Microsoft®
SQL Server®

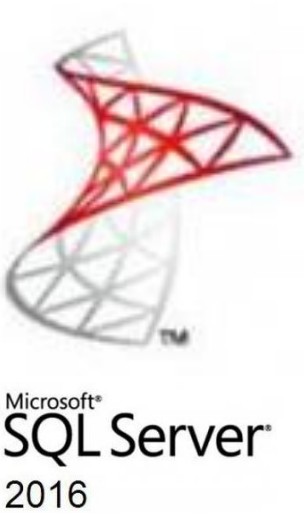
MICROSOFT SQL SERVER DATABASE ADMINISTRATION

Installing, Configuring and Administering one of the most popular RDMS

CORE360 IT SERVICES
TECHNOLOGY | BUSINESS SOLUTIONS | TRAINING

1A F.Jacinto St. Galguerra Cmpd. General T. De Leon, Valenzuela City
Email: johnreygoh@gmail.com
Mobile: +639954334855 | +639228720135
facebook.com/CORE360ITSERVICES
facebook.com/johnrey.goh

Compiled by John Rey Goh for Core360 IT Services
johnreygoh@gmail.com



SQL Server 2016 is a version of Microsoft's relational database management system (RDBMS) that first became available in preview releases during 2015, with general availability on June 1, 2016. SQL Server 2016 is a SQL-based database designed to support a mix of transaction processing, data warehousing and analytics applications in enterprise environments.

Like other RDBMS software, Microsoft SQL Server is built on top of SQL, a standardized programming language that database administrators (DBAs) and other IT professionals use to manage databases and query the data they contain. SQL Server is tied to Transact-SQL (T-SQL), an implementation of SQL from Microsoft that adds a set of proprietary programming extensions to the standard language.

Microsoft offers SQL Server in four primary editions that provide different levels of the bundled services. Two are available free of charge: a full-featured Developer edition for use in database development and testing, and an Express edition that can be used to run small databases with up to 10 GB of disk storage capacity. For larger applications, Microsoft sells an Enterprise edition that includes all of SQL Server's features, as well as a Standard one with a partial feature set and limits on the number of processor cores and memory sizes that users can configure in their database servers.

Overview of MS SQL Server 2016

SQL Server 2016 Editions

SQL Server edition	Definition
Enterprise	The premium offering, SQL Server Enterprise edition delivers comprehensive high-end datacenter capabilities with blazing-fast performance, unlimited virtualization, and end-to-end business intelligence — enabling high service levels for mission-critical workloads and end user access to data insights.
Standard	SQL Server Standard edition delivers basic data management and business intelligence database for departments and small organizations to run their applications and supports common development tools for on-premise and cloud — enabling effective database management with minimal IT resources.
Web	SQL Server Web edition is a low total-cost-of-ownership option for Web hosters and Web VAPs to provide scalability, affordability, and manageability capabilities for small to large scale Web properties.
Developer	SQL Server Developer edition lets developers build any kind of application on top of SQL Server. It includes all the functionality of Enterprise edition, but is licensed for use as a development and test system, not as a production server. SQL Server Developer is an ideal choice for people who build
Express editions	Express edition is the entry-level, free database and is ideal for learning and building desktop and small server data-driven applications. It is the best choice for independent software vendors, developers, and hobbyists building client applications. If you need more advanced database features, SQL Server Express can be seamlessly upgraded to other higher end versions of SQL

Server. SQL Server Express LocalDB, a lightweight version of Express that has all of its programmability features, yet runs in user mode and has a fast, zero-configuration installation and a short list of prerequisites.

Hardware and Software Requirements for Installing SQL Server

The following requirements apply to all installations:

Component	Requirement
.NET Framework	SQL Server 2016 (13.x) RC1 and later require .NET Framework 4.6 for the Database Engine, Master Data Services, or Replication. SQL Server 2016 setup automatically installs .NET Framework. You can also manually install .NET Framework from Microsoft .NET Framework 4.6 (Web Installer) for Windows.

Windows 8.1, and Windows Server 2016 R2 require KB2919355 before installing .NET Framework 4.6.

Network Software	Supported operating systems for SQL Server have built-in network software. Named and default instances of a stand-alone installation support the following network protocols: Shared memory, Named Pipes, TCP/IP and VIA.
Hard Disk	SQL Server requires a minimum of 6 GB of available hard-disk space.
Drive	A DVD drive, as appropriate, is required for installation from disc.
Monitor	SQL Server requires Super-VGA (800x600) or higher resolution monitor.
Internet	Internet functionality requires Internet access (fees may apply).
Memory	Minimum: Express Editions: 512 MB All other editions: 1 GB Recommended: Express Editions: 1 GB *All other editions: At least 4 GB and should be increased as database size increases to ensure optimal performance.
Processor Speed	Minimum: x64 Processor: 1.4 GHz Recommended: 2.0 GHz or faster
Processor Type	x64 Processor: AMD Opteron, AMD Athlon 64, Intel Xeon with Intel EM64T support, Intel Pentium IV with EM64T support

NOTE: SQL SERVER Supports side-by-side installation of different versions. The first installed versions will be set to default. Make sure to set unique named instances for each version. This instance name is specified when connecting to the database engine from the SQL SERVER MANAGEMENT STUDIO.

SQL SERVER USER ACCOUNTS

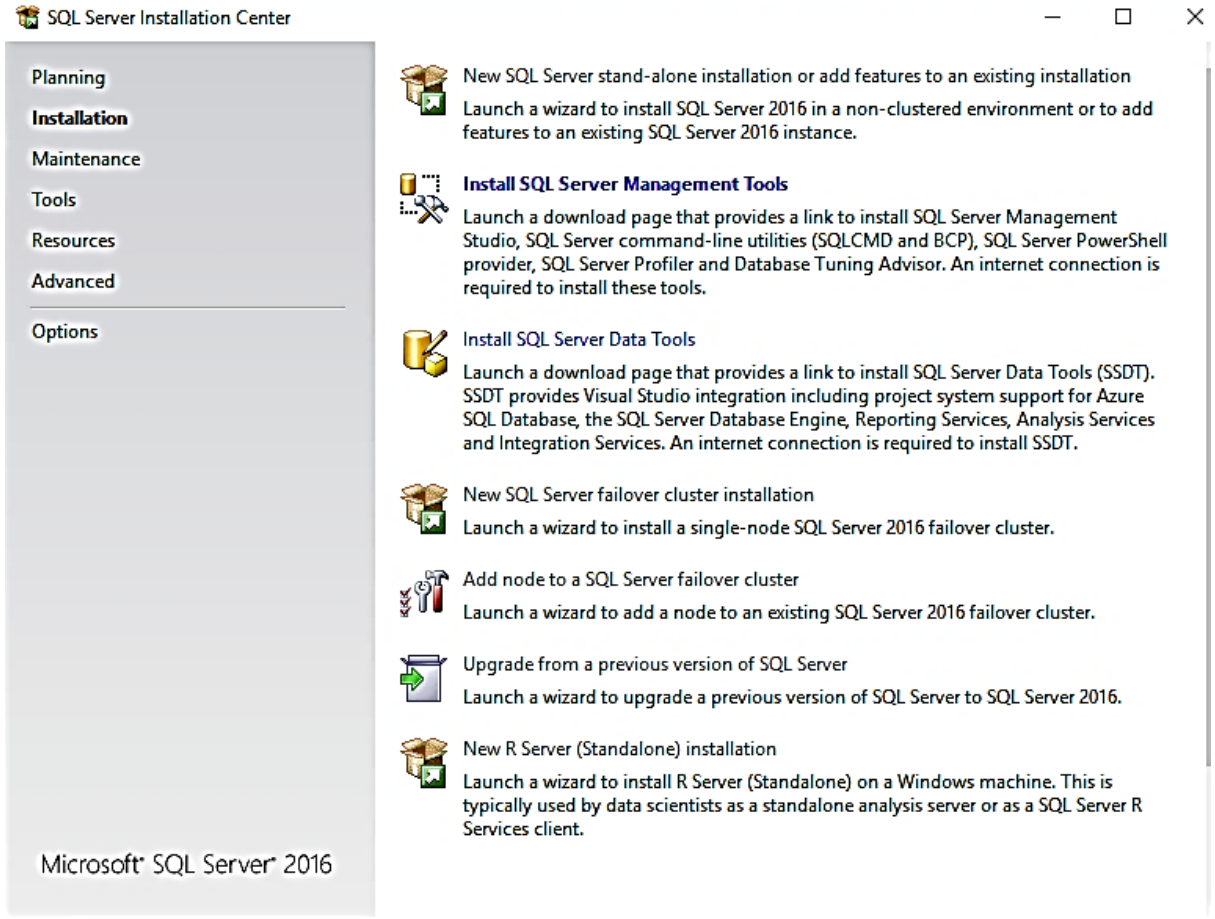
- Windows-Authentication (local users or domain users)
- SQL Server Authentication (application users defined in SQL Server)

1. Installing, configuring and upgrading

SQL SERVER 2016 INSTALLATION

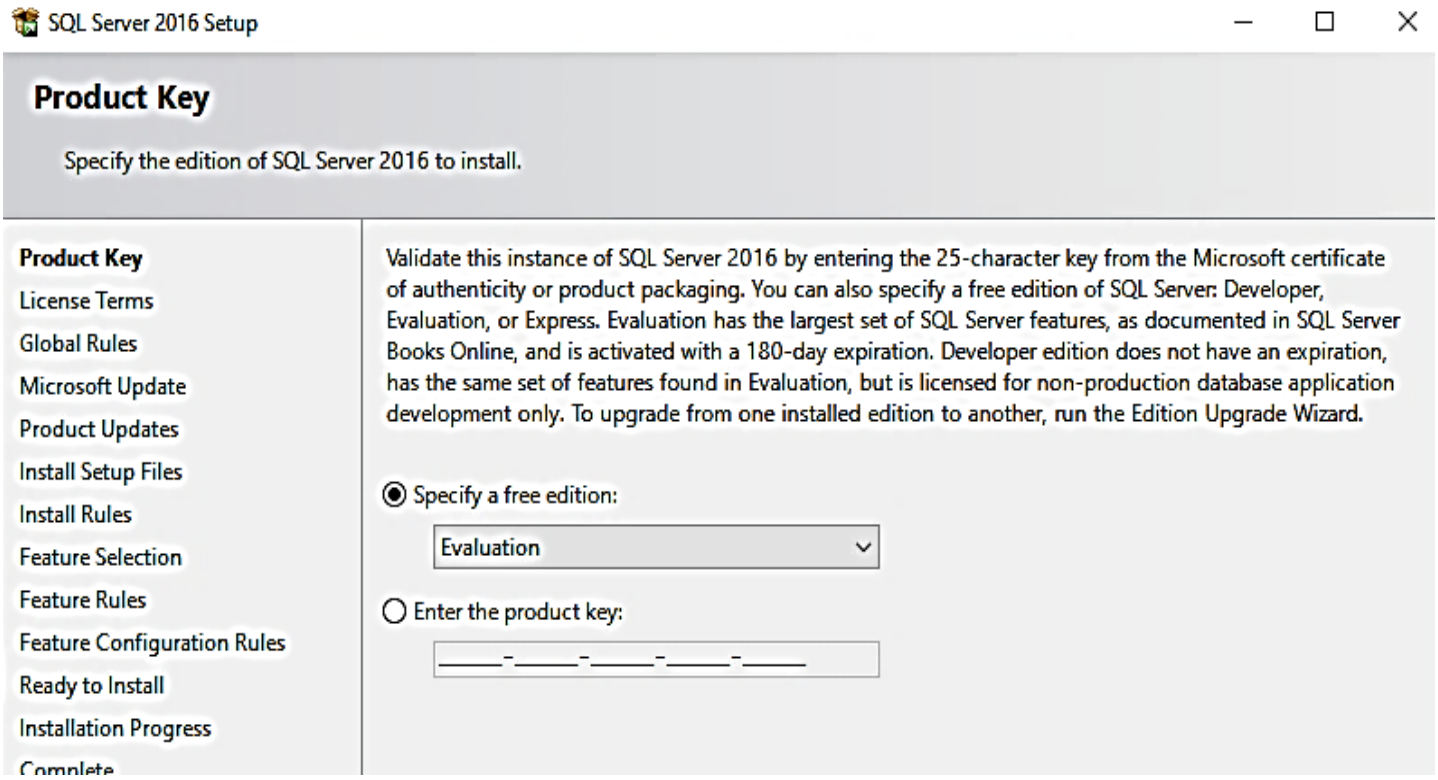
1. Either insert the SQL Server 2016 media into a DVD or CD drive or access it from a local or networked drive. Look in the root folder of the media and double-click setup.exe. If you are using an image file, you will need to use a tool to mount it or software such as WINRAR to extract the contents.

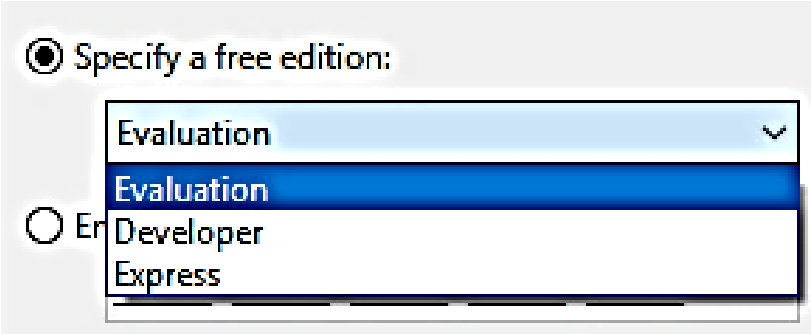
2. The SQL Server Installation 2016 Setup wizard will open. Select Installation from the left navigation area. On the right, click New SQL Server Stand-Alone Installation or Add Features to an Existing Installation, depending on your needs. This is also where you will install SQL Server Management Studio later.



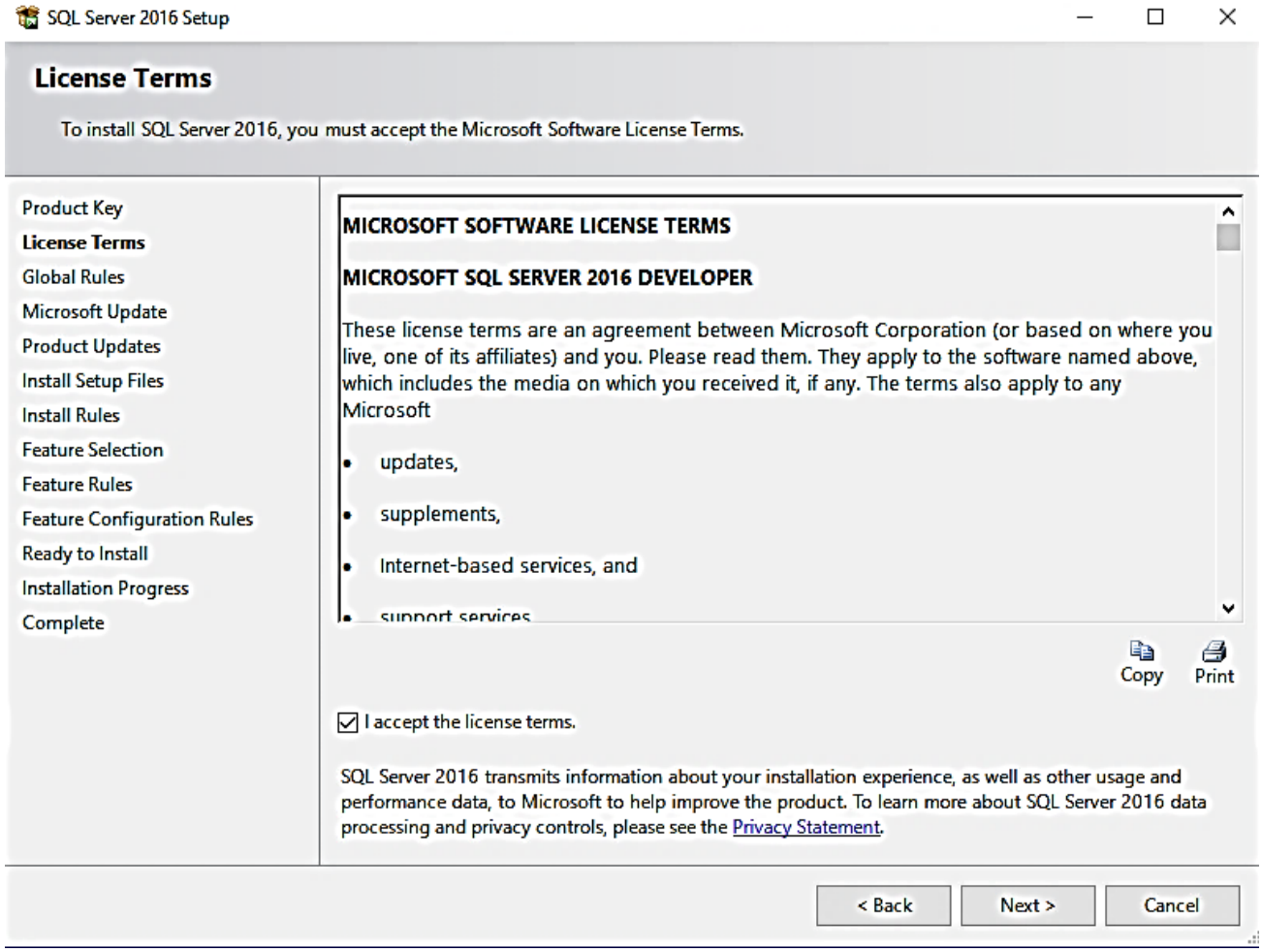
3. The installer will then execute a list of setup support rules. To view a complete list, click the Show Details button. Click OK.

4. If you have a product key, select the radio button labeled Enter the Product Key, and then enter the product key. If you don't have a product key, click the Specify a Free Edition radio button and select Evaluation from the list of available choices. Then click Next.

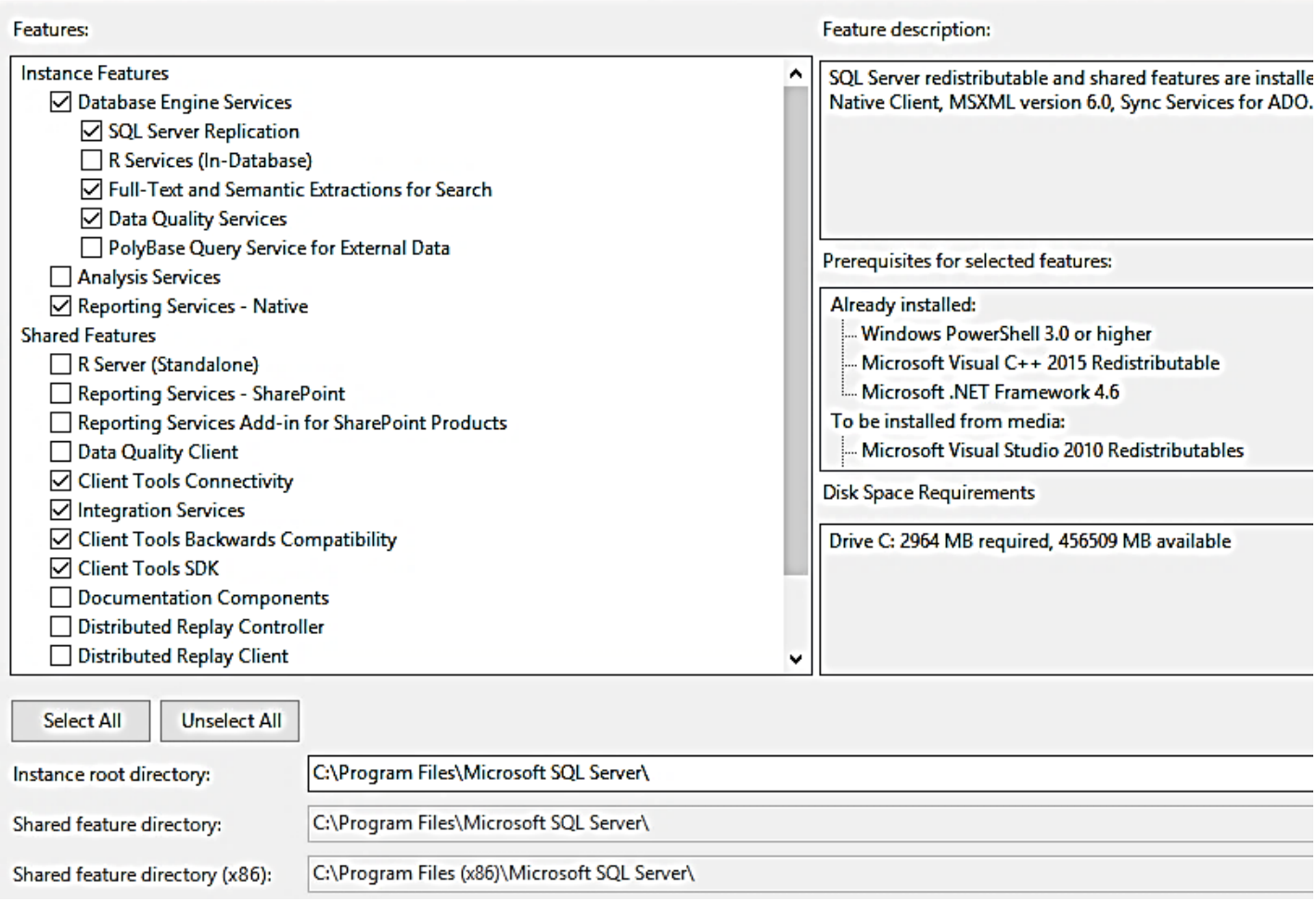




5. On the next page, check the box labeled "I accept the license terms." You also have the option of sending additional information to Microsoft about your installation. Make your choice and click Next. Note that the button labeled Next will not be enabled unless you accept the license terms. The installer will then install the necessary setup files.



6. After the setup files are installed, another set of setup support rules will run. Click Next.
7. Now you must select the server role. Select the SQL Server Feature Installation radio button, and click Next.
8. On the Feature Selection page, select the following. We will focus on these foundation features, but you can install others if you want, as well. Also, if you are installing a second instance of SQL Server 2016, the shared features will already be installed, so the Shared Features options will be grayed out.

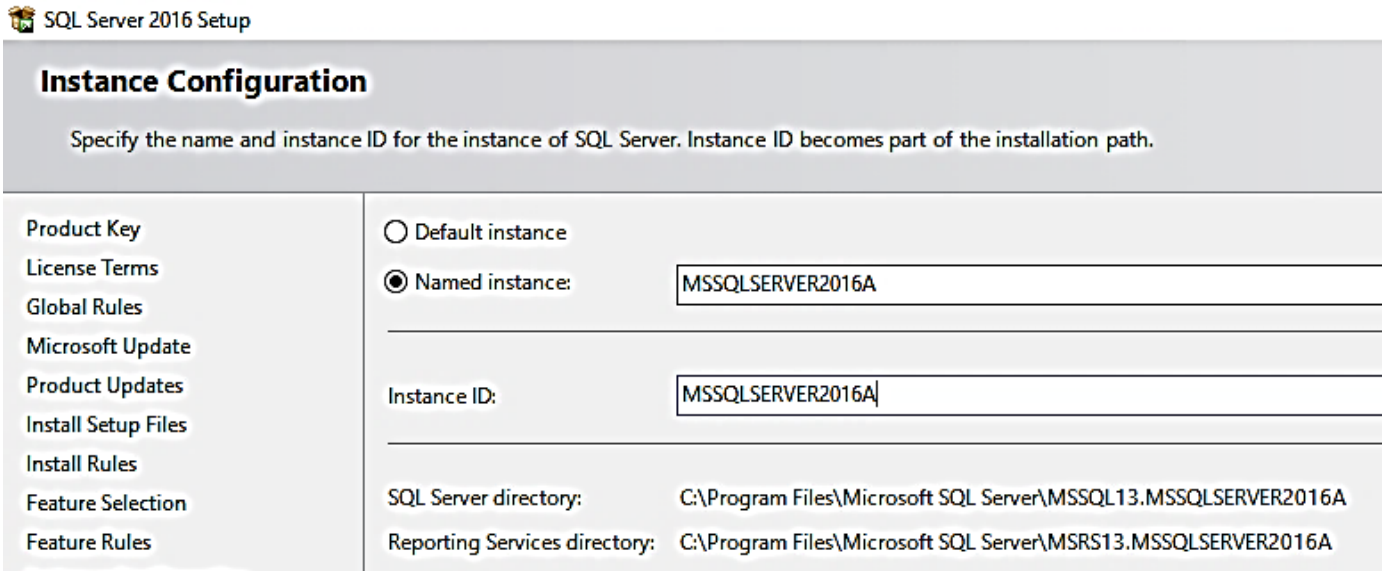


9. Toward the bottom of the page are options for specifying the directory where you want to install the features. Accept the defaults and click Next.

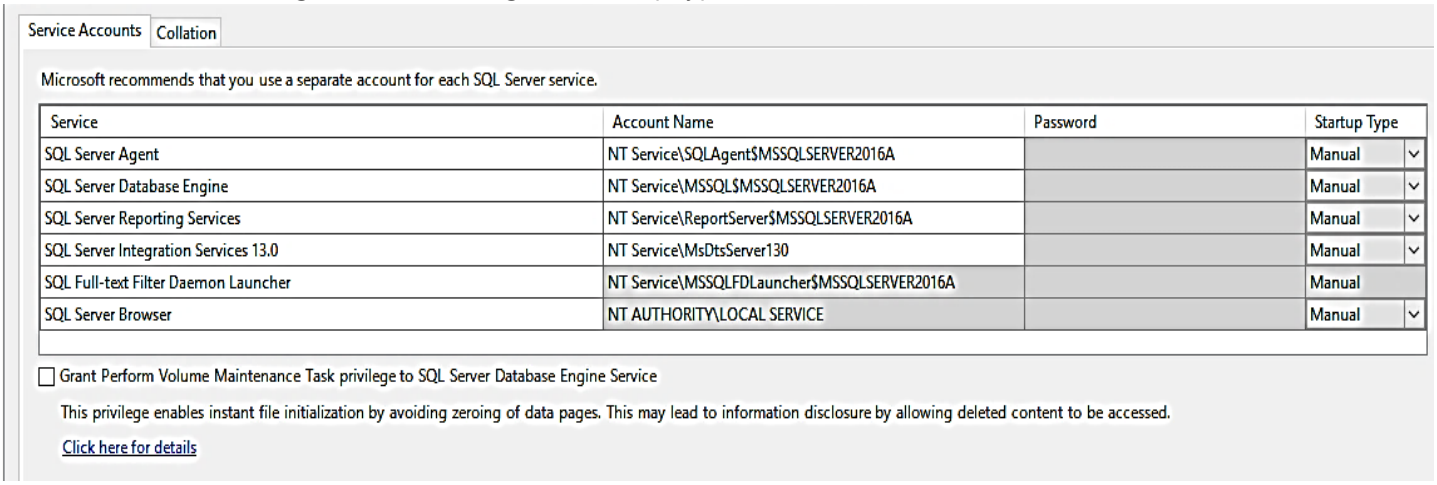
10. A few more installation rules will execute. If you have installed all the proper prerequisites, everything should run successfully. Click Next.

11. On the Instance Configuration page, select whether to install a default installation or a named instance. If a default instance is already installed, your only choice will be to install a named instance. The Named Instance text box will display the name you use to connect to the SQL Server—for example, ServerName\InstanceName\. The instance ID is used to identify installation directories and registry keys for an instance of SQL Server. Click Next.

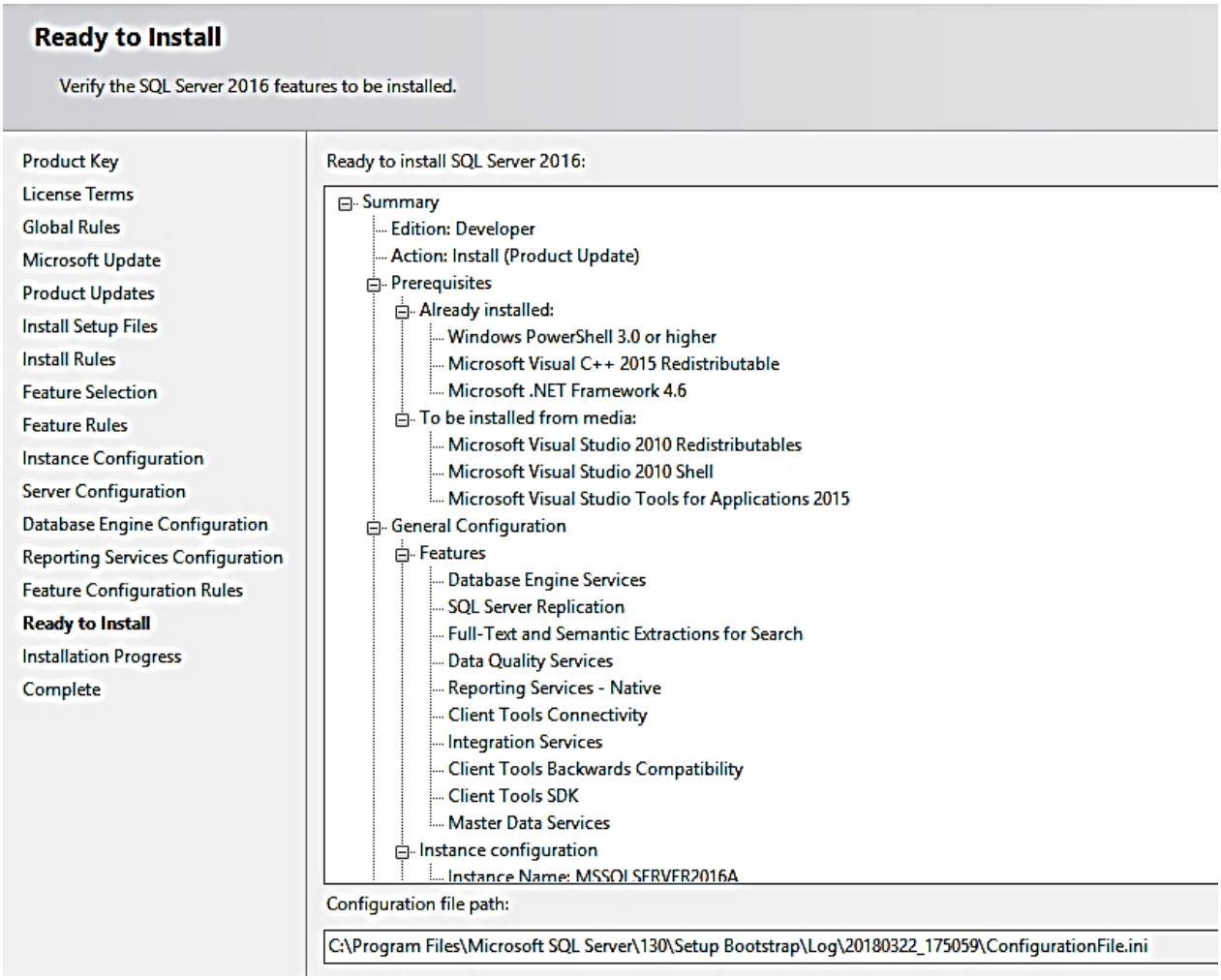
12. The Disk Space Requirements page summarizes how much available space there is and how much is required. Click Next.

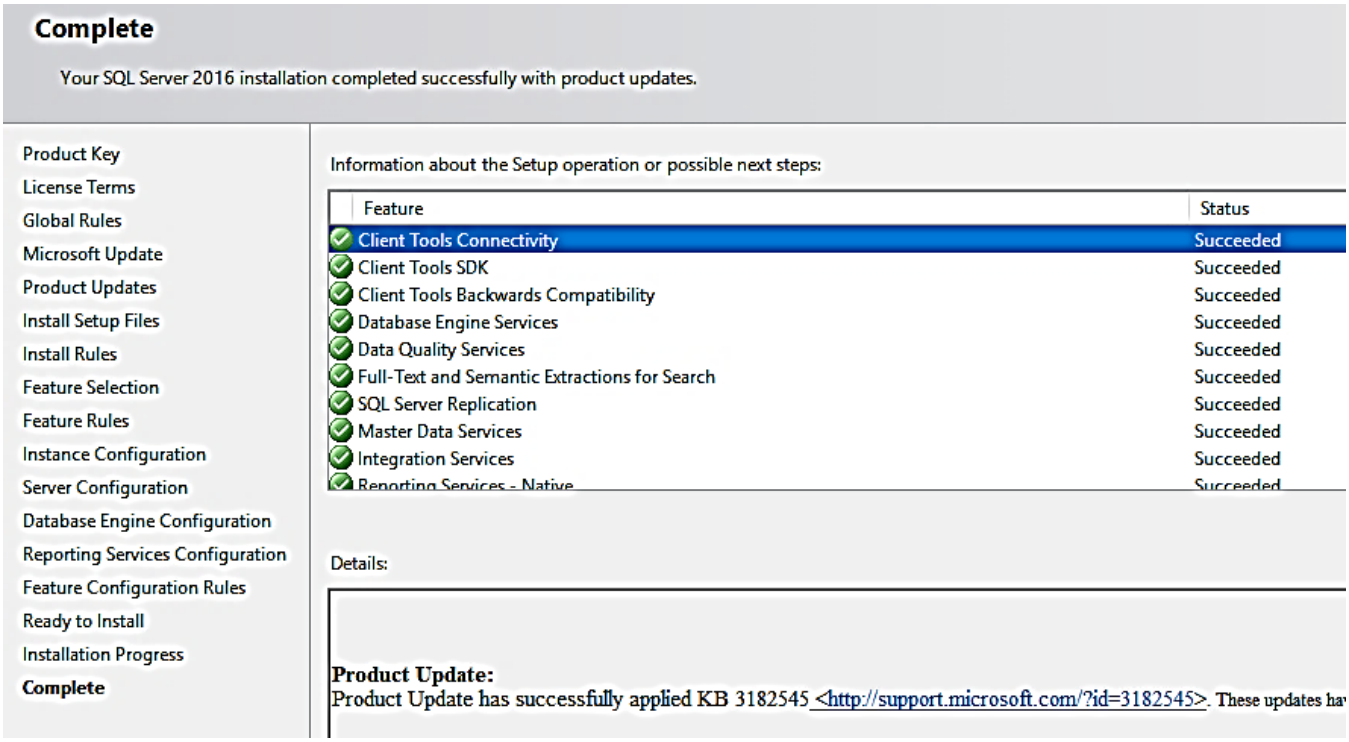


13. On the Server Configuration page, you specify the login accounts and startup types for the SQL Server services. If you want other services to communicate with SQL Server and vice versa, you must specify a domain account, an MSA, or a virtual account as the login account for Database Engine. For now, accept the defaults. For the SQL Server Agent service, change the startup type to Automatic.



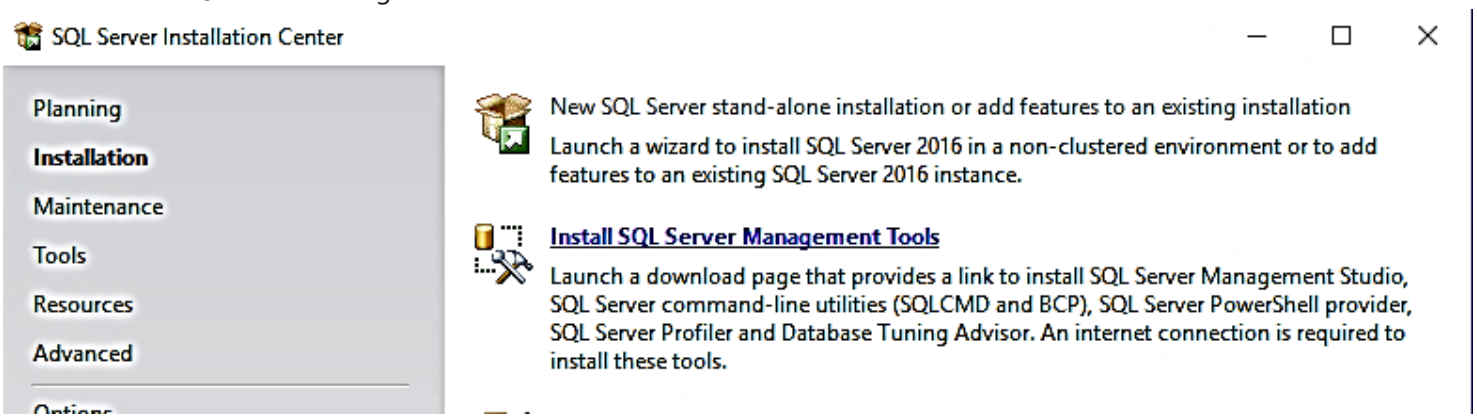
14. On the Database Engine Configuration page, first select your authentication mode. Select the radio button labeled Mixed Mode (SQL Server Authentication and Windows Authentication). Then specify a password for the SA account. Provide a password of your choice. Click the button labeled Add Current User. On the Data Directories tab, you can change the location where the system databases and user databases are stored. For now, accept the defaults.
15. If you want to report errors about the installation, select the check box on the Error Reporting page, and then click Next.
16. One last rules check is run. If everything passes, click Next.
17. You are now ready to install.

