateKey,
  SalesTerritoryRegion,
  SalesTerritoryGroup,
  SalesTerritoryCountry
FROM dbo.DimSalesTerritoryExternal
-- Option 2: Load an existing table
INSERT INTO dbo.DimSalesTerritory
SELECT
  ROW_NUMBER() OVER(ORDER BY (SELECT 1)) +
    (SELECT ISNULL(MAX(SalesTerritoryKey), 0) SK FROM dbo.DimSalesTerritory)
    AS SalesTerritoryKey,
  SalesTerritoryAlternateKey,
  SalesTerritoryRegion,
  SalesTerritoryGroup,
  SalesTerritoryCountry
FROM dbo.DimSalesTerritoryExternal;
```

Note For best performance, run one PolyBase load operation at a time. You can scale up the load performance by increasing DWUs for the SQL Data Warehouse. For additional guidance related to using PolyBase with SQL Data Warehouse, see "Guide for using PolyBase in SQL Data Warehouse" at https://azure.microsoft.com/en-us/documentation/articles/sql-data-warehouse-load-polybase-guide/.

Using SSIS

A third option for loading data into your SQL Data Warehouse is to use SSIS packages. You design packages much as you do when working directly against SQL Server, except you must use an ADO.NET connection manager when connecting to SQL Data Warehouse. In the ADO.NET Destination, enable Use Bulk Insert When Possible for best performance during the data load.

SSIS is a good option for a one-time load of a small data set. However, unlike the PolyBase loading described in the previous section, SSIS connects only to the control node, which can become a bottleneck if data volume is high.

If you are extracting data from on-premises sources to load into SQL Data Warehouse, design your packages to resume package execution at the point of failure without repeating the work that was complete before the failure. Consider implementing the following strategies into your package design as part of your recovery strategy:

- **Dynamically filter source data** Before the Data Flow Task, add one Execute SQL Task to retrieve the minimum key value from the source table and store it in a variable. In a separate Execute SQL Task, get the maximum key value from the target table and store it in another variable. If there is no maximum key value in the target table because the table is empty, store the value of the minimum key instead. Use that latter variable in a WHERE clause appended to the SELECT statement in the data flow's data source object to filter the data and ensure that you load only new data into the SQL Data Warehouse. Be sure to include an ORDER BY clause so that the data is loaded in key order.
- **Add retry logic** Set up one variable in a loop as a counter for the current try and another variable in which you define a constant as a maximum number of retries. Use these variables in a For Loop Container. Place your Data Flow Task (and another task that you want to retry in case of failure) inside the container. Create a failure precedence constraint after the Data Flow Task that connects to a Script Task that forces the package execution to sleep briefly. When the sleep duration ends, the loop starts again if the maximum number of retries has not yet been reached. You must also add a success precedence constraint after the Data Flow task that connects to an Expression Task that updates the current try counter to a value that forces a break out of the loop.

Statistics for SQL Data Warehouse

Like SQL Server, SQL Data Warehouse uses statistics to compare the costs of performing a query in different ways so that it can generate an optimal query plan. Statistics on a single column describe the range and frequency of values in a single column. This information is useful to the query optimizer for estimating how many rows the query will return. Multicolumn statistics provide information about a list of columns, including the single-column statistics for the first column in the list and a density vector for cross-column correlations. This type of statistics helps the query optimizer improve composite joins and grouping operations.

Although SQL Data Warehouse relies heavily on statistics, it does not automatically manage statistics for newly inserted or updated data. To achieve optimal query performance, create statistics on each column of the table after your initial load and update them after each substantial change to the data.

Note You can use the DBCC SHOW_STATISTICS function in SQL Data Warehouse to understand how up to date the statistics in your warehouse are. To learn more, see "DBCC SHOW_STATISTICS (Transact-SQL)" at https://msdn.microsoft.com/library/ms174384.aspx.

Creating statistics

As you develop your SQL Data Warehouse, be sure to create single-column statistics on each column in your table and multicolumn statistics on the columns that are regularly used in join operations and GROUP BY clauses. If you later find that the process to update the single-column statistics takes too long, consider dropping statistics on columns that are less frequently queried or updating the statistics on those columns less often.

