Introducing

Microsoft SQL Server 2016

Mission-Critical Applications, Deeper Insights, Hyperscale Cloud

Preview 2

Stacia Varga, Denny Cherry, Joseph D'Antoni

Introducing Microsoft SQL Server 2016

Mission-Critical Applications, Deeper Insights, Hyperscale Cloud
Preview 2

Stacia Varga, Denny Cherry, and Joseph D'Antoni

Microsoft

Printed and bound in the United States of America.

First Printing

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.

Contents at a glance

Contents

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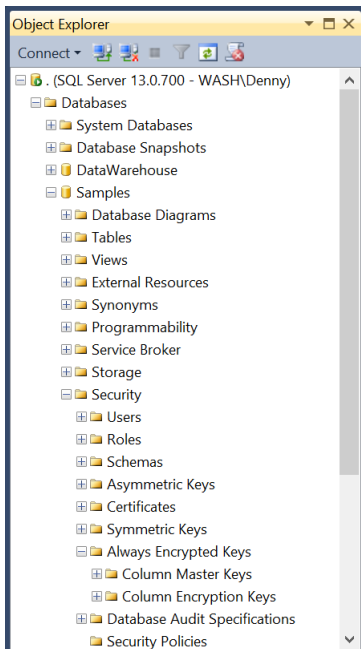
Figure 2-1: Always Encrypted Keys folder in SQL Server 2016 Object Explorer.

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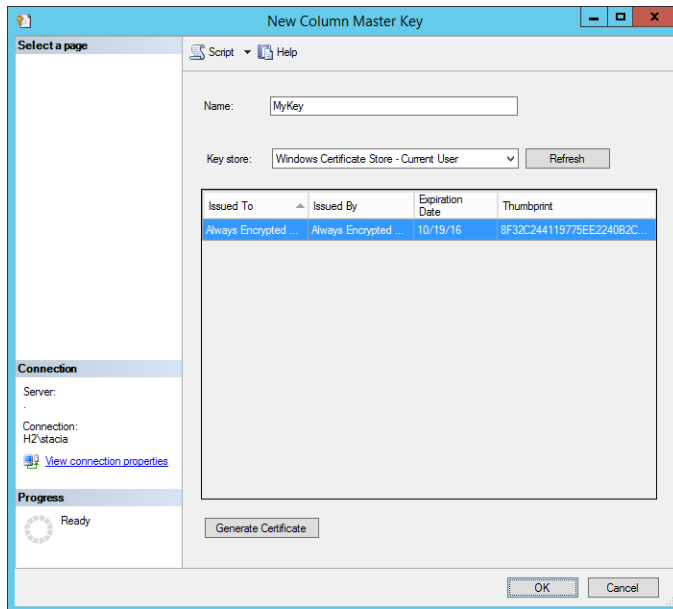
Figure 2-2: New Column Master Key dialog box.

> **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.

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 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 at http://blogs.msdn.com/b/

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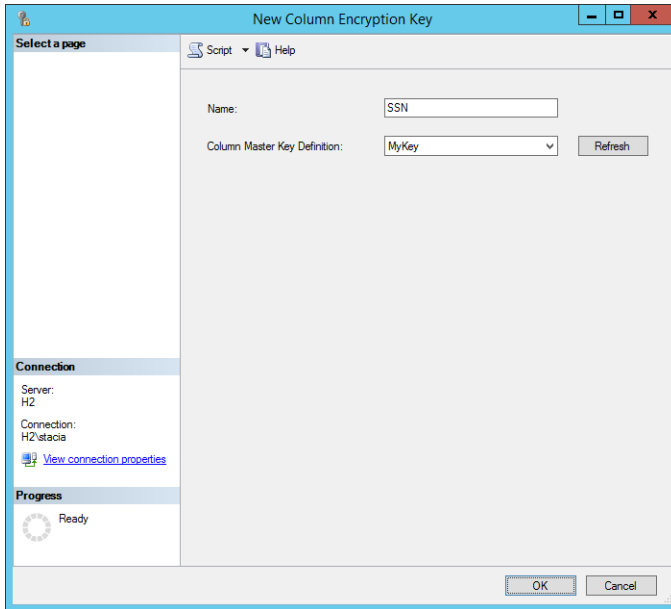
Figure 2-3: New Column Encryption Key dialog box.