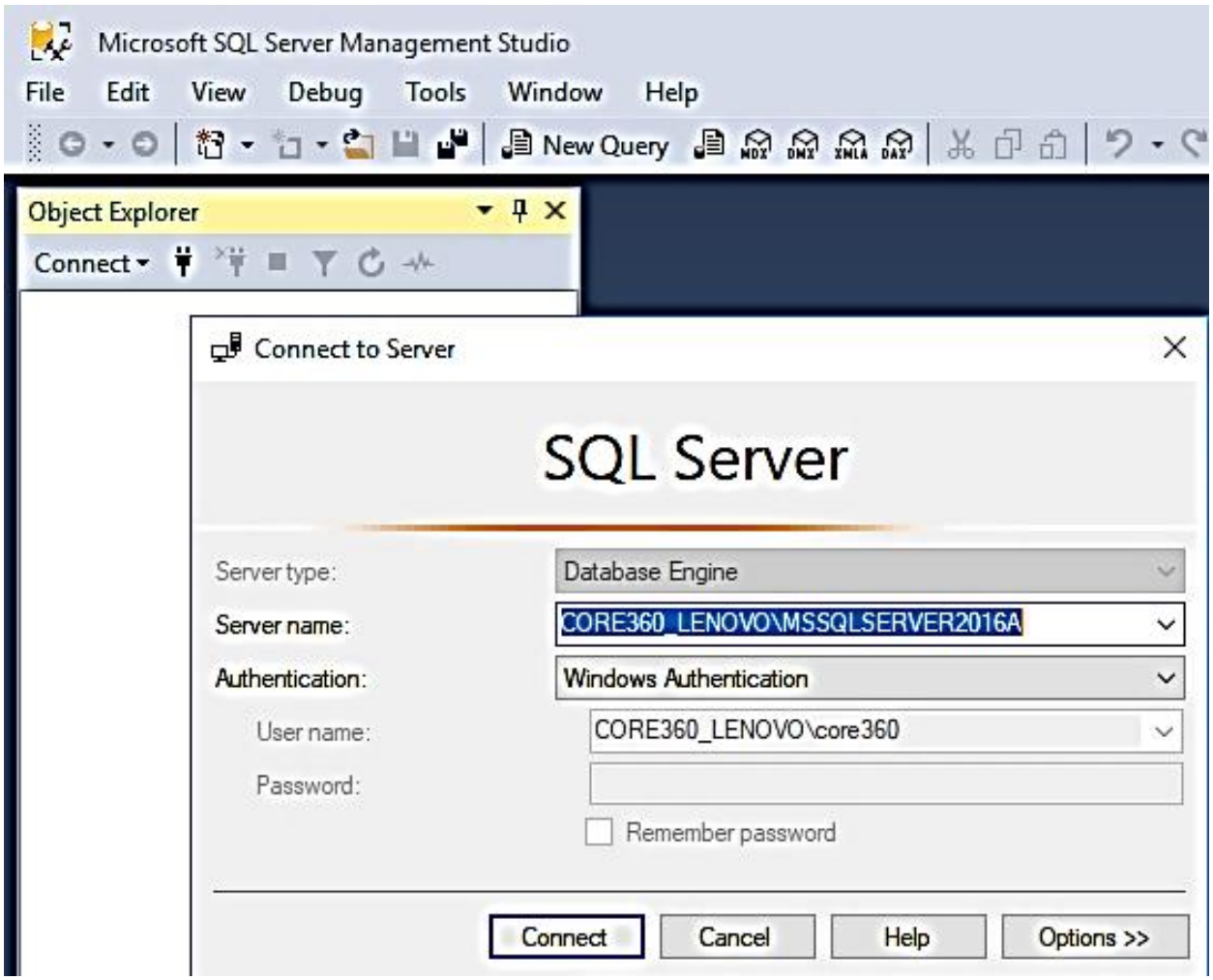

Ready to Install
Verify the SQL Server 2016 features to be installed.
Product Key
License Terms
Global Rules
Microsoft Update
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Reporting Services Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Ready to install SQL Server 2016:
Summary
Edition: Developer
Action: Install (Product Update)
Prerequisites
Already installed:
Windows PowerShell 3.0 or higher
Microsoft Visual C++ 2015 Redistributable
Microsoft .NET Framework 4.6
To be installed from media:
Microsoft Visual Studio 2010 Redistributables
Microsoft Visual Studio 2010 Shell
Microsoft Visual Studio Tools for Applications 2015
General Configuration
Features
Database Engine Services
SQL Server Replication
Full-Text and Semantic Extractions for Search
Data Quality Services
Reporting Services - Native
Client Tools Connectivity
Integration Services
Client Tools Backwards Compatibility
Client Tools SDK
Master Data Services
Instance configuration
Instance Name: MSSQLSERVER2016A
Configuration file path:
C:\Program Files\Microsoft SQL Server\130\Setup Bootstrap\Log\20180322_175059\ConfigurationFile.ini


Complete
Your SQL Server 2016 installation completed successfully with product updates.
Product Key
License Terms
Global Rules
Microsoft Update
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Instance Configuration
Server Configuration
Database Engine Configuration
Reporting Services Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Information about the Setup operation or possible next steps:
Feature
Status
Client Tools Connectivity
Succeeded
Client Tools SDK
Succeeded
Client Tools Backwards Compatibility
Succeeded
Database Engine Services
Succeeded
Data Quality Services
Succeeded
Full-Text and Semantic Extractions for Search
Succeeded
SQL Server Replication
Succeeded
Master Data Services
Succeeded
Integration Services
Succeeded
Reporting Services - Native
Succeeded
Details:
Product Update:
Product Update has successfully applied KB 3182545 <http://support.microsoft.com/?id=3182545>. These updates ha

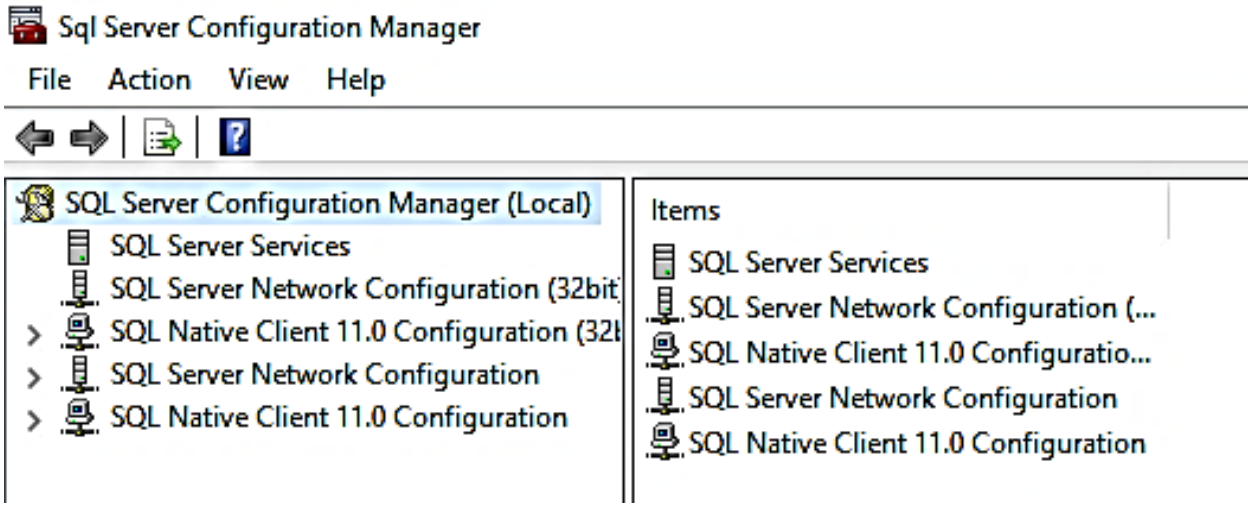
2. Using Management and Configuration tools

✓ Install SQL Server Management Studio





- ✓ **SQL SERVER CONFIGURATION MANAGER**
  a. View and manage services
  b. Set network and protocol configurations



POST-INSTALLATION TASKS

- ✓ Assigning a TCP/IP Port Number to the SQL Server Database Engine

First, you may want to change the SQL Servers default TCP port from 1433 to a different port.

1. Open SQL Server Configuration Manager.
2. In the left-hand navigation pane, expand SQL Server Network Configuration and click Protocols for MSSQLSERVER. If you are changing the port for a non default instance, then you will click Protocols for <Instance Name>.

3. Right-click TCP/IP in the left section.

4. You can configure each specific IP address, or you can configure the port for all IP addresses. To do so, click the IP Addresses tab, scroll to the bottom to locate IPALL, and change the port number to your desired port. Don't change the port, as this will require you to include the port number when connecting to this server.

5. Restart the instance of SQL Server that has been changed.

Once this change is made, you are required to specify this port number when you connect to the SQL Server instance.

3. Designing databases

Understanding sql server system databases

- master
- model
- msdb
- tempdb
- resource
- distribution

master

The master database, as its name suggests, is the primary system database. Without it, SQL Server cannot start. The master database contains the most important information about objects within the SQL Server instance, such as the following:

- ✓ Databases
- ✓ AlwaysON
- ✓ Database mirroring
- ✓ Configurations
- ✓ Logins
- ✓ Resource Governor
- ✓ Endpoints

For example, if you want to quickly obtain a list of all the databases on an instance of SQL Server, you can execute the following query:

```
Select * from sys.master_files
```

model

The model database is exactly what its name implies: a model for all databases that are created on an instance of SQL Server. In other words, it's used as a template each time you create a database. For example, if you want a particular table to exist in every database created on an instance of SQL Server, you will create that table in the model database. As a result, each time a database is created, it will include that table.

msdb

This serves primarily as the back-end database for Microsoft SQL Server Agent. Whenever you create and/or schedule a SQL Server Agent job, the metadata for that job is stored in this database. In addition to SQL Server Agent data, msdb stores information for the following components:

- ✓ Service brokers
- ✓ Alerts
- ✓ Log shipping
- ✓ SSIS packages
- ✓ Utility control point (UCP)
- ✓ Database mail
- ✓ Maintenance plans

tempdb

The tempdb database is a global playground for temporary objects created by the internal processes that run SQL Server and temporary objects that are created by users or applications. These temporary objects included temporary tables and stored procedures, table variables, global temporary tables, and cursors. In addition to temporary objects, tempdb stores row versions for read-committed or snapshot isolation transactions, online index operations, and AFTER triggers. One important thing to note about tempdb is that it is re-created every time SQL Server is restarted. Although you can create objects in tempdb, you should never use it as a database where persisted information is stored.

resource

The resource database is a hidden, read-only database that is usually not discussed very often. The resource database's primary purpose is to improve the upgrade process from one version of SQL Server to the next. All system objects for an instance of SQL Server are stored within the resource database. This database cannot be backed up or restored. You should not attempt to change or move this database unless Microsoft Customer Support directs you to do so.

distribution

This database exists only when you have configured this instance as a distributor for replication. Prior to configuring replication, you must perform this configuration. All metadata and history for the various types of replication are stored within this database.

Understanding SQL Server Database structures

Every SQL Server database consists of two files

- ✓ The data file (.mdf) contains data and database objects such as tables, views, and stored procedures.
- ✓ The log file (.ldf) contains information that assists in the recoverability of transactions in the database.

*the settings can be configured as you create a new database file in ssms.

Activity: Create a database with a custom path and custom database name and log file name

Activity: Create a database using a T-sql query

```
USE master;
CREATE DATABASE sampledb1
ON PRIMARY
(NAME=' sampledb1 ', FILENAME = 'C:\SQLDATA\sampledb1.mdf', SIZE=10MB, MAXSIZE=20, FILEGROWTH=10%)
LOG ON (NAME='SBSChp4TSQL_log', FILENAME = 'C:\SQLLog\sampledb1 _log.ldf',  SIZE=10MB, MAXSIZE=200,
FILEGROWTH=20%);
```

Activity: Create a database using a solution script file

Adding files and filegroups

Note: primary data files are (.mdf) while you can also create secondary data files (.ndf) for user-defined objects or to hold exceeding capacity from primary data files.

Activity: create a new filegroup and add a secondary database file (.ndf) in the new filegroup

1. Your database properties → filegroup → add filegroup, set as default (optional)
2. Your database properties → file → add → set logical name, filegroup and set path and filename where to save.

Activity: create a new filegroup and add a secondary database file (.ndf) in the new filegroup using T-SQL

```
USE master;
ALTER DATABASE sampledb1
    ADD FILEGROUP samplegroup1;

ALTER DATABASE sampledb1
    ADD File
    (
        NAME=samplefile1,
        FILENAME = 'C:\SQLDATA\samplefile1.ndf',
        SIZE=10MB,
        MAXSIZE=20,
        FILEGROWTH=10%
    )
TO FILEGROUP samplegroup1;
```

Detaching and Attaching SQL Server databases

a. To detach and attach a database
   - ✓ Rclick your database → detach (check "drop connections" and "update statistics"), you can now copy your files.
   - ✓ Rclick the databases folder → attach database → browse for your database file. The log file must also be on the same directory as the database file so it could be automatically attached as well.

b. To detach and attach a database using TSQL

```
USE Master;
EXEC sp_detach_db @dbname = 'sampledb1';
```

```
USE master;
CREATE DATABASE sampledb1 ON
(FILENAME = 'C:\SQLData\sampledb1.mdf'),
(FILENAME = 'C:\SQLData\samplefile1.ndf'),
(FILENAME = 'C:\SQLLog\sampledb1_Log.ldf')
FOR ATTACH;
```

Understanding database recovery models

To set recovery model for your database:

Database properties → options → recovery model (simple,full,bulk-logged)

The model determines how precisely a database may be restored.

Simple model

The simple model does not allow for transaction log backups. As a result, you cannot restore a database to a point in time. Your database is vulnerable to data loss when using this model. That said, using this model does ease the task of administration because SQL Server will reclaim space automatically from the transaction log.

Full model

With the full model, data loss is minimal when the transaction log is backed up on a regular basis. Every transaction is fully logged to the transaction log, and the transaction log will continue to grow until it is backed up. While this model does add administrative overhead, your data is protected from data loss.

Bulk-logged model

When you use the bulk-logged model, bulk operations are minimally logged, which reduces the size of the transaction log. Note that this does not eliminate the need to back up the transaction log. Unlike in the full recovery model, in the bulk-logged model you can restore only to the end of any backup; you cannot restore to some point in time.

Plan and adopt a Naming Standard for your design

The following are some best practices:

a. **General standards**
   - ✓ Do not use spaces within any object or column name.
   - ✓ Underscore characters are acceptable, but be aware that they can present some challenges with visualization tools.
   - ✓ Use PascalCase, which means capitalizing the first letter of each word that is used to name an object or column.
   - ✓ Do not use reserved keywords. Plural table and column names are acceptable.

b. **Table naming standards**
   - ✓ Names should reflect the contents of the table.
   - ✓ Names must be unique to the database and the schema.

c. **Column naming standards**
   - ✓ Names should be unique to each table.
   - ✓ Names should reflect the business use.
   - ✓ Select the appropriate data type, as discussed later in this chapter.

Understanding Schemas

Schemas offer another level of containment and organization within a database. By default, the "database owner or [dbo]" schema is used and any objects created are added to this schema.

Activity: Create a new schema for your database objects

Expand your database folder → expand security folder → rclick Schema folder (new schema) → type a name for the schema and set the user (ex. "dbo") for the schema.

Activity: Create a new schema for your database objects using TSQL

```
USE sampledb1;
GO
CREATE SCHEMA Sales;
GO
CREATE SCHEMA HumanResources;
GO
```

Note: create schema prior to tables. If not possible, make sure to move the tables to their schemas after creation.

4. Creating tables

Understanding SQL SERVER Data Types

The four categories of data types in SQL SERVER:

NUMERIC STRINGS DATE AND TIME OTHER

NUMERIC

Data Type	Range	Storage
bigint	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8 bytes
int	–2,147,483,648 to 2,147,483,647	4 bytes
smallint	–32,768 to 32,767	2 bytes
tinyint	0 to 255	1 byte
money	–922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
smallmoney	–214,748.3648 to 214,748.3647	4 bytes

Here are the types with decimal places:

- **decimal**
  *they have precision (number of digits you can store on both sides of the decimal places) and scale (number of digits stored in the right side).

Precision	Storage
1–9	5 bytes
10–19	9 bytes
20–28	13 bytes
29–38	17 bytes

- **float**

n**value**	**Precision**	**Storage**
1–24	7 digits	4 bytes
25–53	15 digits	8 bytes

STRING

The character string subcategory will store non-Unicode data. The three types are as follows:

- char(n) Fixed-length string data type with a string length between 1 and 8,000.
- varchar(n) Variable-length string data type that can store up to 2 GB of data.
- text Deprecated data type. Replace it with a varchar(max).

The Unicode string subcategory will store both Unicode and non-Unicode data. The three types are as follows:

- nchar(n) Fixed-length string data type with a string length between 1 and 4,000.
- nvarchar(n) Variable-length string data type that can store up to 2 GB of data.
- ntext Deprecated data type. Replace it with nvarchar(max).

The binary string subcategory will store binary data. The three types are as follows:

- binary(n) Fixed-length binary data type with a string length between 1 and 8,000.
- varbinary(n) Variable-length binary data type with a string length up to 2 GB.
- image Deprecated data type. Replace with varbinary(max).

As a best practice, you should use the fixed-length (char, nchar, binary) data types across all subcategories when the values being stored are a consistent size. When the values are not consistent, you should use the variable-length data types (varchar, nvarchar, varbinary).

Date and time data types

Date and time data types are used widely in SQL Server databases. They offer the convenience of storing the date and time in various ways. There are six date and time data types.

- time(n)

  This data type stores the time of day without time-zone awareness based on a 24-hour clock. time accepts one argument, which is fractional seconds precision. You can only provide values between 0 and 7. As the number increases, so does the fractional precision. If you specify a data type of time(2), you can store a value similar to 11:51:04:24. Changing 2 to 3 increases the precision to three numbers, similar to 11:51:04:245.

- date

  This data type stores a date value between 01-01-01 and 12-31-9999.

- smalldatetime

  This data type stores a date and time value. The value of the date is between 1/1/1900 and 6/6/2079. The time precision is down to seconds. A value of 4/1/2016 11:15:04 can be stored using this data type.

- datetime

  This data type is similar to smalldatetime, but it offers a larger date range and a higher level of precision with regard to time. It offers the same date range as the date parameter, 01-01-01 to 12-31-9999, and it has a more precise value of time. A value of 4/1/2016 11:15:04:888 can be stored using this data type.

- datetime2(n)

  This data type is similar to datetime, but it offers extended flexibility of time. Unlike with datetime, you can control the fractional second precision with a value. You can only provide values between 0 and 7. If you specify a data type of datetime2(2), you can store a value similar to 4/1/2016 11:51:04:24. Changing 2 to 3 increases the precision to three numbers, similar to 4/1/2016 11:51:04:24.

- datetimeoffset

  This data type includes all the capabilities of datetime2, and it also has time-zone awareness. This makes it unique among the date and time data types. Using this data type, you can store the time-zone offset along with the date and time. A value of 4/1/2016 03:10:24 -06:00 can be stored using this data type.

OTHER DATA TYPES

Data Type	Description
cursor	A temporary copy of data that will be used for recursive or iterative processes. Of all the data types, this is the only one that cannot be included as part of a table.
rowversion(timestamp)	This data type automatically generates an 8-byte value similar to 0x0000000000000001. *rowversion* replaces the *timestamp* data type, which has been deprecated. This data type is typically used to detect changes in data.

hierarchyid	This is a positional data type. It represents a position in a hierarchy. *hierarchyid* is used to organize data such as a bill of materials and organizational charts.
sql_variant	This is the chameleon of data types. *sql_variant* can assume the identity of just about any data type in the list of SQL Server data types. Prior to performing any types of operations on it, you must convert it to the respective data type. For example, if you want perform addition, you must cast this data type to an int or some other numeric data type that supports that operation.
xml	You can store actual XML data using this data type.
geospatial	SQL Server supports two geospatial data types: GEOGRAPHY and GEOMETRY. GEOGRAPHY represents data in a round-earth coordinate system. GEOMETRY is a flat or planar data type in which you can store points, lines, and other geometric figures.
filestream	This data type allows you to store common unstructured data such as documents and images. SQL Server has been coupled with the NTFS file system, allowing the storage of *varbinary(max)* on the file system.

Understanding Column Properties

Common properties include:

1. Data type
2. Allow null
3. Identity
4. Sequence (new in 2016)
5. Primary key

Activity: create a table using SMSS

Activity: create a table using TSQL

```
USE sampledb1;
CREATE TABLE HumanResources.Address
(
    AddressID int NOT NULL IDENTITY(1,1),
    StreetAddress varchar(125) NOT NULL,
    StreetAddress2 varchar(75) NULL,
    City varchar(100) NOT NULL,
    State char(2) NOT NULL,
    EmployeeID int NOT NULL
) ON [Filegroup];
```

Altering database tables

Activity: modify table design using ssms (rclick table → design).

*some changes may need reconstruction of the table. By default, it won't be saved.

To fix this:

Tools → options → designer → prevent saving changes that require table re-creation

Activity: modify table design using TSQL

```
USE sampledb1;
ALTER TABLE HumanResources.Employee
    ADD Gender char(1) NOT NULL;
```

Understanding computed columns

These are columns that have values derived from other columns from the result of a computation.

To set a computed column:

Column properties· computed column specification· formula(ex. Column1 + Column2)

Activity: add a computed column

Activity: add a computed column using TSQL

```
USE sampledb1;
ALTER TABLE HumanResources.Employee
   ADD FullName         AS LastName+', '+FirstName;
```

Adding constraints to tables

- Primary key constraints(unique,readonly if identity)
- Default constraints
- Unique constraints
- Check constraints
- Foreign key constraints

Primary key constraints

Rclick column in table design view → set as primary key

Default constraints

*Columns with a default specific value (set default value in column properties)

Unique constraints

In table designer view· rclick column → indexes and keys → add → specify column and make unique

Check constraints

*Uses logical expressions to check for a value in that column In table designer → rclick column → check constraint → add → expression (ex. [gender]='male' or [gender]='female')

Or by using TSQL

```
USE sampledb1;
ALTER TABLE HumanResources.Employee
    ADD CONSTRAINT PK_HumanResourcesEmployee_EmployeeID
        PRIMARY KEY (EmployeeID);

ALTER TABLE HumanResources.[Address]
    ADD CONSTRAINT PK_HumanResourcesAddress_AddressID
        PRIMARY KEY (AddressID);ALTER TABLE HumanResources.Employee
    ADD CONSTRAINT DF_HumanResourcesEmployee_Active_True DEFAULT(1) FOR Active;

ALTER TABLE HumanResources.Employee
    ADD CONSTRAINT UQ_HumanResourcesEmployee_SocialSecurityNumber
    UNIQUE (SocialSecurityNumber);
```

Foreign key constraints

*enforces the referential integrity of data. This is a column that can only accept data that is also existing on another column.

Activity: add or set foreign key on a second table

1. create another table with a primary key first
2. create another table with a column similar to the first table's primary key
3. on the second table, we will set its primary key to become a foreign key constraint:
   *expand your table → rclick keys folder → add foreign key
   *set name (ex. FK_TABLENAME_PK_TO_TABLENAME_FK) and click the ellipsis for Table and columns specification (point to the primary key reference and the foreign key reference).
   *try entering data to your foreignkey

Activity: add or set foreign key on a second table using TSQL

```
USE sampledb1;
ALTER TABLE HumanResources.Address
    ADD CONSTRAINT FK_Employee_To_Address_On_EmployeeID
        FOREIGN KEY (EmployeeID)
            REFERENCES HumanResources.Employee(EmployeeID);
```

Creating database diagrams

Expand your database → database diagram (rclick new database diagram) → create objects when prompted

Note: After creating the diagram, you can see it in your database diagrams folder. Rclick on the diagram view you can show relationship labels or copy it to the clipboard

5. Building and maintaining indexes

An index is an on-disk data structure that is based on tables and views. Indexes make the retrieval of data faster and efficient, in most cases. However, overloading a table or view with indexes could adversely affect the performance of other operations such as inserts or updates.

Indexes can be classified in two types: clustered and non-clustered

Clustered index:

a. default for primary key
b. only one clustered index per table
c. saves the rows in your disk in order of the clustered index
d. useful for performance boost if the clustered index column have multiple similar entries, these entries will be placed one-after-another in a cluster
e. a disadvantage would be is that if you update a row it will then move all the rows to reorder them.

Activity: create a clustered index:

1. Expand your database → expand your table → rclick indexes folder (make sure table designers view are closed prior to this) → new index → clustered index
2. Set index name, click add → select column to be indexed → ok → set other properties such as sort order (ascending or descending)

Activity: create a clustered index using TSQL:

```
USE AdventureWorks2016;
CREATE CLUSTERED INDEX CIX_DatabaseLog_PostTime
ON dbo.DatabaseLog (
    PostTime DESC
)
WITH(DROP_EXISTING = ON);
```

Non-clustered index:

a. Multiple non-clustered indexes can be set on a table
b. Saves the rows in order that they were created but creates a pointer to each record.
c. Slower searches but faster insert, delete, and update performances

Activity: create a non-clustered index:

1. Expand your database→expand your table→rclick indexes folder(make sure table designers view are closed prior to this)→new index→non-clustered index
2. Set index name, click add→select column to be indexed→ok→set other properties such as sort order(ascending or descending)

Activity: create a non-clustere index using TSQL:

```
USE AdventureWorks2016;
CREATE NONCLUSTERED INDEX IX_SalesOrderHeader_DueDate
ON Sales.SalesOrderHeader
(
    DueDate
) ;
```

More index properties:

Option	Description
IGNORE_DUP_KEY	During a multirow insert that contains duplicate key values, setting this option to ON will ensure that only one unique row is inserted and the integrity of the index is not violated.
STATISTICS_NORECOMPUTE	Statistics are vital to SQL Server with regard to determining how a query will be executed. As such, statistics need to be updated regularly. You can stop statistics from automatically recomputing by setting this option to ON.
ONLINE	Index maintenance is pivotal. You must rebuild and reorganize your indexes on a regular basis. However, when these operations are performing, users cannot access the data. Setting this option to ON allows data access. Please note that you must own the Enterprise version of SQL Server to use this option; Chapter 21 discusses this option further.
ALLOW_ROW_LOCKS	When accessing data, if this option is set to ON, SQL Server will lock the accessed rows.
ALLOW_PAGE_LOCKS	When accessing data, if this option is set to ON, SQL Server will lock the accessed pages.
MAX_DOP	Using this option, you can control how many processors are used during index creation.
DATA_COMPRESSION	This option is available only in the Enterprise version of SQL Server. There are two types of compression: ROW and PAGE. Both are discussed in detail in Chapter 7, "SQL Server Compression."

Disabling and dropping indexes

Activity: disable index

Rclick the selected index in your indexes folder→disable

Activity: disable index using TSQL

```
USE AdventureWorks2016;
ALTER INDEX IX_SalesOrderHeader_OrderDate
    ON Sales.SalesOrderHeader DISABLE;
```

Activity: delete index

Rclick the selected index in your indexes folder→delete

Activity: delete index using TSQL

```
USE AdventureWorks2016;
DROP INDEX CIX_DatabaseLog_PostTime
    ON dbo.DatabaseLog;
```

Advanced Database Design Topics

1. Table compression

Note: Server 2016 enterprise edition only!

Data Types Affected by Compression

smallint	int	bigint	decimal	numeric
real	float	money	smallmoney	bit
datetime	datetime2	datetimeoffset	char	nchar
binary	rowversion			

Understanding row compression

*identifies data type of each column, converts it to variable length then trim off excess variable sizes

Activity: perform row compression

Rclick the selected table→storage→manage compression→select compression type(row)→use same compression type for all partitions→calculate→next→run immediately

Activity: perform row compression using TSQL

```
USE AdventureWorks2016
ALTER TABLE [Sales].[SalesOrderHeader]
    REBUILD WITH(DATA_COMPRESSION = ROW);
```

Understanding page compression

*page compression performs

Row compression

*same process as above

Prefix compression

*gets common data prefixes, adds it to a page header, then replaces the values with those prefixes with pointers to the page header

Dictionary compression

*gets common dictionary words, adds it to the page header, then replaces the values with those words with pointers to the page header

Activity: perform page compression

Rclick the selected table→storage→manage compression→select compression type(page)→use same compression type for all partitions→calculate→next→run immediately

Activity: perform page compression using TSQL

```
USE AdventureWorks2016
ALTER TABLE Sales.SalesOrderDetail
    REBUILD WITH(DATA_COMPRESSION = PAGE);
```

2. Table partitioning

Note: SQL SERVER ENTERPRISE FEATURE ONLY!

Understanding table partitioning

Table partitioning is a way to divide a large table into smaller, more manageable parts without having to create separate tables for each part. Data in a partitioned table is physically stored in groups of rows called partitions and each partition can be accessed and maintained separately. Partitioning is not visible to end users, a partitioned table behaves like one logical table when queried.

An alternative to partitioned tables (for those who don't have Enterprise Edition) is to create separate tables for each group of rows, union the tables in a view and then query the view instead of the tables. This is called a partitioned view.

Data in a partitioned table is partitioned based on a single column, the partition column, often called the partition key. Only one column can be used as the partition column, but it is possible to use a computed column.

Activity: create a table, preferably with date timestamp and add different filegroups. Prior to the next activity.

Activity: partition a table

Rclick the selected table
→ storage
→ create partition
→ select partition column

→ create and name a partition function
→ create a partition schema
→ map partitions to filegroups
→ set boundaries
→ create a script (optional but recommended) or run immediately

Activity: view all partitions
Select * from sys.partitions

Activity: view all partitions in a given table using table's object id.
Select * from sys.partitions where object_id=object_id('dbo.employees')

Activity: view other statistics pertaining the partitions
Select * from sys.dm_db_partition_stats where object_id = object_id('dbo.employees')

Activity: view partition function ID
Select * from sys.partition_function

Activity: view row count for each boundary on a given partition function
Select * from sys.partition_range_values

3. Database snapshots

*** A database snapshot is a static, read-only copy of an existing Microsoft SQL Server database taken at that point of time.**

Prerequisites

- Database snapshots are supported only in the Enterprise version of SQL Server 2016. The source and the snapshot database must reside on the same SQL Server instance.
- The source database cannot be dropped, detached, or restored.
- Source database files cannot be dropped.
- Performance could be negatively affected due to increased I/O on the source.

Limitations

- The snapshots must reside on the same server as the source database.
- Snapshots cannot be backed up, restored, or detached.
- Changes in the source database will cause the snapshot database to grow. Therefore, you should ensure that you have disk space available equal to the size of your source database.
- If a snapshot runs out of space, it must be deleted and re-created.

Activity: create a database snapshot using SMSS

Activity: create a database snapshot using TSQL

```
USE master;
```

```
CREATE DATABASE SBSChp4TSQL_snapshot_42016200
ON
(
    NAME = SBSChp4TSQL1,
    FILENAME = 'C:\SQLDATA\SBSChp4TSQL1_snapshot_data.ss' ),
(
    NAME = SBSChp4TSQL2,
    FILENAME = 'C:\SQLDATA\SBSChp4TSQL2_snapshot_data.ss'
)
AS SNAPSHOT OF SBSChp4TSQL;
```

Activity: deleting a database snapshot using TSQL

```
USE master;
DROP DATABASE sampledb1_snapshot_42016400;
```

Activity: reverting to a database snapshot

```
USE master;
RESTORE DATABASE sampledb1 FROM DATABASE_SNAPSHOT = 'sampledb1_snapshot_42016400';
```

***more errors and limitations you might encounter while reverting to a snapshot**

- The source database must have only one snapshot.
- If any of the files are read-only or offline, you cannot revert to a snapshot.
- Any changes that occurred after the reverted snapshot was taken will be lost.

4. Using the SELECT statement

Writing a SELECT statement

While the SELECT statement offers a plethora of arguments that can make it very complex, in its simplest form it consists of two keywords: a list of columns and a table name.

1. Open Microsoft SQL Server Management Studio (SSMS) and connect to a server.
2. In Object Explorer, expand the Databases folder.
3. Expand the AdventureWorks2016 database.
4. Expand the Tables folder.
5. Expand the HumanResources.Department table.
6. Open the query editor in SSMS.
7. In the query editor, enter and execute the following T-SQL code:

```
USE AdventureWorks2016;
SELECT
FROM
```

8. Click and drag the Columns folder underneath the table in Object Explorer to the right of the keyword SELECT. Ensure that there is a space after the SELECT keyword.

9. Click and drag the HumanResources.Department table from Object Explorer to the right of the keyword FROM. Ensure that there is a space after the FROM keyword.

```
USE AdventureWorks2016;
SELECT DepartmentID, Name, GroupName, ModifiedDate FROM [HumanResources].[Department]
```

10. Execute the query and review the results.

Sorting results

Now that you can retrieve data, you may want to do certain things with the results of the query. For example, often you'll want to sort data. To sort data in SQL Server 2016, you use the ORDER BY clause.

This clause sorts the data in the order specified, either ascending (ASC) or descending (DESC).

```
USE AdventureWorks2016;
SELECT *
FROM [HumanResources].[Department]
ORDER BY DepartmentID DESC
```

Note If a clustered index exists on the table, and ORDER BY is not specified, the results are usually returned in the order specified when the clustered index was created. However, when logic is relying on data ordered in a specific way, you should always specify ORDER BY.

Filtering data with the WHERE clause

There are several different implementations of the query clause, among them the following, which are discussed in the sections that follow:

- Comparison operators
- The BETWEEN operator
- A WHERE clause with multiple conditions
- A search for a list of values
- A wildcard search

Using comparison operators

SQL Server offers several comparison operators, such as = (equals), < (less than), > (greater than), and >= (greater than or equal to), among others. Coupling these operators with the WHERE clause can assist you in limiting data in several ways.

```
USE AdventureWorks2016;
SELECT *
FROM [HumanResources].[Department]
WHERE DepartmentID = 4
```

```
USE AdventureWorks2016;
SELECT *
FROM [HumanResources].[Department]
WHERE DepartmentID > 4
```

Using the BETWEEN operator

In some cases, you may want to search your data for a sequential range of data. For example, you may want to return all the sales from May 1, 2007, through December 12, 2007. You could use a couple of the comparison operations, but SQL Server offers you a more elegant solution: the BETWEEN operator.

```
USE AdventureWorks2016;
SELECT
    AccountNumber,
    SalesOrderID,
    OrderDate
FROM Sales.SalesOrderHeader
WHERE
    OrderDate BETWEEN '5/1/2007' AND '12/31/2007'
```

Note When using the BETWEEN operator, note that it is an inclusive range for numbers, which means that the two values specified in the clause will be included in the filter.

In the previous example, the result set would include all order data that occurred between 5/1/2001 12 AM and 12/31/2007 12 AM. As such, it would not return anything that happened during the day of 12/31/2007 after midnight.

If you were required to include the values for that data, you could write the query as follows:

```
SELECT
    AccountNumber,
    SalesOrderID,
    OrderDate
FROM Sales.SalesOrderHeader
WHERE
    OrderDate >= '5/1/2007 00:00:00' AND OrderDate<= '12/31/2007'
```

Using the WHERE clause with multiple conditions

Sometimes you may need to specify multiple filters in one statement. For example, perhaps you want to find sales in a certain date range but also only for a specific product. In that case, you can use the AND or OR operator to combine both filters.

```
USE AdventureWorks2016;
SELECT
    SalesOrderDetailID,
    OrderQty,
    ProductID,
    ModifiedDate
FROM Sales.SalesOrderDetail s
WHERE       ModifiedDate BETWEEN '5/1/2007' AND '12/31/2007' AND
    ProductID = 809
```

Searching for a list of values

Another typical scenario involves retrieving a result set based on a list of values. For example, you may need to return all sales for a particular list of products. Using the IN operator, SQL Server determines whether items in a specified list match the specified value.

```
USE AdventureWorks2016;
SELECT
    SalesOrderDetailID,
    OrderQty,
    ProductID,
    ModifiedDate
FROM Sales.SalesOrderDetail s
WHERE
    ProductID IN (776, 778, 747, 809)
```

Using a wildcard search

The final variation of the WHERE clause covered here is the wildcard search. For example, suppose you want to return all the departments at your organization that start with the letters PR. To do this, you use a LIKE comparison. When you use LIKE, SQL Server can determine if a specified character or string of characters matches a value in your database.

```
USE AdventureWorks2016;
SELECT
    * FROM HumanResources.Department
WHERE
    Name LIKE 'Pr%'
```

The Pr% used in the preceding query tells SQL Server to return all departments whose name starts with Pr and any following characters.

The LIKE syntax does not use a typical regular expression wildcard set. As demonstrated in the previous script, the % represents any string of zero or more characters. In addition, other wildcard characters are as follows:

- _ (a single character)
- [abc] (a single character in a set)
- [^abc] (a single character not in the set)

Creating aliases

You can create aliases, which can be a shorter or more understandable name, for table and column names, making it easier to work with aggregations, expressions, and queries that involve multiple tables. In addition, your database may contain very cryptic column names, and you may want to provide names that are more meaningful to applications and end users. Using aliases allows you to rename or shorten the names of tables and columns.

```
USE AdventureWorks2016;
SELECT
    DepartmentID,
    Name AS DepartmentName,
    GroupName AS DepartmentGroupName
FROM HumanResources.Department AS d
```

In the preceding query, the Name column is renamed DepartmentName, the GroupName column is renamed DepartmentGroupName, and the table has been aliased as simply d. Now you can reference the table as d instead of the entire table name throughout the query. The use of table aliases is explained in the next section.

Note The AS keyword used in the previous query is optional when aliasing items within a SQL Server query. This means the original table name cannot be used any longer, so HumanResources.Department.DepartmentName will no longer resolve in the query.

Using the JOIN operator to return data from multiple tables

You have focused primarily on retrieving data from a single table thus far. In practice, it is highly unlikely that your queries will reference just one table—most of the time, you will be required to return data from multiple tables. To do this, you use the JOIN operator.

We will focus on the three most commonly used:

- INNER
- LEFT OUTER
- RIGHT OUTER

Using INNER JOIN

Of the three most commonly used JOIN operators, INNER JOIN is the one that you will likely use on a regular basis. INNER JOIN is an equality match between two or more tables. For example, assume you have a table that

contains products and another that contains sales, and you want to find only the products that have been sold. Basically, you are looking for the intersection of the two tables on some value

```
USE AdventureWorks2016;
SELECT
    p.FirstName,
    p.LastName,      ea.EmailAddress
FROM Person.Person AS p
INNER JOIN Person.EmailAddress AS ea
    ON p.BusinessEntityID = ea.BusinessEntityID
```

The INNER JOIN keywords have been included, which allows you to specify a second table in the query. The INNER JOIN or any JOIN must be coupled with the ON keyword. In the ON clause, you specify which column or columns will be used to connect (JOIN) the two tables. The key to successfully joining any two tables is to identify their intersecting data, which is commonly aligned across primary key and foreign key. If you want to perform a LEFT OUTER or RIGHT OUTER JOIN in the preceding query, you replace INNER with either LEFT OUTER or RIGHT OUTER.

Using OUTER JOINs

```
SELECT
    p.ProductID,      sd.ProductID,
    p.Name AS ProductName,      sd.OrderQty,      sd.UnitPrice
FROM Production.Product AS p
LEFT OUTER JOIN Sales.SalesOrderDetail sd      ON p.ProductID = sd.ProductID
```

Using TOP

The TOP keyword limits the number of rows that are returned in a result to either a specific number of rows or a specific percentage of rows. TOP should always be used with the ORDER BY clause. In most cases, you will be looking for the highest or lowest set of values for a given column and sorting the data will provide you with that information. For example, if you want to return the top five sales in your Sales table, you add TOP (5) immediately following the keyword SELECT. In addition, you include an ORDER BY clause specifying the column that contained the actual sales value for each row as the ordering column.

```
USE AdventureWorks2016;
SELECT TOP(5)
    SalesOrderID,
    OrderDate,
    SalesOrderNumber,
    TotalDue
FROM Sales.SalesOrderHeader
ORDER BY
    TotalDue DESC
```

Using DISTINCT and NULL

DISTINCT returns a unique or distinct list of values of each specified column in a SELECT statement. If there are any duplicate values in the list, all but one duplicate value will be removed

```
USE AdventureWorks2016;
SELECT DISTINCT
    p.Name AS ProductName FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail sd
    ON p.ProductID = sd.ProductID
```

By placing DISTINCT immediately following the SELECT keyword, you remove any duplicates from the list.