Maintaining statistics

You should have a plan to update statistics on a regular basis in your data warehouse. When the distribution of the data changes, the statistics might no longer be valid and could result in query performance that is less than optimal. Unfortunately, you cannot simply measure how up to date statistics are based on their age. A column in which the number of distinct values increases dramatically after each data load, such as a date column, requires a more frequent update of statistics as compared to a column with a relatively small and static number of distinct values, such as a column for gender, which might never require an update. Consequently, the methodology you use to update

statistics depends on the characteristics of your data. As you develop your plan, consider the following guidelines:

- Update at least one statistics object in each table containing data to update the row and page count information automatically as part of the same operation.
- Regularly review columns that appear often in JOIN, GROUP BY, ORDER BY, and DISTINCT clauses.
- An ascending key column for which new values are added frequently, such as a transaction date column, likely requires more frequent updates in order to include these new values in the statistics histogram.
- If a column has a set of values that remain static, reduce the frequency of updates or eliminate updates altogether.
- Because each statistics object is updated in a series, avoid using UPDATE STATISTICS <TABLE_NAME> for wide tables. Instead, consider using UPDATE STATISTICS <TABLE_NAME> (STATISTIC_OBJECT) to update specific objects as needed.

Integration options

After you load data into SQL Data Warehouse, your users can use a variety of tools to gain deeper insights into that data. SQL Data Warehouse integrates easily with other components of the Microsoft analytics and business-intelligence stack. In this section, we take a closer look at integrating with the following cloud-based tools:

- Azure Machine Learning
- Azure Stream Analytics
- Power BI

Note Considering the volume of data stored in a typical data warehouse and the higher volume of data recommended for SQL Data Warehouse, the use of cloud-based analysis tools helps you avoid both data egress charges and latency associated with bringing data back to on-premises installations.

Predictive modeling in Azure Machine Learning

Azure Machine Learning is a managed analytics service that you can use to build complex predictive models. You can use SQL Data Warehouse as both a source and a destination for your machine-learning models.

To retrieve data from SQL Data Warehouse for your experiment, expand the Data Input and Output module category and then drag the Reader module to the experiment canvas. In the Properties window, shown in Figure 9-8, select Azure SQL Database in the Data Source drop-down list and then provide the following connection details in the respective text boxes: Database Server Name, Database Name, Server User Account Name, Server User Account Password, and Database Query. You have the option to select the Accent Any Server Certificate (Insecure) check box if you want to omit the step to review the site certificate before accessing your data, although this action is not recommended. The Properties window also includes the Use Cached Results check box that you can select if you want to read the data once and then use the cached results for each subsequent run of your experiment.

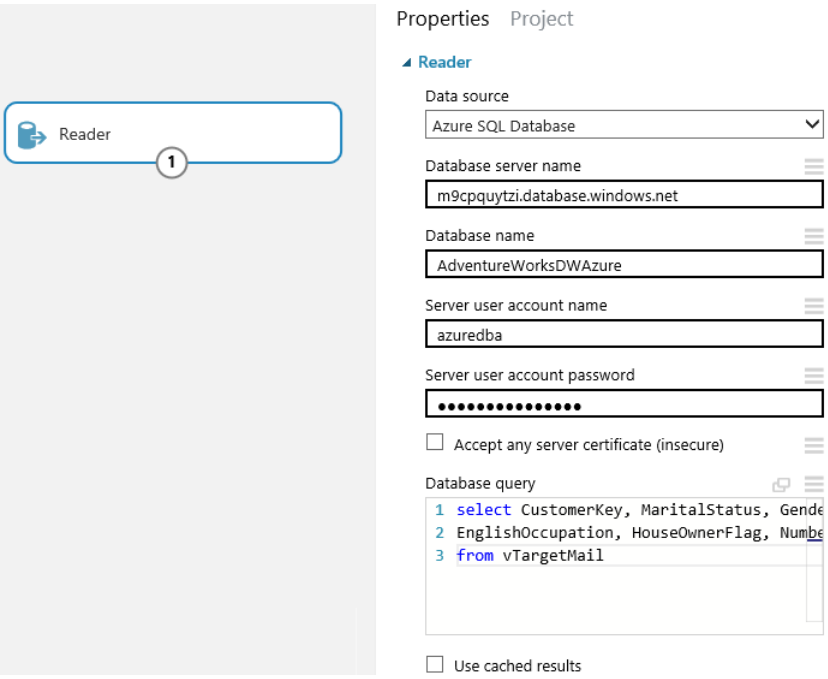


Figure 9-8: Using a Reader module to retrieve data from SQL Data Warehouse.