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 =
0x016E0080006300750072007200720065006E00740075007300650072002F006D0079002F0064006500330061003700370003
000660032003500065006200640003600300037003100330030003500620037003700660062003100390038006400310061
0065003400330034006500036003000310034004D74119935C902E59F57A96C3E6F770826D247135FFFA759B5B013DF4DA
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.

```
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. 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.

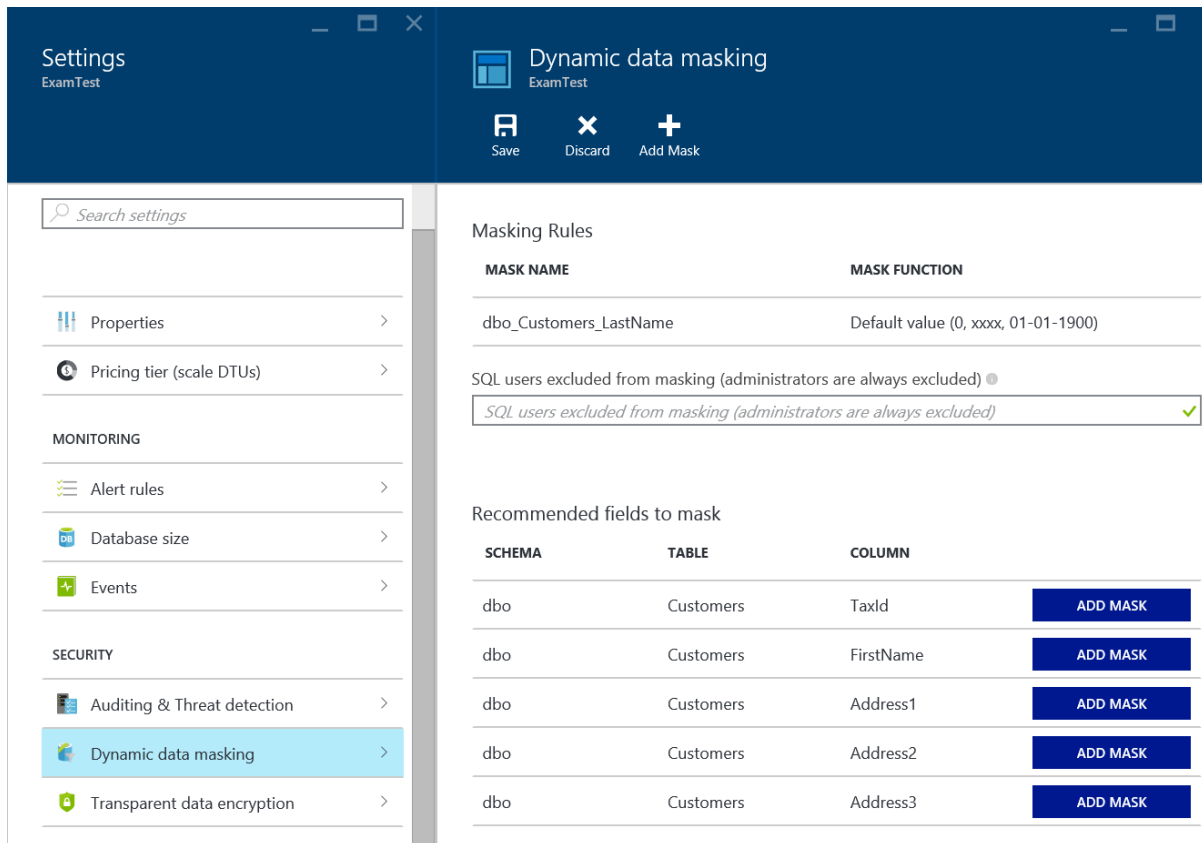Figure 2.5: Configuring dynamic data masking for a SQL Database in the Azure portal.

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.

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.

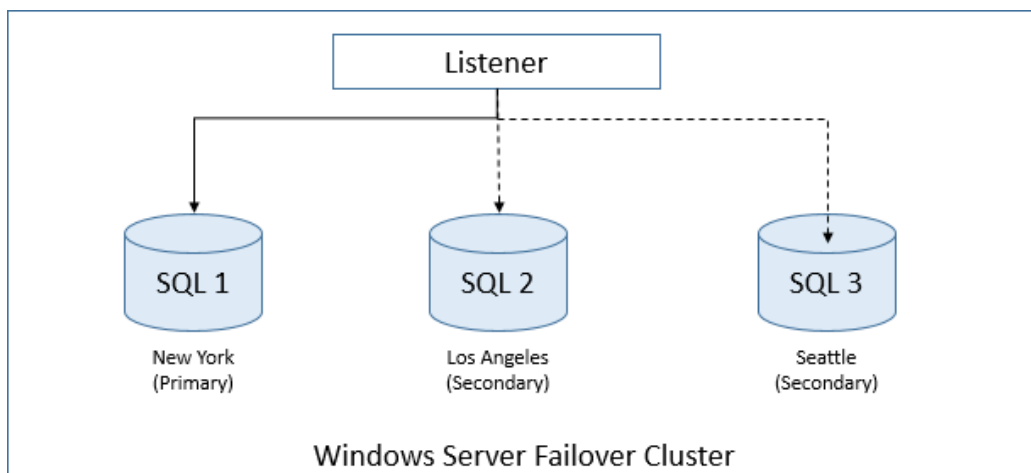
Figure 3-1: An AlwaysOn Availability Group with a primary replica and two secondary replicas.

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.
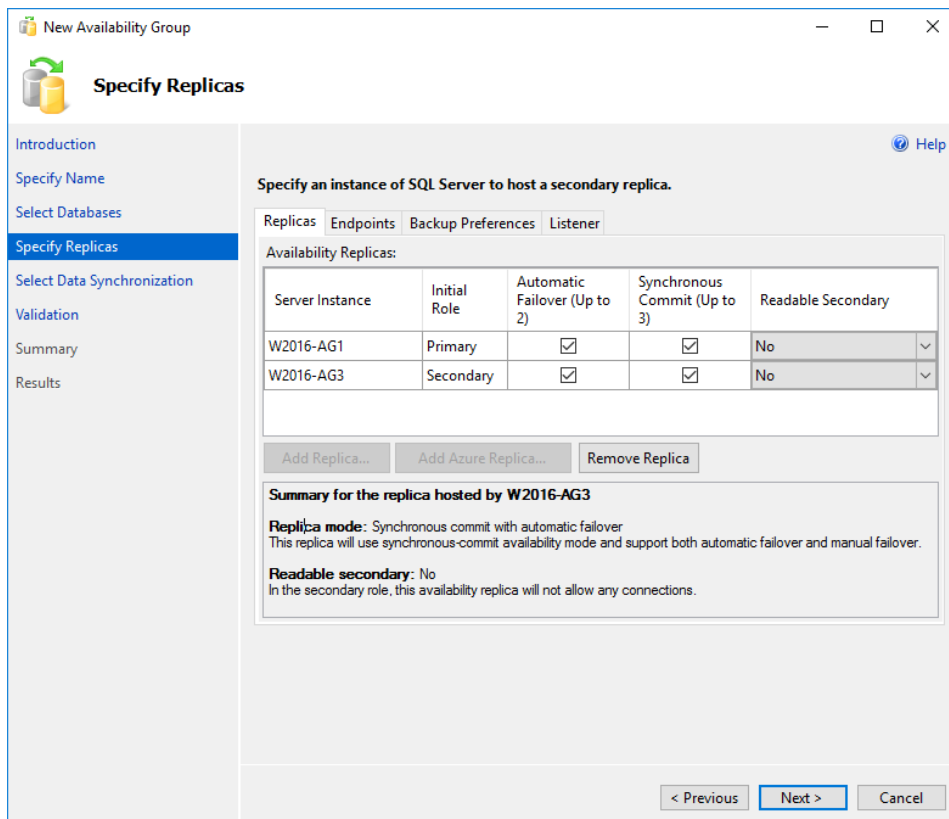

Figure 3-2: Configuring replicas for a basic availability group.

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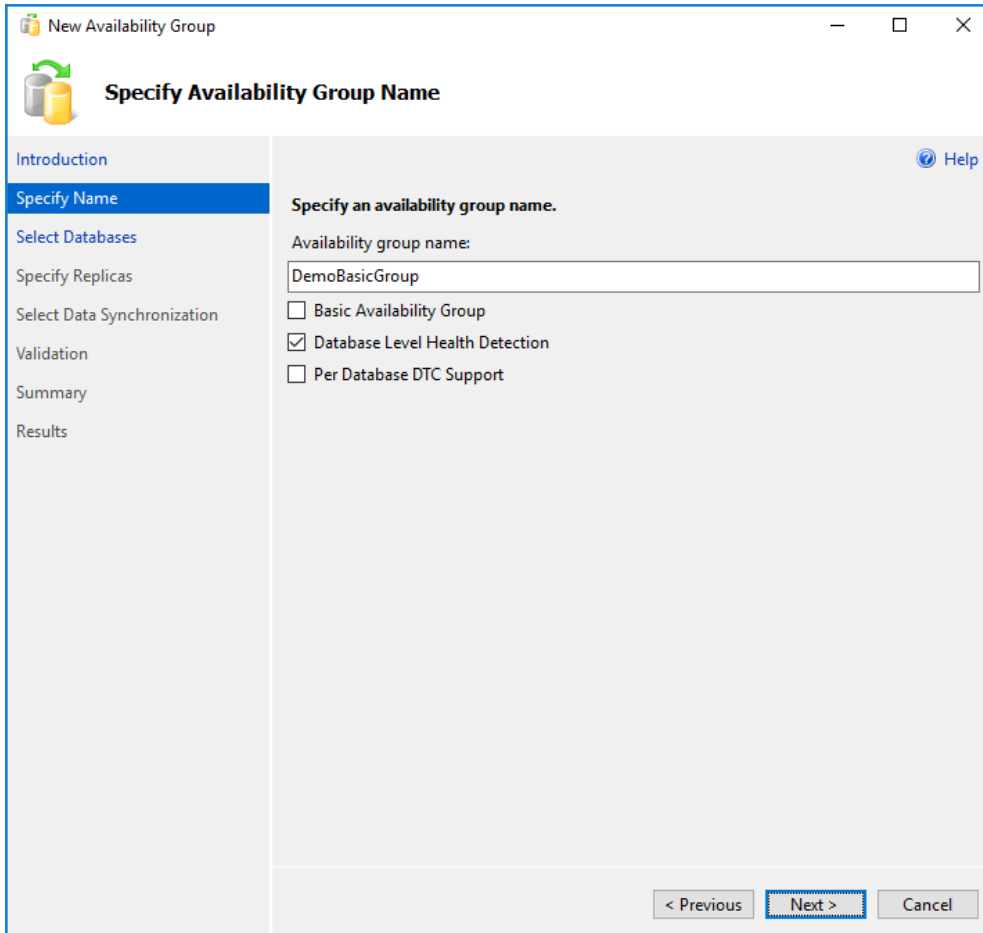
Figure 3-3: Creating a new availability group with database-level health detection.

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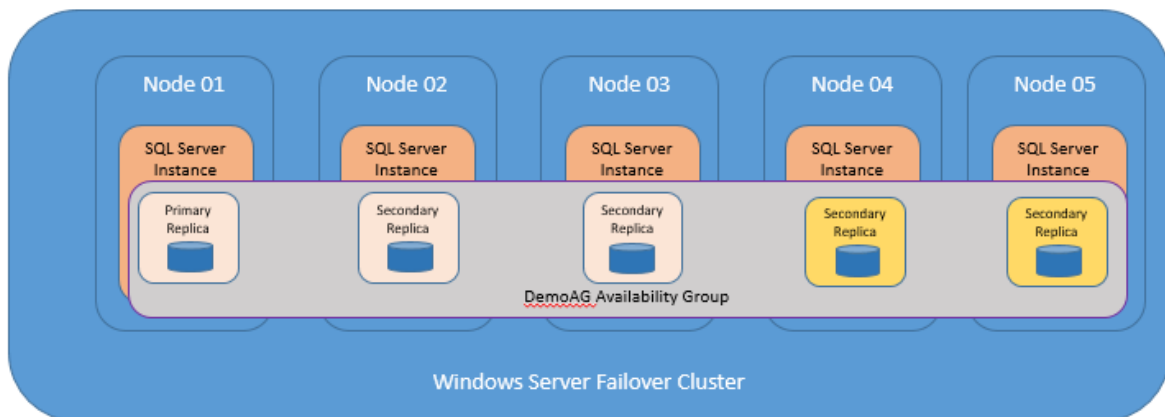
Figure 3-4: Availability Group topology with three automatic failover targets.

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.
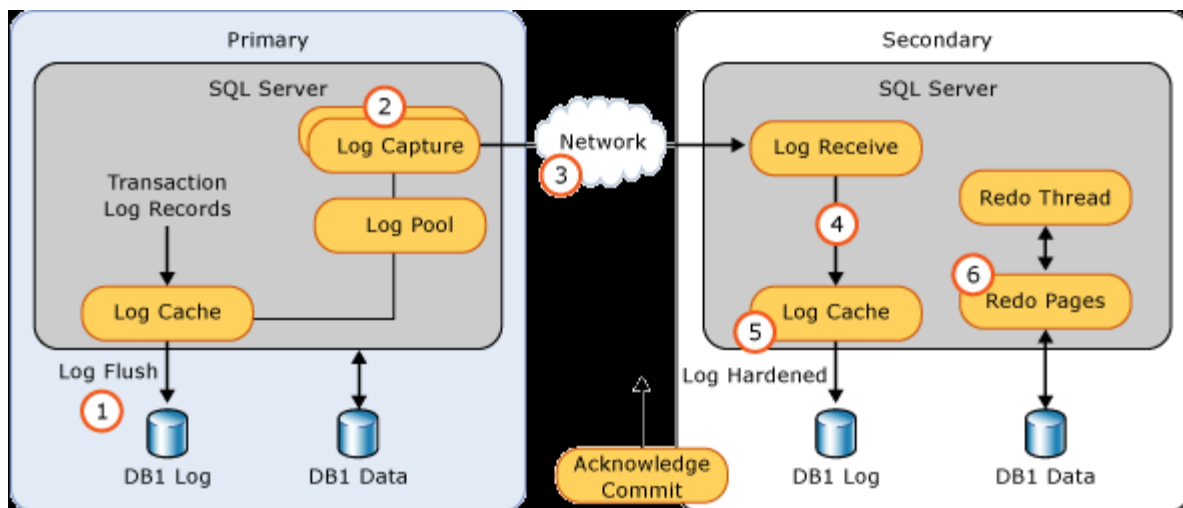

Figure 3-5: Log transport operations for AlwaysOn Availability Groups.

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. You can 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.
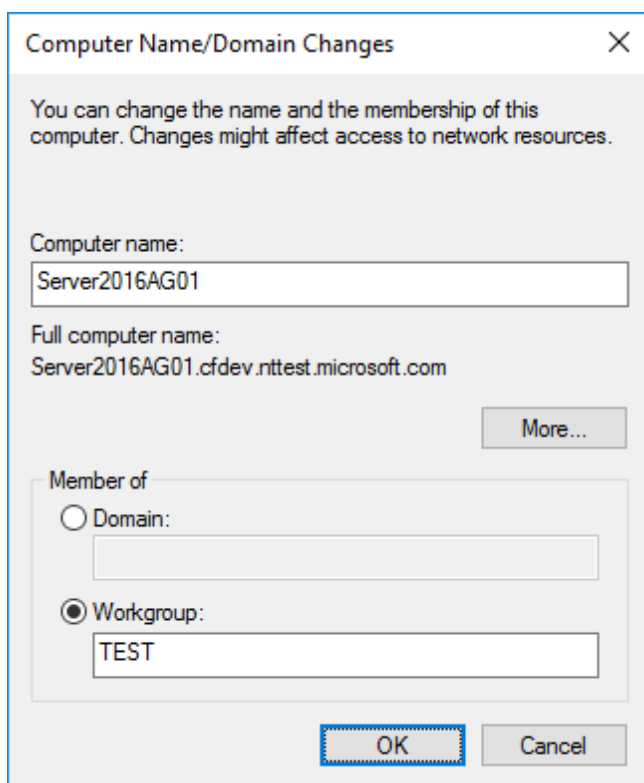

Figure 3-6: A server with a DNS suffix assigned to a workgroup.

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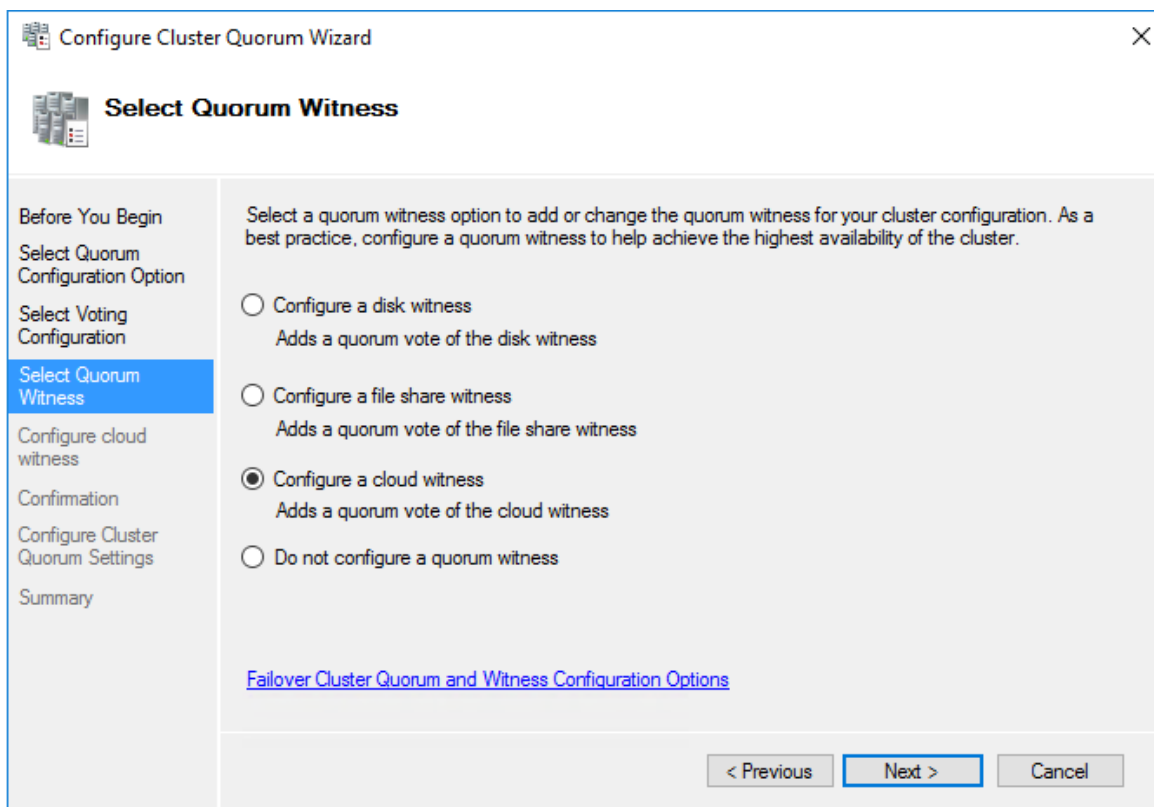
Figure 3-7: Creating a cloud witness for a cluster quorum.

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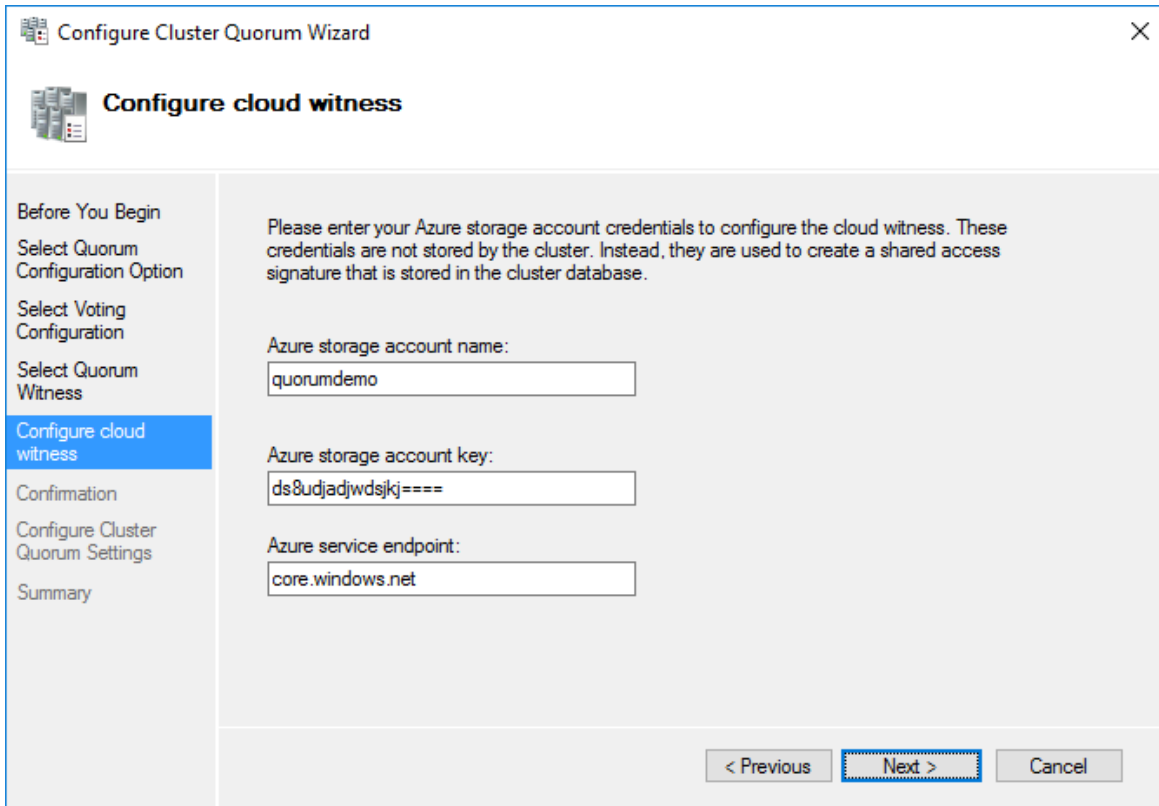
Figure 3-8: Addition of Azure credentials to cloud witness configuration.

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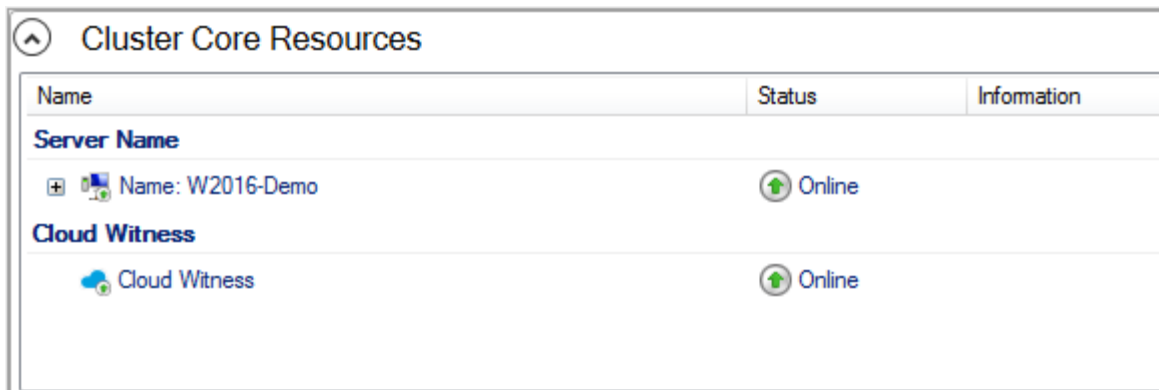
Figure 3-9: Successful addition of a cloud witness to cluster core resources.

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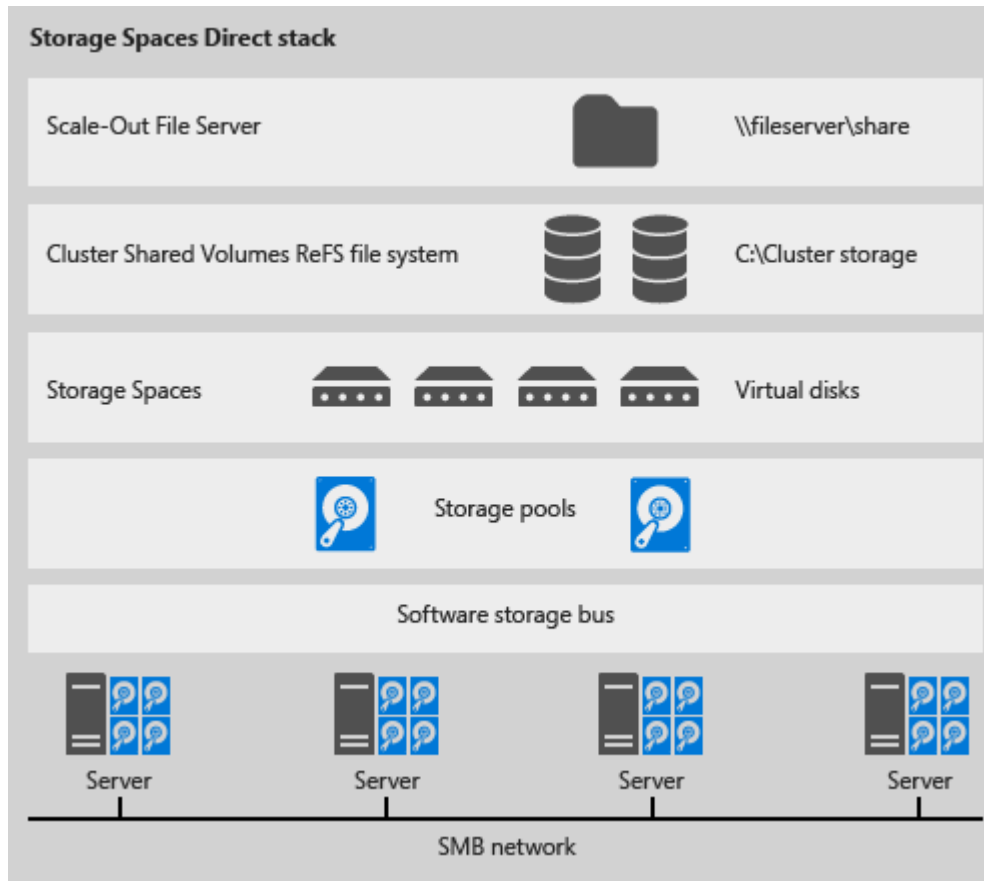
Figure 3-10: Storage options in Storage Spaces Direct.

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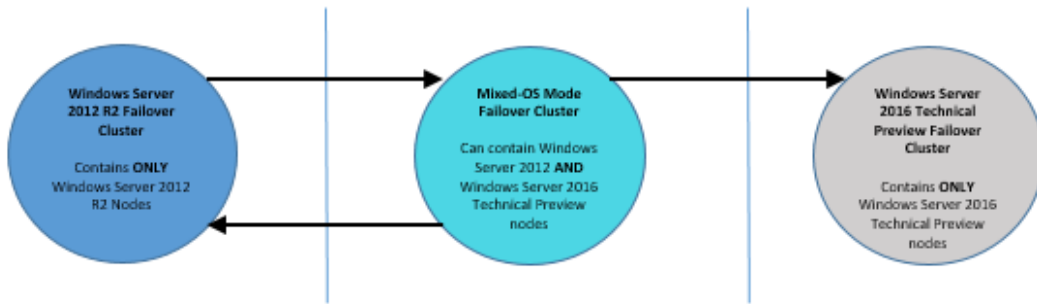
Figure 3-11: State transitions during a rolling operating system upgrade.

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.

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.
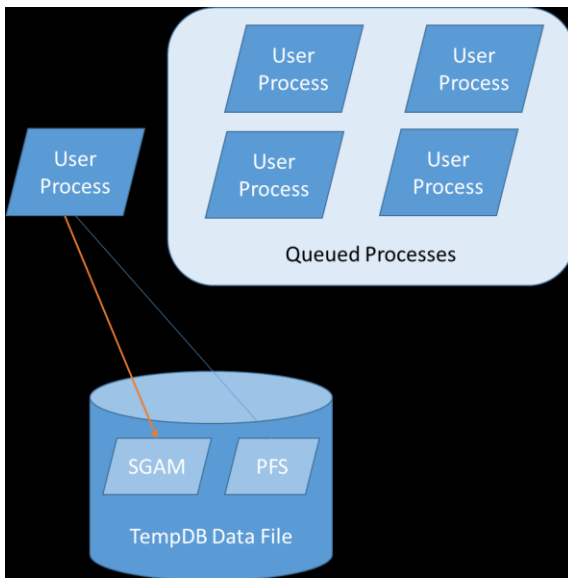
Figure 4-1: Contention in TempDB.

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.

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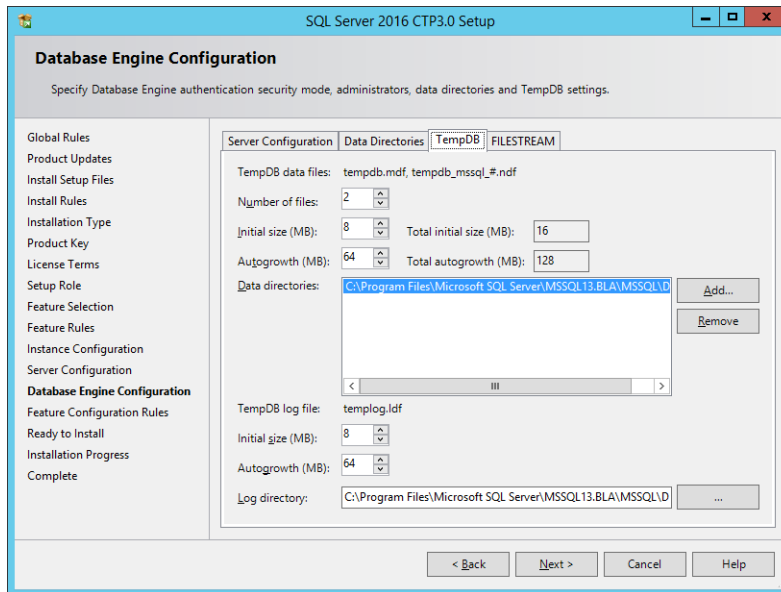
Figure 4-2: Configuring TempDB in SQL Server 2016.

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.
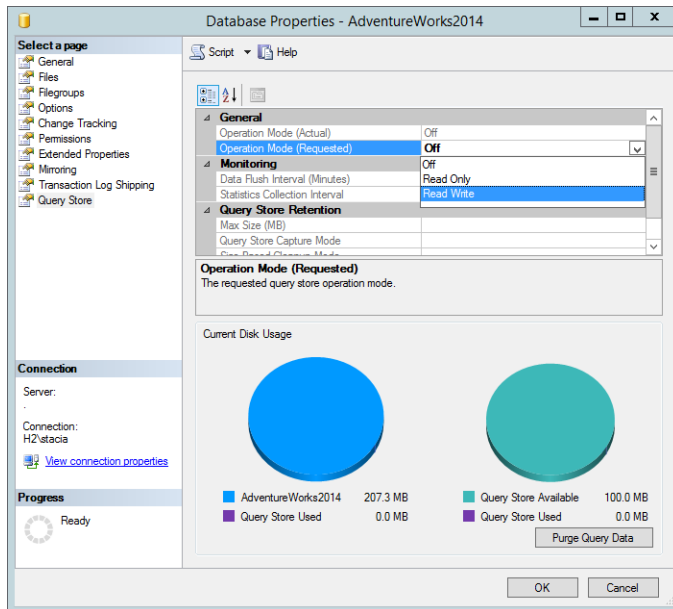
Figure 4-3: Enabling Query Store.

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.