Moreover, if you want to limit the result to only products that have not shipped, you add the WHERE clause. In this example, the WHERE clause needs to identify all the SalesOrderProduct rows that contain a NULL CarrierTrackingNumber.

The NULL value is a special value. It is actually not a value at all—it's the absence of a value. As a result, there are special comparison values that can be used when referencing it in a WHERE clause. If you are searching for NULLs, you would use the following:

```
WHERE <column name> IS NULL
```

If you are searching for columns that are NOT NULL, you would use this:

```
WHERE <column name> IS NOT NULL
```

Using UNION to combine result sets

Often you will have two SELECT statements that may need to be combined into one result for consumption by an application or end user. Using the UNION keyword, you can accomplish just that. UNION has two variations:

- Just UNION, which removes any duplicate rows in your result set.
- UNION ALL, which includes duplicates. If duplicates are possible, you should use UNION ALL; it is much faster because it does not have to include DISTINCT.

Example of UNION:

```
SELECT column1, column2 FROM TABLE1
UNION
SELECT column1, column2 FROM TABLE2
```

Using Transact-SQL (TSQL) Scripting

1. Data retrieval techniques

Paging data

Using OFFSET and FETCH, you can write a single query that returns data one page at a time to a client application or end user:

- OFFSET Denotes how many rows to skip before the query starts returning rows
- FETCH Specifies how many rows to return after OFFSET has been processed

OFFSET is synonymous with the page number and FETCH with the number of rows that will be displayed per page.

Both OFFSET and FETCH have a few additional arguments that must be included in the syntax. The following is sample syntax for writing a paging query:

```
SELECT <column list>
FROM <table name>
ORDER BY <column name>
OFFSET <rows to start on> ROWS
FETCH NEXT <number of rows to return> ROW ONLY
```

Activity: implement paging

Example:

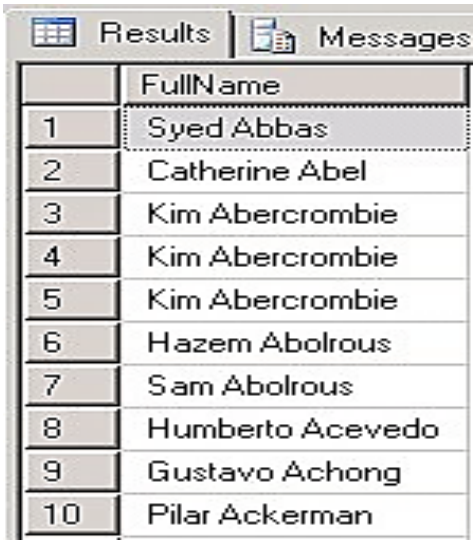
```
USE AdventureWorks2016;
SELECT
    ProductID,
    ProductNumber,
    Name AS ProductName,
    ListPrice
FROM Production.Product
ORDER BY ProductID
OFFSET 10 ROWS
FETCH NEXT 10 ROWS ONLY
```

Writing expressions

Sample TSQL:

```
USE AdventureWorks2016;
SELECT      FirstName+' '+LastName AS FullName
FROM Person.Person
```

Sample output of TSQL:



Results | Messages

	FullName
1	Syed Abbas
2	Catherine Abel
3	Kim Abercrombie
4	Kim Abercrombie
5	Kim Abercrombie
6	Hazem Abolrous
7	Sam Abolrous
8	Humberto Acevedo
9	Gustavo Achong
10	Pilar Ackerman

Another example of an expression used to return computed data:

```
USE AdventureWorks2016;
SELECT      (SubTotal+TaxAmt)*1.05
AS TotalDue
FROM SaleS.SalesOrderHeader
```

Using variables

Sample variable declaration and setting of value

```
DECLARE @varname int
SET @varname=<value>
```

Sample usage of variable in a query

```
USE AdventureWorks2016;
DECLARE @ProductID int = 1;

SELECT
    ProductID,
    ProductNumber,
    Name AS ProductName
FROM Production.Product
WHERE ProductID = @ProductID
```

2. Modifying data

Activity: inserting single row of data

```
USE AdventureWorks2016;
INSERT INTO HumanResources.Department(Name, GroupName, ModifiedDate)
VALUES('Payroll', 'Executive General and Administration', '6/12/2016');
```

Activity: inserting multiple rows of data

```
USE AdventureWorks2016;
INSERT INTO HumanResources.Department
VALUES
    ('International Sales', 'Sales and Marketing', '5/26/2016'),
    ('Media Control', 'Quality Assurance', '5/26/2016')
```

Activity: insert multiple rows from an existing table into another table using a SELECT statement

```
USE AdventureWorks2016;
INSERT INTO HumanResources.Department(Name, GroupName, ModifiedDate)
SELECT
    Name+' USA', GroupName, ModifiedDate
FROM HumanResources.Department
WHERE DepartmentID IN (20, 19)
```

Activity: update a single row

```
USE AdventureWorks2016;
UPDATE HumanResources.Department
SET Name = Name +' Europe'
WHERE DepartmentID = 19
```

Activity: update rows while referencing other tables

```
UPDATE Production.Product
SET ListPrice = p.ListPrice * 1.05
FROM Production.Product p
INNER JOIN Production.ProductSubcategory ps
    ON p.ProductSubcategoryID = ps.ProductSubcategoryID
WHERE ps.Name = 'Socks'
```

Activity: deleting a single row

```
USE AdventureWorks2016;
DELETE FROM HumanResources.Department
WHERE DepartmentID = 22
```

Activity: deleting multiple records while referencing other tables

```
USE AdventureWorks2016;
DELETE FROM HumanResources.Department
FROM HumanResources.Department d
LEFT OUTER JOIN HumanResources.EmployeeDepartmentHistory ed
    ON d.DepartmentID = ed.DepartmentID
WHERE ed.DepartmentID IS NULL
```

Activity: removing all records from a table

```
USE AdventureWorks2016;
TRUNCATE TABLE dbo.Department
```

Activity: returning output data on affected rows

```
*returns a sample table that shows the affected modified rows

USE AdventureWorks2016;
INSERT INTO HumanResources.Department
    OUTPUT inserted.DepartmentID, inserted.Name, inserted.GroupName, inserted.
ModifiedDate
VALUES('International Marketing', 'Sales and Marketing', '5/26/2016');
```

Activity: viewing the hidden deleted table when returning output from update statement

```
USE AdventureWorks2016;
UPDATE HumanResources.Department
SET Name = Name +' Europe'
OUTPUT
    deleted.Name AS OldName,      inserted.Name AS UpdateValue
WHERE DepartmentID = 25
```

Activity: insert output data in a table

```
USE AdventureWorks2016;
GO
CREATE TABLE dbo.Department_Audit
(
    DepartmentID int NOT NULL,
    Name nvarchar(50) NOT NULL,
    GroupName nvarchar(50) NOT NULL,
    DeletedDate datetime NOT NULL
        CONSTRAINT DF_Department_Audit_DeletedDate_Today DEFAULT(GETDATE()) )
```

Activity: deletes a single row from a table and also inserts the contents of the deleted row into another table

```
USE AdventureWorks2016;
DELETE FROM dbo.Department
OUTPUT deleted.Departmentid, deleted.Name, deleted.GroupName
INTO dbo.Department_Audit(DepartmentID, Name, GroupName)
```

```
WHERE DepartmentID = 16
```

3. Built-in scalar functions

DATE AND TIME FUNCTIONS

Activity: return date and time

```
SELECT GETDATE() AS GETDATE, SYSDATETIME() AS SYSDATETIME
```

Activity: return datetime parts

```
SELECT
    DAY(GETDATE()) AS DAY,
    MONTH(GETDATE()) AS MONTH,
    YEAR(GETDATE()) AS YEAR,
    DATENAME(WEEKDAY, GETDATE()) AS DATENAMEWeekDay,
    DATEPART(M, GETDATE()) AS DATEPART,
    DATEPART(WEEKDAY, GETDATE()) AS DatePartWeekDay,
    DATENAME(MONTH, GETDATE()) AS DateNameMonth
```

Activity: using datepart

*datepart arguments:

datepart	Abbreviations
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
week	wk, ww
weekday	dw
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs

nanosecond	ns
TZoffset	tz
ISO_WEEK	Isowk, isoww

DIFERENCING,MODIFYING AND VALIDATING DATE VALUES

Activity: get date difference, add / subtract to a date, get last day of the month and check if given date is valid or not

```
SELECT
DATEDIFF(dd, GETDATE(), '5/26/2013') AS DaysUntilMyBirthday,
DATEADD(y, 1, GETDATE()) AS DateAdd,
EOMONTH(GETDATE()) AS EOMonth, --New to SQL Server 2016
ISDATE(GETDATE()) AS IsValidDate,
ISDATE('13/1/2122') AS InvalidDate
```

Using conversion functions

Activity: using the CAST()

Syntax

```
CAST([column] as data type)
```

Activity: using the CONVERT()

Syntax

```
CONVERT(data type, [column])
```

***using the functions above fails the statement if the column data is unconvertible. The ff. functions generate null values instead**

Syntax

```
TRY_CAST([column] as data type)
TRY_CONVERT(data type, [column])
```

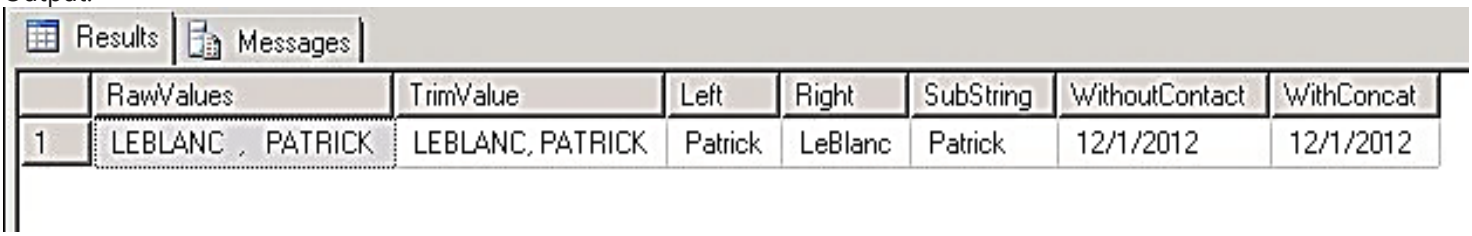
Using string functions

Activity: demonstrate common string functions

```
SELECT
    'LEBLANC '+', '+' PATRICK' RawValues,
    RTRIM('LEBLANC ')+', '+LTRIM(' PATRICK') TrimValue,
    LEFT('PatrickDTomorr', 7) [Left],
    RIGHT('DTomorrLeBlanc', 7) [Right],
    SUBSTRING('DTomorrPatrick',8,LEN('DTomorrPatrick')) [SubString],
    '12/'+CAST(1 AS VARCHAR)+'/2016' WithoutConcat,
```

```
    CONCAT('12/',1,'/2016') WithConcat
```

Output:



	RawValues	TrimValue	Left	Right	SubString	WithoutContact	WithConcat
1	LEBLANC , PATRICK	LEBLANC, PATRICK	Patrick	LeBlanc	Patrick	12/1/2012	12/1/2012

Creating Other Database Objects

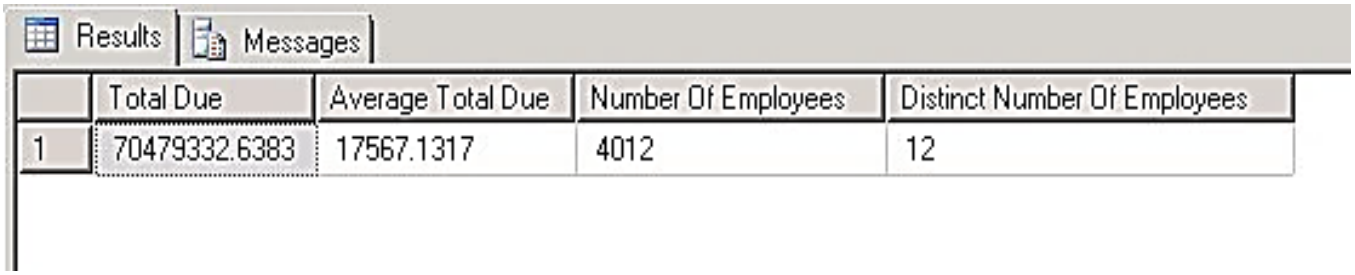
1. Advanced TSQL Scripting

Activity: Using aggregate functions

(*adding DISTINCT as a function parameter gets only distinct values for the expression)

```
USE AdventureWorks2016;
SELECT
    SUM(poh.TotalDue) AS [Total Due],
    AVG(poh.TotalDue) AS [Average Total Due],
    COUNT(poh.EmployeeID) [Number Of Employees],
    COUNT(DISTINCT poh.EmployeeID) [Distinct Number Of Employees]
FROM Purchasing.PurchaseOrderHeader poh
```

Sample output



	Total Due	Average Total Due	Number Of Employees	Distinct Number Of Employees
1	70479332.6383	17567.1317	4012	12

Activity: performing aggregate functions with grouping

```
USE AdventureWorks2016;
SELECT
    sm.Name AS ShippingMethod,
    SUM(poh.TotalDue) AS [Total Due],
    AVG(poh.TotalDue) AS [Average Total Due],
    COUNT(poh.EmployeeID) AS [Number Of Employees],
    COUNT(DISTINCT poh.EmployeeID) AS [Distinct Number Of Employees]
FROM Purchasing.PurchaseOrderHeader poh
```

```
GROUP BY sm.Name
```

Sample output

Results | Messages

	ShippingMethod	Total Due	Average Total Due	Number Of Employees	Distinct Number Of Employees
1	XRQ - TRUCK GROUND	3330909.2897	5655.194	589	12
2	ZY - EXPRESS	14874601.7677	22709.3156	655	12
3	OVERSEAS - DELUXE	8002938.997	50018.3687	160	12
4	OVERNIGHT J-FAST	11965191.1871	11027.8259	1085	12
5	CARGO TRANSPORT 5	32305691.3968	21211.8787	1523	12

Activity: using the HAVING clause

```
USE AdventureWorks2016;
SELECT
    sm.Name AS ShippingMethod,
    YEAR(poh.OrderDate) OrderYear,
    SUM(poh.TotalDue) AS [Total Due],
    AVG(poh.TotalDue) AS [Average Total Due],
    COUNT(poh.EmployeeID) AS [Number Of Employees],
    COUNT(DISTINCT poh.EmployeeID) AS [Distinct Number Of Employees]
FROM Purchasing.PurchaseOrderHeader poh
GROUP BY sm.Name,YEAR(poh.OrderDate)
HAVING SUM(poh.TotalDue) > 5000000
```

Sample output:

Results | Messages

	ShippingMethod	OrderYear	Total Due	Average Total Due	Number Of Employees	Distinct Number Of Employees
1	ZY - EXPRESS	2008	10078436.6734	22853.5978	441	12
2	OVERSEAS - DELUXE	2008	6261920.1304	55415.2223	113	12
3	OVERNIGHT J-FAST	2008	7896355.8603	10951.9498	721	12
4	CARGO TRANSPORT 5	2007	8073385.8419	20861.4621	387	12
5	CARGO TRANSPORT 5	2008	22004527.508	21426.0248	1027	12

Handling TSQL errors

Syntax:

```
BEGIN TRY
{ sql_statement |statement_block}
END TRY

BEGIN CATCH
    [{ sql_statement |statement_block}]
END CATCH
```

Activity: demonstrate error handling

```
BEGIN TRY
    SELECT 1/0;
END TRY
BEGIN CATCH
    THROW;
 END CATCH
```

Controlling flow keywords

- BEGIN…END

```
BEGIN
{
  sql_Statement | statement_block
}
END
```

- IF…ELSE

```
IF Boolean_expression { sql_statement | statement_block }
 [ ELSE { sql_statement | statement_block } ]
```

- RETURN
  *immediately completes a query

- WHILE

```
WHILE Boolean_expression
  { sql_statement | statement_block | BREAK | CONTINUE }
```

Example:

```
DECLARE @count int = 0
WHILE (@count < 10)
BEGIN
    SET @count = @count + 1;
        IF(@count < 5)
        BEGIN
            SELECT @count AS Counter
            CONTINUE;
        END
        ELSE
            BREAK;
END
```

2. Views

*a table derived from another table

Activity: Create a view
Expand your database→rclick views folder→new view→select tables to use→select columns to use in the view→save

*close the view designer then execute query to display the view

```
USE AdventureWorks2016;
SELECT * FROM dbo.vwEmployeeInformation
```

3. User-defined functions

Activity: create a function

Expand your database→programmability→rclick functions folder→new function

```
CREATE FUNCTION dbo.GetEmployeeAge
(
    @BirthDate datetime
)
RETURNS int
AS
BEGIN
    -- Declare the return variable here
    DECLARE @Age int
    -- Add the T-SQL statements to compute the return value here
    SELECT @Age = DATEDIFF(DAY, @BirthDate, GETDATE())
    -- Return the result of the function
    RETURN @Age
END
GO
```

Activity: using the function

```
USE AdventureWorks2016;
SELECT TOP(10)
     p.FirstName,
          p.LastName,
          e.BirthDate,
          dbo.GetEmployeeAge(BirthDate) EmployeeAge
FROM HumanResources.Employee e
    ON e.BusinessEntityID = p.BusinessEntityID
```

Activity: modify the function

Expand your database→programmability→functions→scalar-valued functions→rclick your custom function (modify)

Activity: delete a function using TSQL

```
DROP FUNCTION <function name>
```

4. Stored procedures

Activity: create a stored procedure

Expand your database→programmability→rclick stored procedures→new stored procedure

Sample:

```
CREATE PROCEDURE dbo.PurchaseOrderInformation
AS
BEGIN
    SELECT
        poh.PurchaseOrderID, pod.PurchaseOrderDetailID,
        poh.OrderDate, poh.TotalDue, pod.ReceivedQty, p.Name ProductName
    FROM Purchasing.PurchaseOrderHeader poh

END
GO
```

Activity: execute a stored procedure

```
EXECUTE | EXEC procedure_name
```

Activity: create a stored procedure with parameters

Sample:

```
Create procedure dbo.addrecord
@my_id int,
@my_name
As
Begin
Insert into tablename values (@my_id,@my_name)
End
Go
```

Activity: execute a stored procedure with parameters

Sample:

```
EXECUTE | EXEC procedure_name @param1='',@param2=''
```

Activity: deleting stored procedure

```
DROP PROCEDURE <procedure name>
```

5. Data manipulation triggers

*executes another set of action upon manipulating data (ex. inserting new record)

Activity: create a trigger

Expand your table-->rclick triggers folder-->new trigger

```
USE AdventureWorks2016;
GO
CREATE TRIGGER HumanResources.iCheckModifedDate
ON HumanResources.Department
FOR INSERT
AS
```

```
BEGIN
    DECLARE @modifieddate datetime, @DepartmentID int
    SELECT @modifieddate = modifieddate, @DepartmentID = departmentid FROM inserted;
    IF(DATEDIFF(Day, @modifiedDate, getdate()) > 0)
        BEGIN
            UPDATE HumanResources.Department
            SET ModifiedDate = GETDATE()
            WHERE DepartmentID = @DepartmentID
        END
END
```

Try inserting a new record (intentionally adding a wrong date)

```
USE AdventureWorks2016;
INSERT INTO HumanResources.Department
VALUES('Executive Marketing', 'Executive General and Administration', '2/12/2011');

SELECT *
FROM HumanResources.Department
```

Output:

15	15	Shipping and Receiving	Inventory Management	2002-06-01 00:00:00.000
16	16	Executive	Executive General and Administration	2002-06-01 00:00:00.000
17	17	Payroll	Executive General and Administration	2012-08-11 12:54:13.533
18	18	International Sales	Sales and Marketing	2012-08-11 12:54:13.533
19	19	Media Control Europe	Quality Assurance	2012-09-11 14:00:12.660

Activity: alter the trigger

Activity: delete the trigger

```
DROP TRIGGER <trigger name>
```

Activity: disable / re-enable trigger

```
USE AdventureWorks2016;
--Disable a Trigger with T-SQL
DISABLE TRIGGER HumanResources.iCheckModifedDate
ON HumanResources.Department;
--Enable a Trigger with T-SQL
ENABLE TRIGGER HumanResources.iCheckModifedDate ON HumanResources.Department;
```

6. Replication

There are typically three types of servers in a replication topology:

- Publisher Database server that contains the source data.

- Subscriber Database server where the data and database objects are copied.
- Distributor Database server that stores changes. This information is stored in a database called the distribution database.

These are two commonly used types of replication:

- Snapshot replication is typically used with the following:
  - Small amounts of data
  - High latency or intermittent network connections
  - Data that changes infrequently
  - Copies of data that can be an hour, day, week, or month old

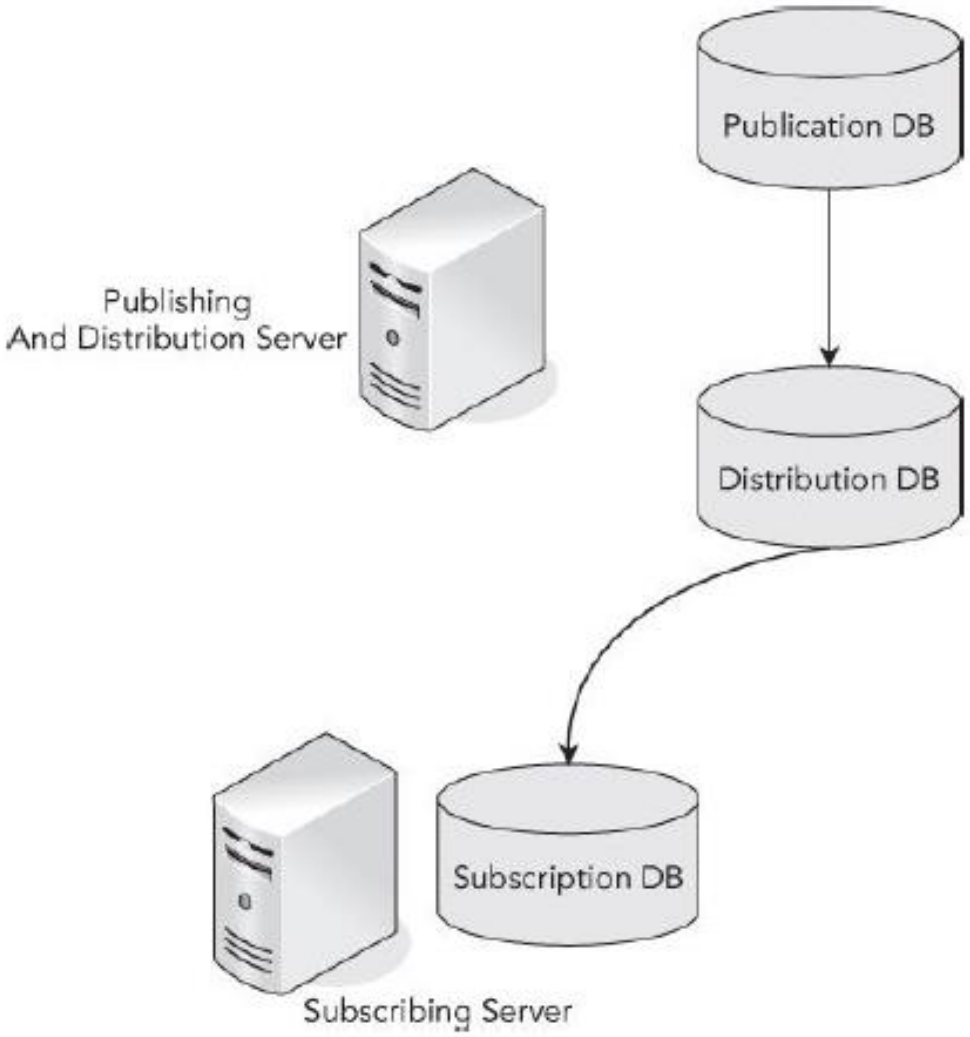
- Transactional replication is typically used in situations that require the following:
  - Near real-time data on one or more subscribers
  - Data that has to be incrementally loaded
  - Data that is highly transactional or that changes frequently

REPLICATION MODELS

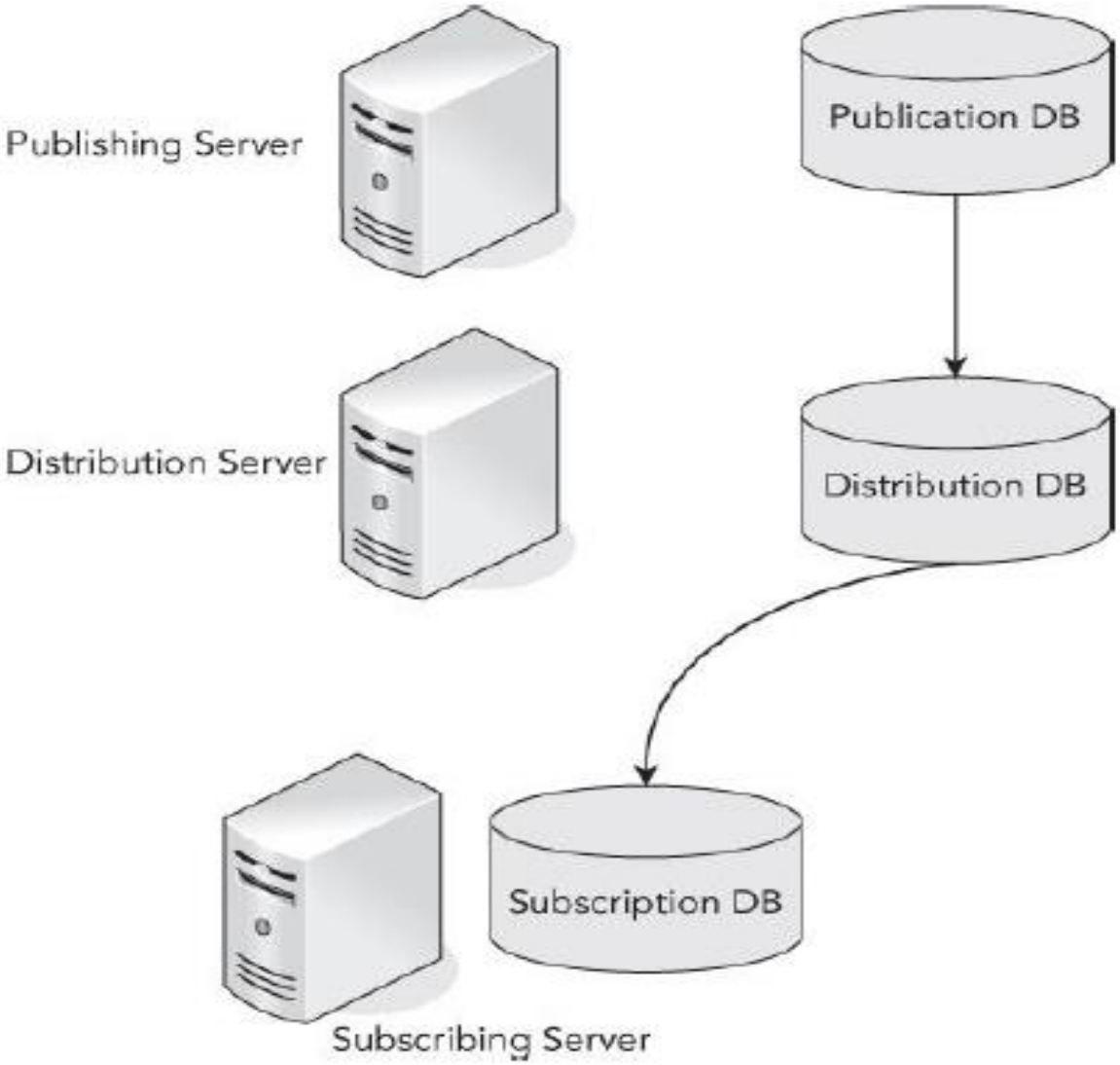
You can set up replication in quite a few different ways. This section covers some of the most common replication topologies.

Single Publisher, One or More Subscribers

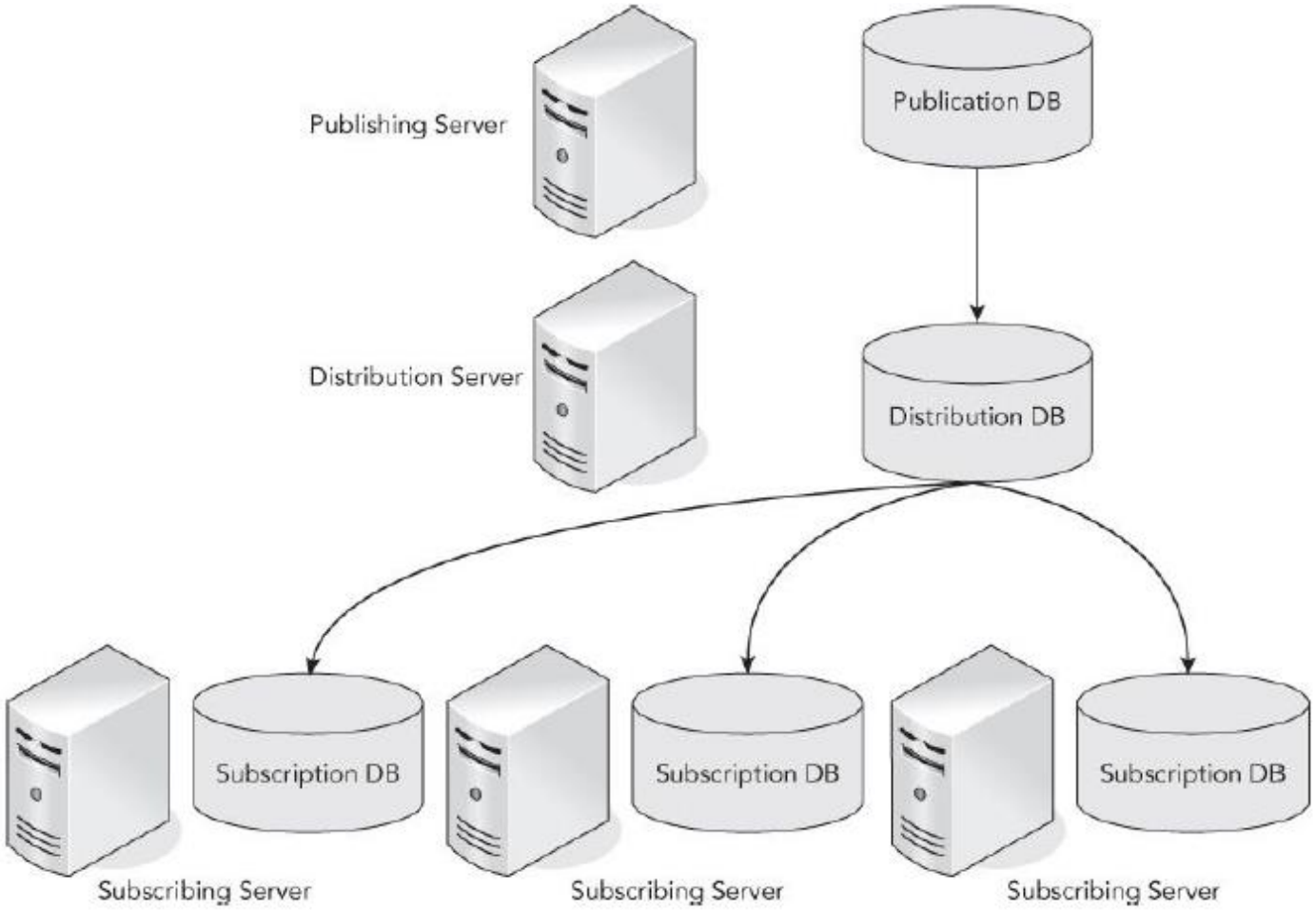
A Single Publisher model is perhaps the simplest topology to use with a single publishing database that has one or more subscription databases. You might use this topology where you need to keep a hot standby system, or distribute data from a central office to multiple field offices.



The image below shows a more advanced option with a separate server for the distribution database. Use this option when you need more performance at the distributor than you can get from a single server acting as the publisher and distributor.

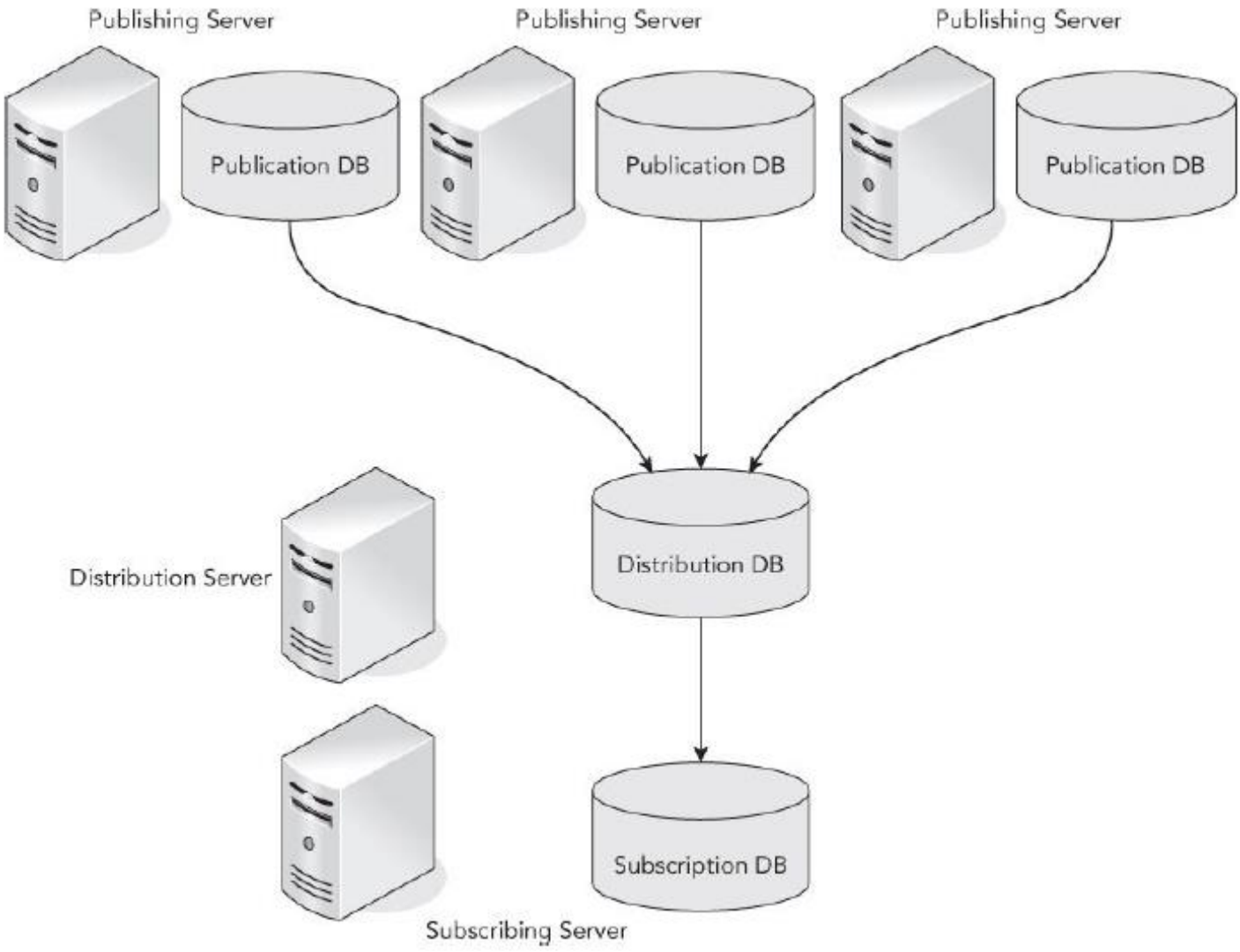


The image below shows the next variant of the Single Publisher model, with multiple subscription databases.



Multiple Publishers, Single Subscriber

A Point of Service (POS) application is a good example of a Multiple Publisher model. A POS application has multiple publishers, but only one subscriber. In a POS, it is often necessary to send data from the many POS terminals in a store to a central system either in the store, or at the head office where the individual transactions can be consolidated.



Setting Up Snapshot Replication

To keep things simple, you only create two new databases in this replication example: a new database called Publisher that is the publisher, and a new database called Subscriber that is the subscriber. For the purposes of this example, you create a sales schema, and, in that schema, you create a single table, Cars. You then insert a small set of rows into that table. The scenario is based on a car company that sells fuel-efficient hybrid cars in the United States, China, and Sweden. You set up snapshot replication between database servers in the United States and China to refresh data.

You must implement a distributor before you can create publications and subscribe to publications, so first create the distributor that you need for all types of replications.

Setting Up Distribution

As mentioned earlier, a distributor consists of a distribution database (where replication history, status, and other important information are stored) and a shared folder (where data and articles can be stored, retrieved, and refreshed).

The distributor database can live on either the Publisher (the default location), or a separate server. For low volumes of transactions, using the Publisher works well. For a higher volume of transactions, using a separate server will give better performance.

To begin setting up distribution, you must find out the domain name and account that will be used during the process to run various replication agents, such as the Snapshot agent, Log Reader agent, and Queue Reader agent. For the purpose of this example, you can just choose to impersonate the SQL Server Agent account, but when you put things into production, a dedicated domain account is recommended for security reasons.

Following is a step-by-step process of how to set up distribution:

1. Using SQL Server Management Studio, connect to the distributor server. Expand the server in Object Explorer.
2. Right-click Replication and select Configure Distribution.

WARNING
If you use a fresh default install of SQL Server 2014/2016, by default, the Agent XPs are disabled. This prevents you from starting the SQL Agent service. If SQL Agent is not running, you see an error when you try to complete step 2. To prevent this error, enable the Agent XPs using the following script, and then start the SQL Agent service, either from SSMS or SQL Server
Configuration Manager:

```
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'Agent XPs', 1;
GO
RECONFIGURE
GO
```

After executing this script, the SQL Server Agent node in SSMS will no longer have the “Agent XPs disabled” tag appended.