Similarly, you can write the result set from your experiment to SQL Data Warehouse by using the Writer module, as shown in Figure 9-9. You provide the same connection information as you do for the Reader module and also supply a comma-delimited list of columns in the result set from the module preceding the Writer module in the workflow, the name of the target table in SQL Data Warehouse, a comma-delimited list of columns in the target table, and the number of rows to write at a time.

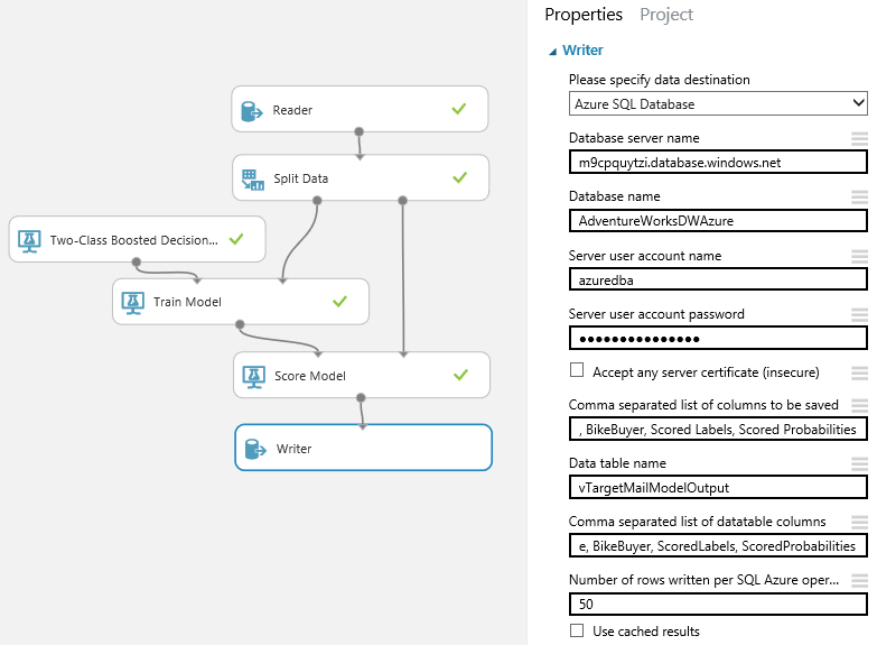


Figure 9-9: Using a Writer module to store data in SQL Data Warehouse.

Note To learn more about Azure Machine Learning, see "Introduction to machine learning on Microsoft Azure" at https://azure.microsoft.com/en-us/documentation/articles/machine-learning-what-is-machine-learning/.

Working with streaming data in Azure Stream Analytics

Azure Stream Analytics is a complex, fully managed infrastructure that you can use to process and consume real-time event data generated from an Azure Event Hub. You can use SQL Data Warehouse as an output sink for the streaming data so that you can apply other tools later for deeper analysis.

To use Stream Analytics, you must create and configure an event hub and have access to streaming data. Then use the Azure Management Portal to create a new Stream Analytics job. A job uses an input to define the source of the streaming data, a query to define which columns to capture from the streaming data and whether to aggregate the data, and an output to define the target for the query results. Click the Inputs, Query, and Outputs tiles in the Stream Analytics Job blade to create the respective objects.