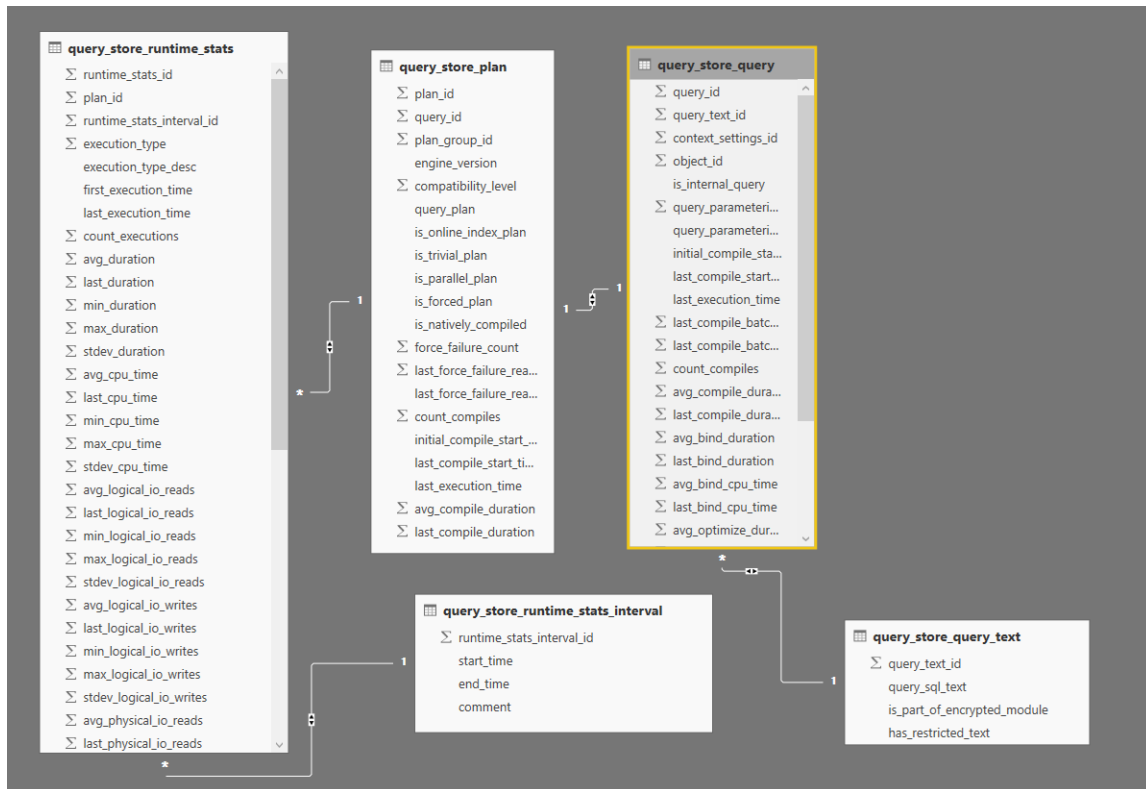
Figure 4-4: Query Store catalog views.

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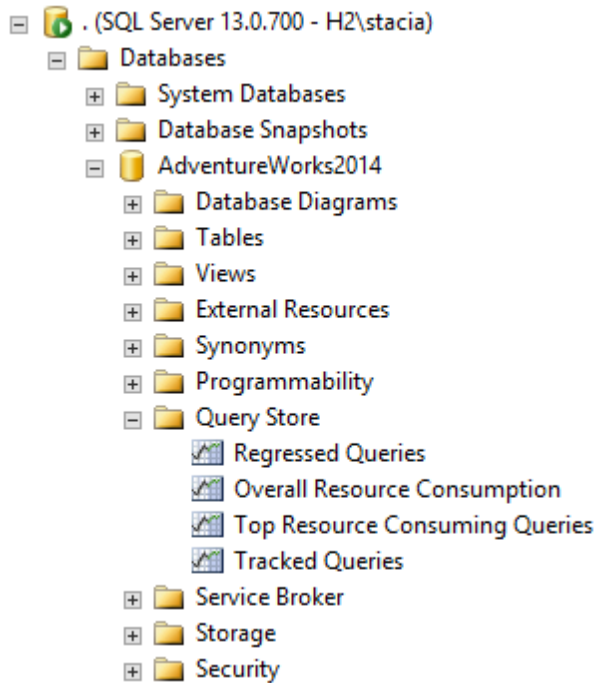
Figure 4-5: Query Store dashboards available in SSMS.

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.
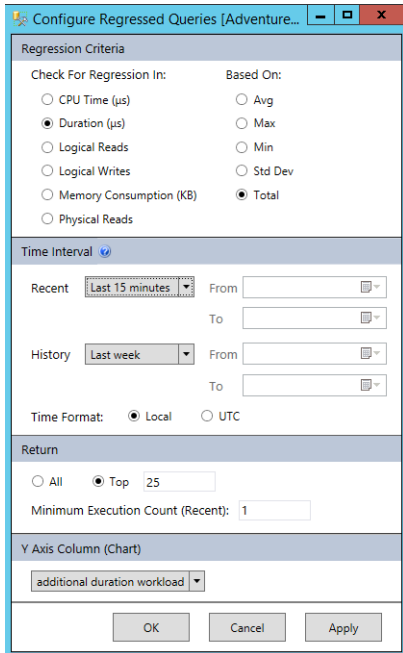
Figure 4-6: Configuring a Query Store dashboard.

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.

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.

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.
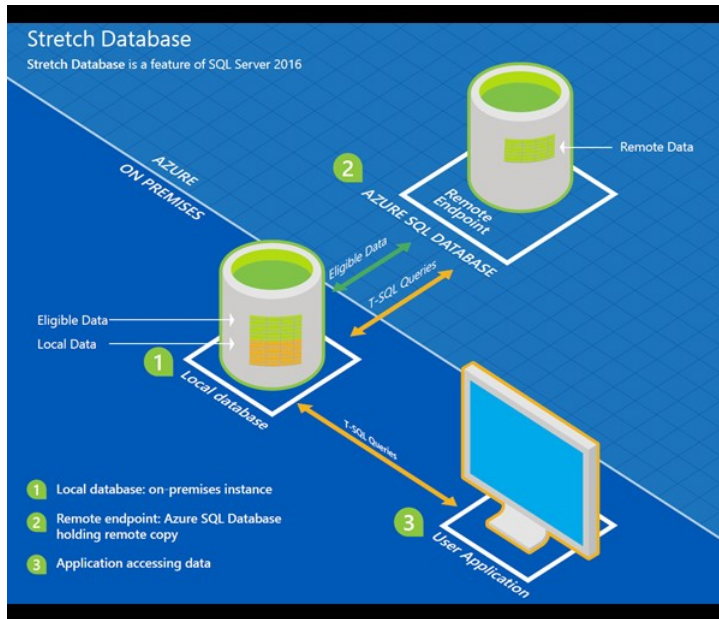
Figure 4-7: Stretch Database architecture.

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.
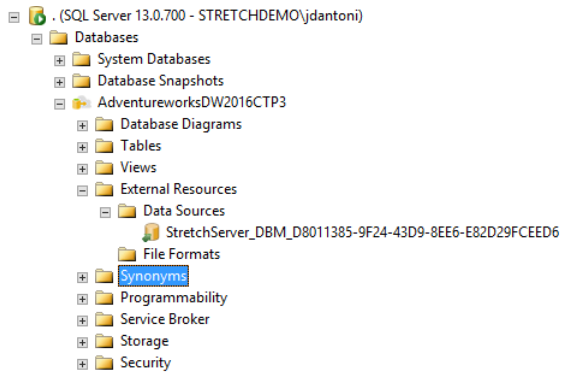
Figure 4-8: External resource for Stretch Database.

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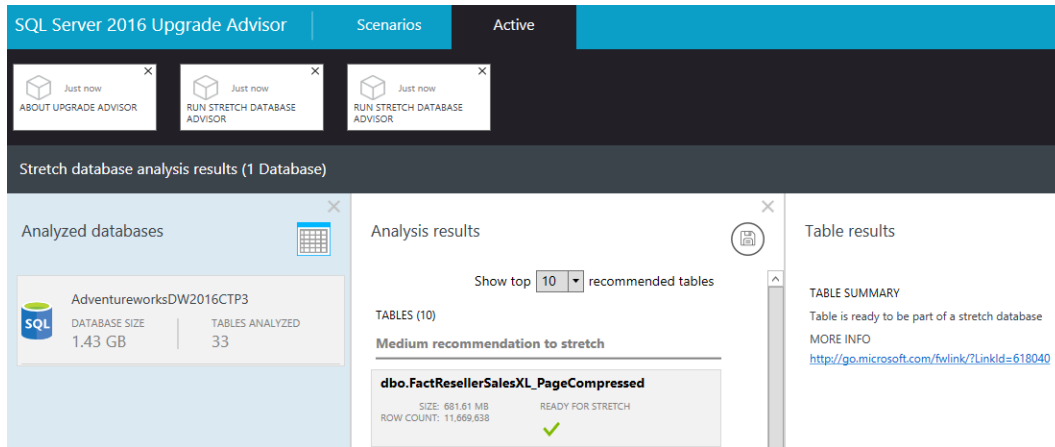
Figure 4-9: Analyzing candidates for Stretch Database in SQL Server 2016 Upgrade Advisor.

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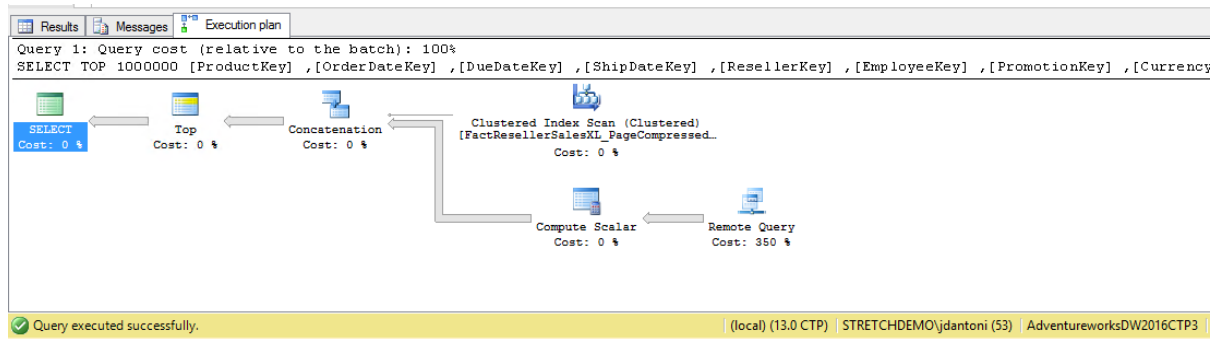
Figure 4-11: Reviewing the execution plan for a stretch table.

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.

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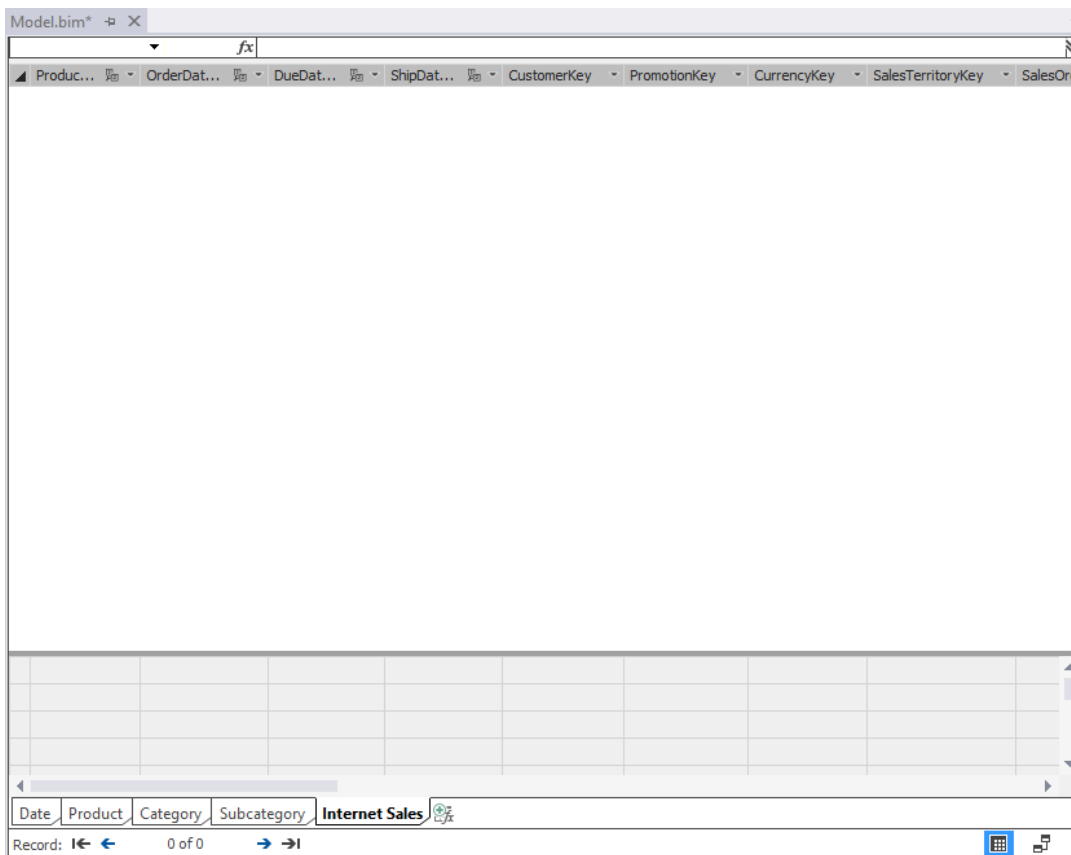
Figure 6-1: Working with a model in DirectQuery mode.

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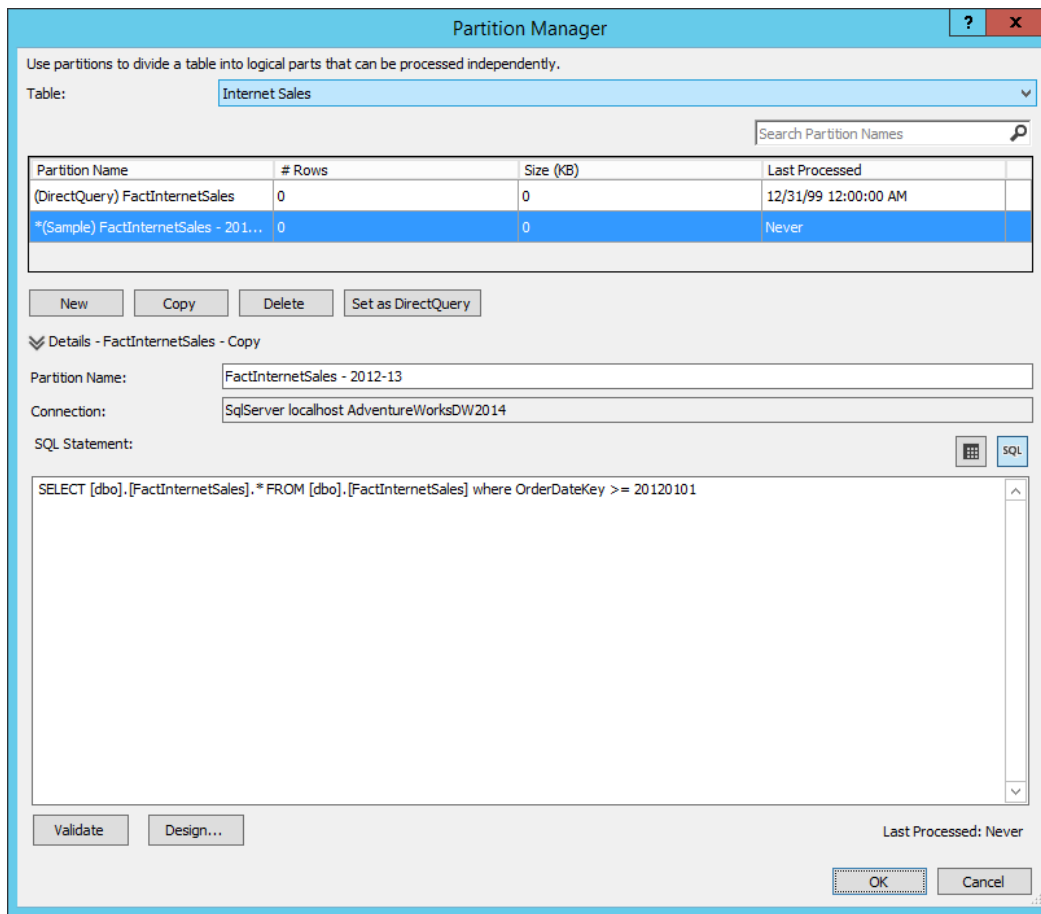
Figure 6-2: Configuring sample data for a partition in DirectQuery mode.

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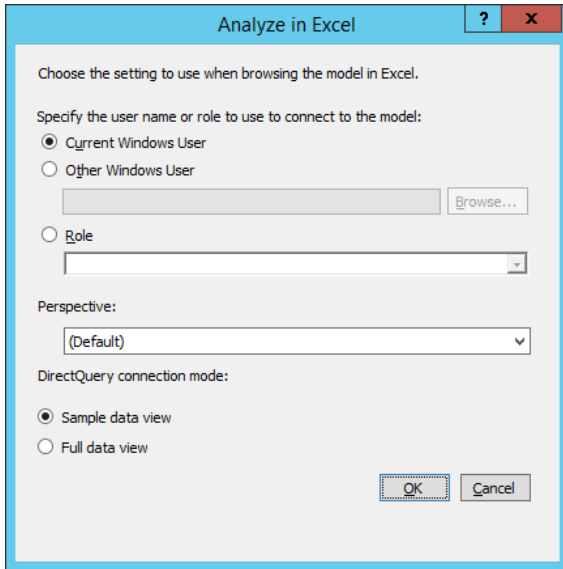
Figure 6-3: Using the sample data view in DirectQuery mode to analyze data in Excel.

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.

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.

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
2	M	16,898
3	R	15,552
4	S	23,358
5	T	4,590
6	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.

	A	B	C
1			
2			
3	Row Labels ▼	Sales Count	Distinct Product Count
4	⊟M	16,898	112
5	2011	390	112
6	2012	1,064	112
7	2013	14,584	112
8	2014	860	112
9	⊟R	15,552	162
10	2011	1,788	162
11	2012	2,156	162
12	2013	11,101	162
13	2014	507	162
14	⊟S	23,358	54
15	2011		54
16	2012		54
17	2013	22,011	54
18	2014	1,347	54
19	⊟T	4,590	52
20	2011		52
21	2012		52
22	2013	4,389	52
23	2014	201	52
24	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.
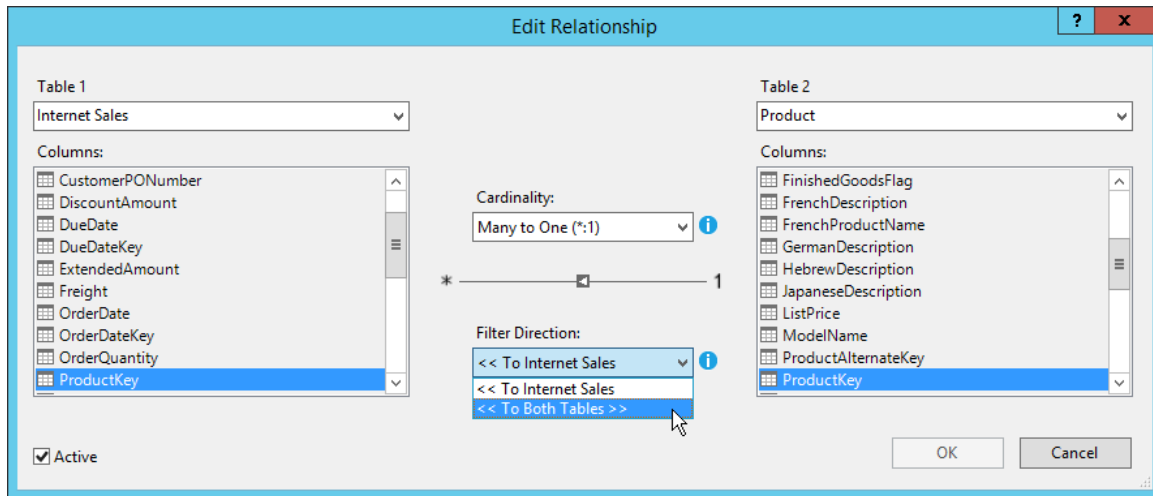
Figure 6-10: Setting a bidirectional cross-filter in a table relationship.

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.

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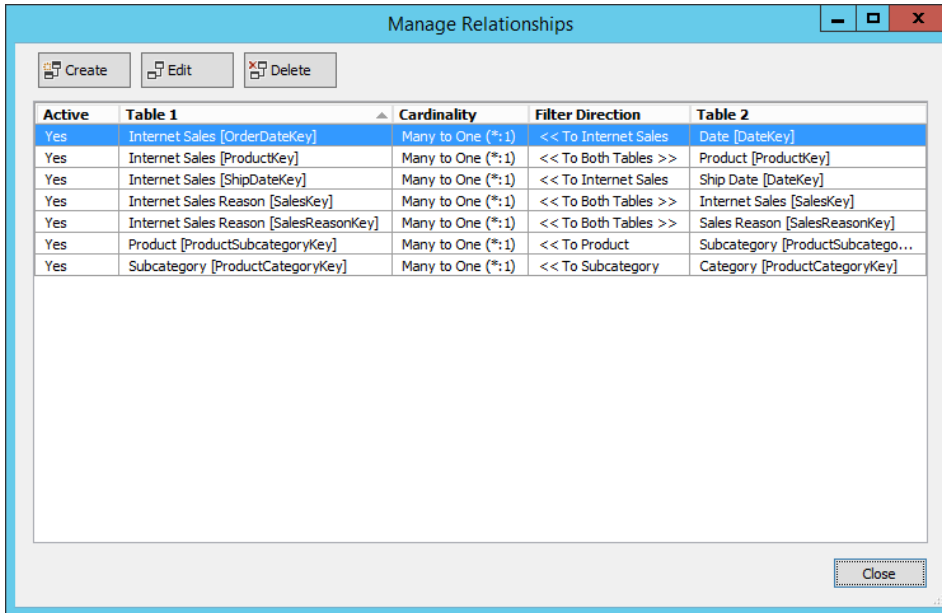
Figure 6-12: Defining a many-to-many relationship.

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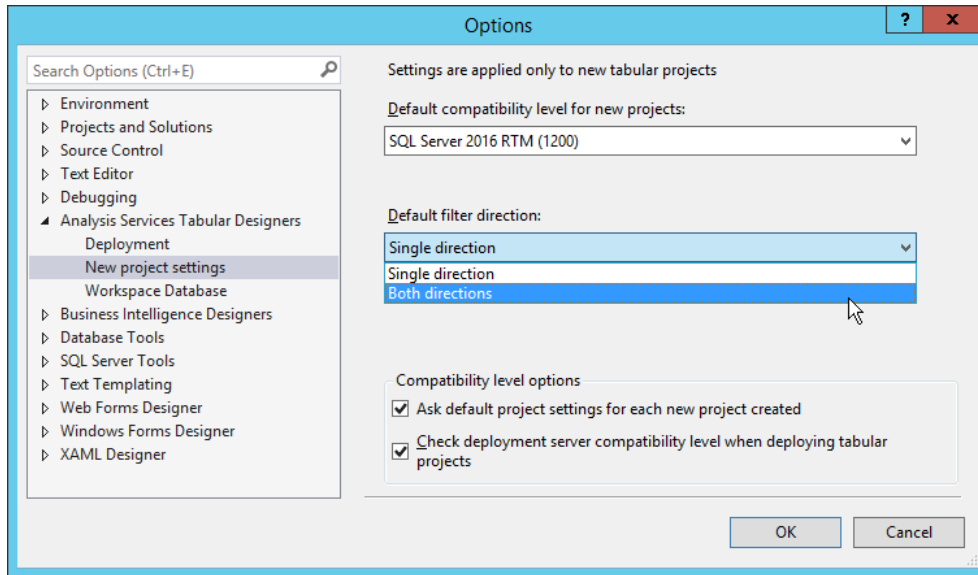
Figure 6-14: Setting the default filter direction for new projects.

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 kth percentile of values in a range, where k is in the range 0 to 1, exclusive.
	PERCENTILE.INC()	Returns the kth percentile of values in a range, where k is in the range 0 to 1, inclusive.
	PERCENTILEX.EXC()	Returns the kth 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.