3. At the Welcome screen, click Next. You see the Distributor screen, where you pick which server to use as the distributor. This example uses the current machine. For a more complex topology, you can choose a separate server.

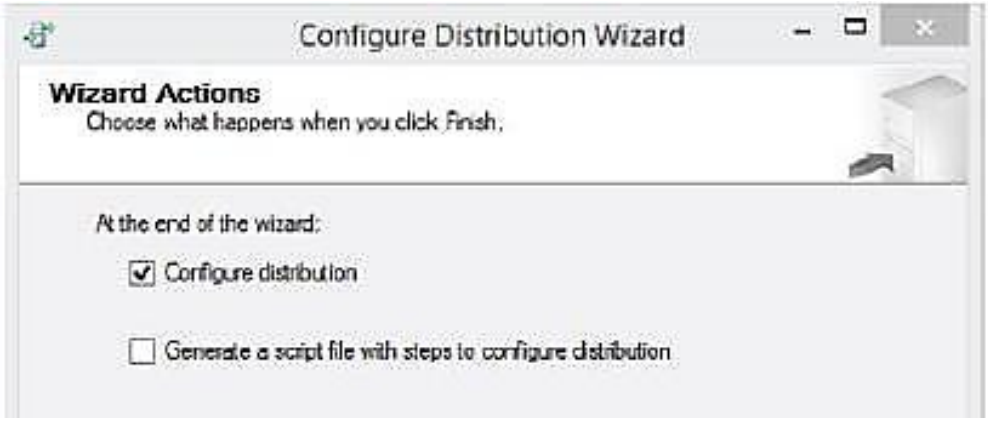
4. Click Next. The SQL Server Agent Start page appears. This enables you to configure the SQL Server Agent to start automatically or manually. Select the option to start the agent automatically.

5. Click Next. The Snapshot Folder screen appears. Here you can enter the snapshot folder. If you want to use a pull subscription, the path you enter here should be a network path. This example sets up a local replication topology, so a network path is not necessary.

6. After you pick the snapshot folder, move to the Distribution Database screen where you configure the name of the distribution database (distribution, in this example), and the folder locations for the database and log files. Click Next to continue.

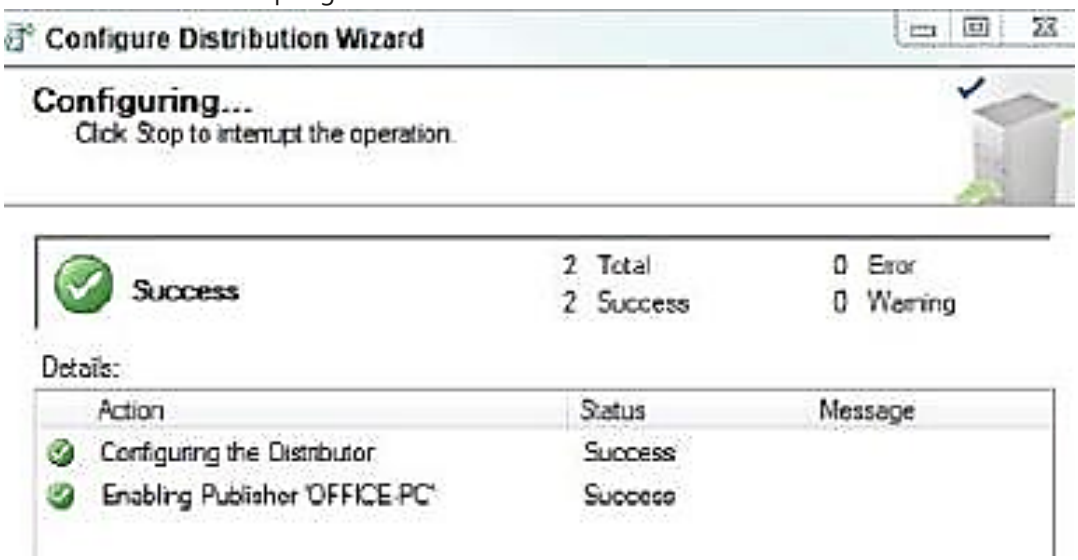
7. Next you see the Publishers screen, where you set the list of servers that can use this distribution database.

8. Next, define what the wizard should do from the Wizard Actions screen. Choose to either configure the distribution database or create scripts. If you want to create scripts, select both options.

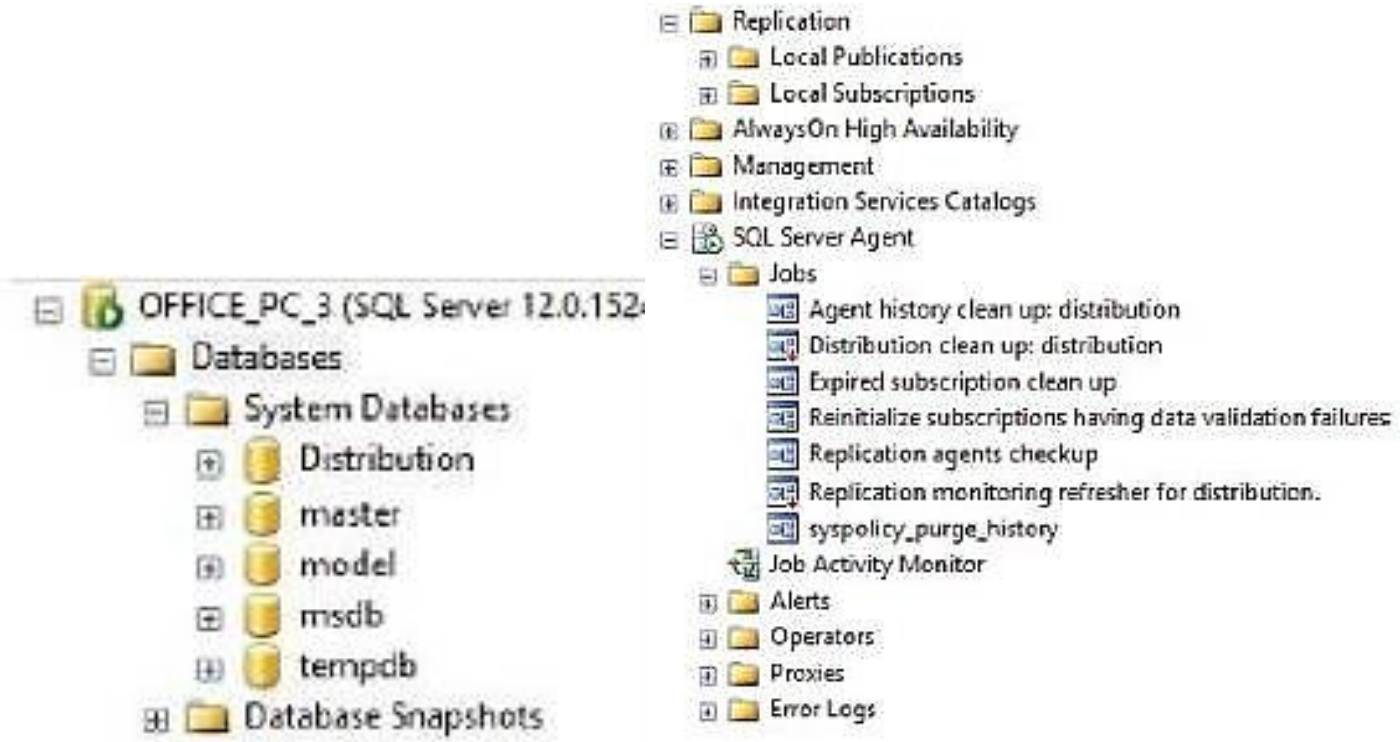


9. The next page of the wizard displays a list of actions the SQL Server can perform. Confirm that the list of actions is what you expect and click Finish. If it's not correct, go back and fix the incorrect settings. You then see a screen indicating the progress the wizard has made at executing the actions you selected.

10. The image below shows the wizard actions completed successfully. If any errors are reported, investigate and resolve each one before attempting to rerun the wizard.



After executing the wizard you can browse to the new distribution database using the Server Explorer in Management Studio. The new distribution database appears under System Databases, as shown in the image below. Additionally, you can expand the SQL Server Agent in Object Explorer and open the Jobs folder, where you see that a number of new jobs were created for the replication.



Implementing Snapshot Replication

Now that you've set up the distributor, you can use that for the replication later. Before you set up snapshot replication, though, you must create the databases to use as the publisher and subscriber.

In this example, you create these databases on a single server.

To create them manually, follow these steps:

1. Right-click the Databases node and select the New Database option.
2. Make sure the publication database is called Publisher, and the subscription database is called Subscriber. You can leave all other settings at default for now.
3. After creating the publisher, you also need to create some tables and load some data.

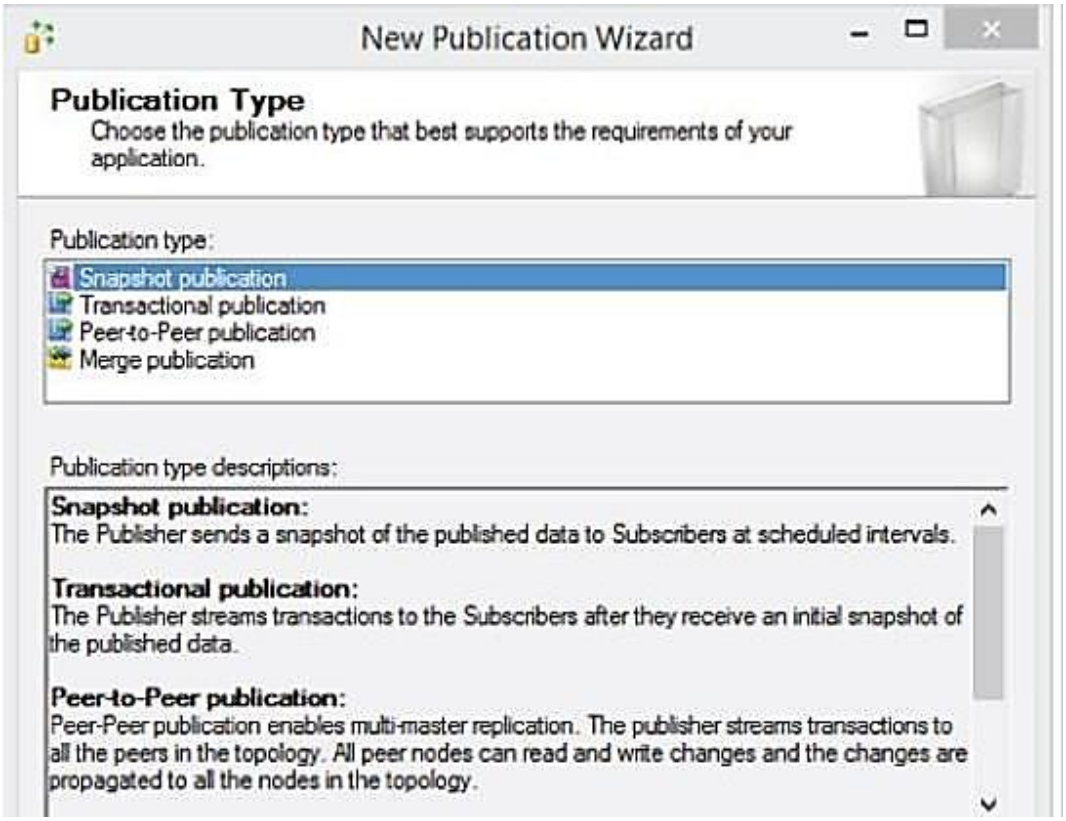
Setting Up Snapshot Publication

The best way to set up Snapshot Publication is to use Management Studio and elect to have everything scripted at the end of the process. Follow these steps:

1. Within Management Studio, while connected to the server where the publication database resides, expand the Replication folder. Right-click Local Publications and select New Publication.

2. You see the Publication Wizard welcome screen; click Next. On the Publication Database screen, pick a database for publication. Select Publisher as your database to create a publication, and click Next.

3. The image below shows the Publication Type screen where you select the type of replication. In this case, select Snapshot Publication and click Next to continue.



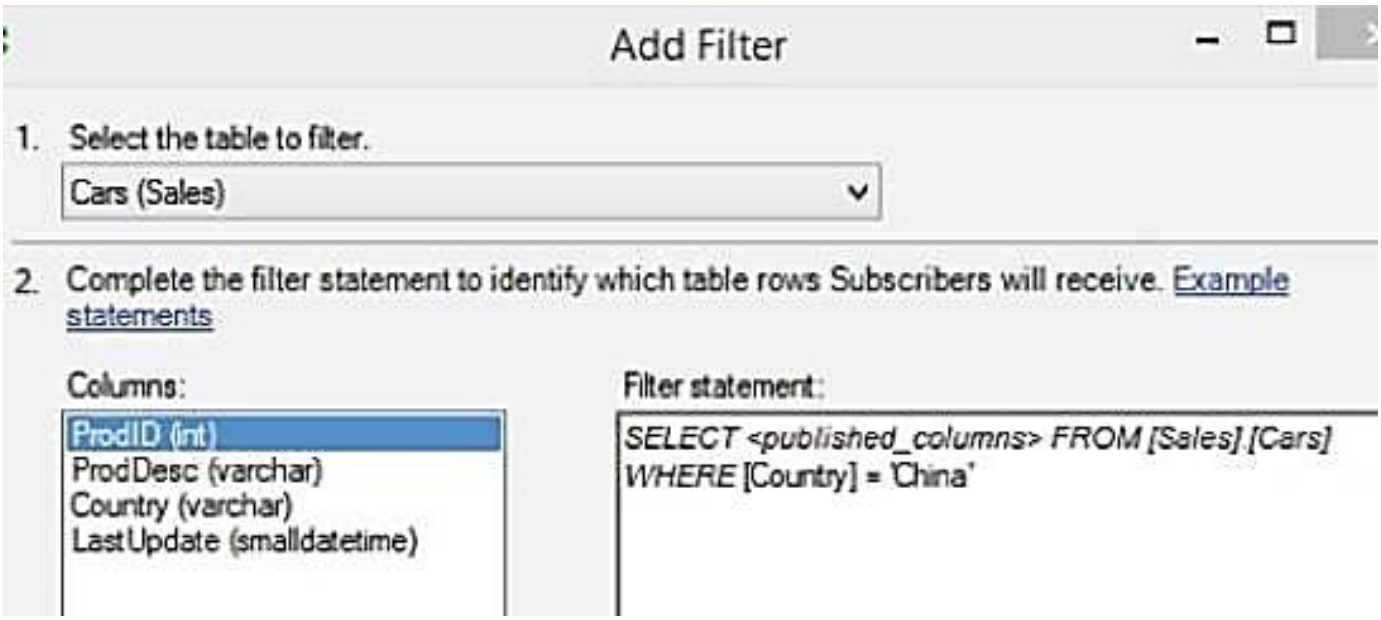
4. Next is the Articles screen where you select the tables in this article. By default, nothing is selected, so you must expand the Tables node and pick the tables you want to publish. Once you expand, select the Cars table you created earlier for publication and its children, as shown in the image below. You can see that the Cars table under the Sales schema is selected. In other scenarios, if necessary, you can pick and choose columns of the table for publication by unchecking the box next to the column name.

You can also set properties of articles that you choose to publish. These properties affect how the article is published and some behaviors when it is synchronized with the subscriber. Don't change the default properties here; however, they can be useful in other situations. Click Next to continue.

5. On the Filter Table Rows page, the wizard gives you an option to filter out rows. Click the Add button to display the Add Filter page.

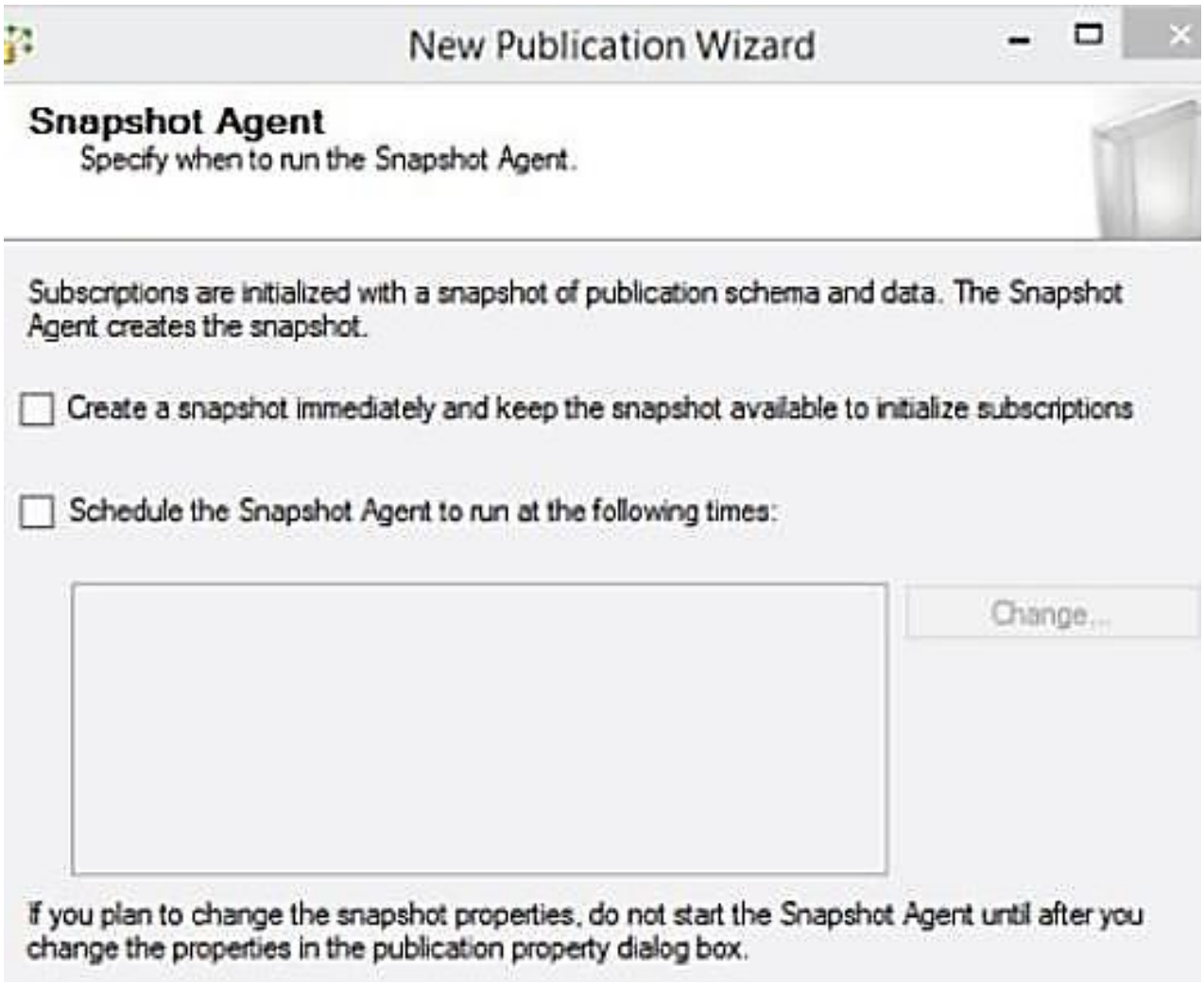
6. As mentioned earlier, because you want to replicate data to the Chinese market, you need to filter the cars by country. To do this, add a WHERE clause to the "filter statement" to match the following SQL text, as shown in the image below.

```
SELECT <published_columns>
FROM [Sales].[Cars]
WHERE [Country] = 'China'
```



After you click OK, the filter is applied. This returns you to the previous screen, which now has a filter defined, as shown in the image below. Click Next to continue.

7. The image below shows the Snapshot Agent screen, where you define how the snapshot should be created, and when it should be scheduled to run. In this case, don't schedule it or create one immediately. Instead, in the example, you will invoke it manually by running a SQL Server Agent job yourself. Click Next to continue.

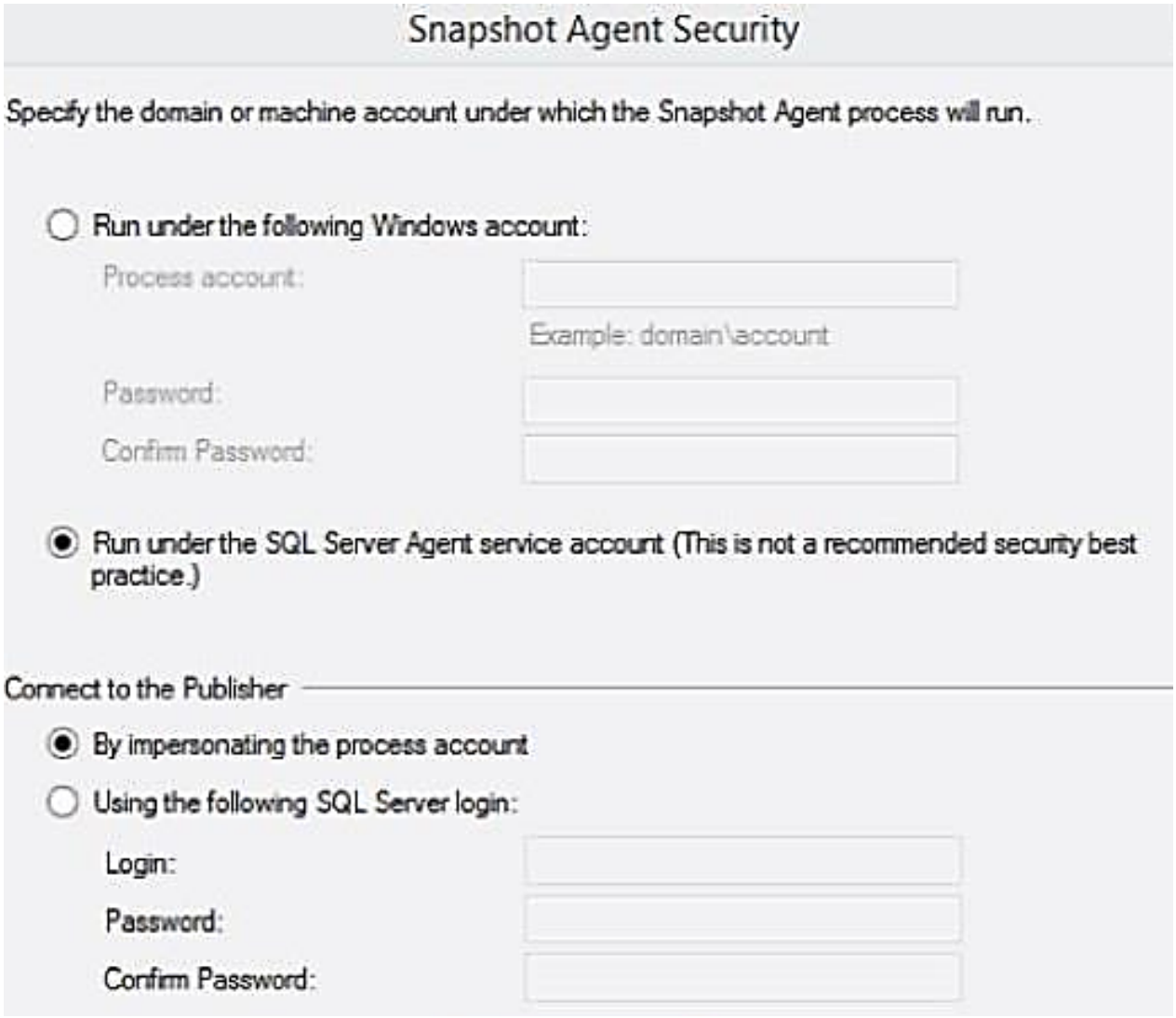


8. The next screen is the Snapshot Agent Security screen, where you can specify the accounts used to run the Snapshot Agent job and to connect to the publisher. As mentioned earlier, different replication models call for different agents to be run. Click the Security Settings button to configure the accounts needed for your replication model.

It is convenient to have a dedicated domain account with a secure password that doesn't need to be changed often for this purpose. However, if you don't have that access, you can choose to impersonate the SQL Server Agent account, as mentioned previously. Even though you will do so for this exercise, it is a best practice to always use a dedicated account.

If you chose to use a dedicated Windows account, then the account must be a member of the db_owner database role in the distribution database and have write permissions on the snapshot share.

Enter the account details on the Snapshot Agent Security screen, as shown in the image below. After the account is set, click OK. On the Agent Security page, click Next.



9. The image below shows the Wizard Actions screen, where you can specify whether you want the wizard to create the publication, or script the creation. For now, leave these settings on the default, and just create the publication.

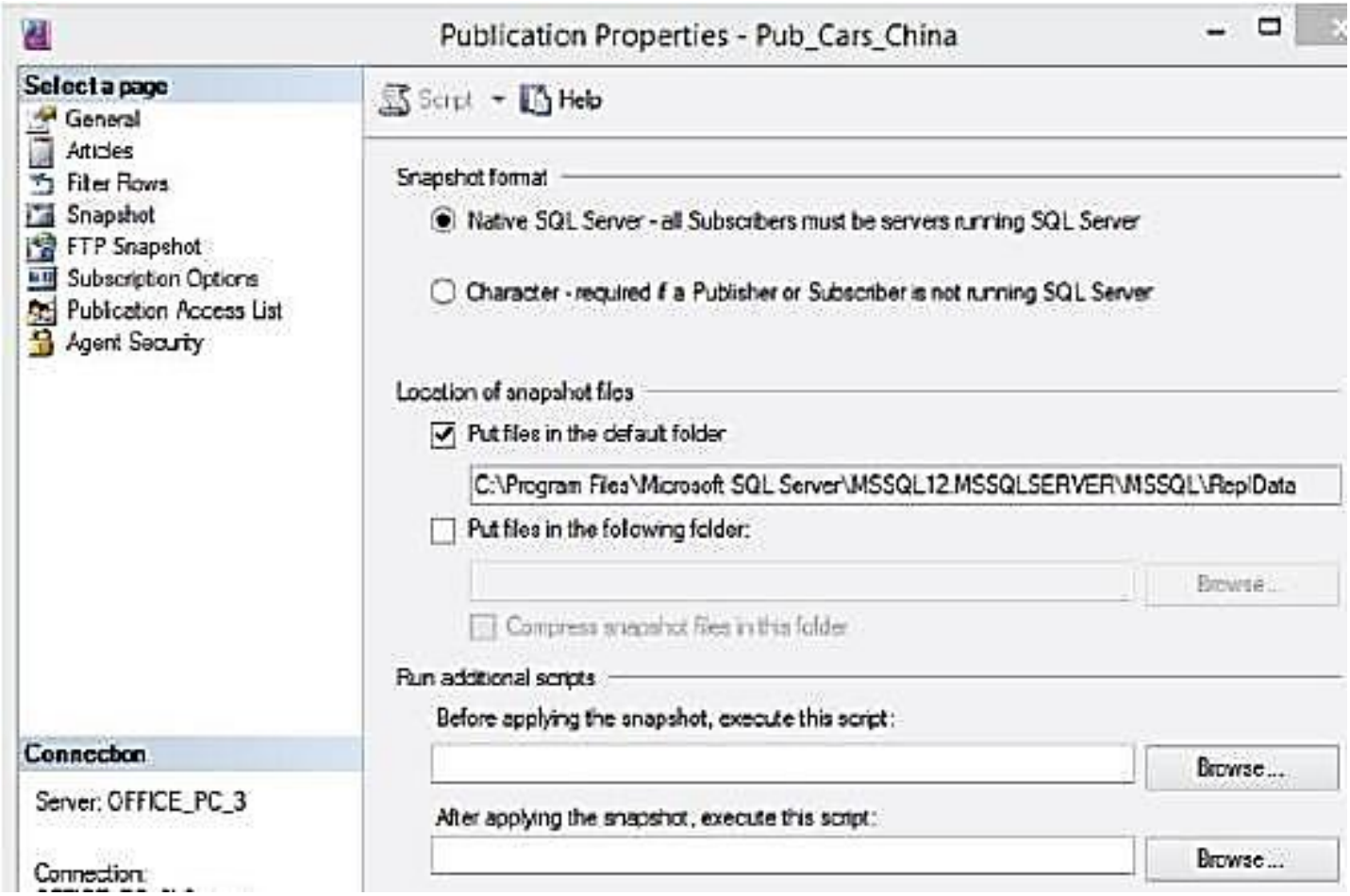
10. The next screen is the wizard action confirmation screen. Here you can assign a name to this publication. Enter the name Pub_Cars_China in the Publication Name text box. Review the actions specified here, and if anything is incorrect go back and fix it. When everything is correct, click Finish.

11. The next screen (shown in the image below) confirms your actions, which, in this case, are to Create the Publication and then add the articles (of which you have just one). The status should indicate success, showing that you have now successfully created a new publication. If any errors are reported, investigate and resolve them before attempting to execute the actions again.

The rest of the process is quite similar to the distributor creation documented earlier. After you click Finish, the Snapshot Publication is created. Again, you can use Object Explorer to see the publication and SQL Server jobs created during this process.

At this point, no files or folders in the snapshot folder have been created, and the snapshot has not been created because no one is subscribed to the snapshot. To get SQL Server to do anything further, you must subscribe to the new publication. When there is a subscriber, SQL Server creates all the necessary files.

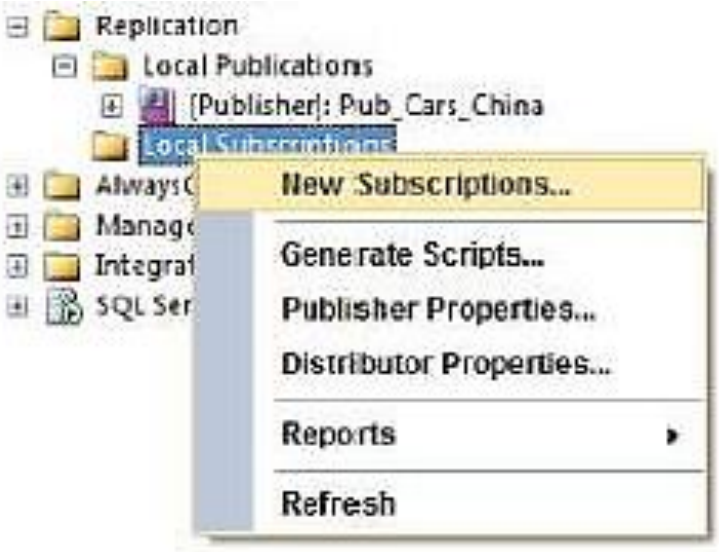
You can find these in the shared folder defined when the distributor was set up earlier. The default location is C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER\MSSQL\ReplData. If you specified a different location during setup, you can always retrieve the current location by expanding the Replication node in SQL Server Management Studio Object Explorer, expanding Local
Publications, and selecting the relevant publication. Right-click and select Properties. On the Snapshot page (shown in the image below), you see a section titled "Location of snapshot files" that shows you the current location.



Setting Up a Subscription to the Snapshot Publication

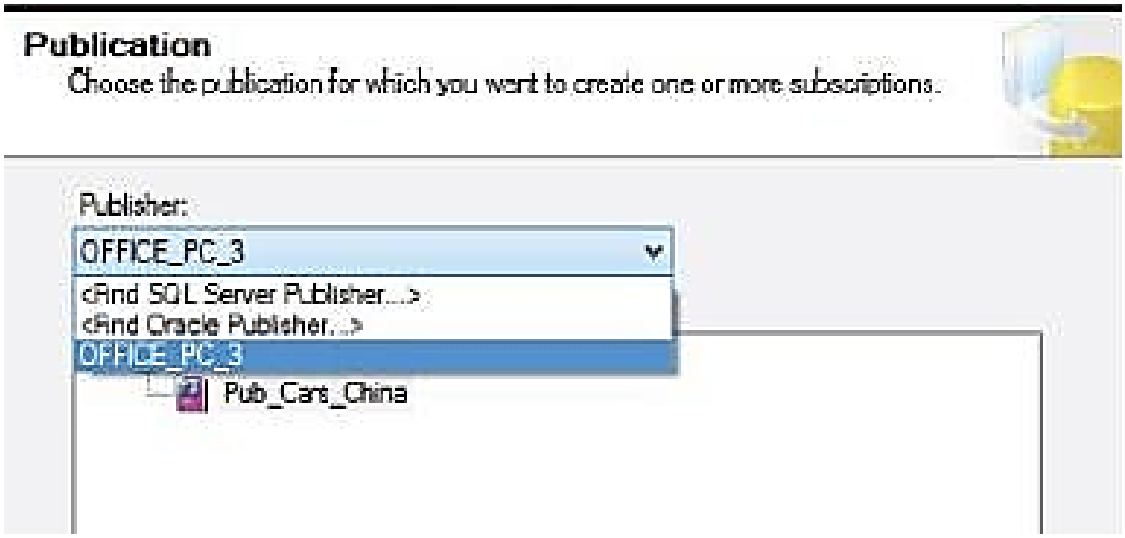
Now that a Snapshot Publication is created, you can subscribe to it from a different database, or a different server. Follow these steps:

1. If you are not already connected to it, connect to the subscription server using SQL Server Management Studio, and expand the Replication node in Object Explorer. Right-click Local Subscription and select New Subscriptions.

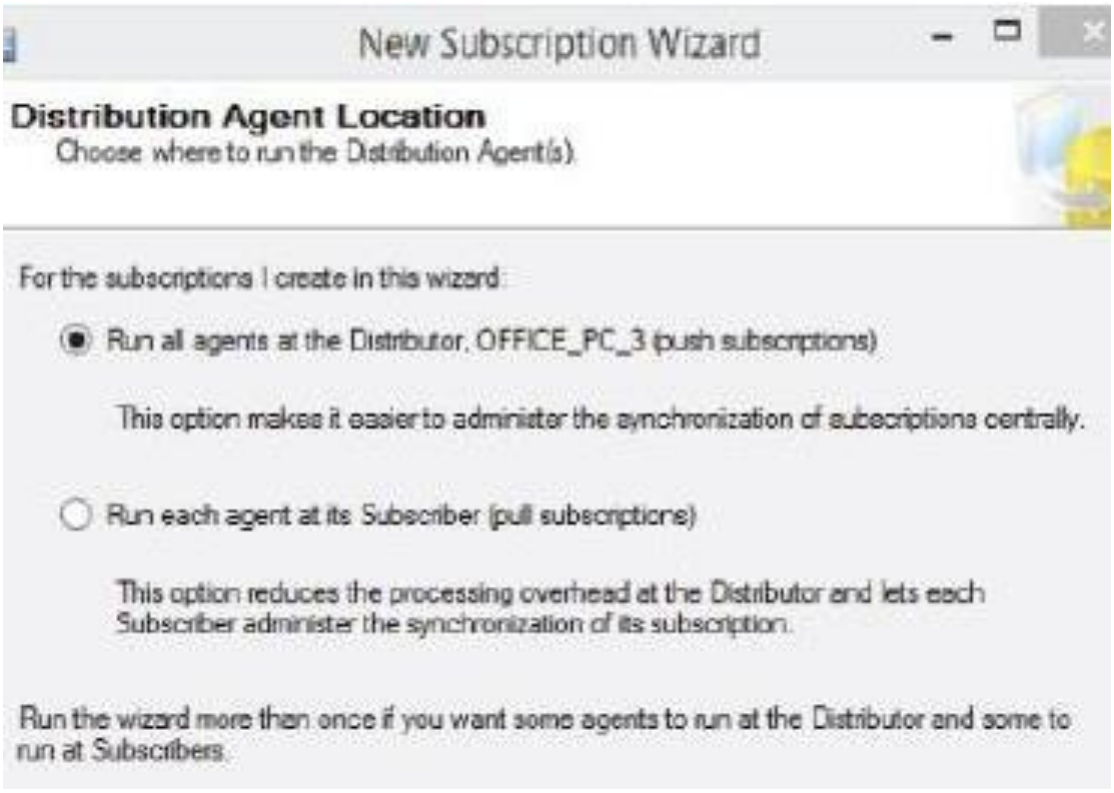


2. You see the welcome screen of the New Subscription Wizard. Click Next to choose the publication you want to subscribe to. This example has the publisher and subscriber on the same server, so the page should already be populated with the available publications. In a real scenario in which the publisher and subscribers are on different servers, you must find the publishing server. Do this by selecting "<Find SQL Server Publisher…>" from the drop-down list (see image below).

You see the typical Connect to Server window that is common in SQL Server Management Studio. To connect to the server where you set up the Snapshot Publication earlier, select the publishing server, and click Connect. You then see the Publication screen. Expand the database (Publisher) and select the publication you just created by the name you gave it: Pub_Cars_China. Click Next to continue.

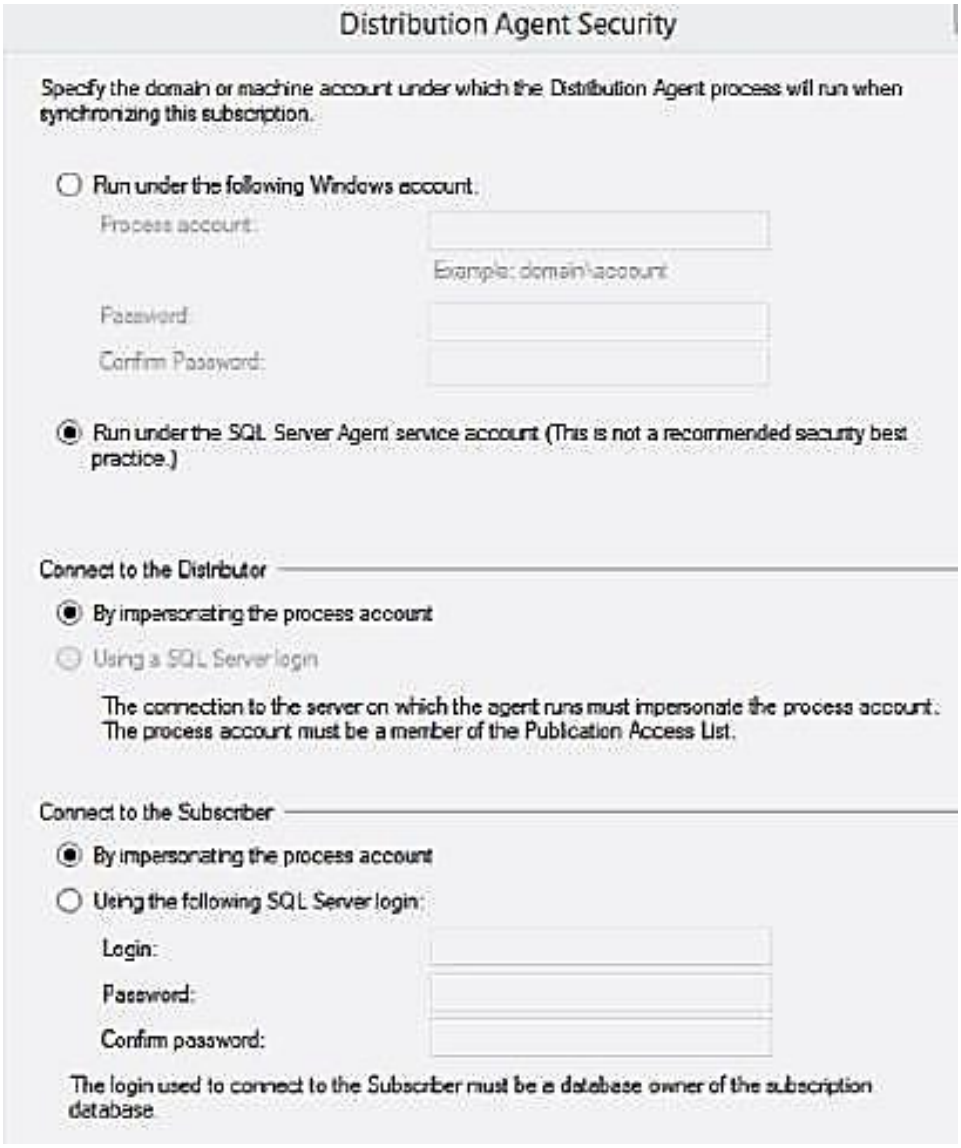


3. The next screen is the Distribution Agent Location screen, shown below. Here, select the location for the distribution agents to execute. This can be either at the distributor for a push subscription or at the subscriber for a pull subscription. Make your subscription a push subscription by selecting the first option, "Run all agents at the Distributor," and click Next.