To store the query results in your SQL Data Warehouse, use a SELECT…INTO…FROM statement similar to the one shown in Example 9-9 in the query definition for the job. In the INTO clause, you reference the output alias that you create in the next step, and you use the input alias in the FROM clause.

Note A sample event-stream generator is available for download. You can learn how to create and configure an event hub and use the sample generator at "Get started using Azure Stream Analytics: Real-time fraud detection," https://azure.microsoft.com/en-us/documentation/articles/stream-analytics-get-started/.

Example 9-9: Querying an event stream to store results in a SQL Data Warehouse

```
SELECT System.Timestamp as WindowEnd, CategoryName, COUNT(*) as IncidentCount
INTO SQLDWOutput
FROM CategoryDataStream
TIMESTAMP BY IncidentTime
GROUP BY TUMBLINGWINDOW(s, 5), CategoryName;
```

When you create the output to your SQL Data Warehouse, you provide the pertinent connection information and the name of the target table, as shown in Figure 9-10. The output alias must match the alias in the INTO clause in the query.

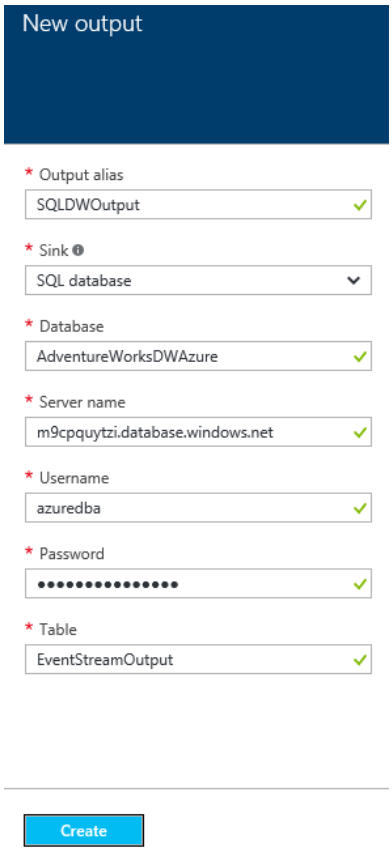


Figure 9-10: Creating an Analytics Stream job output in Azure Management Portal.

After you start the job by clicking the Start button in the Analytics Stream Job blade, the job connects to the streaming data and executes the query. You can then use a query tool to see the results in the target table in your SQL Data Warehouse.

Note Additional information about Azure Stream Analytics is available at "What is Stream Analytics?," https://azure.microsoft.com/en-us/documentation/articles/stream-analytics-introduction/.

Analyzing data by using Power BI

To create reports and rich visualizations using your SQL Data Warehouse, you can use Power BI directly from the Azure Management Portal, or you can connect to it as a data source in Power BI. In an on-premises SQL Server data warehouse, foreign-key relationships are typically added to tables, and Power BI can use these relationships as it builds visualizations. However, SQL Data Warehouse does not support foreign keys. Therefore, consider adding views to your data warehouse to support the types of analysis your users perform.

In the blade for your SQL Data Warehouse in the Azure Management Portal, the toolbar includes the Open In Power BI button. When you click this button, a new browser tab opens with a prompt to sign in to Power BI or to sign up if you do not yet have a Power BI account. After you log in, a window displays prompts for the server name hosting your SQL Data Warehouse and the database name with the values automatically populated. In addition, you can turn on the Enable Advanced Options switch to specify a refresh frequency and define custom filters. For a custom filter, you can type in a comma-delimited list of tables to access or write a SELECT statement. When you click Next, you must provide credentials for your data warehouse and then click Sign In. After this authentication step, a SQL Data

Warehouse tile is displayed in your default dashboard in the Power BI window. Click this tile to access the Power BI report designer.

A more common way for users to access SQL Data Warehouse from Power BI is to click the Get Data button in Power BI, click Databases, and then click the Azure SQL Data Warehouse tile, shown in Figure 9-11. Next, click the Connect button to open the prompt for the server, database, and advanced options described earlier. If you are using Power BI Desktop, click Get Data, select Microsoft Azure SQL Data Warehouse in the list of sources, click Connect, supply connection and credentials information, and select the tables or views to use in Power BI.