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 and reuse the code in another platform, such as Hadoop. The workload shifts from the desktop to the server, where R Services performs multithreaded, multicore, and parallelized multiprocessor computations at high speed while maintaining the necessary levels of security for your data. 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 provide the links to download the requisite files and additional information necessary to prepare your environment to use R Services.

Server configuration

In the server environment, you start by installing the following components on the computer hosting SQL Server 2016:

- **Advanced Analytics Extensions**   A database engine feature that you select during installation of a SQL Server instance.

- **Microsoft R Open (MRO) 3.2.2 for Revolution R Enterprise 7.5.0**   An enhanced open-source R distribution that you download from https://www.microsoft.com/en-us/download/details.aspx?id=49525. This is a prerequisite for Revolution R Enterprise.

- **Revolution R Enterprise 7.5.0 (RRE-7.5.0)**   A collection of libraries that you download from https://www.microsoft.com/en-us/download/details.aspx?id=49505. This provides connectivity to SQL Server and executes R packages in a high-performance, parallel architecture.

At the time of writing, you must perform several postinstallation tasks. Because this information is subject to change, refer to "Post-Installation Server Configuration (SQL Server R Services)" at https://msdn.microsoft.com/library/mt590536(SQL.130).aspx for the most up-to-date instructions.

> **Note**   The default memory-allocation settings might allow SQL Server to consume most of the available physical memory without leaving adequate memory for R. 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:

- **Microsoft .NET Framework 3.5**   If this version of the .NET Framework is not yet installed on the client workstation, you can manually install it by following the instructions at https://msdn.microsoft.com/en-us/library/hh506443(v=vs.110).aspx.

- **MRO 3.2.2 for RRE-7.5.0**   Download this component from https://www.microsoft.com/en-us/download/details.aspx?id=49525, and install it to load the open-source R run-time libraries onto the workstation. This is a prerequisite for RRE.

- **RRE-7.5.0**   Download this component from https://www.microsoft.com/en-us/download/details.aspx?id=49505, and install it to add connectivity to SQL Server and enhanced R packages for high-performance.

- **An R integrated development environment (IDE)**   You can use any R IDE that you prefer.

> **Note**   At the time of writing, the integration of Revolution R into SQL Server is not complete. A list of current limitations is available at https://msdn.microsoft.com/en-US/library/mt590540.aspx.

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.

User permissions

Before users can begin executing R on a database, you must ensure that each user has read permissions to the data. In addition, you must add each user to the *db_rrerole* in SQL Server Management Studio (SSMS) by running the code shown in Example 6-3.

Example 6-3: Adding a user to the *db_rrerole*