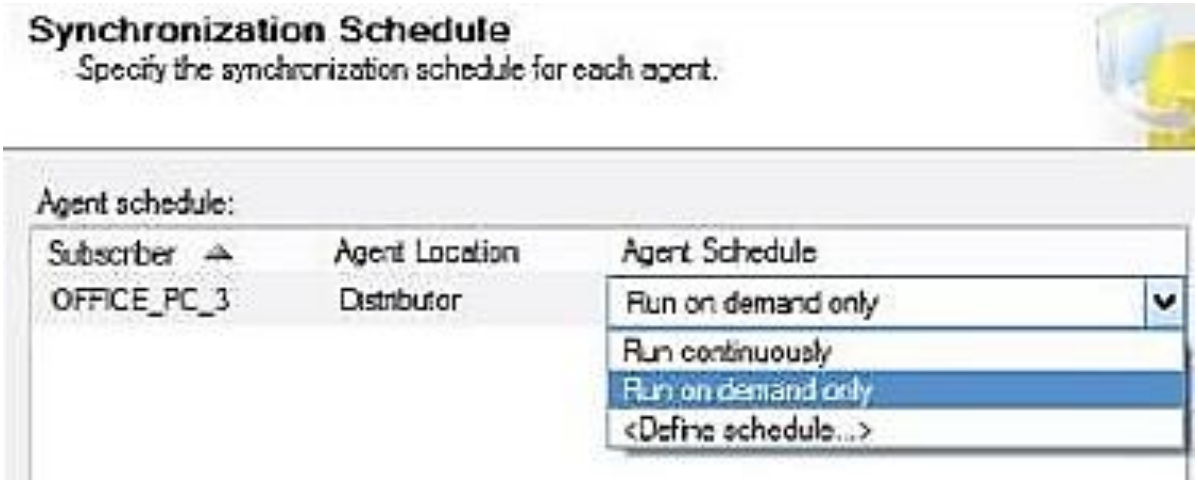


4. The next screen is the Subscribers screen (shown in the image below), where you can set the subscriber properties. Select the server where the subscriber will be; in this case, it is the current server. Then select the subscription database. If you want to add additional subscribers, you can do so on this page as well by using the Add Subscriber button. Select Next to move to the next screen.

5. The image below shows the Distribution Agent Security screen, where you see the familiar screen used for setting agent security settings. Click the ellipsis (…) to set up the security options. If you want to specify a different domain account for Distribution Agent Security, fill out the account information here.



6. Specify the synchronization schedule. Because this is just your first test example, make it simple and set it to run on demand. Click the drop-down and select "Run on demand only,". Click Next to continue.



7. The image below shows the Initialize Subscriptions screen where you specify when the snapshot is initialized. Check the box to select initialization; and in the drop-down, choose to initialize immediately.

8. Next is the Wizard Actions screen shown earlier, where you can choose to create the subscription and script the creation. Make your selection and click Next to continue.

9. From the confirmation screen that appears, confirm that the actions are as you expect. If not, go back and fix anything before executing the wizard actions you selected. When you are satisfied, click Finish.

10. The next screen shows the actions and their status. You should see three actions, with the last of these indicating a warning. Clicking the message for this action should reveal a message indicating that the snapshot is not available. Although you marked the subscription to initialize immediately, it must wait until you manually run the Snapshot agent to create a snapshot that is available to be applied. If this step shows any errors, investigate and resolve them before attempting to rerun the wizard.

Database Maintenance

1. Backups

Creating a full backup

1. Open SSMS and connect to an instance of SQL Server.
2. In Object Explorer, expand the server tree.
3. Expand the Databases folder.
4. Right-click the AdventureWorks2012 database.
5. Select Tasks | Back Up.
6. In the Back Up dialog box, you can select a different database to back up, but just accept the selected database. You can also change the backup type from Full to Differential or Transaction Log. For now, accept Full.

7. Directly below the backup type, you can specify if you want to take a Copy-Only Backup. Copy-only does not start a new backup chain, so if this option is specified, you cannot take a differential backup.

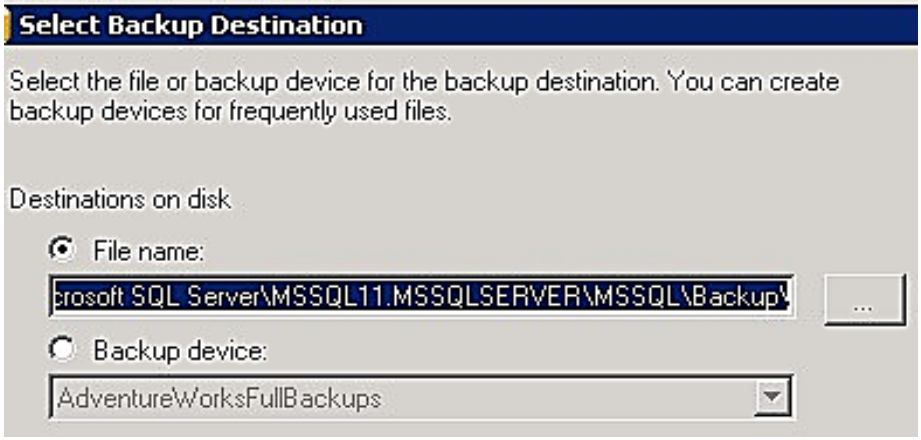
8. In the next section, Backup Set, you can specify a name and description for the database backup. For now, accept the default and enter a description if preferred.

9. Also in the Backup Set section, you can specify when the backup will expire and can be overwritten by SQL Server. It should also be noted that the backup files can be removed by using other tools and through the operating system. Accept the default of 0, which means the backup never expires.

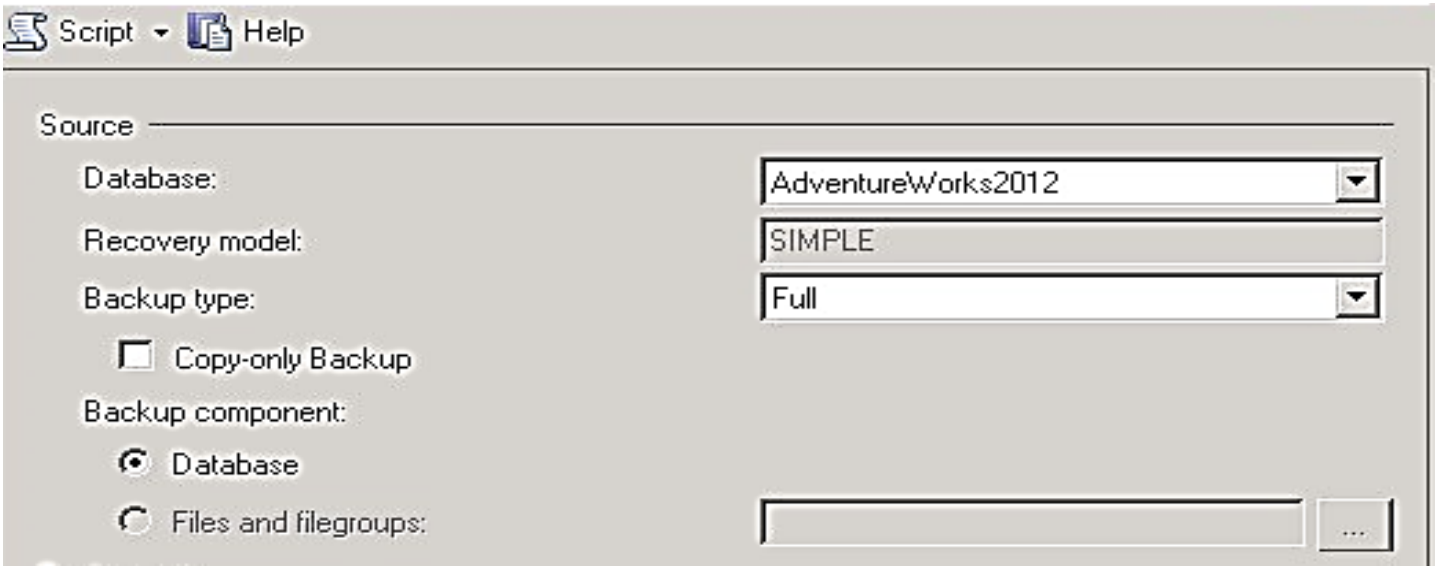
10. In the Destination section, you can specify whether you will back up to disk or tape. The tape option will not be enabled unless you have a device attached to the server.

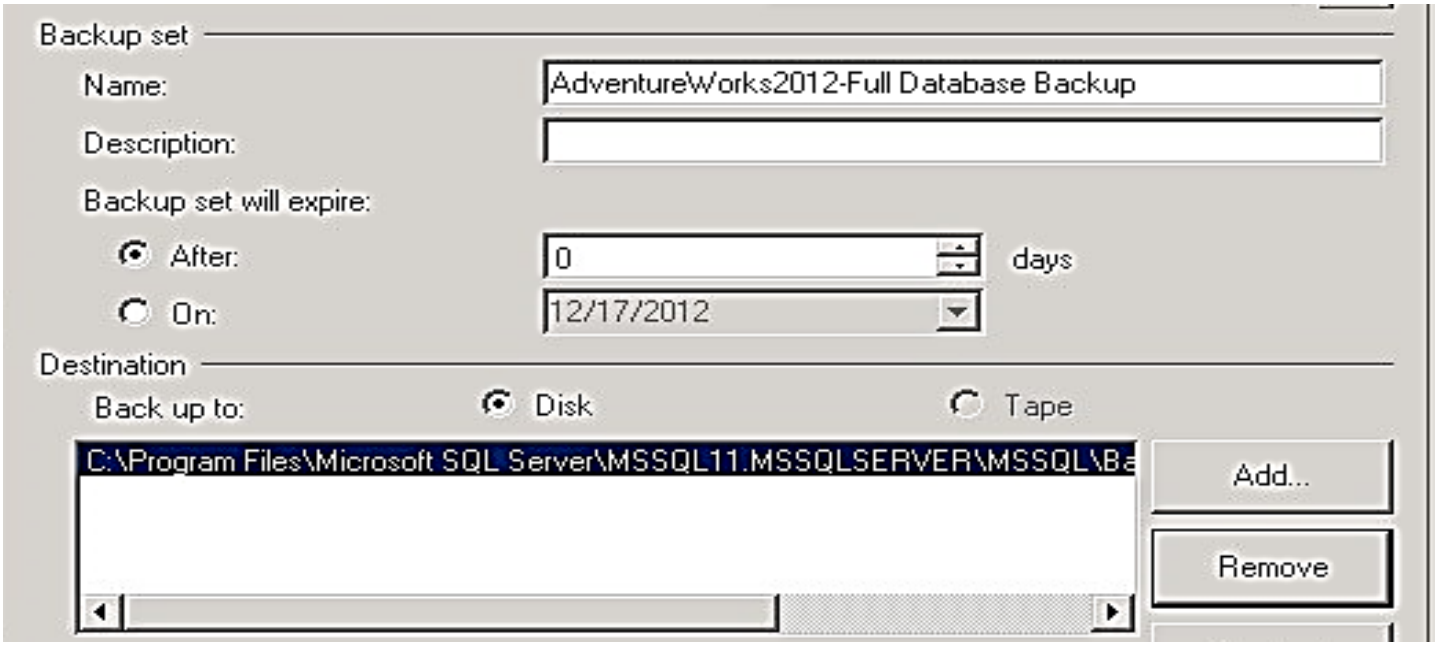
11. Be default, the configured backup location is automatically included. If you had not created a device, the backup could have been stored directly in the location specified. Click Remove to delete that location.

12. Click Add. In the Select Backup Destination dialog box, select File Name.



13. Click the ellipsis, and in the Locate Database Files dialog box, browse to this location: C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Backup\.





14. In the File Name text box in the Locate Database Files dialog box, enter AdventureWorks2012.bak and click OK twice.

15. In the Select a Page pane of the Back Up Database dialog box, select Options.

16. In the first section, Overwrite Media, select the Overwrite All Existing Backup Sets option. By selecting this option, you will empty the backup set. If you accept the default choice to append, you will add a new backup file to the media set during each subsequent backup.

17. In addition to deciding whether to overwrite, you can also choose to check the media set name and backup expiration. If you select this box, you can either enter an existing media set name or leave the media set name blank to create new one. Leave this check box cleared.

18. You can also decide to back up to a new media set. Do not select this option—accept the default, Back Up to the Existing Media Set.

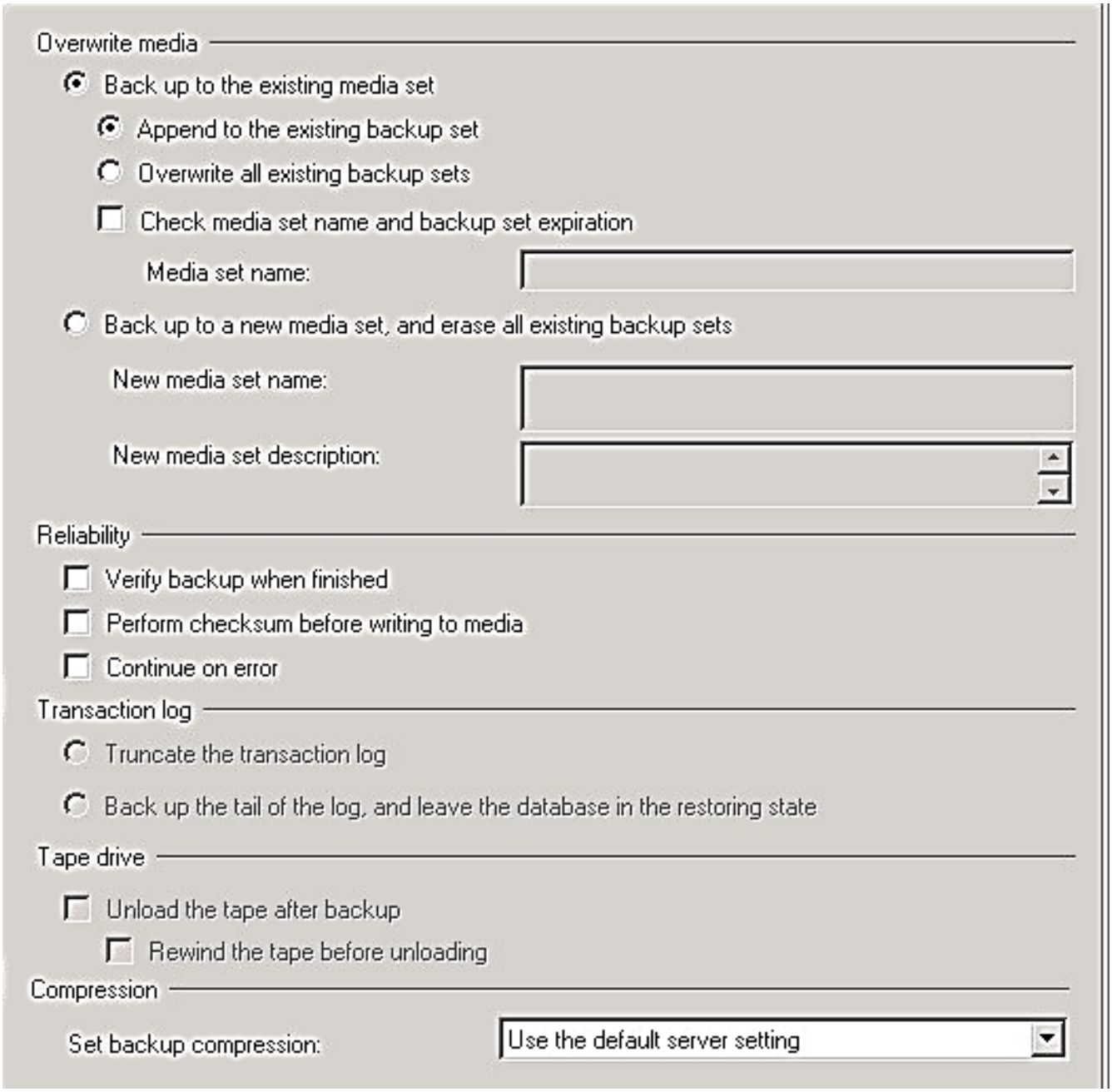
19. In the Reliability section, you can specify three options:

- Verify Backup When Finished
- Perform Checksum Before Writing to Media
- Continue on Error

Leave these check boxes cleared for now.

20. Since you are performing a full backup, the Transaction Log section is not enabled, and since a tape drive is not available, the Tape Drive section is also not enabled.

21. Finally, you can decide whether or not to compress a backup. If you specify to compress the backup, SQL Server will reduce the size of the backup, which ultimately saves space. The amount of space saved depends on several factors, primary the type of data within the database. Compression can be set at the server level or individually. For now, accept the default.



22. Click OK and the backup begins. Once the backup is complete, browse to this directory: C:\Program Files\Microsoft SQL Server\ MSSQL11.MSSQLSERVER\MSSQL\Backup. Here you will see the backup file.

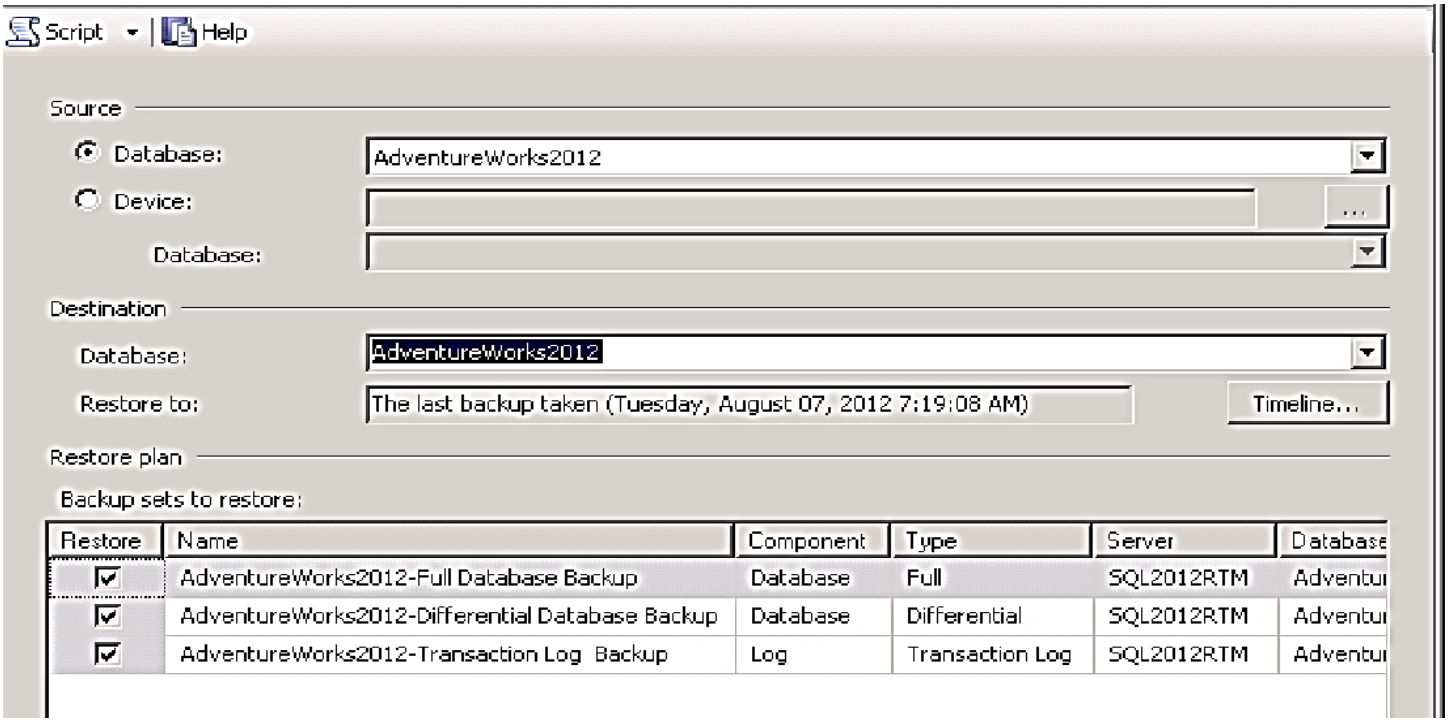
Creating differential backup

1. Open SSMS and connect to an instance of SQL Server.
2. In Object Explorer, expand the server tree.
3. Expand the Databases folder.
4. Right-click the AdventureWorks2012 database.
5. Select Tasks | Back Up.
6. In the Back Up Database dialog box, select Differential from the Backup Type drop-down list. Accept all the other defaults.
7. In the Destination section, click Remove to remove any existing items.
8. Click Add and select File Name.
9. Click the ellipsis, and in the Locate Database Files dialog box, browse to this location: C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Backup\. In the File Name text box in the Locate Database Files dialog box, enter AdventureWorks2012DiffBackups.bak and click OK twice.
10. If you browse to the directory C:\Program Files\Microsoft SQL Server\ MSSQL11.MSSQLSERVER\MSSQL\Backup, you should see two files. One is the full backup, and the other is the differential backup. Notice how much smaller the differential backup is than the full backup.

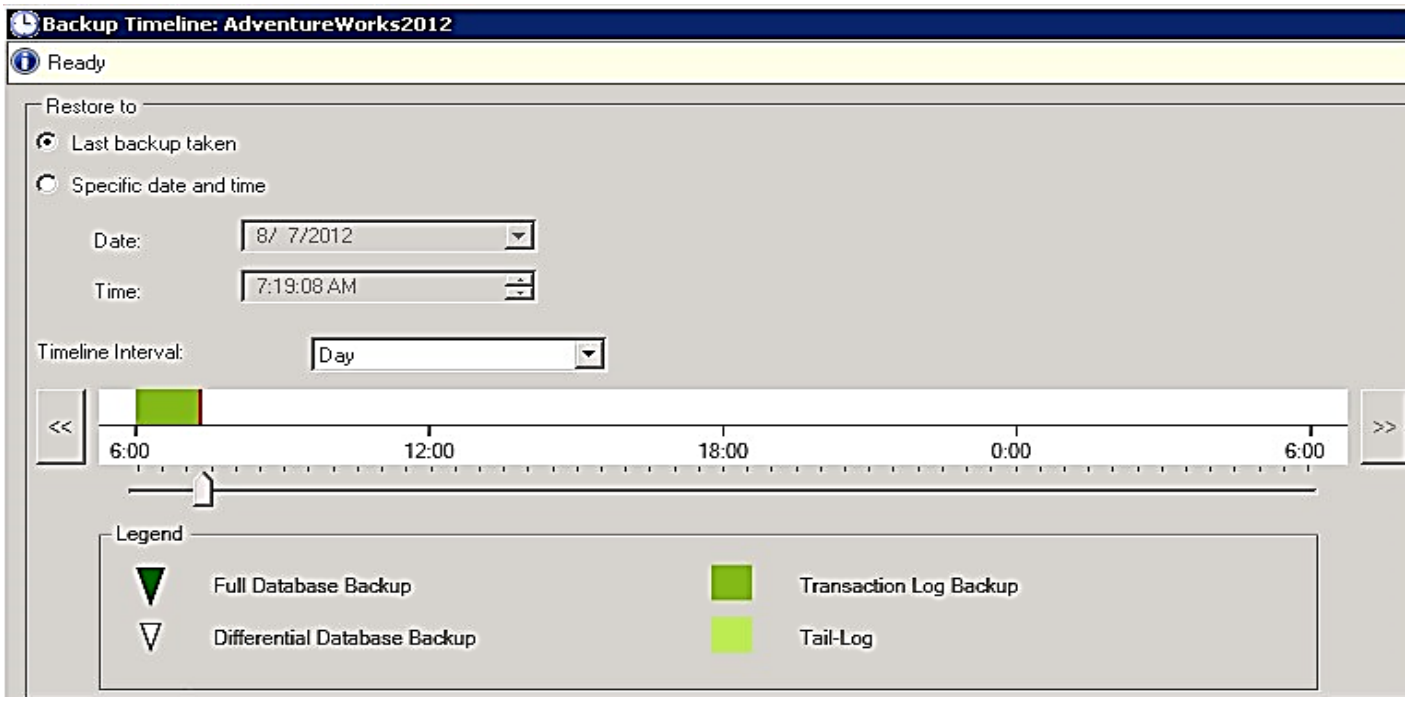
Restore database from backup

1. Open SSMS and connect to an instance of SQL Server.
2. In Object Explorer, expand the server tree.
3. Right-click the AdventureWorks2012 database.
4. Select Tasks | Restore | Database.

Since you are using the GUI, it automatically recognizes all the backups that have been taken and places them in the correct restore order.



5. SQL Server 2016 introduces new functionality to the Restore Database dialog box. If you click the Timeline button next to the Restore To text box in the Source section, the Backup Timeline dialog box opens.



6. Using the timeline, you can easily restore to a specific point in time either by selecting the Specific Date and Time option and specifying a date or by using the slider. Click Cancel.
7. In the Select a Page pane of the Restore Database dialog box, select Files. If you want to move the database files to a new location during the restore, you can use this page.
8. Accept the defaults and select Options from the Select a Page pane.

You can specify several options in this dialog box, but the one that you will focus on is the Recovery State. If you were restoring only a single backup file and wanted to restore additional files later in another operation, you would select RESTORE WITH NORECOVERY from the drop-down list.

9. Ensure that RESTORE WITH RECOVERY is selected from the Recovery State drop-down list.
10. Before a database can restored, all existing connections must be closed. To ensure that all connections are closed, select the Close Existing Connections to Destination Database option in the Server Connections section.
11. Click OK.

2. Managing and maintaining indexes and statistics

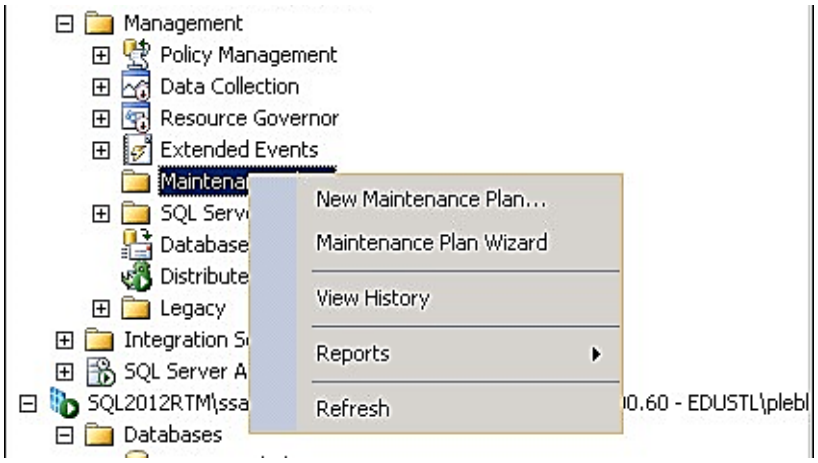
Activity: reorganize index to solve small fragmentation issues

Activity: rebuild index if there is high fragmentation

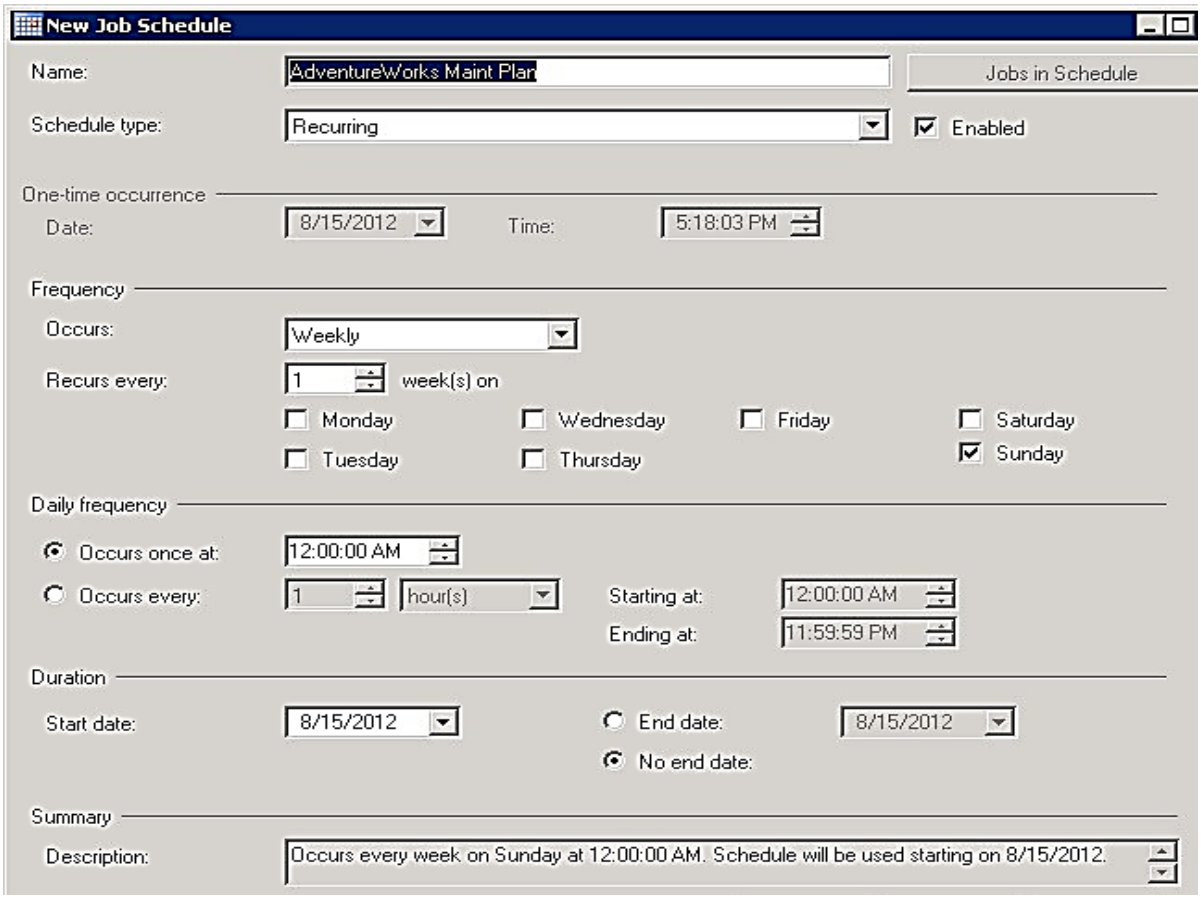
3. Maintenance plans

Activity: using the maintenance plan wizard

1. Open SSMS and connect to a server.
2. In Object Explorer, expand the server tree.
3. Expand the Management folder.
4. Right-click the Maintenance Plans folder and select Maintenance Plan Wizard.



5. Click Next on the Maintenance Plan Wizard Introduction page.
6. On the Select Plan Properties page, enter AdventureWorks Maint Plan in the Name text box.
7. Ensure that the Single Schedule for the Entire Plan or No Schedule option is selected.
8. Click the Change button to create a schedule. The New Job Schedule dialog box appears.



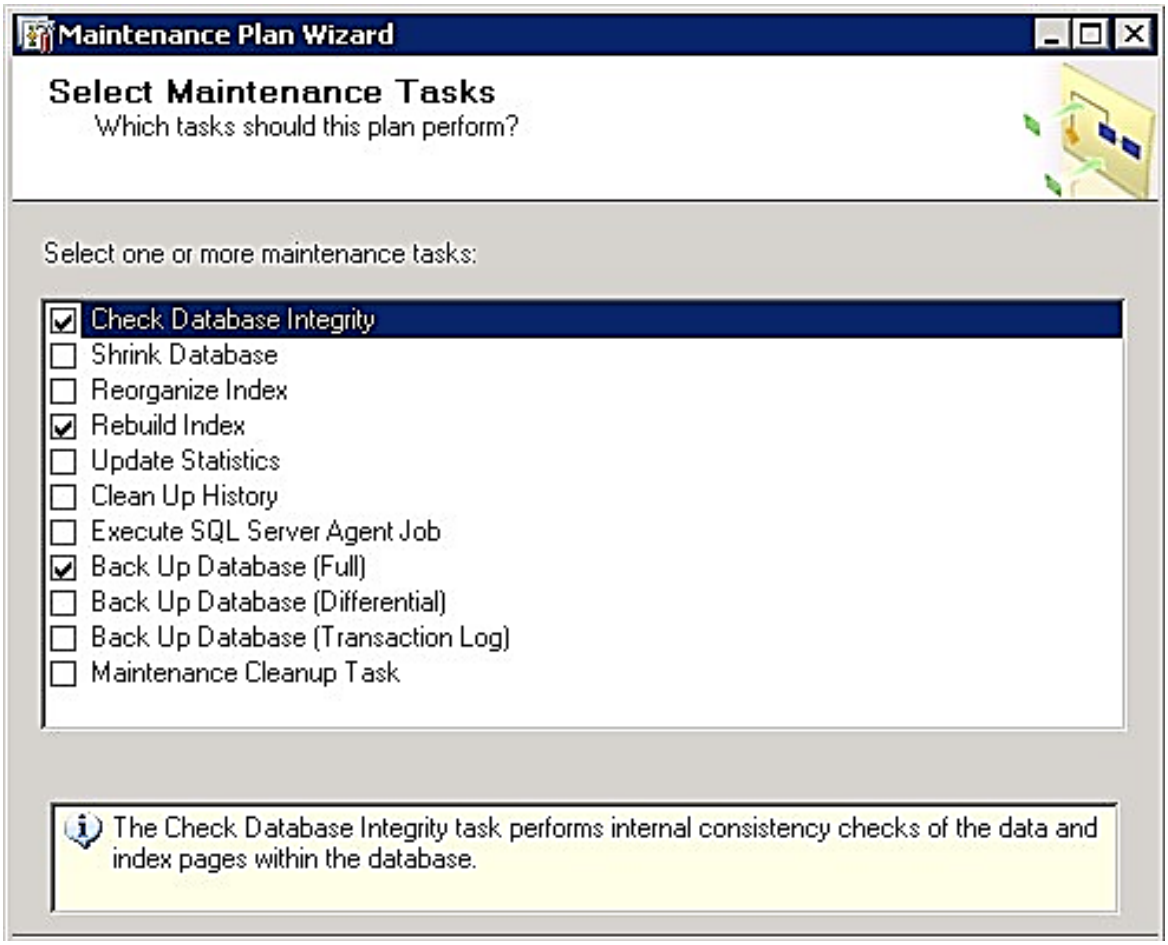
9. In the New Job Schedule dialog box, enter Nightly AW Maint Schedule in the Name text box.

10. In the Frequency section, change the Occurs drop-down list to Daily. Several further options are available, and you should review each section, but for now accept all the other defaults.

11. Click OK.

12. Click Next.

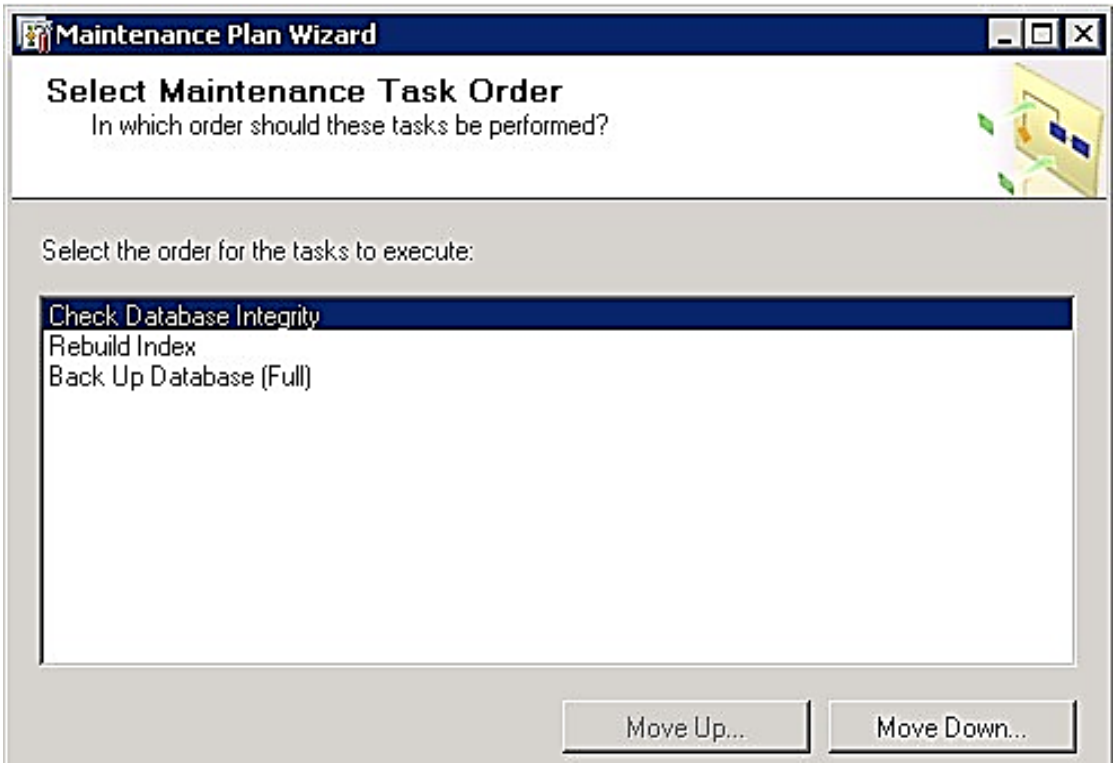
13. On the Select Maintenance Tasks page, select the following options:
    - Check Database Integrity
    - Rebuild Index
    - Back Up Database (Full)



You have several options to choose from, but for the sake of brevity, only a few are selected. You can create additional maintenance plans to perform other tasks. For example, you can create a new maintenance plan to perform differential or transaction log backups.