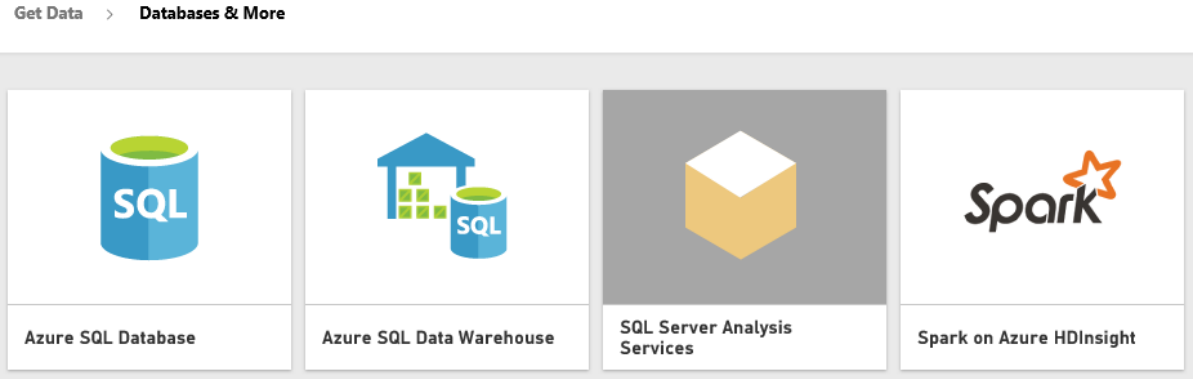


Figure 9-11: Getting data for Power BI.

Regardless of which path you take to connect to your data warehouse, you can then build reports and dashboards by using visualizations based on data from your data warehouse, as shown in Figure 9-12. You can expand tables in the Fields panel to find the column names that you want to use in a visualization. You can choose a different type of visualization in the Visualizations panel to view your data from a different perspective. You can also resize the visualization or move it to a different location on the page. A report page can even contain multiple visualizations. Click an empty spot on the page and select fields from the Fields panel to create a new visualization.

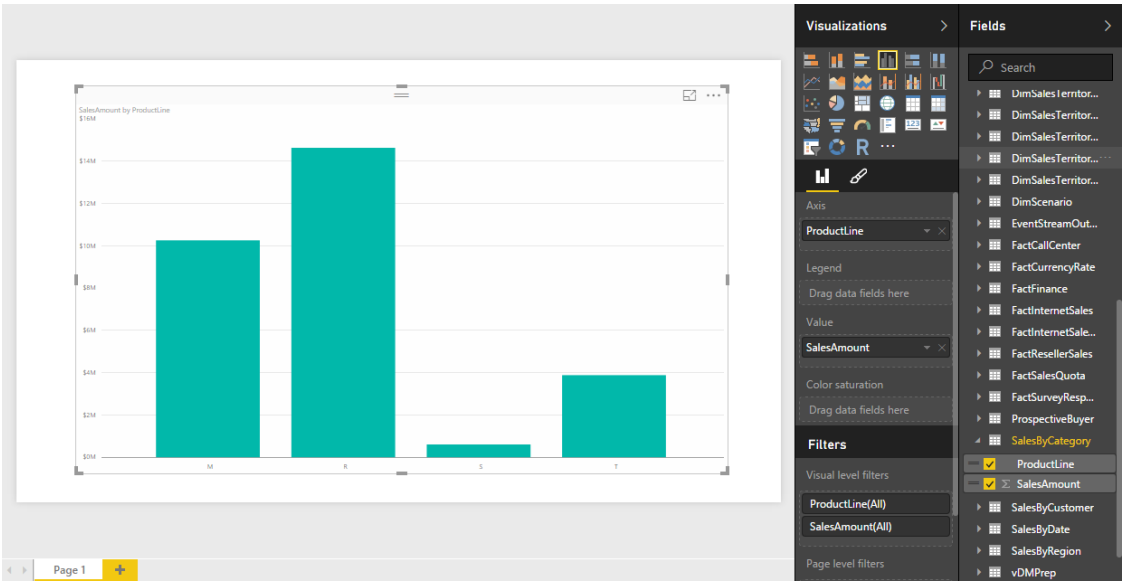


Figure 9-12: Visualizing SQL Data Warehouse data in Power BI.