```
USE [master]
GO
CREATE USER [<user name>] FOR LOGIN [<login name>] WITH
    DEFAULT_SCHEMA=[db_rrerole];
ALTER ROLE [db_rrerole] ADD MEMBER [<user name>];
```

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 6-4.

Example 6-4: 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 6-4, 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 6-4, 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 6-5 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 6-5: 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 6-14. 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 6-14: 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 6-15, 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 6-15: Executing the *rxSummary* function in the R console.

You can also use graphical objects such as a histogram to explore your data. Figure 6-16 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)
```
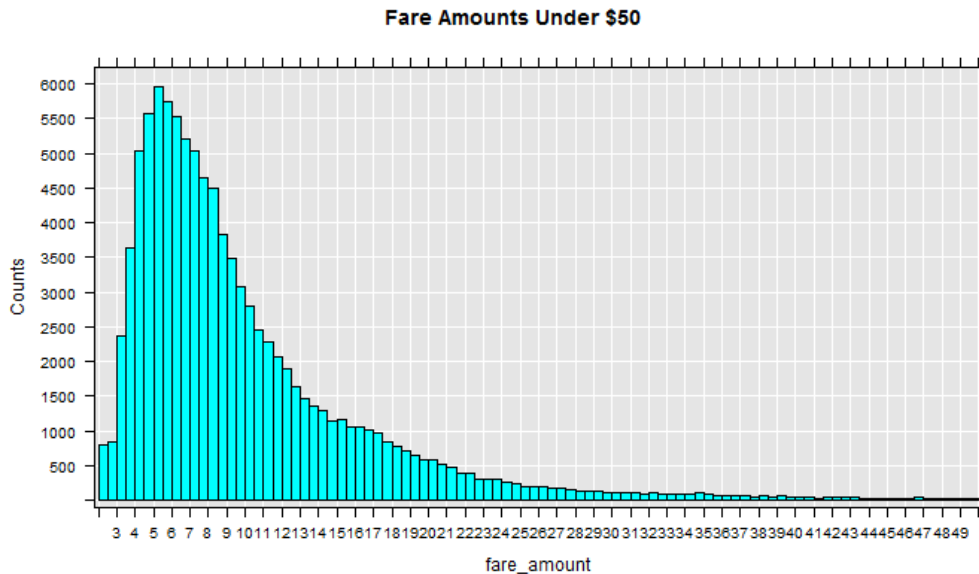
Figure 6-16: Viewing the plot created by executing the *rxHistogram* function.