14. **Click Next.**

On the Select Maintenance Task Order page, you can order the tasks.



15. Change the order to the following:
    - Back Up Database (Full)
    - Rebuild Index
    - Check Database Integrity
16. Click Next.
17. On the Define Back Up Database (Full) Task page, select AdventureWorks2012 from the Database(s) drop-down list. Click OK.

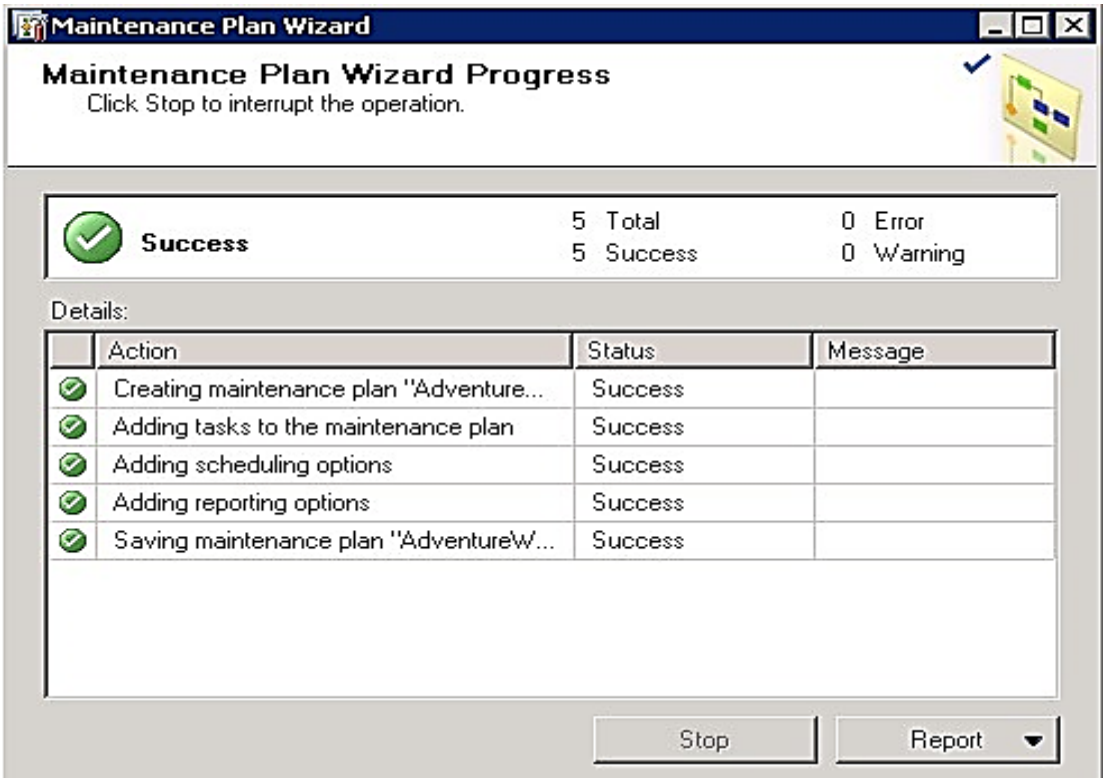
18. Accept all the other defaults and click Next.
19. On the Define Rebuild Index Task page, select AdventureWorks2012 from the Database(s) drop-down list.
20. In the Advanced Options section, select the Keep Index Online While Reindexing check box, and select the Rebuild Indexes Offline option.

21. Click Next.
22. On the Define Database Integrity Task page, select AdventureWorks2012 from the Database(s) drop-down list.

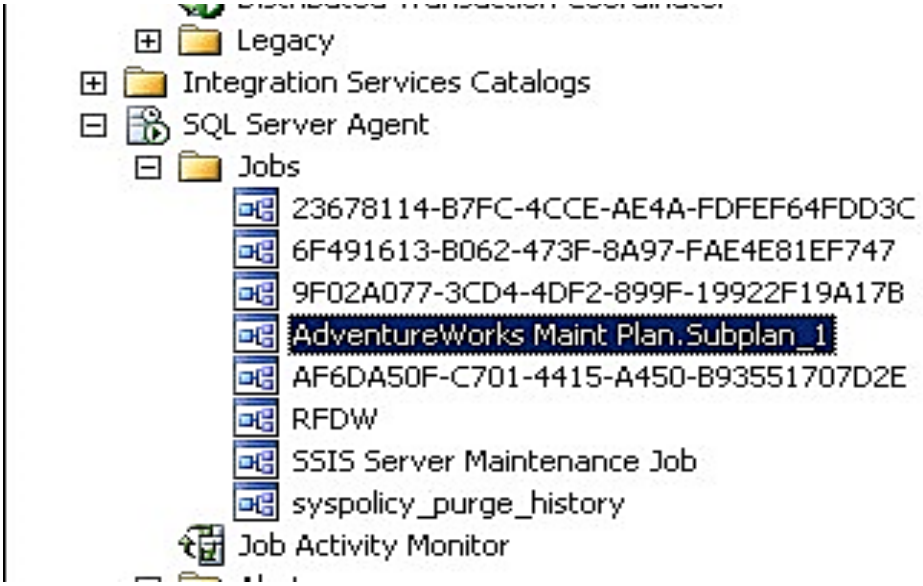
23. Click Next.
24. On the Select Report Operations page, you can specify if and where you want to create a log file. Also, if you have any operators created, you can have the report emailed.

25. Accept the defaults and click Next.
26. Review your selected options on the Complete the Wizard page and click Finish.

The Maintenance Plan Wizard Progress page appears, and each action should successfully complete.



27. Click Close.
28. Finally, to verify that a job was created, in Object Explorer expand the SQL Server Agent Jobs folder.

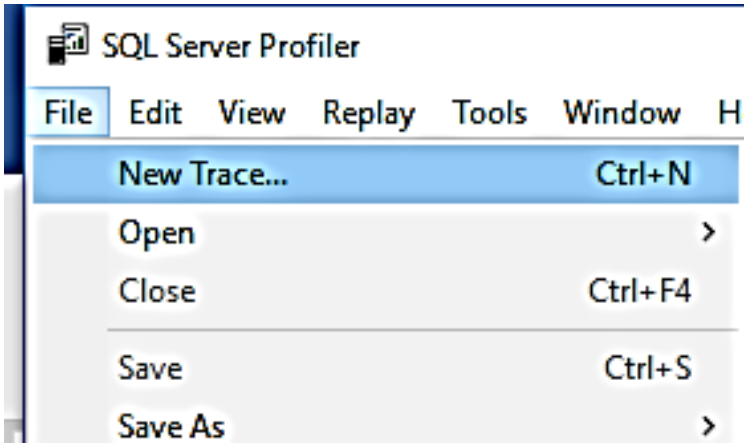


29. Right-click the job and select Properties to view the details.

Database Management

1.SQL Server Profiler

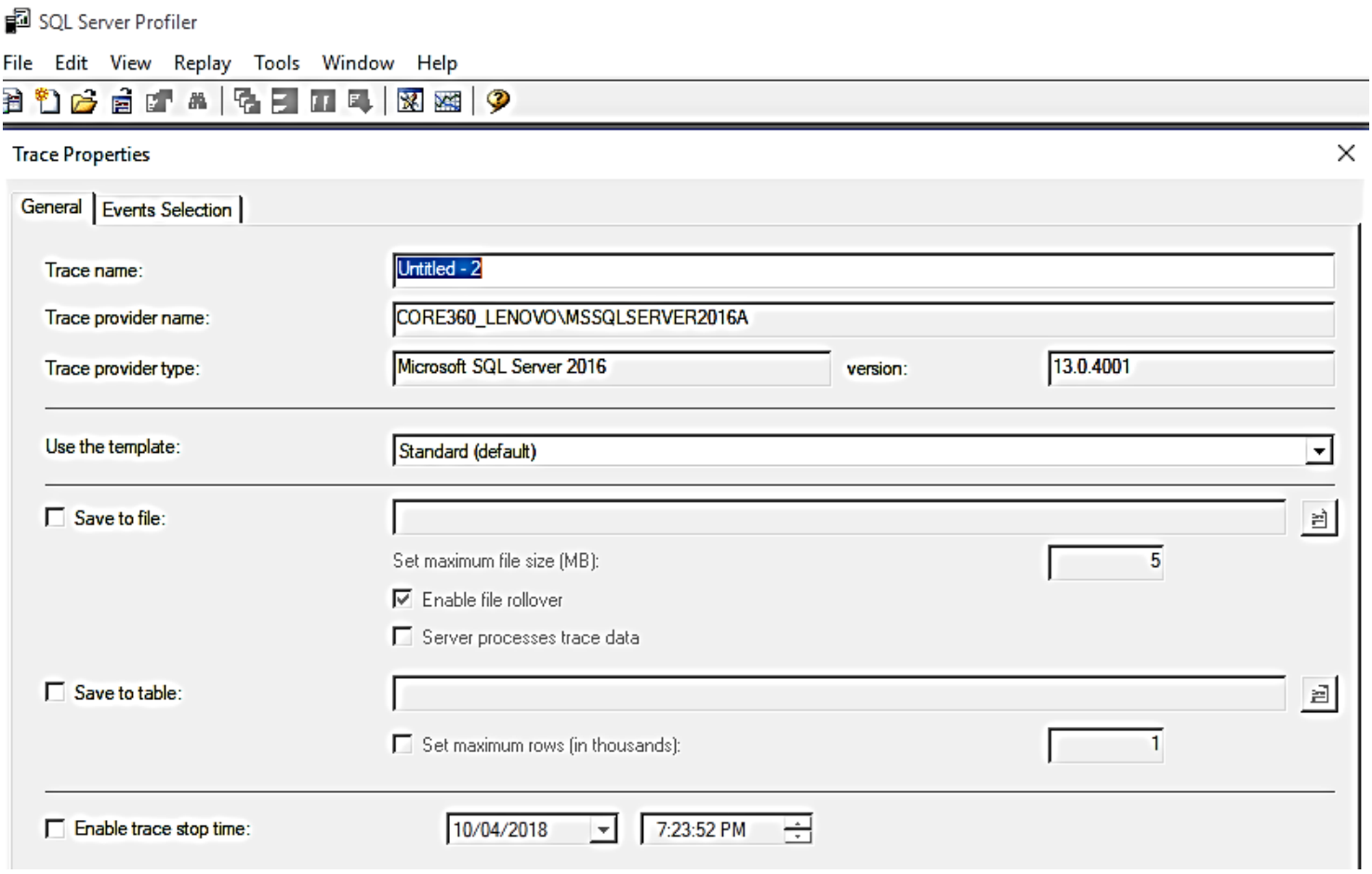
1. Click Start | All Programs | SQL Server Tools 2016 | SQL Server Profiler.
2. Select File | New Trace from the SQL Server Profiler menu.



3. In the Connect to Server dialog box, enter your server name in the Server Name text box.
4. Click Connect.
5. In the Trace Properties dialog box, enter Long Running Stored Procedures in the Trace Name text box.
6. Ensure that Standard (Default) is selected in the Use the Template drop-down list.
7. In the next section, you can save the trace to either a file or a table. This is important when you plan on analyzing the data later. You can always save the trace to a file or table later if you fail to make a selection now. For now, don't select either. The trace will be displayed in the GUI only for now, but later you can save it to another location.

8. The final option is to enable a trace stop time. While this is not a requirement if you are going to run the trace interactively, you should definitely consider including a trace stop time. This will ensure that the trace stops capturing information and does not affect the performance of your server.

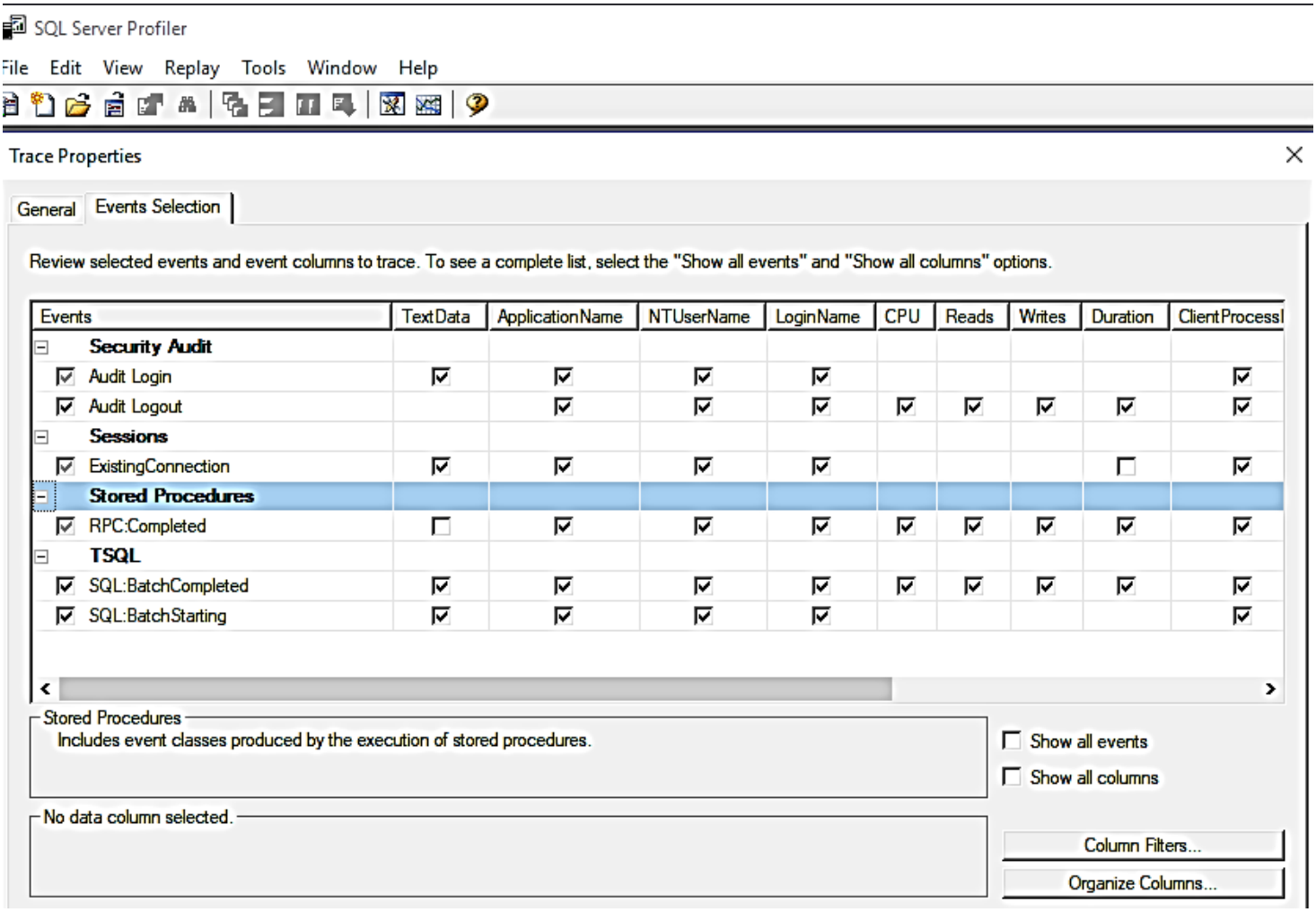
Select the Enable Trace Stop Time check box. By default, the time in the box is one hour from the current time. You can change it to the time that meets your requirements.



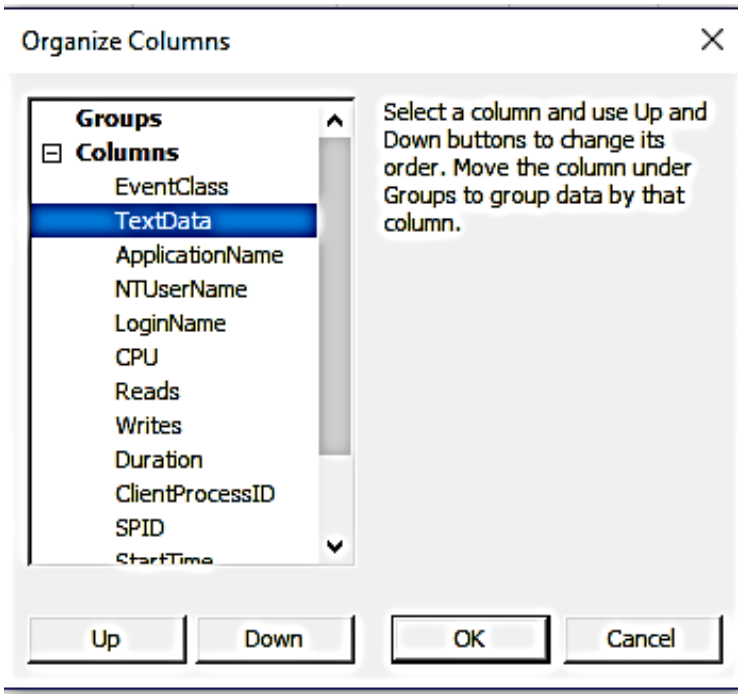
9. Click the Events Selection tab at the top of the Trace Properties dialog box.

This is where you select the different events that will be captured by the trace. Each event is grouped by event category.

10. Clear all events in the Events list.
11. Select the Show All Columns check box.
12. Select the Show All Events check box.
13. Scroll down to the Stored Procedures event category and select the box next to the SP:Completed event. Scroll down further to the T-SQL event category and select the box next to the SP:StmtCompleted event.
14. Clear the Show All Events check box.



15. Click the Organize Columns button.
16. Scroll down the list until you locate TextData. Select it and click the Up button until it is at the top of the list.



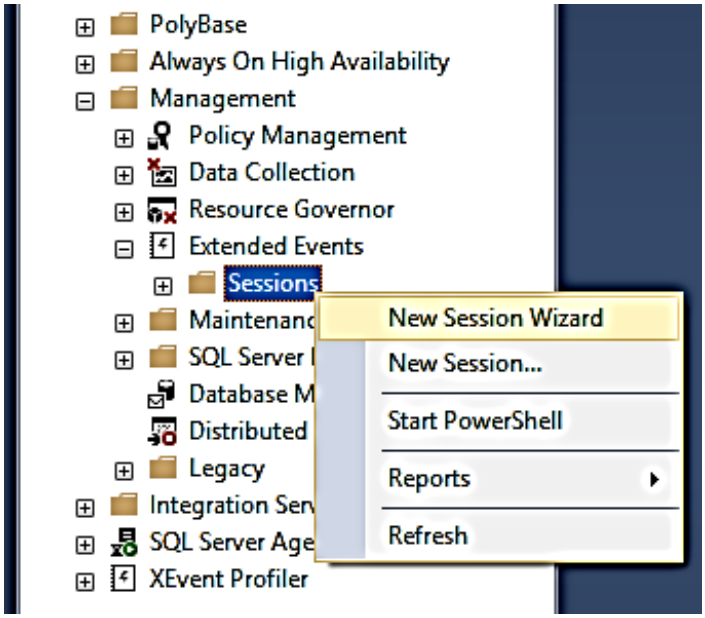
17. Click OK.
18. Click the Run button.

Depending on what is currently using your SQL Server instance, you may or may not see any events being generated.

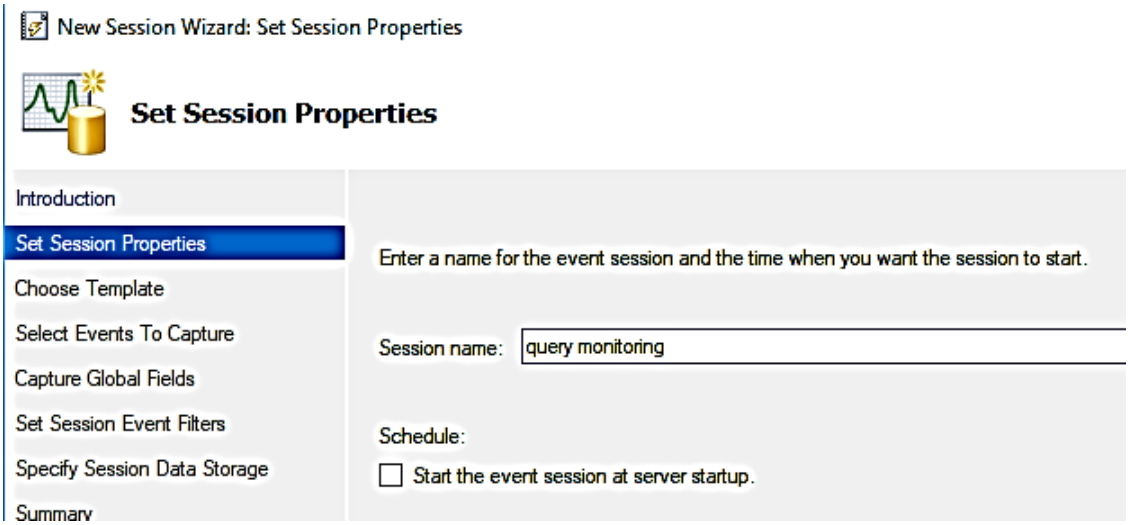
19. To stop the trace. In the menu bar, select File | Stop Trace.

2. Extended events

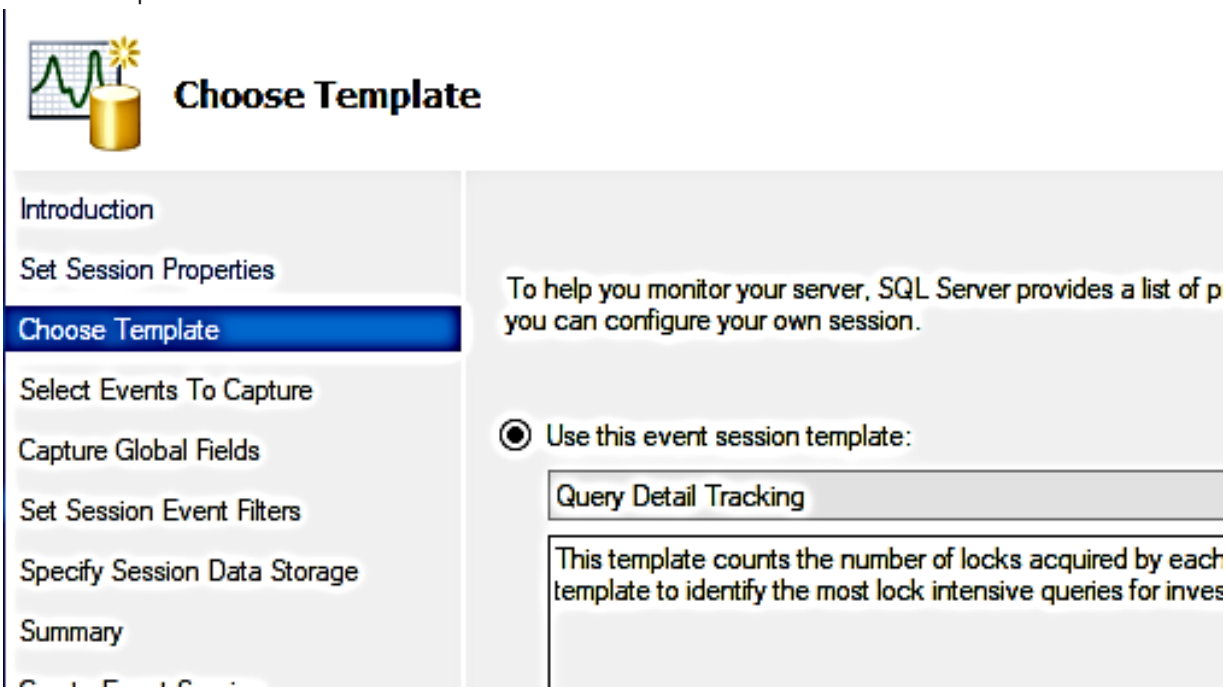
1. Open SQL Server Management Studio (SSMS) and connect to a server.
2. Open Object Explorer if it is not already open.
3. Expand the server node, and expand the Management folder.
4. Expand the Extended Events folder, and then expand the Sessions folder.
5. Right-click the Sessions folder and click New Session Wizard.



6. Skip past the Introduction page by clicking Next.
7. Type Query Monitoring in the Session Name text box. If you are attempting to troubleshoot a startup issue, selecting the Start the Event Session at Server Startup check box may come in handy on this page. For now, leave it cleared. Click Next.



8. On the next page, select the Use This Event Session Template option, and then select Query Detail Tracking from the drop-down list.



9. Click Next.

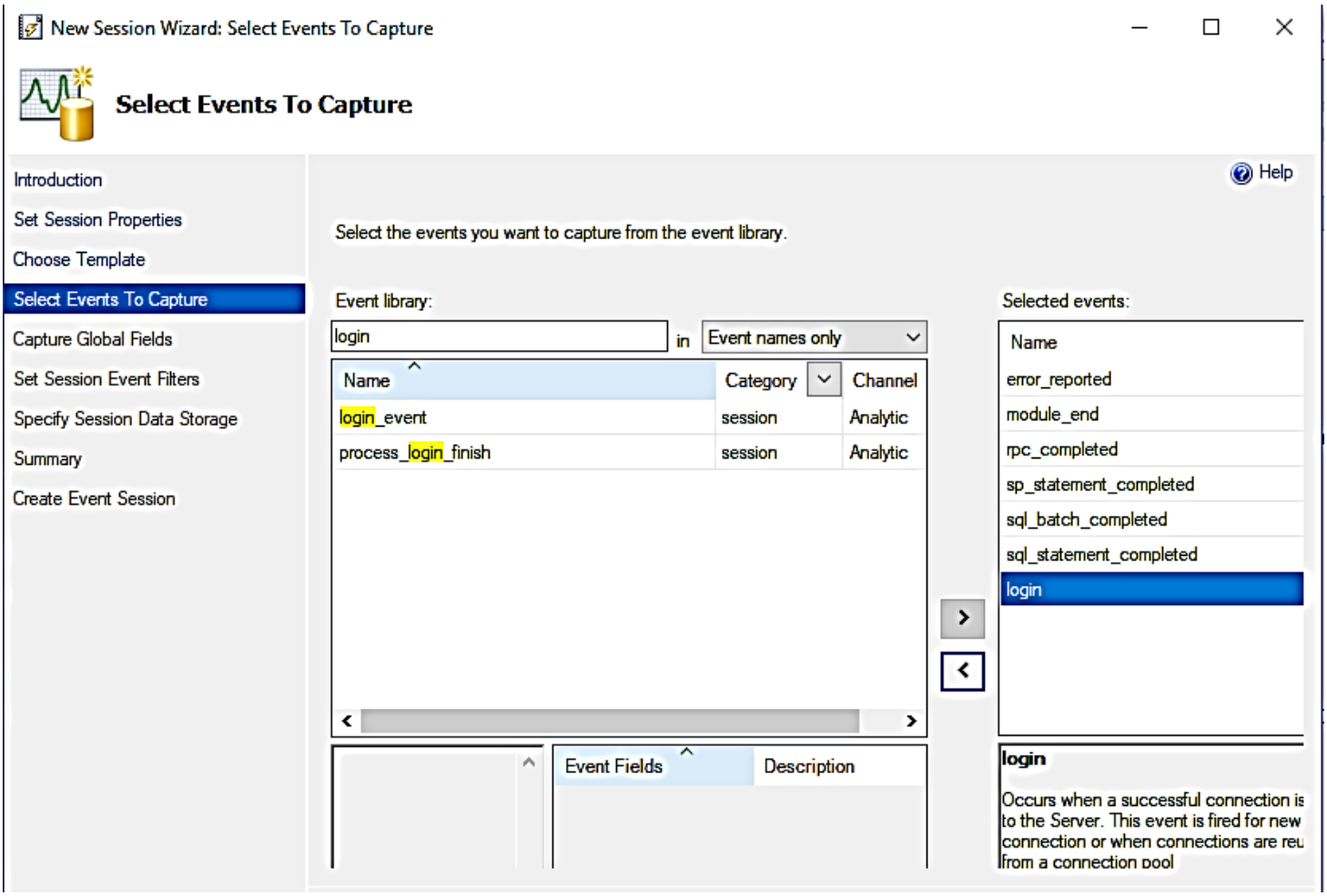
On the Select Events to Capture page, note that the template has already selected a halfdozen events to capture in this Extended Events session. These events are in the Selected Events box on the right side of the page.

10. On the left side of the Select Events to Capture page, type login into the Event Library search text box.
Notice how the library is quickly filtered, allowing you to search for commands easily.

11. Click the results row for login.
Note that below the search results window, a detailed description of the event is provided, as well as fields predetermined for this event.

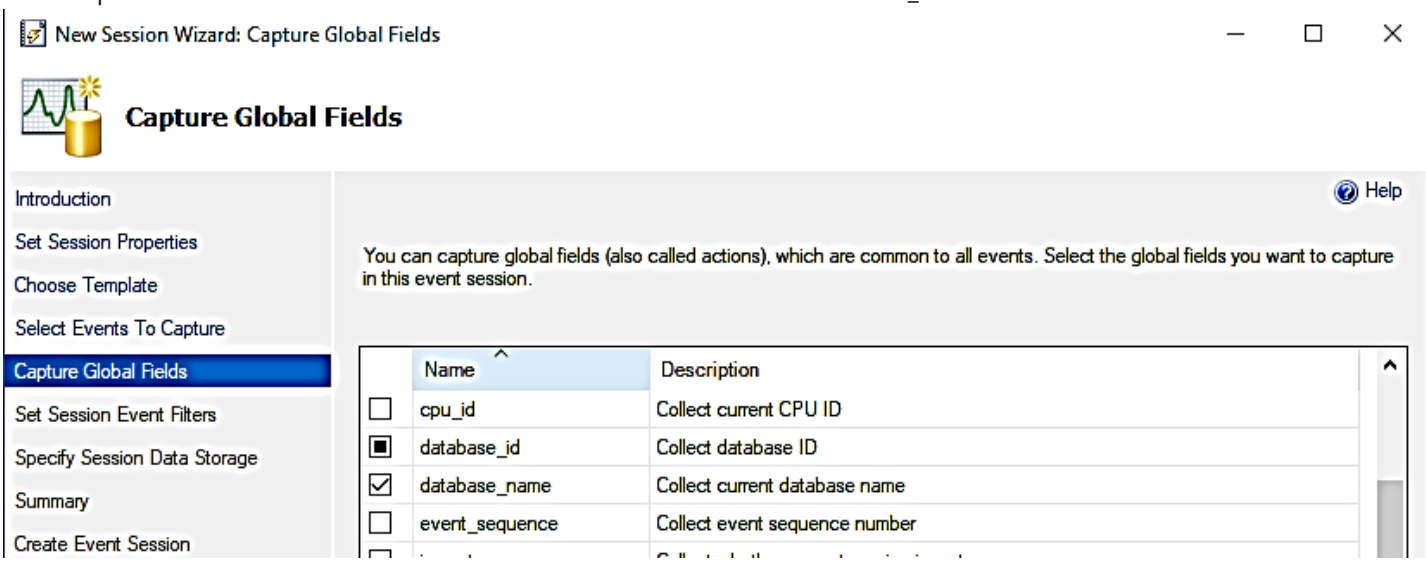
12. Click the > arrow to move the login event into the Selected Events window.
13. Click module_end in the Selected Events window.

14. Click the < arrow to remove the module_end event from the Selected Events window. Now the module_end event will not show up in your session data.



15. Click Next.

16. The Capture Global Fields page already has some of these global fields (or actions) selected for you, based on the template. Scroll down in this window and select the box next to database_name. Click Next.



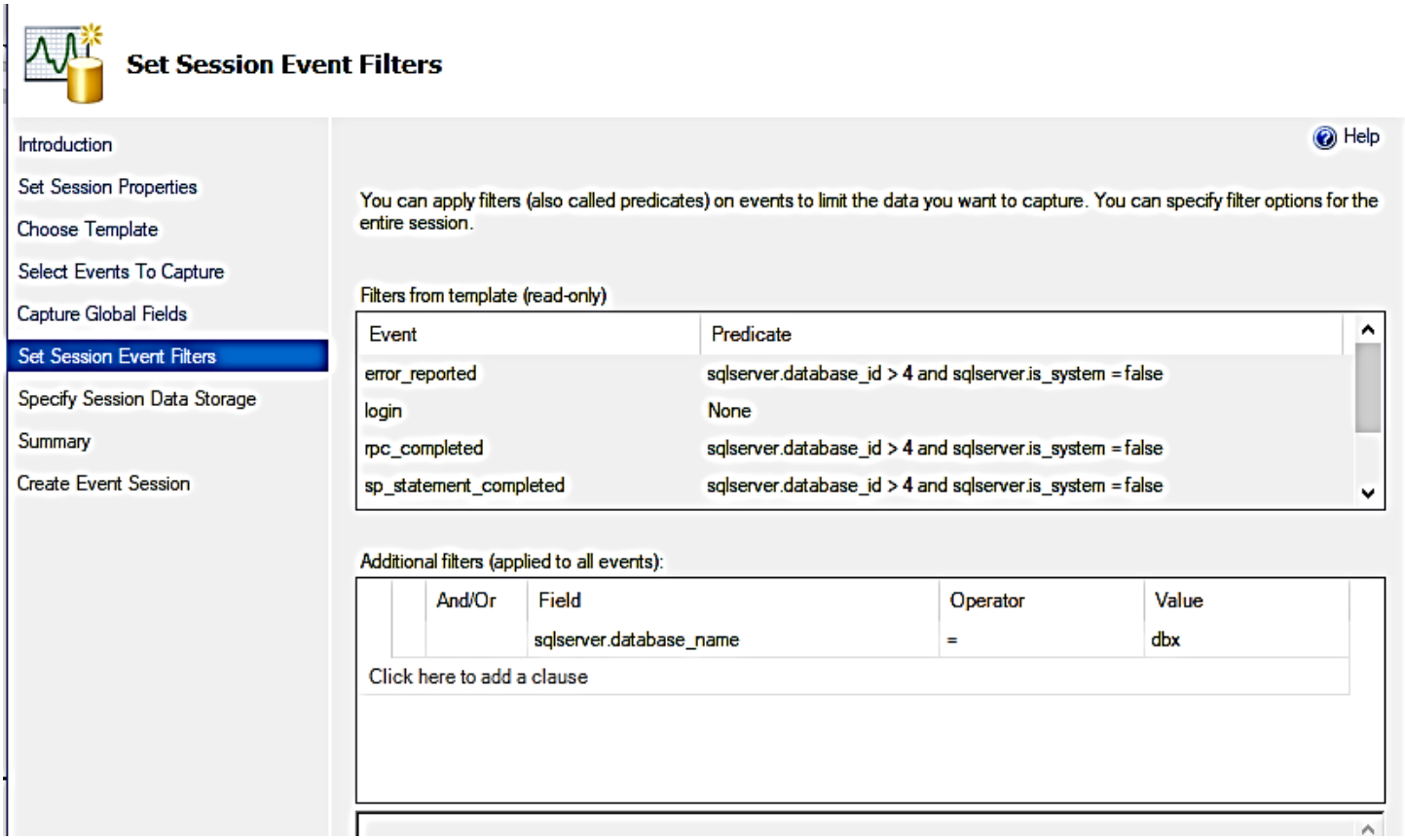
17. The Set Session Event Filters page allows you to limit the activity returned by the session. In this case, you'll add a filter to show only activity in your database.

In the white text box located in the Additional Filters (Applied to All Events) section, click the Click Here to Add a Clause area.

18. In the Field drop-down box, select sqlserver.database_name.

19. You can change the selection in the Operator drop-down to any of a variety of comparison choices. For now, leave this selection as equals (=).

20. Type your database name in the text box under Value.
This will filter session data to only include events where the request's database_name equals your database name. You could just as easily have set up a filter on the database_id of your database, the user name, or the client host_name.

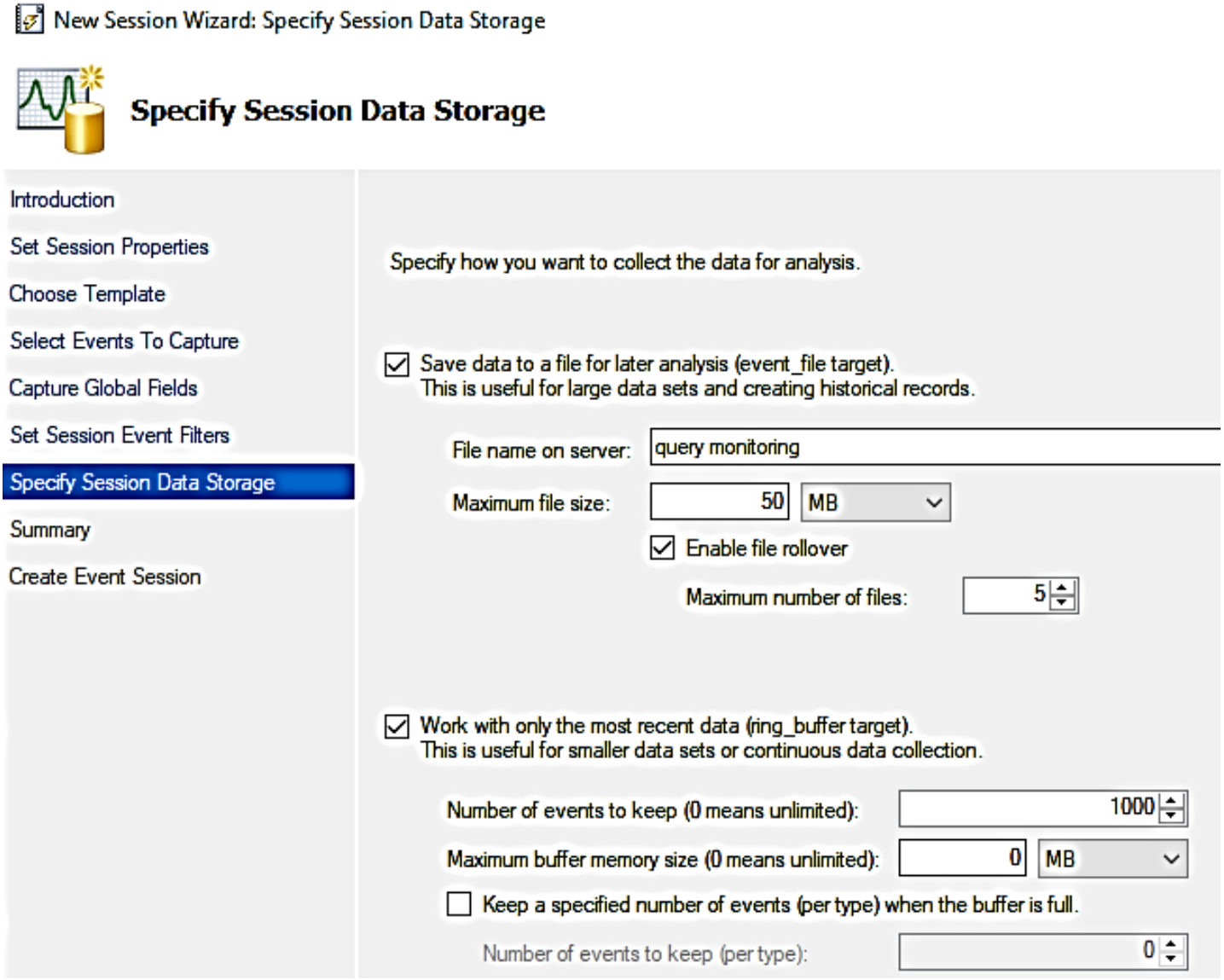


21. Click Next.
On the Specify Session Data Storage page, you'll see two of the three primary targets. The ring_buffer Target check box is enabled by default—this allows for continuous, rolling, in memory storage of event data. However, the fastest and most versatile target is event_file.