Note To successfully use Power BI, you must be sure that the client IP addresses for users are configured in the firewall settings for the server and enable the Allow Access To Azure Services

option. You can learn more about Power BI at "Get Started with Power BI," https://powerbi.microsoft.com/en-us/documentation/powerbi-service-get-started/.

Building business-intelligence solutions with SQL Server components

Because SQL Data Warehouse is built with much of the same technology available in the SQL Server database engine, the two products share many common features. Although SQL Data Warehouse does not fully implement T-SQL at the time of this writing, it does include commands that are commonly used in data-warehouse workloads so that you can quickly get started writing stored procedures and queries for SQL Data Warehouse. Other tools useful for on-premises business-intelligence solutions based on SQL Server, such as SQL Server Analysis Services and SQL Server Reporting Services, are fully compatible with SQL Data Warehouse. However, if you are exporting data to work with these tools on-premises, pay careful attention to your data egress, as charges for data leaving the cloud do apply. Instead, consider using Azure IaaS Virtual Machines for these components when building your business-intelligence solutions.

About the authors

Stacia Varga is a consultant, educator, mentor, and writer who has specialized in business-intelligence solutions since 1999. During that time she authored or coauthored several books about BI as Stacia Misner. Her last book was *Introducing Microsoft SQL Server 2014* (Microsoft Press, 2014). She has also written articles for *SQL Server Magazine* and Technet and has produced multiple BI video courses available through Pluralsight. In addition, she has been recognized for her contributions to the technical community as a Microsoft Data Platform MVP since 2011. Stacia provides consulting and custom education services through her company, Data Inspirations; speaks frequently at conferences serving the SQL Server community worldwide; and serves as the chapter leader of her local PASS user group, SQL Server Society of Las Vegas. She holds a BA in social sciences from Washington State University. Stacia writes about her experiences with BI at blog.datainspirations.com and tweets as @_StaciaV_.

Joseph D'Antoni is a principal consultant for Denny Cherry and Associates Consulting. He is well versed in SQL Server performance tuning and database infrastructure design, with more than a decade of experience working in both Fortune 500 and smaller firms. Joseph is a frequent speaker at major technical events worldwide. In addition, he blogs about a variety of technology topics at joeydantoni.com and tweets as @jdanton. Joseph holds a BS in computer information systems from Louisiana Tech and an MBA from North Carolina State University.

Denny Cherry is the owner, founder, and principal consultant for Denny Cherry and Associates Consulting. His primary areas of focus are system architecture, performance tuning, and data replication. Denny has been recognized in the technical community as a Microsoft Data Platform MVP, VMware vExpert, and EMC Elect. He holds certifications for SQL Server from the MCDBA for SQL Server 2000 up through Microsoft Certified Master for SQL Server 2008. He is also a Microsoft Certified Trainer. Denny has written dozens of articles for *SQL Server Magazine*, Technet, and SearchSQLServer.com, among others. In addition, he has authored and coauthored multiple books, including *The Basics of Digital Privacy: Simple Tools to Protect Your Personal Information and Your Identity Online* (Syngress, 2013) and *Securing SQL Server: Protecting Your Database from Attackers,* 2nd Edition (Syngress, 2012). Denny speaks at events worldwide, blogs at www.dcac.co/blogs, and tweets as @mrdenny.

From technical overviews to drilldowns on special topics, get *free* ebooks from Microsoft Press at:

www.microsoftvirtualacademy.com/ebooks

Download your free ebooks in PDF, EPUB, and/or Mobi for Kindle formats.

Look for other great resources at Microsoft Virtual Academy, where you can learn new skills and help advance your career with free Microsoft training delivered by experts.

Microsoft Press