Note To achieve the results shown in Figure 6-16, 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 6-6.

Example 6-6: 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 6-7.

Example 6-7: 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*.

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 overlaid on the map, as shown in Figure 6-17.

---

[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.
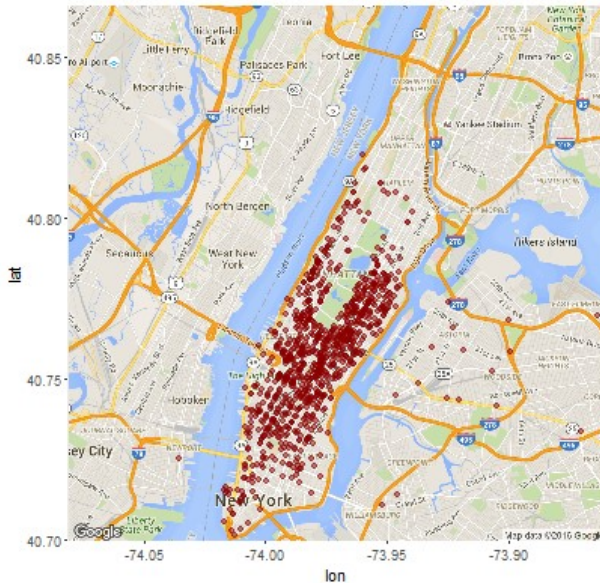
Figure 6-17: Viewing the map plot created by executing the custom *mapPlot* function.

> **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 6-8 in your R IDE.

Example 6-8: 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 6-9, the goal is to predict whether a tip was given. The *summary* function provides statistical information about the resulting model, as shown in Figure 6-18.

Example 6-9: 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 6-18: 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 6-10. 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 6-10: Predicting values

```
scoredOutput <- RxSqlServerData(
    connectionString = connStr,
    table = "taxiScoreOutput")
```

```
rxPredict(modelObject = logitObj, data = modelDataSource, outData = scoredOutput,
          predVarNames = "Score", type = "response", writeModelVars = TRUE, overwrite = TRUE)
```

Figure 6-19 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 6-19: 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 6-11 shows the code to install and load this library by using the *install.packages* and *library* functions.

Example 6-11: 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 6-20.
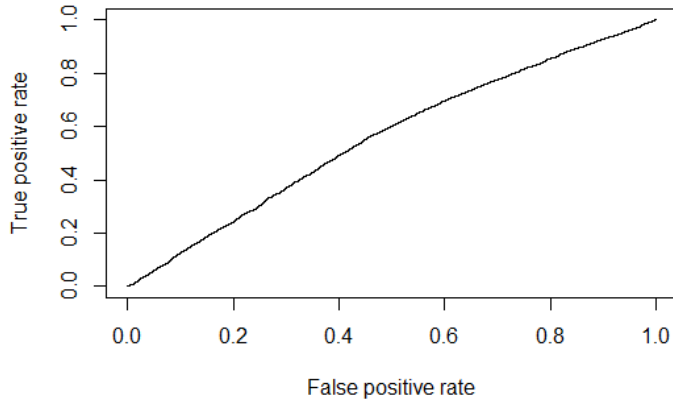
Figure 6-20: Viewing the plot of the prediction model performance.