22. Select the box next to Save Data to a File for Later Analysis, and then clear the box next to Work with Only the Most Recent Data.

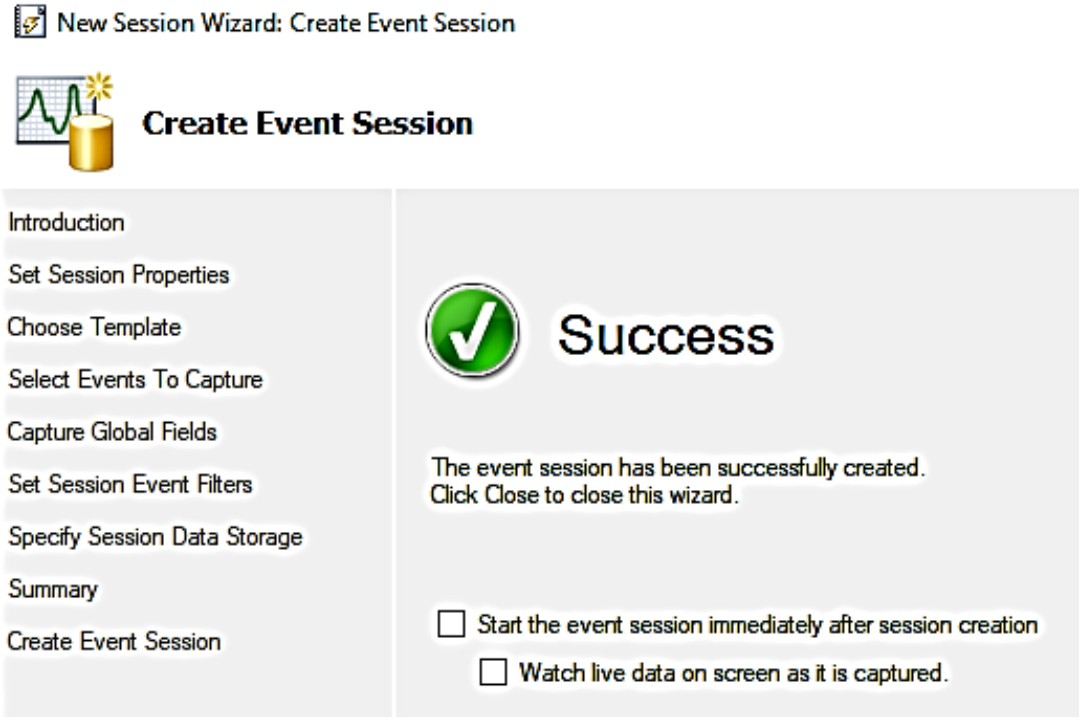
By default, the location of event_file is <instance path>\MSSQL\Log\Query Monitoring.xel. You can also limit the file size and have the file roll over to prevent one file from becoming too large. This behavior is very similar to the SQL Server Profiler setting with the same name.

23. Change the Maximum File Size setting to 50 MB, and then clear the Enable File Rollover option.



24. Click Next and then click Finish.
25. On the Create Event Session page, despite the appearance of a large green circle with a check mark in it, you're not done yet. The session has not yet been created.

26. Select the Start the Event Session Immediately After Session Creation check box.



27. Select the Watch Live Data on Screen As It Is Captured check box.

This is the third main way to access data, and it will open a new screen for the SSMS streaming data provider. This is the easiest way to access data, but it should be used only briefly—it also has the most overhead of the three Extended Events targets.

28. Click Close.

Use Extended events session to monitor activity

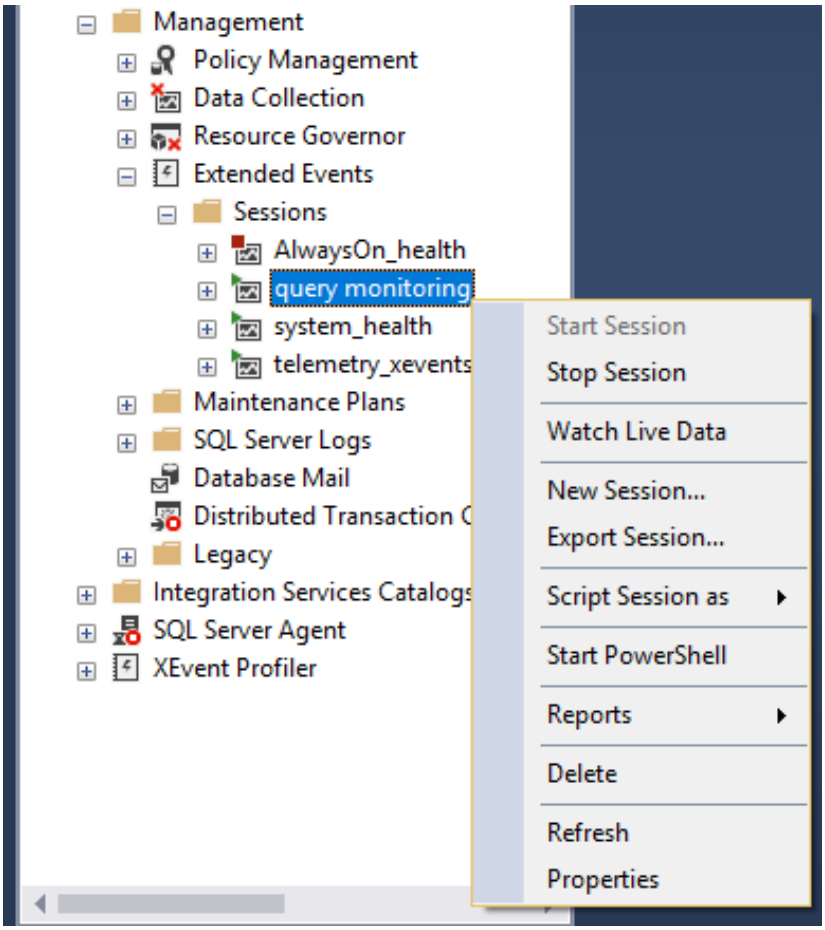
You should see the Query Monitoring: Live Data window is already running, and you may see it showing some activity in your database. To generate some activity, you'll run a simple SELECT statement.

1. In Object Explorer, expand the Databases folder.
2. Right-click your database, and then click New Query. Execute the following query:

```
USE <your database>;
select * from [your table].[TransactionHistory]
```

3. In SSMS, click the Query Monitoring: Live Data tab.

You should see a set of events just generated by yourself. Because this buffer reads the data asynchronously, there may be as much as a five-second delay.



4. Locate the record where the Name column is sql_batch_completed, and click that record.

Information about that event will appear in the Details tab.

5. On the Details tab, locate the row with a Field value of batch_text. This will show the query you executed in the other tab, as it was captured by the Extended Events session.

6. Close the Query Monitoring: Live Data tab. Note that this does not stop the Extended Events session; it is still running on SQL Server.

Now you'll stop the Extended Events session back in Object Explorer.

7. If it's not already open, expand the server folder, expand Management, expand Extended Events, and expand Sessions. Right-click the Query Monitoring session and then click Stop Session.

Note that the icon changes from a green arrow pointing right to a red arrow pointing down.

8. Navigate to the operating system folder where you stored the.xel file, which should be <instance path>\MSSQL\Log\.

Note that the actual file name has a uniquely identifying string appended to it—for example, Query Monitoring_0_129993306334000000.xel.

9. Double-click the file. A new SSMS window opens with this file in the tab.

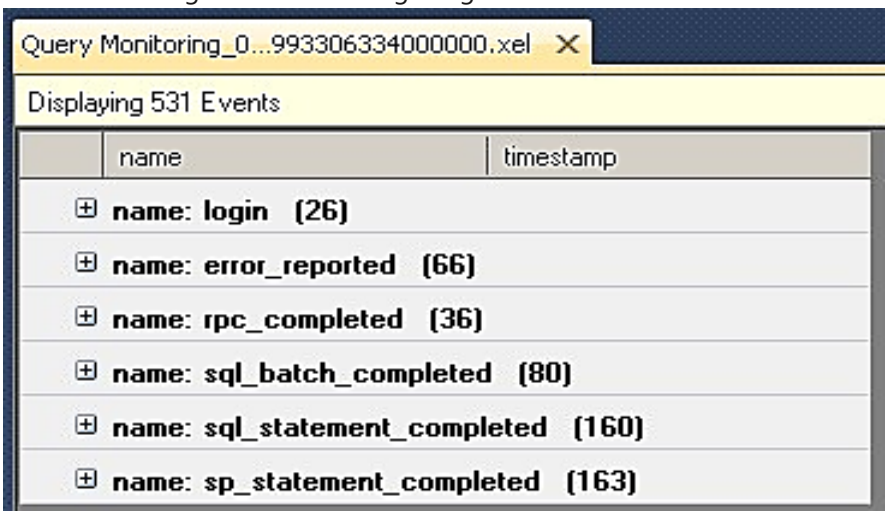
This interface is similar to the Live Data tab you saw earlier, and you can view the same details for each recorded event.

10. In the SSMS toolbar, click the Grouping button. The Grouping dialog box opens.

11. Under Available Columns, click Name, and then click the > button to move Name to the Columns Grouped On window.

12. Click OK.

On a lengthy Extended Events session, grouping the event name will offer additional insight into the recorded events, which would look something like the following image.



13. On the toolbar located above the tab, click the Filters button.
14. In the Filters window, click the Set Time filter.
15. Use the sliders to restrict the current display of events to a time frame precise to the second. You can add other filters here to help you make sense of a large Extended Events event_file data set.
16. Click OK to accept your settings and to close the Filters window.
17. On the toolbar located above the tab, click the Choose Columns button.

In the Choose Columns window, you can add more fields to the dataset.

18. Locate database_name, duration, batch_text, and session_id in the Available Columns screen, and use the > arrow to move each to the Selected Columns window. Use the up and down arrows to arrange the columns in any order.
19. Click OK.
20. Back in the Query Monitoring tab, expand the sql_batch_completed group.

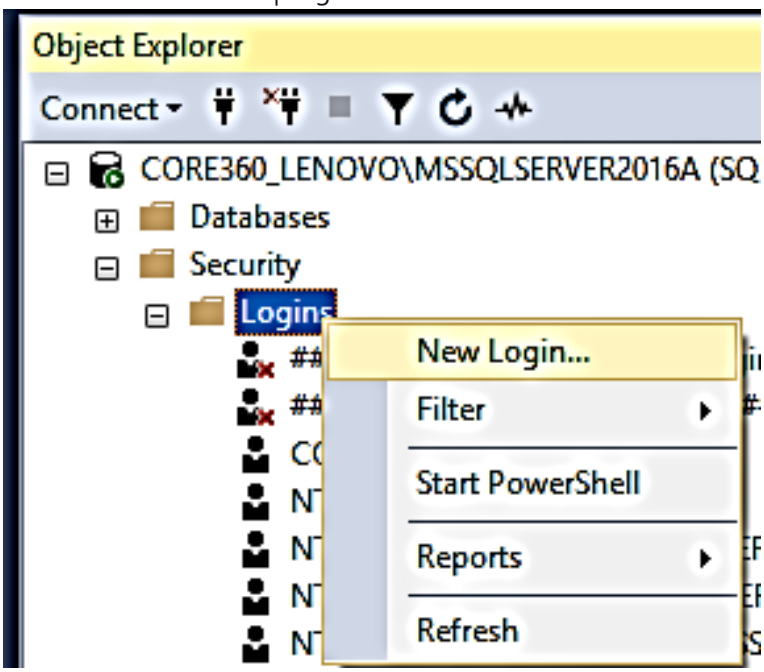
You can now see more data for each event.

3. SQL Server security

Creating server logins

1. Open SQL Server Management Studio (SSMS) and connect to a server.
2. Open Object Explorer if it is not already open.
3. Expand the server tree.

4. Right-click the Security folder and select New | Login.



5. On the General page, enter a Windows account in the Login Name text box. You must enter the account in the following format: domain\username (if using active directory). For example, if the Windows account alias is jdoe, you will enter domain\jdoe. Alternatively, you can click the Search button to find the Windows entity that you want to enter.

6. Since this is a Windows-based account, ensure that the Windows Authentication option is selected.
If this was a SQL Server login, you would select the SQL Server Authentication option. With this option selected, a password must be entered and confirmed using the Password and Confirm Password text boxes.

7. You have the choice to enforce Windows Active Directory password polices and password expiration policies on this account by selecting the Enforce Password Policy and Enforce Password Expiration check boxes. This means that whatever polices have been configured for passwords and password expirations by your Active Directory administrator will be enforced for this SQL Server login.

8. Finally, you can specify that the password must be changed when the account is initially used by selecting the User Must Change Password at Next Login option. For now, ensure that the Windows Authentication option is selected.

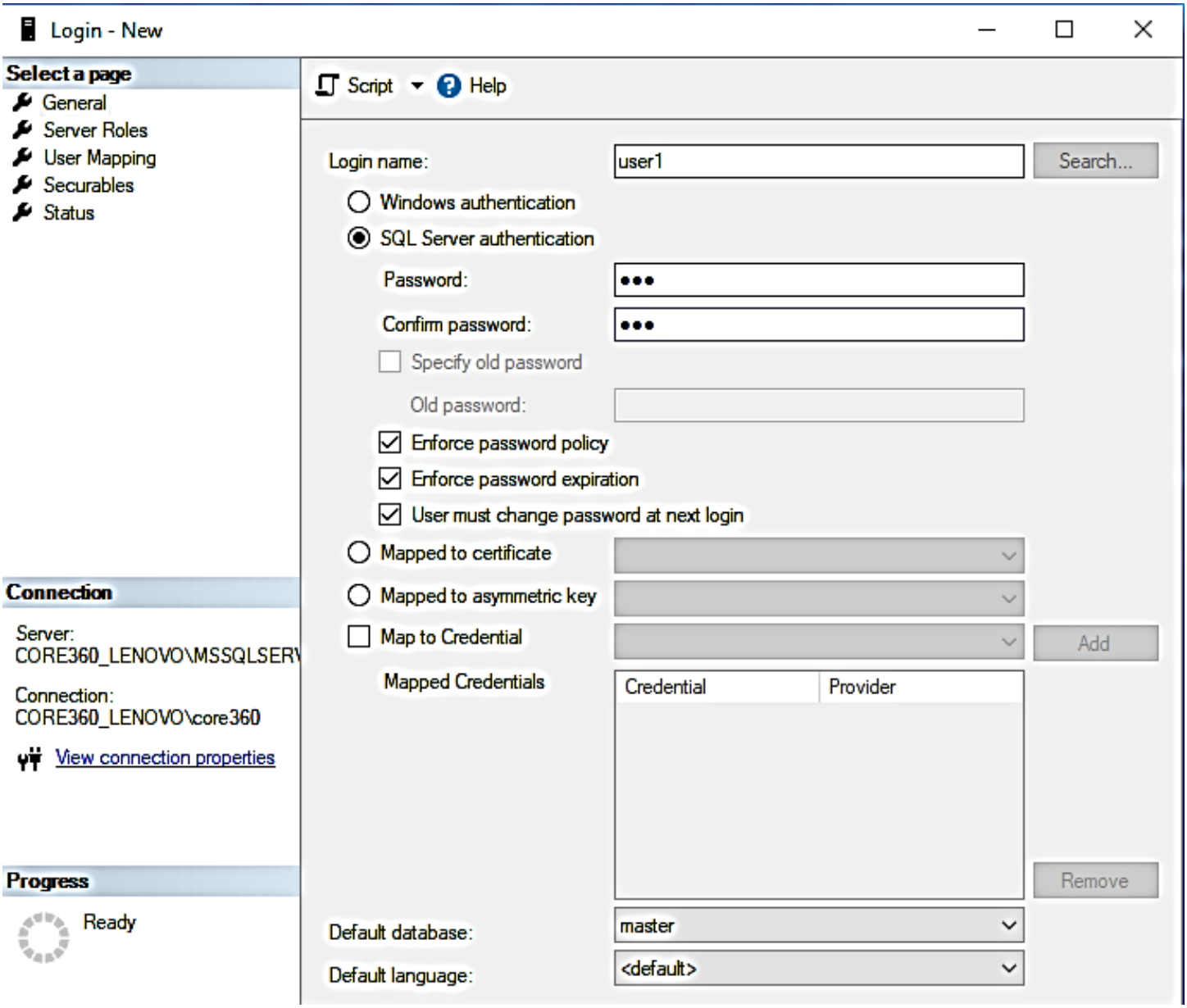
9. Leave the next options, Mapped to Certificate and Mapped to Asymmetric Key, cleared. You can select only one or the other.

10. Selecting the Map to Credential check box allows you to map this login to the credential of another account. This option allows a SQL Server login to access resources external to SQL Server under the context of the login specified in the credential. For now, leave the box cleared.

11. When creating an account that will have database-level access, you should specify a default database using the Default Database drop-down list. When first authenticated to the SQL Server instance, a new connection will be in the default database's context. For example, if a new query window is opened by SSMS, the current database displayed will be the default database. If this is a login that will be used for data access, as a best practice always select a user-

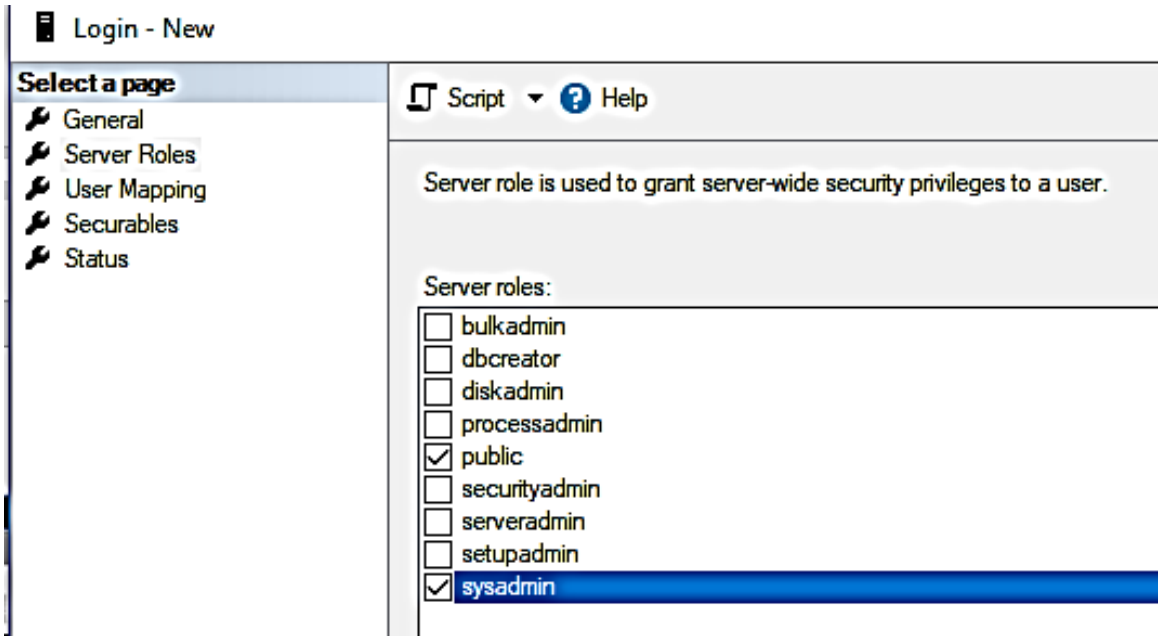
created database and not a system database. Assume the person who owns this account will be the DBA for this instance, so the default database can remain master.

12. The final drop-down list, Default Language, by default uses the default language configured on the server. Alternatively, you can select a language. For the purposes of this exercise, use the default.



13. In the Select a Page pane, select Server Roles.

14. A list of built-in server roles is displayed on this page. The public role is selected by default. A description of each server role is provided in the "Creating user-defined server roles" section of this chapter. For now, select the box next to the sysadmin server role, which allows the system administrator to execute any task against the server.



15. In the Select a Page pane, select User Mapping.
16. If you want to explicitly grant this user access to a specific database, you can select that database from the list displayed at the top of the page. Additionally, you can assign the user to built-in database roles.
17. In the Select a Page pane, select Securables.

This page lists items that can be secured and their corresponding permissions or the permissions that can be assigned to that login. Security is typically set at the server or database level.

However, SQL Server provides you with the ability to set a much finer level of security.

18. Click the Search button. You are presented with three choices:
    a. You can add access to specific server-level items or you can provide access to every item by choosing the The Server 'Your Server Name' option.
    b. If you select the Specific Objects option, you can add items of different types.
    c. If you select All Objects of the Types option, you can holistically grant permissions to all objects of a specific type.

19. For the purposes of this exercise, click Cancel.
20. In the Select a Page pane, select Status. On this page, you have the ability to grant or deny the login permission to connect to the server. Ensure that the Grant option is selected.
21. Ensure that the Enabled option is selected.
22. Finally, if this was a SQL Server account, you could lock out the account, just as you can with Active Directory. Click the OK button and the login will be created.

To create a Windows-based login using T-SQL

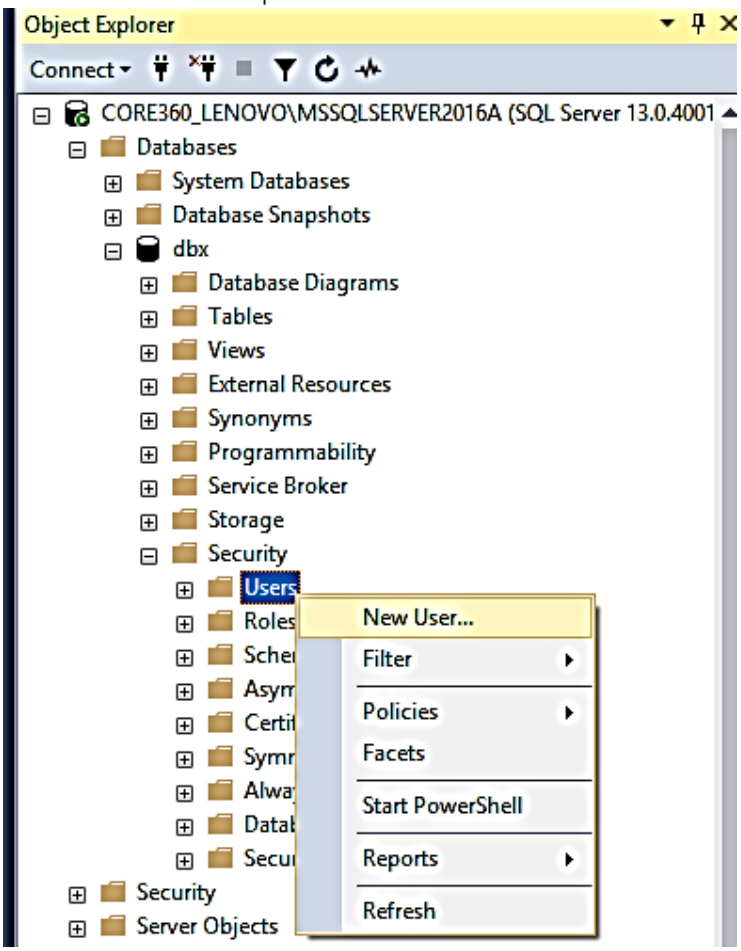
```
USE [master]
GO
CREATE LOGIN [DOMAIN\jdoe] FROM WINDOWS WITH DEFAULT_DATABASE=[master]
GO
ALTER SERVER ROLE [sysadmin] ADD MEMBER [DOMAIN\jdoe] GO
```

To create a SQL-based login using T-SQL

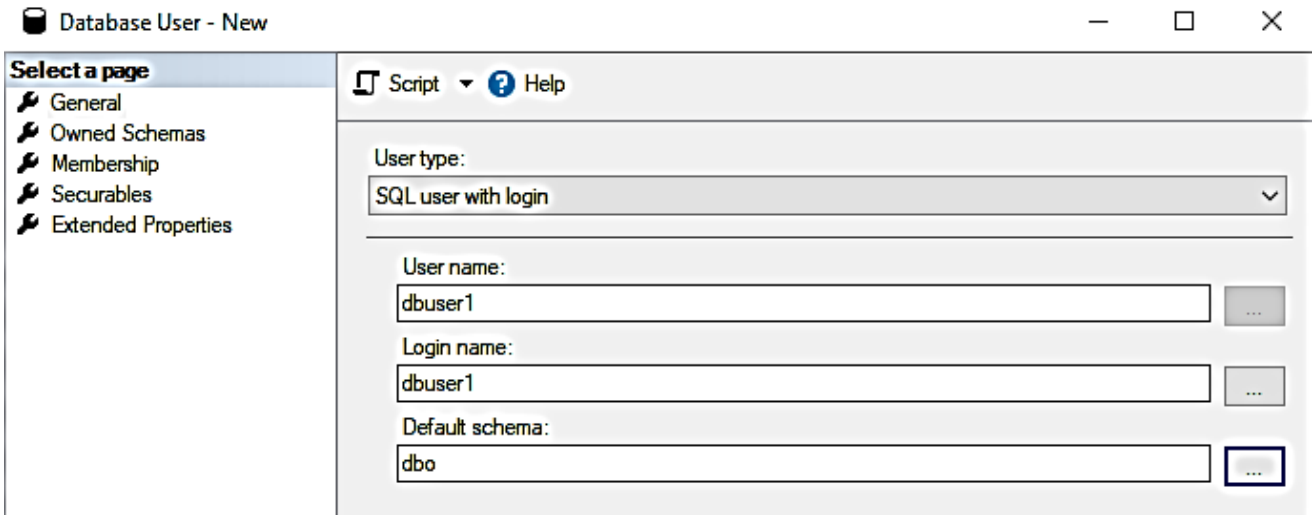
```
USE [master]
GO
CREATE LOGIN [JDOE] WITH PASSWORD=N'password'
    MUST_CHANGE,
    DEFAULT_DATABASE=[master],
    CHECK_EXPIRATION=ON,
    CHECK_POLICY=ON
GO
ALTER SERVER ROLE [sysadmin] ADD MEMBER [JDOE] GO
```

Creating database users

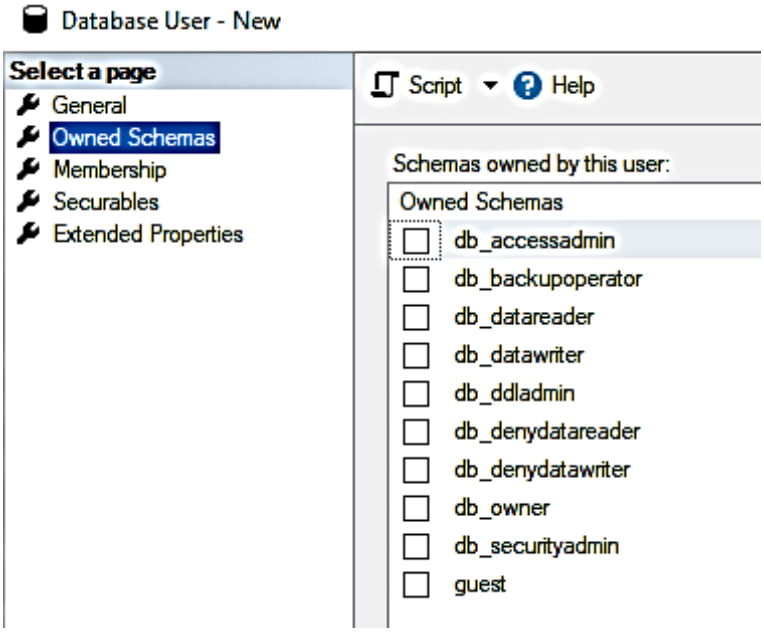
1. Open SSMS and connect to a server.
2. Open Object Explorer if it is not already open.
3. Expand the server tree.
4. Expand the Databases folder.
5. Expand your database.
6. Right-click the Security folder and select New | User from the menu. The Database User dialog box opens.



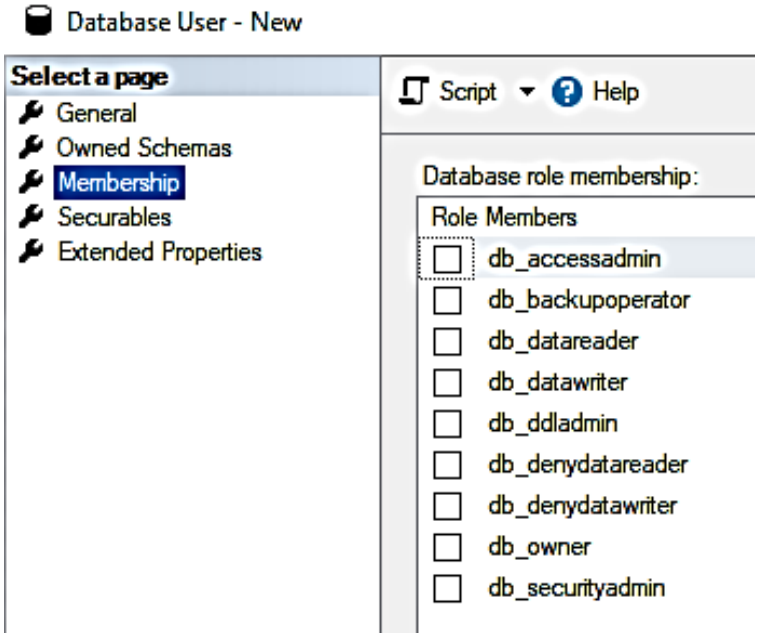
7. In the User drop-down list, select Windows User.
8. In the User Name and Login Name text box, enter dbuser1.
9. In the Default Schema text box, enter dbo.



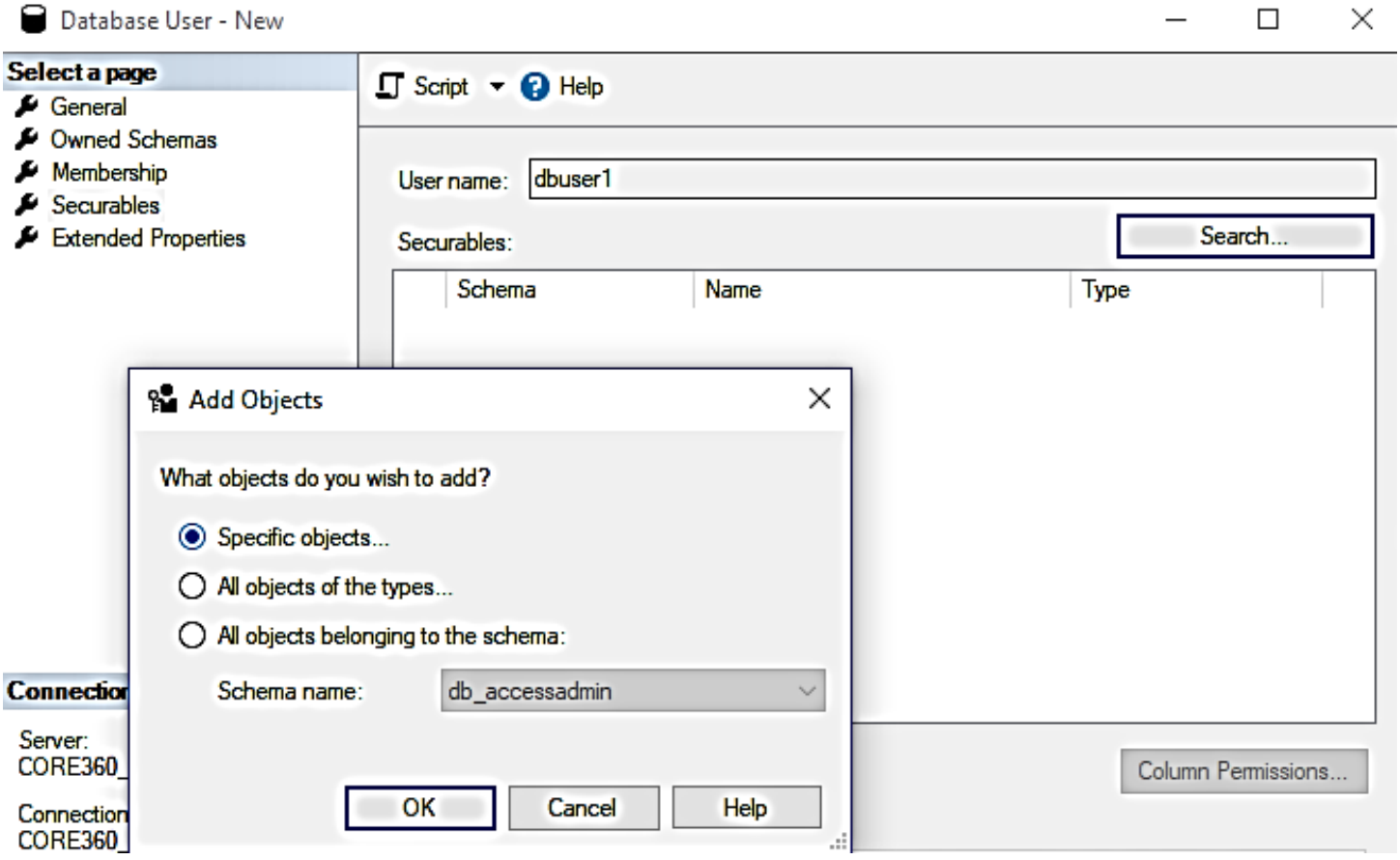
10. In the Select a Page pane, select Owned Schemas. Specifying a user as an owner of a schema gives that user full control. Do not select any schemas.



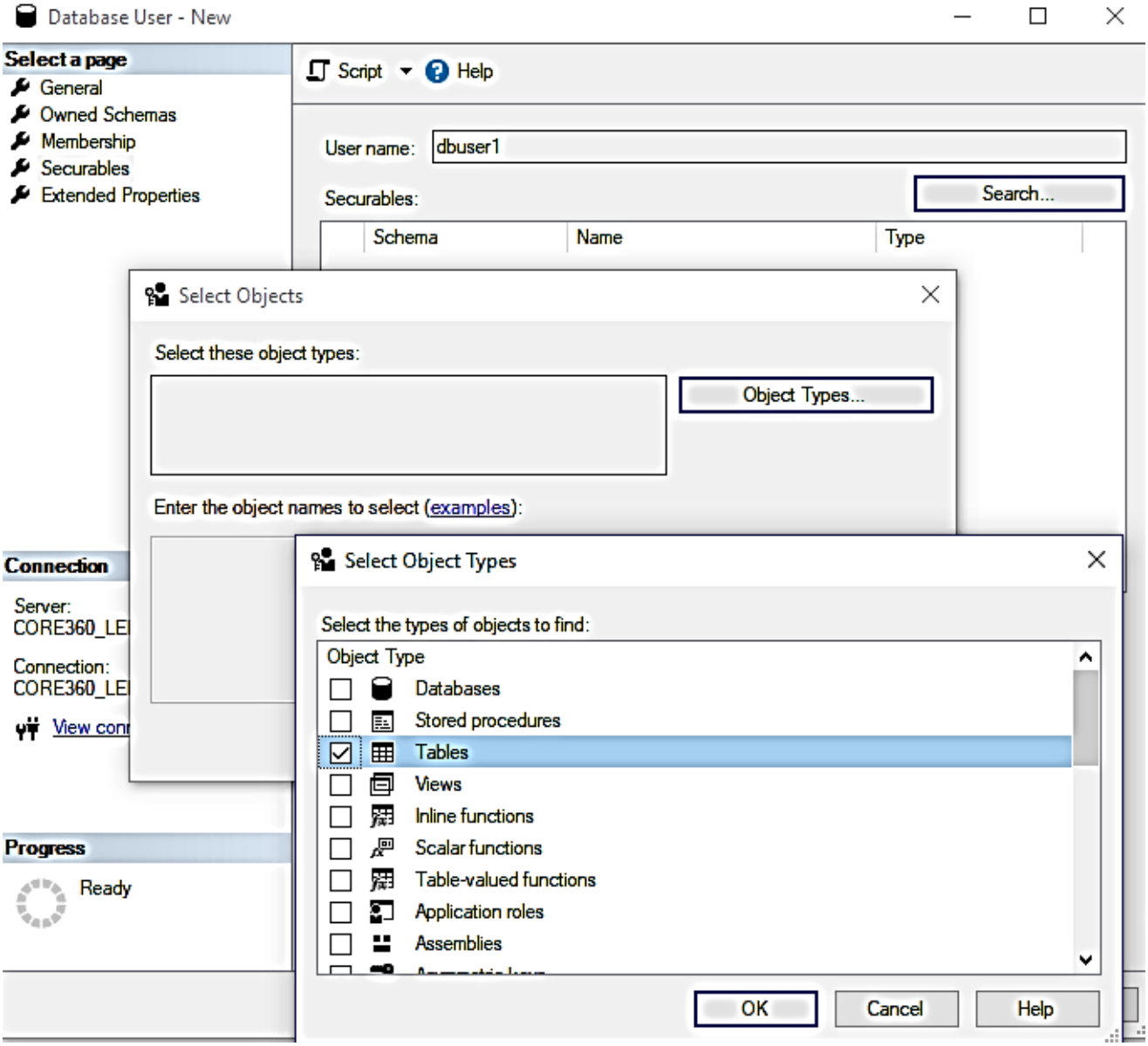
11. In the Select a Page pane, select Membership. Selecting a role from the list grants the user all the permissions of that role. Do not select any roles.