> **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 6-12. 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 6-12: 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 6-13. 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 6-13: 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 6-14 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 6-14: 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
```

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. 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 dataset 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.
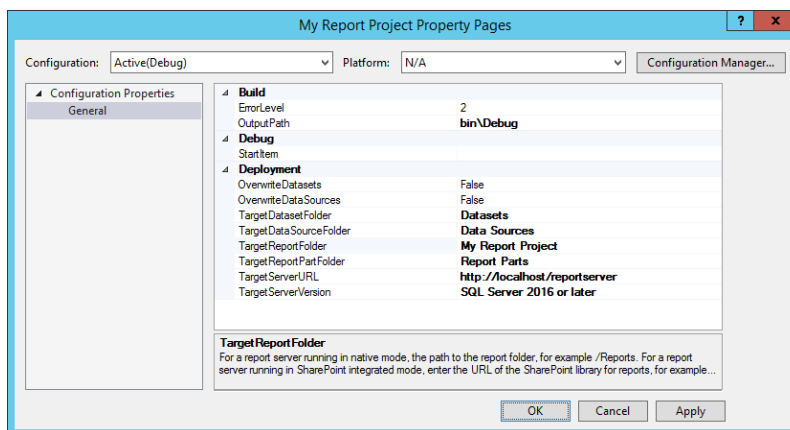

Figure 7-1: A new default value for the TargetServerVersion property in the project's properties.

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/confirmation.aspx?id=49528 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.
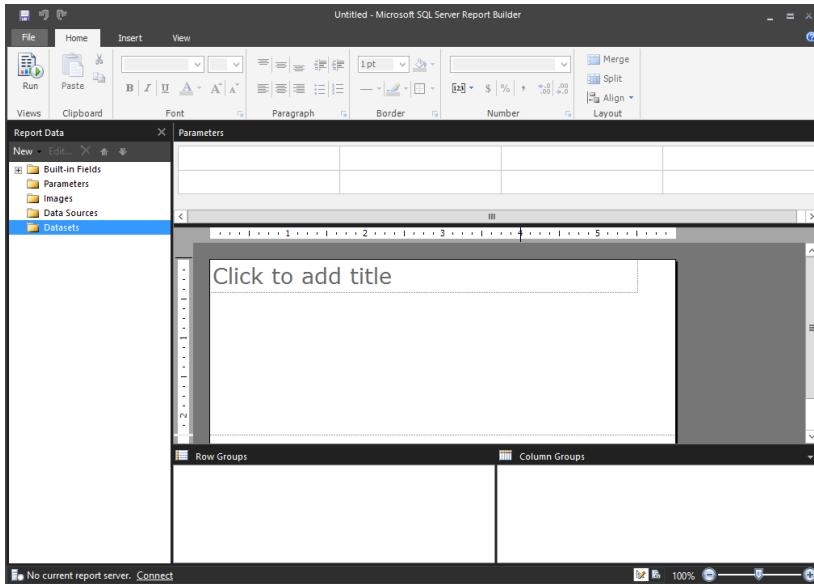
Figure 7-2: New Report Builder interface.

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.

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/region 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/region'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.
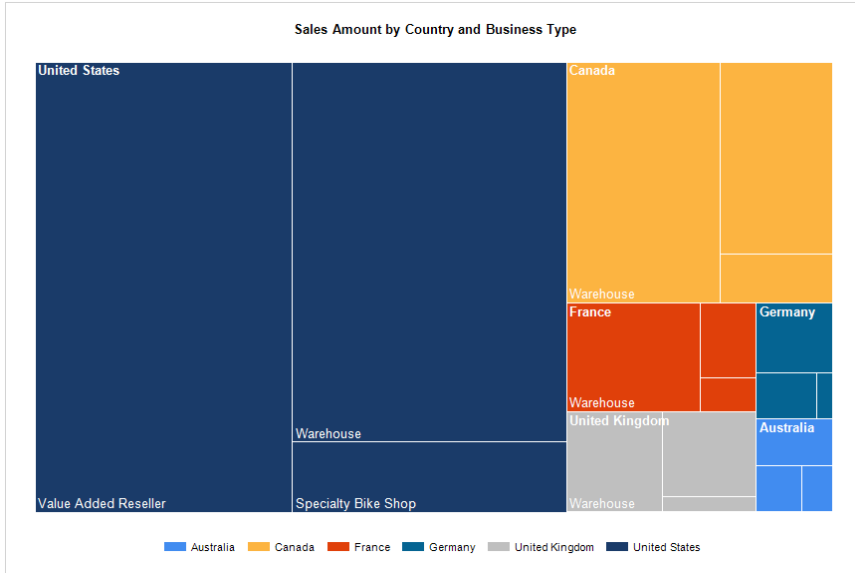
Figure 7-3: Tree map showing sales hierarchically by country/region and by business type.

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.
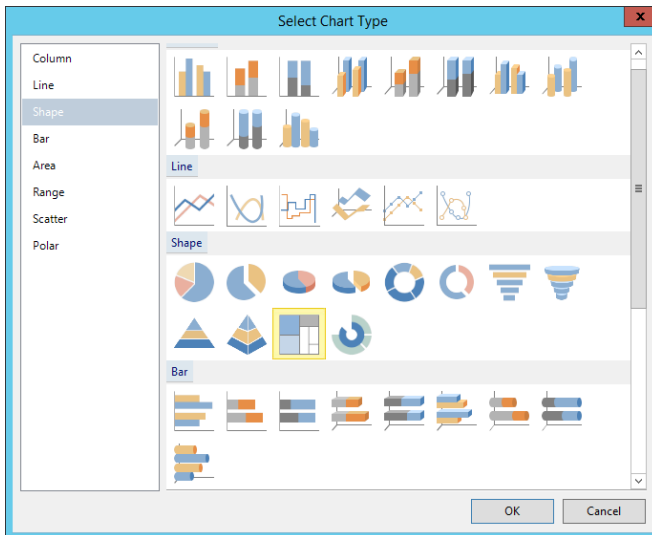

Figure 7-4: Selection of a tree map in the Select Chart Type dialog box.

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/region is identifiable by color in the tree map. Within each country's/region's rectangle, each distinct value within a category group is represented by a separate rectangle. In this case, each country's/region'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/region's total sales determines the size of its rectangle.
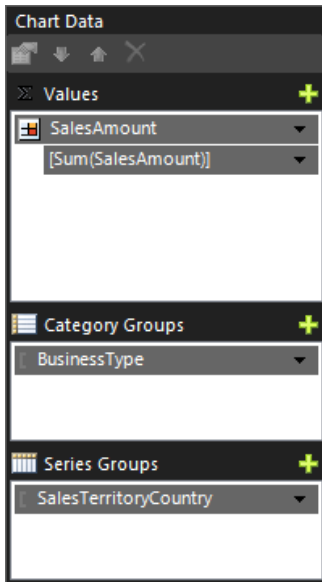
Figure 7-5: Configuring the Chart Data pane for a tree map.

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")
```
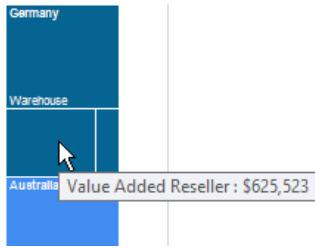
Figure 7-6: Tooltip displayed above a selected rectangle in a tree map.

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.
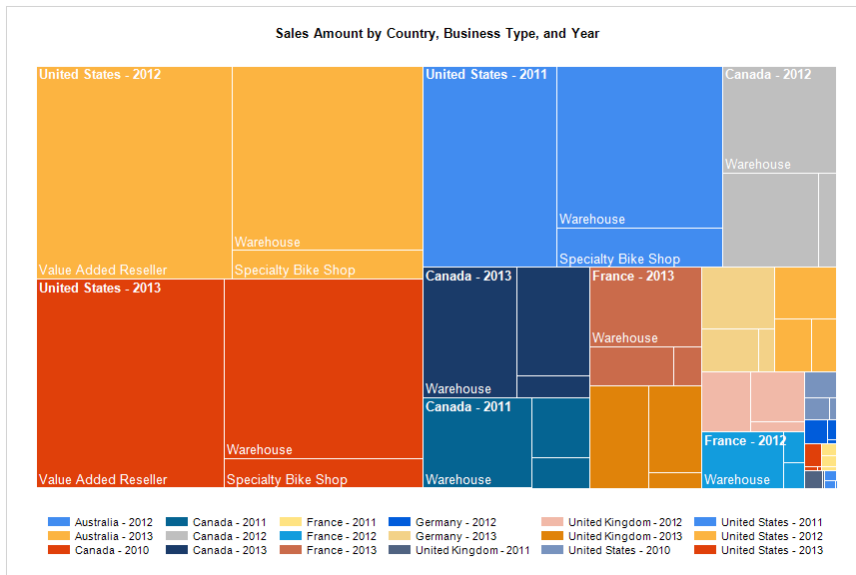

Figure 7-7: Tree map displaying two series groups.

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/region, again sorting the countries/regions from largest to smallest in clockwise order. The outer circle further subdivides the countries/regions 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.
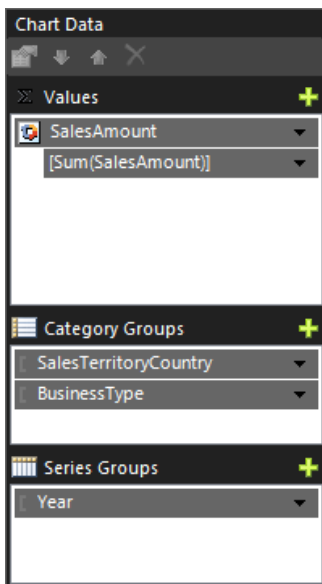

Figure 7-9: Chart Data pane configured for a sunburst chart.

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.
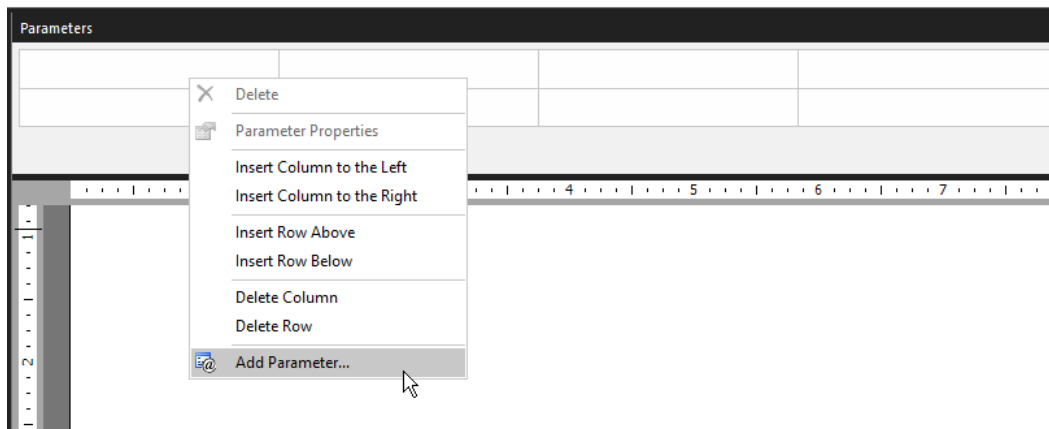

Figure 7-10: Adding a new parameter to a report by using the Parameters pane in Report Builder.

> **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, but these tools are currently in various states of integration. To create mobile reports, you use the SQL Server Mobile Report Publisher (which you can download from the Microsoft Store for Windows 8 and Windows 10).

> **Note** The Mobile Report Publisher is not available at the time of this writing. This section will be updated with more details about Mobile Report Publisher in the final version of this ebook.

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.

KPI development

In the CTP 3.2 release of SQL Server 2016, you use the Reporting Services web portal to create KPIs. From the main portal page at http://<yourserver>/reports, click the Preview The New Reporting Services link at the top of the page, click New in the toolbar, and then click KPI. A new KPI screen is displayed, as shown in Figure 7-13.

Figure 7-13: 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 dataset on the report server, or leave its value empty. (If you choose to use a shared dataset, 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** Datasets 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 7-14. The web portal includes a Favorites page on which you can organize reports by type: KPIs, mobile reports, and paginated reports. You can switch to the Browse page to view reports by navigating through folders.
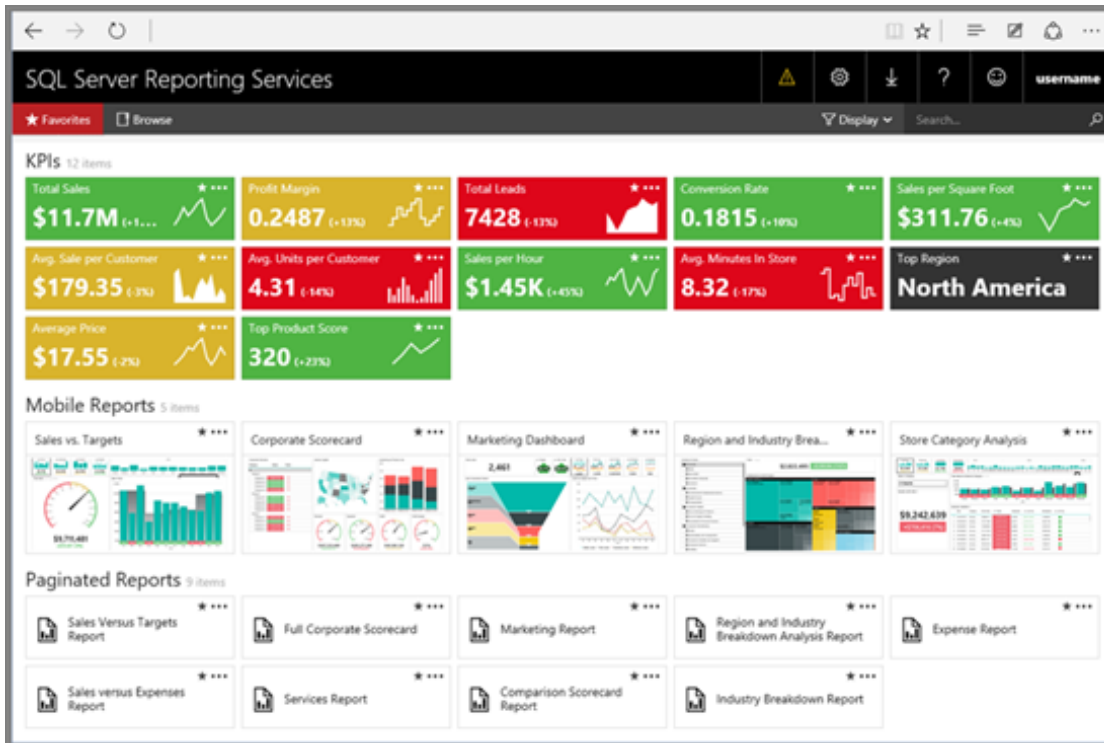
Figure 7-14: The home page of the new Reporting Services web portal displaying the Favorites tab.

Note   Mobile reports are not available in the new web portal in SQL Server 2016 CTP 3.2 but will be available in a future release of SQL Server 2016. This section will be updated in the final ebook.

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-

15. This dialog box allows you to adjust the paper size and page orientation by using the respective drop-down lists before printing your report.
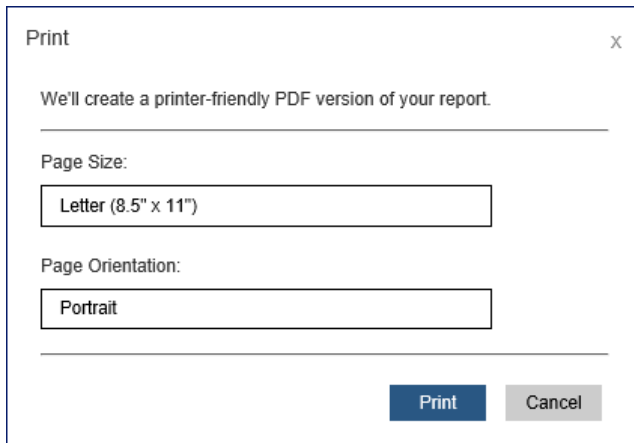

Figure 7-15: Print dialog box for browser without PDF control.

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 7-16. 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 7-16: 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 7-17.
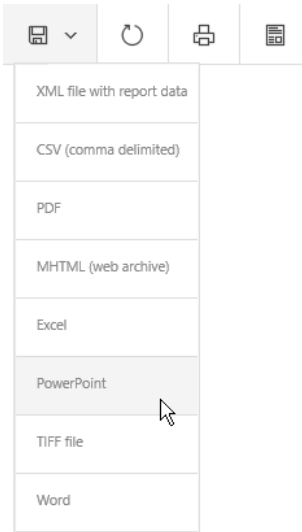
Figure 7-17: Choosing PowerPoint as an option for exporting a report.

> **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 7-18, 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 7-18, 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.
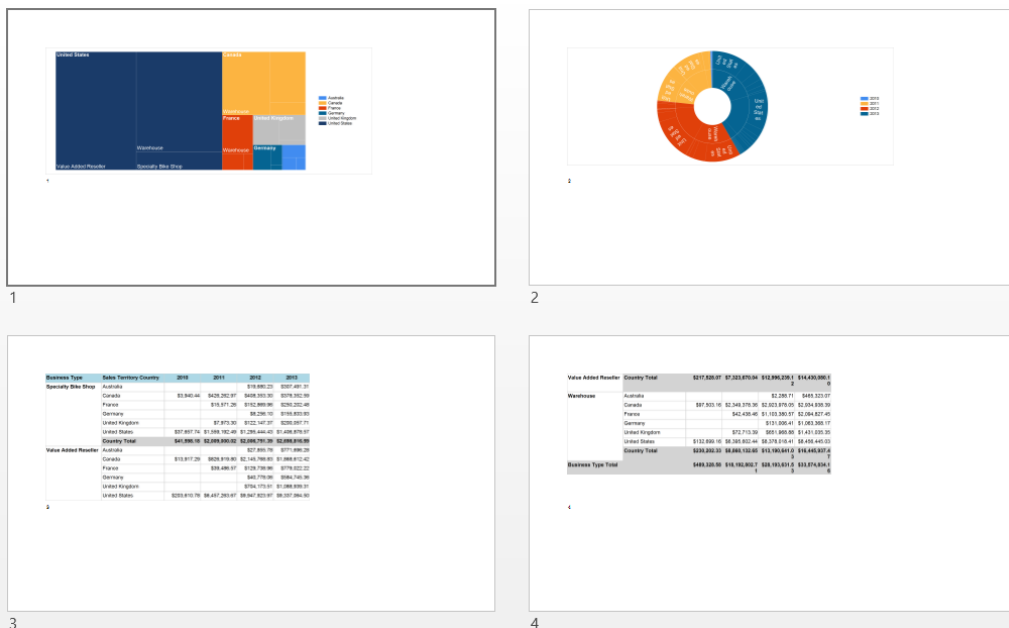

Figure 7-18: A report rendered as a PowerPoint file.

> **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 7-19.
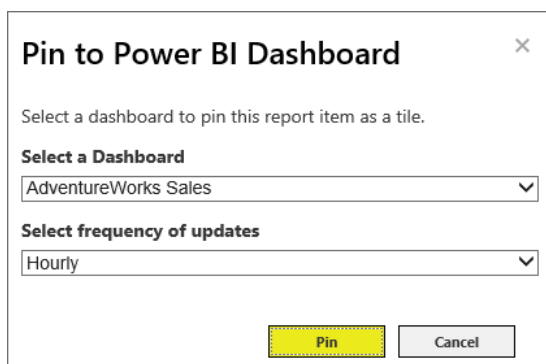

Figure 7-19: Selecting a dashboard for pinning a report.

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 7-20, 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.
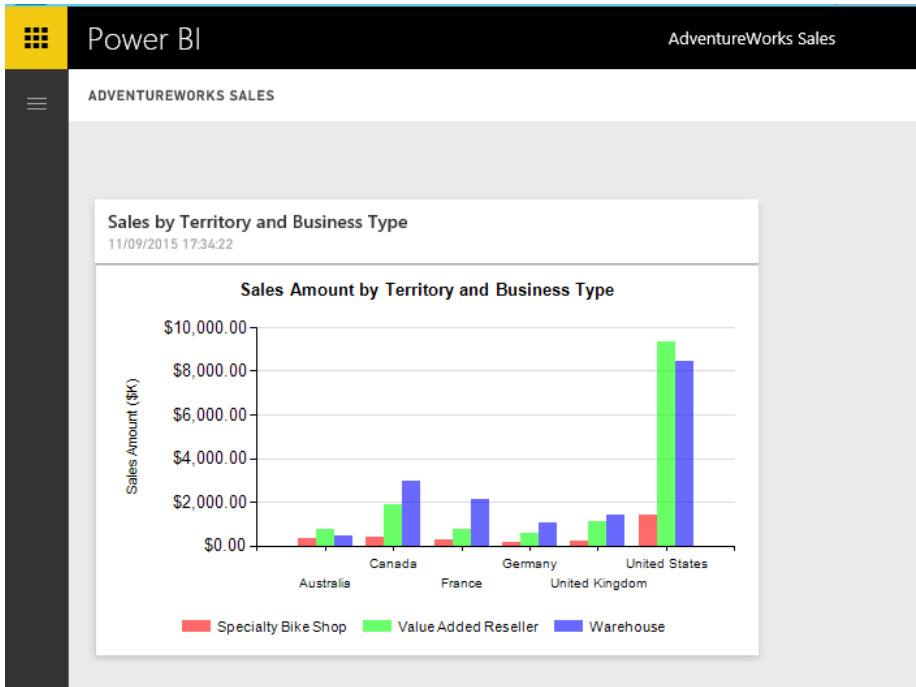
Figure 7-20: Displaying a Reporting Services report as a report tile in a Power BI dashboard.

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 Subscription Properties section, as shown in Figure 7-21, 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 7-21: A portion of a subscription definition showing the new Subscription Properties section.

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 7-22. You can sort subscriptions by the Description column by clicking the column header.

Figure 7-22: 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 icon to the left of the subscription displays a warning symbol and the Status column value changes to Disabled, as shown in Figure 7-23.

Figure 7-23: 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 7-24.
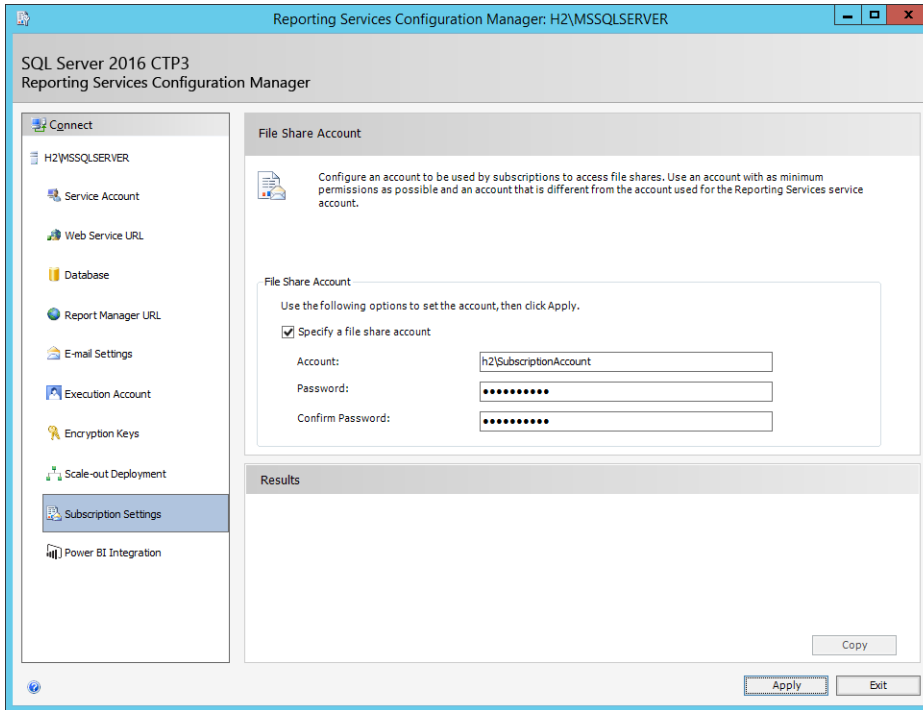

Figure 7-24: Subscription Settings page in Reporting Services Configuration Manager.

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 7-25. 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 7-25: The Use File Share Account option when configuring a file share subscription.

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.

Free ebooks

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

Hear about it first.

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

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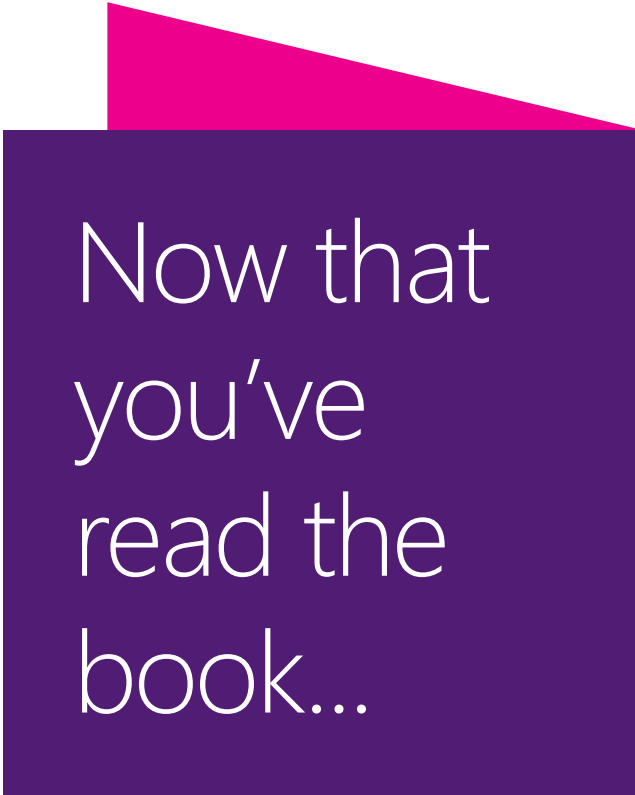- Special offers
- Free eBooks
- How-to articles

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

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!

■■ Microsoft