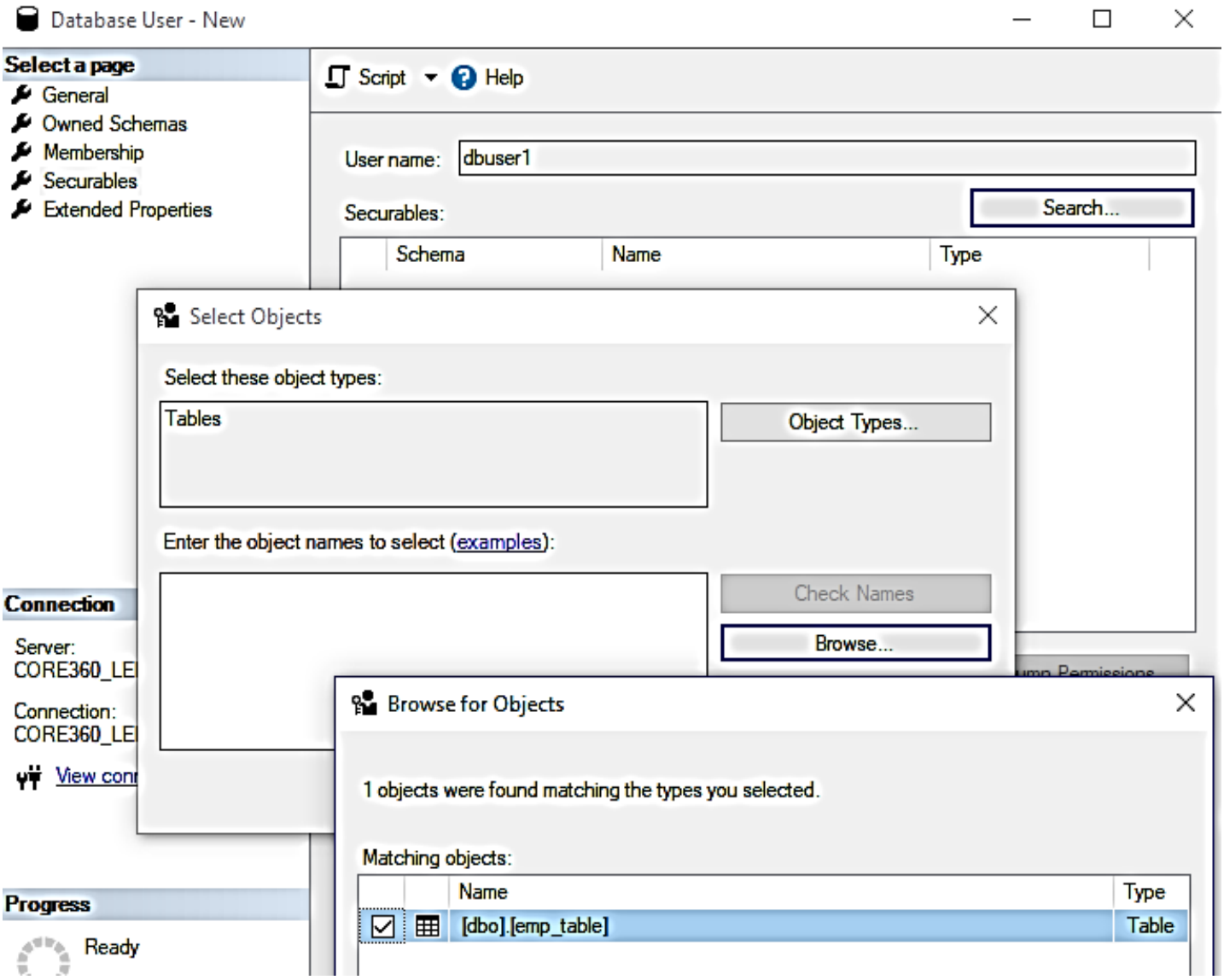
12. In the Select a Page pane, select Securables.
13. On the Securables page, click the Search button. Very similar to when you created a database user, you can grant or deny access at the same levels.
14. With the Specific Objects option selected, click OK.



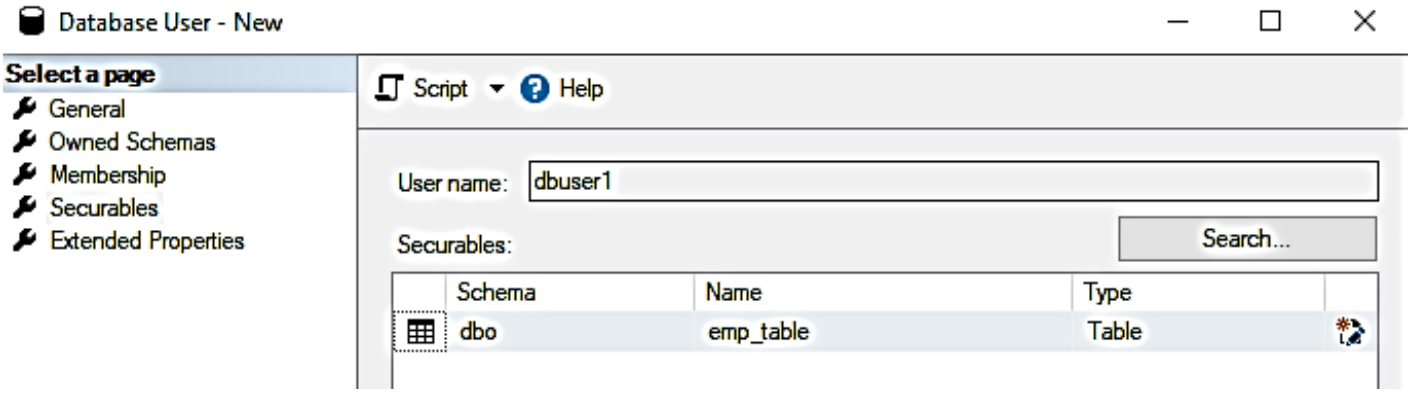
15. In the Select Objects dialog box, click the Object Types button.
16. In the Select Object Types dialog box, select Tables and click OK.

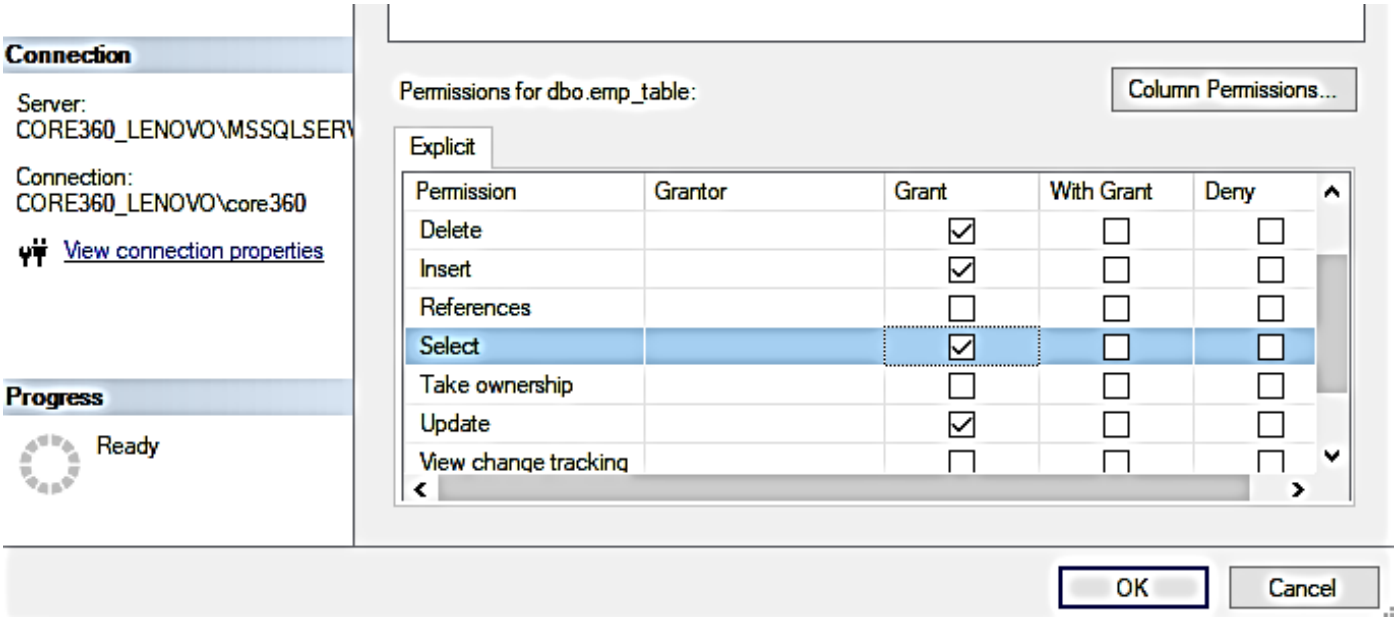


17. Click the Browse button.
18. Select the boxes next to the tables and click OK twice.



19. In the Securables section, select the table.
20. In the Explicit section, select the boxes in the Grant section for the following permissions: Delete, Insert, Select, and Update.





To create a user using T-SQL

```
USE [AdventureWorks2012]
GO
CREATE USER [domaim\jdoe] FOR LOGIN [domaim\jdoe] WITH DEFAULT_SCHEMA=[dbo] GO
use [AdventureWorks2012]
GO
GRANT DELETE ON [dbo].[DatabaseLog] TO [domaim\jdoe] GO
use [AdventureWorks2012]
GO
GRANT INSERT ON [dbo].[DatabaseLog] TO [domaim\jdoe]
GO
```

```
use [AdventureWorks2012]
GO
GRANT SELECT ON [dbo].[DatabaseLog] TO [domaim\jdoe] GO
use [AdventureWorks2012]
GO
GRANT UPDATE ON [dbo].[DatabaseLog] TO [domaim\jdoe] GO
use [AdventureWorks2012]
GO
GRANT DELETE ON [dbo].[AWBuildVersion] TO [domaim\jdoe] GO
use [AdventureWorks2012]
GO
GRANT INSERT ON [dbo].[AWBuildVersion] TO [domaim\jdoe] GO
use [AdventureWorks2012]
GO
GRANT SELECT ON [dbo].[AWBuildVersion] TO [domaim\jdoe] GO
```

```
use [AdventureWorks2012]
GO
GRANT UPDATE ON [dbo].[AWBuildVersion] TO [domaim\jdoe]
GO
```

To deny permission, replace the GRANT keyword with DENY. Also, if you want to not GRANT or DENY, you can replace either with REVOKE. Remember that DENY will take precedence over GRANT.
To remove an existing GRANT or DENY permission on an object, use REVOKE in a T-SQL script.

4. Resource Governor

* By using Resource Governor, you can create a workload group mapped to a resource pool that governs how the resources such as CPU and memory are allocated.

To enable Resource Governor using T-SQL, execute the following query:

```
USE [master]
GO
ALTER RESOURCE GOVERNOR RECONFIGURE;
GO
```

To disable Resource Governor using T-SQL, execute the following query:

```
USE [master]
GO
ALTER RESOURCE GOVERNOR DISABLE;
GO
```

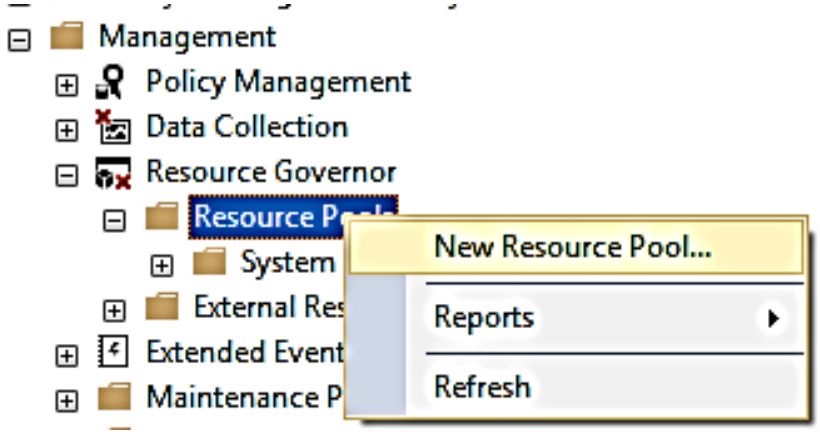
Creating Resource Pools

A resource pool represents the amount of system resources (memory or CPU) available to the server. Two predefined pools exist by default:

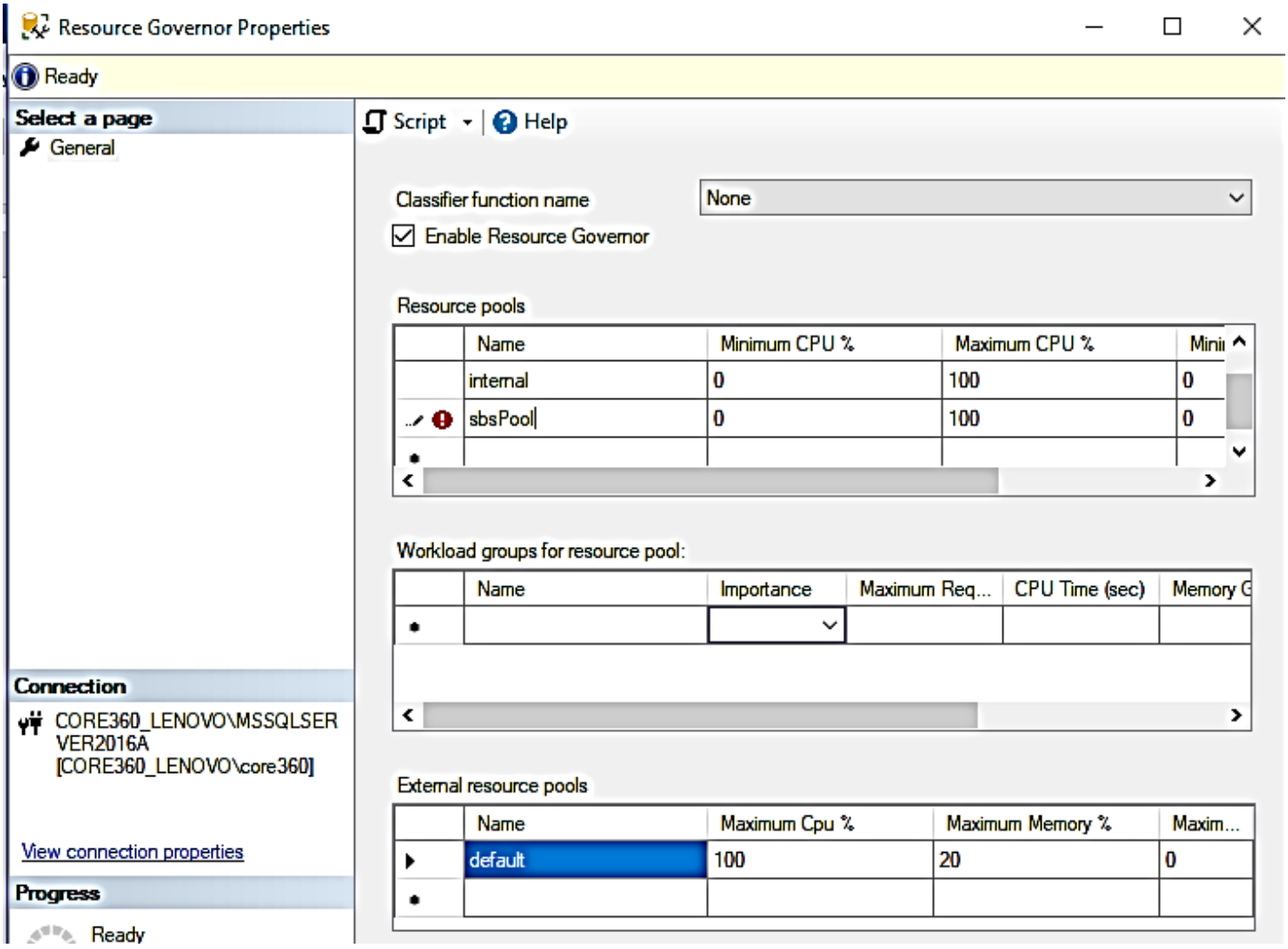
- The internal pool cannot be dropped or altered. It represents the pool that critical SQL Server functions workloads are placed in, and it has priority over all other pools.
- The default pool is a user-defined pool that cannot be dropped, but it can be altered. In addition, user-defined workload groups can be mapped to it.

Creating a resource pool is the first step in configuring Resource Governor to control workload resource usage. When configuring a resource pool, you can specify the minimum and maximum server memory and CPU.

1. Open SSMS and connect to a server.
2. Open Object Explorer if it is not already open.
3. Expand the server tree.
4. Expand the Management folder.
5. Expand Resource Governor.
6. Right-click the Resource Pools folder and select New Resource Pool.

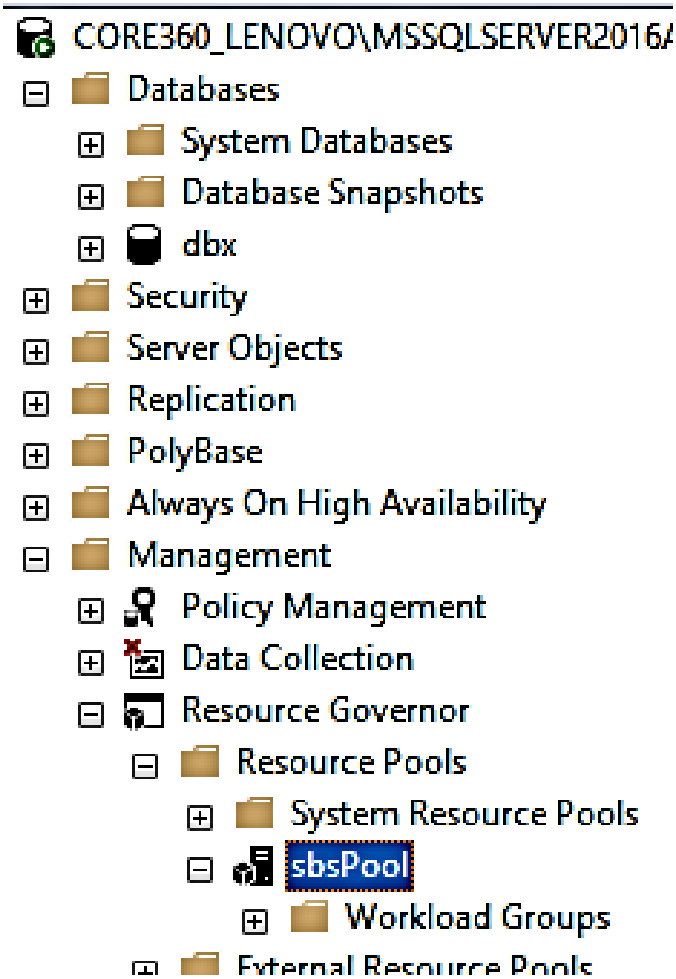


7. The Resource Governor Properties dialog box opens.
8. In the Resource Pools table, click in the column labeled Name on the row with the red-highlighted exclamation point.
9. In the Name column, type sbsPool.



10. In the Minimum CPU% column of the same row, enter 20.
11. In the Maximum CPU% column of the same row, enter 50.

12. In the Minimum Memory % column of the same row, enter 20.
13. In the Maximum Memory % column of the same row, enter 50.
14. Click OK.
15. Right-click the Resource Pools folder and select Refresh.
16. Expand the Resource Pools folder and you will see the new pool, sbsPool.



To create a resource pool using T-SQL, execute the following query:

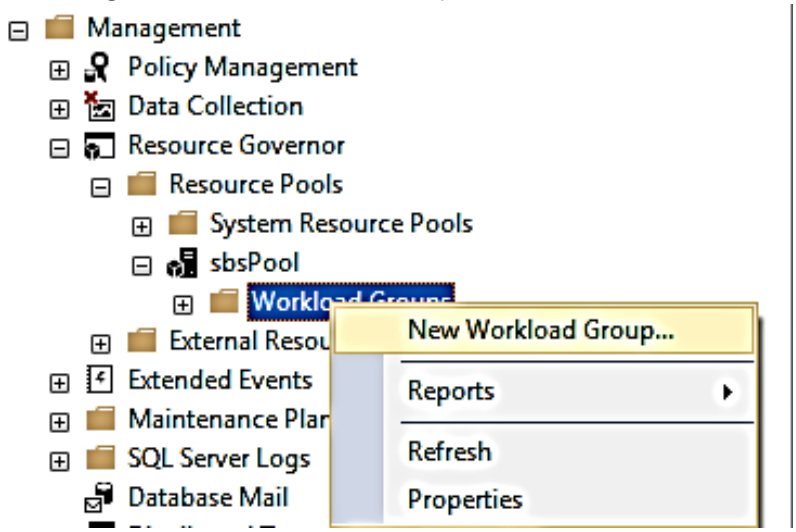
```
Use master;
CREATE RESOURCE POOL [sbsPool] WITH(min_cpu_percent=20,          max_cpu_percent=50,
min_memory_percent=20,          max_memory_percent=50)
GO
ALTER RESOURCE GOVERNOR RECONFIGURE;
GO
```

Creating a workload group

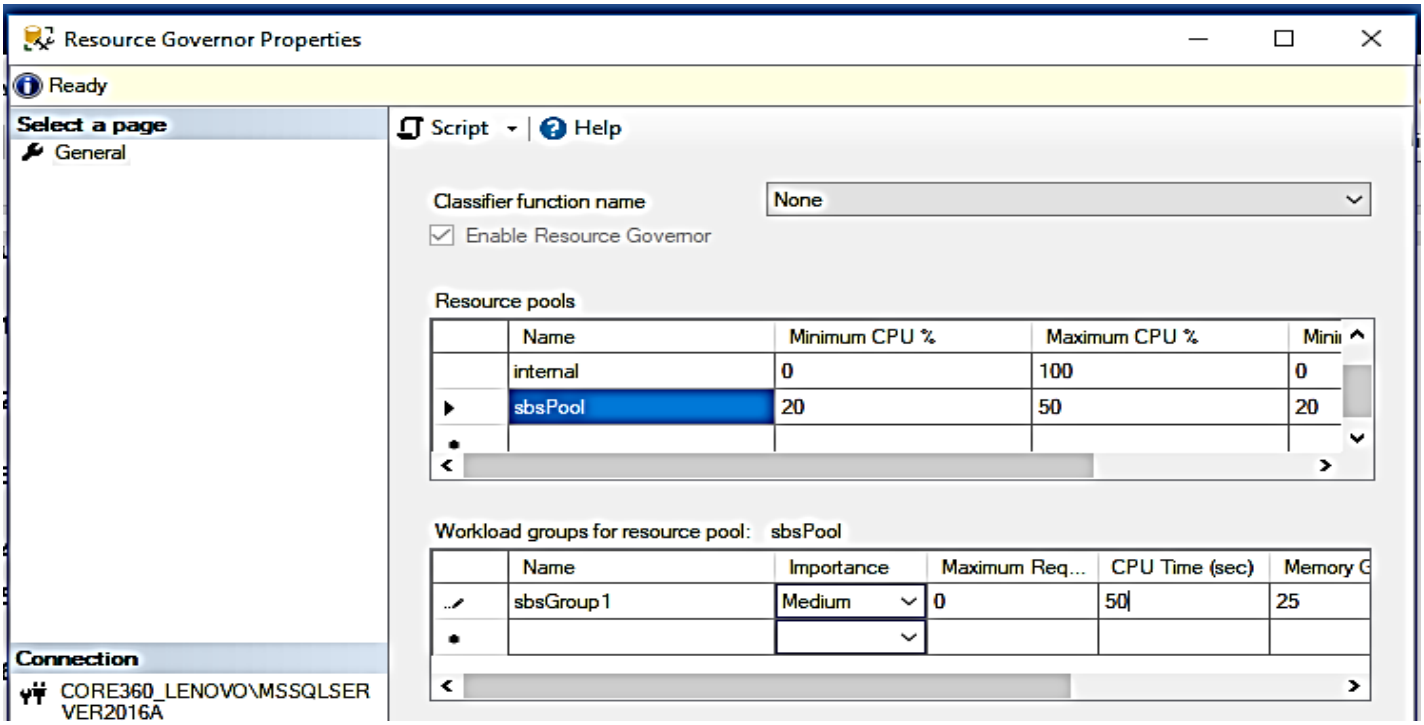
After the pool is created, the next step is to create a workload that will be mapped to that pool.

Argument	Description	Default value
importance	Request the level of importance relative to all requests. The value can be High, Medium, or Low.	Medium
request_max_memory_grant_percent	Maximum amount of memory provided to a given request relative to the group. Must be greater than zero. The memory grant should not be changed, and note that it is for a single query in a group.	25
request_max_memory_grant_timeout_sec	Amount of time a request can wait for memory.	0
request_max_cpu_time_sec	Maximum amount of CPU time provided to a given request relative to a group. A zero value means it defaults to the global maximum degree of parallelism.	Global value
max_dop	Maximum degree of parallelism for each request.	0
group_max_requests	Maximum number of concurrent requests that are permitted to execute within a workgroup.	0 (unlimited)

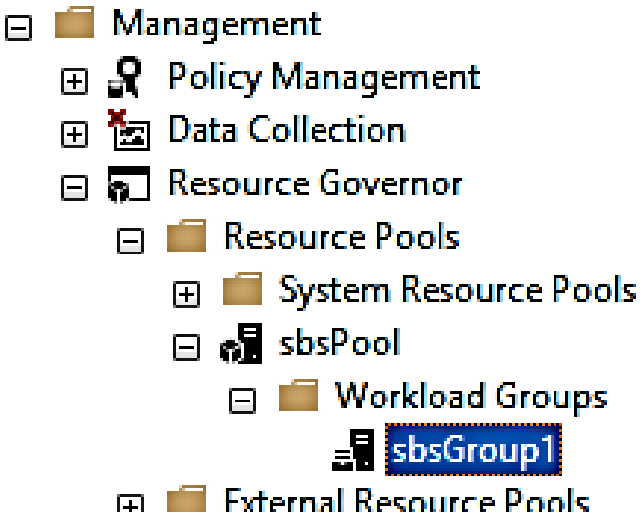
1. Open SSMS and connect to a server.
2. Open Object Explorer if it is not already open.
3. Expand the server tree.
4. Expand the Management folder.
5. Expand Resource Governor.
6. Expand the Resource Pools folder.
7. Expand the sbsPool resource pool.
8. Right-click the Workload Groups folder and select New Workload Group from the menu.



9. In the Workload Groups for Resource Pool table, click in the column labeled Name on the row with the red-highlighted exclamation point.
10. In the Name column, type sbsGroup1.
11. In the Importance column, select Medium.
12. In the Maximum Requests column, accept 0.
13. In the CPU Time (Sec) column, enter 50.
14. In the Memory Grant % column, enter 25.
15. Accept the default for Grant Time-out (Sec) and Degree of Parallelism.
16. Click OK.



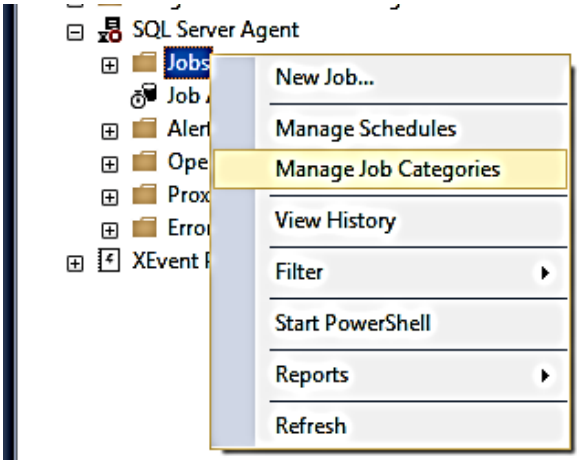
17. Right-click the Workload Groups folder and select Refresh. You will now see the new group.

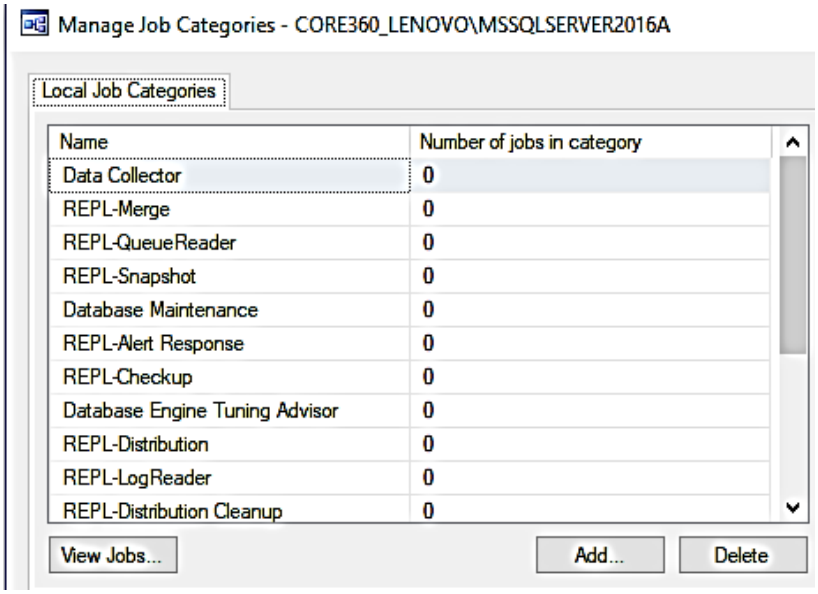


5. Scheduling jobs to perform maintenance activities using SQL Server Agent

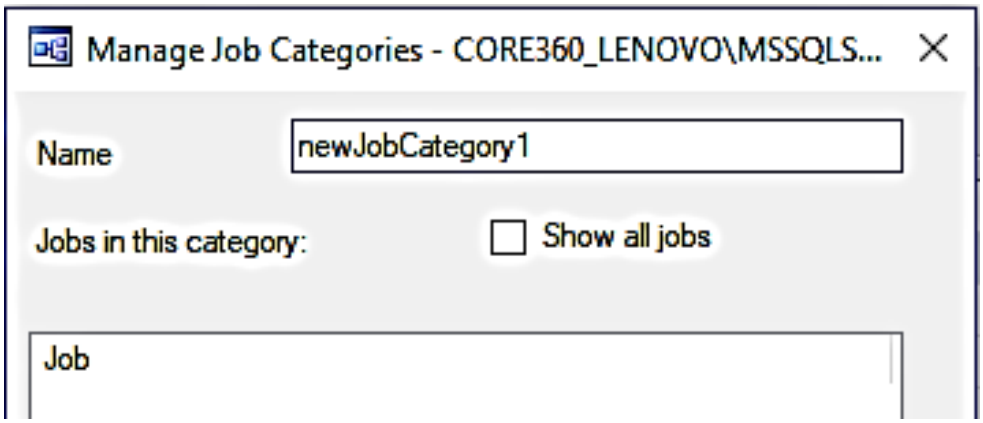
When you create a job, you can place it into a job category. Each job can be in only one category. Several predefined job categories are available, including [Uncategorized (Local)] and Database Engine Tuning Advisor. You can also create your own job categories by following these steps:

1. From the Object Explorer window of SQL Server Management Studio, open the SQL Server Agent item in the tree view, and right-click the Jobs node.
2. From the menu, select Manage Job Categories.





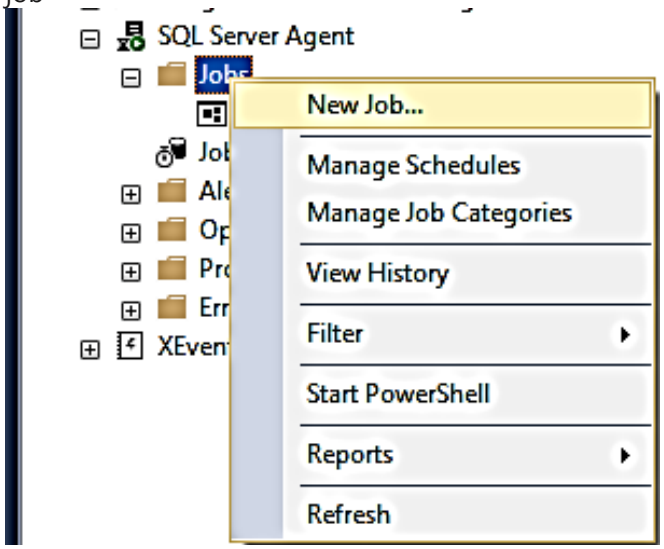
3. Click Add



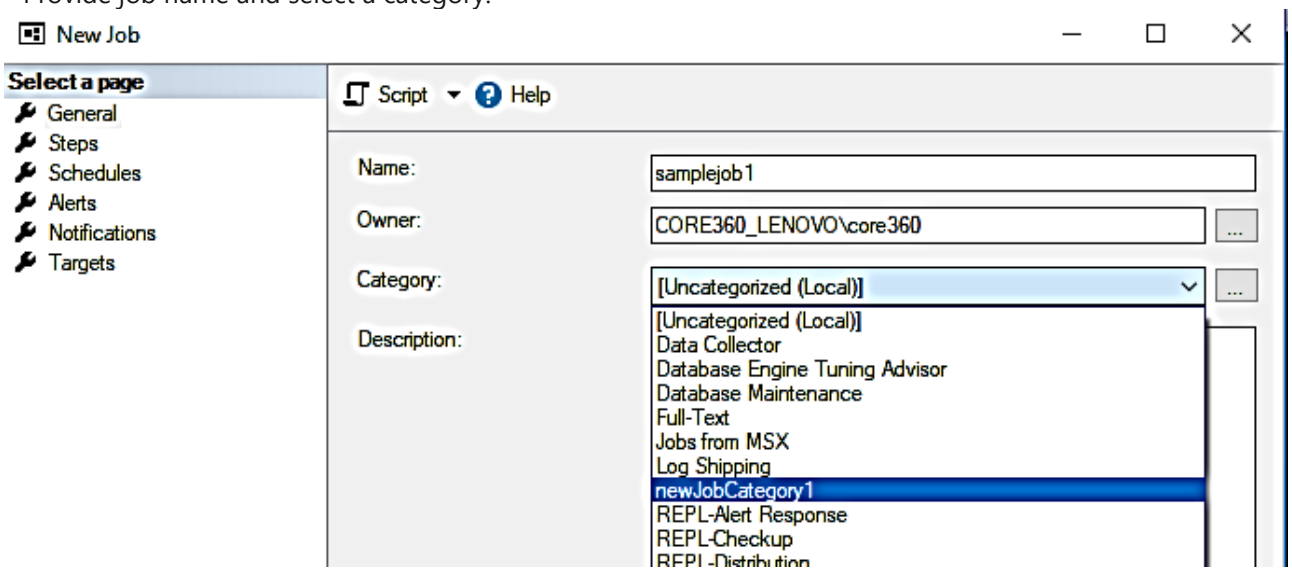
4. Enter the new job category name into the Name field, and click OK.

Creating a new Job

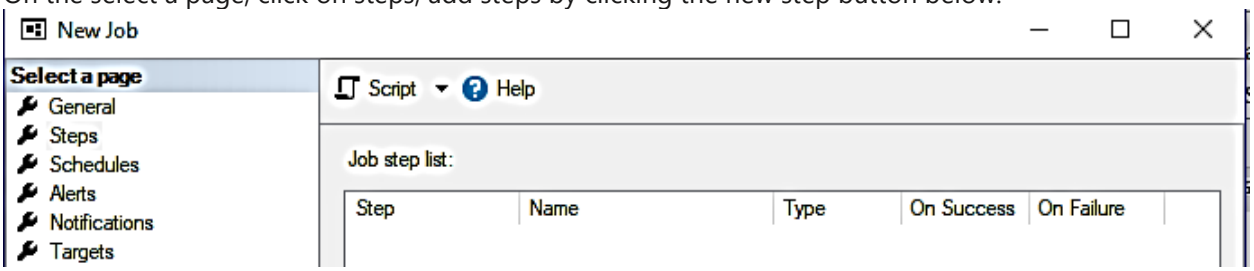
1. Rclick jobs folder → new job

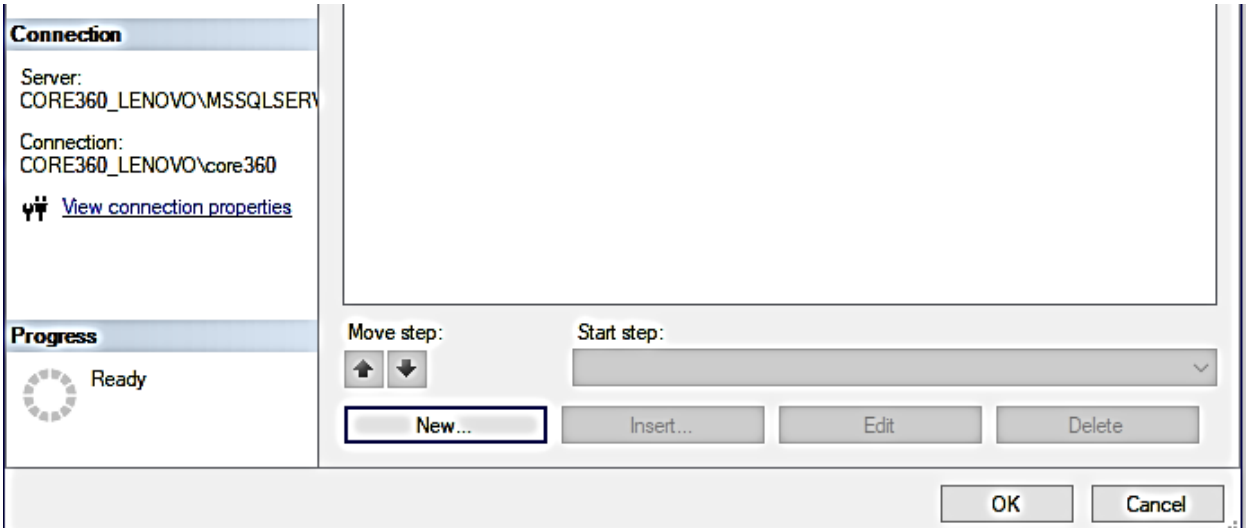


2. Provide job name and select a category.

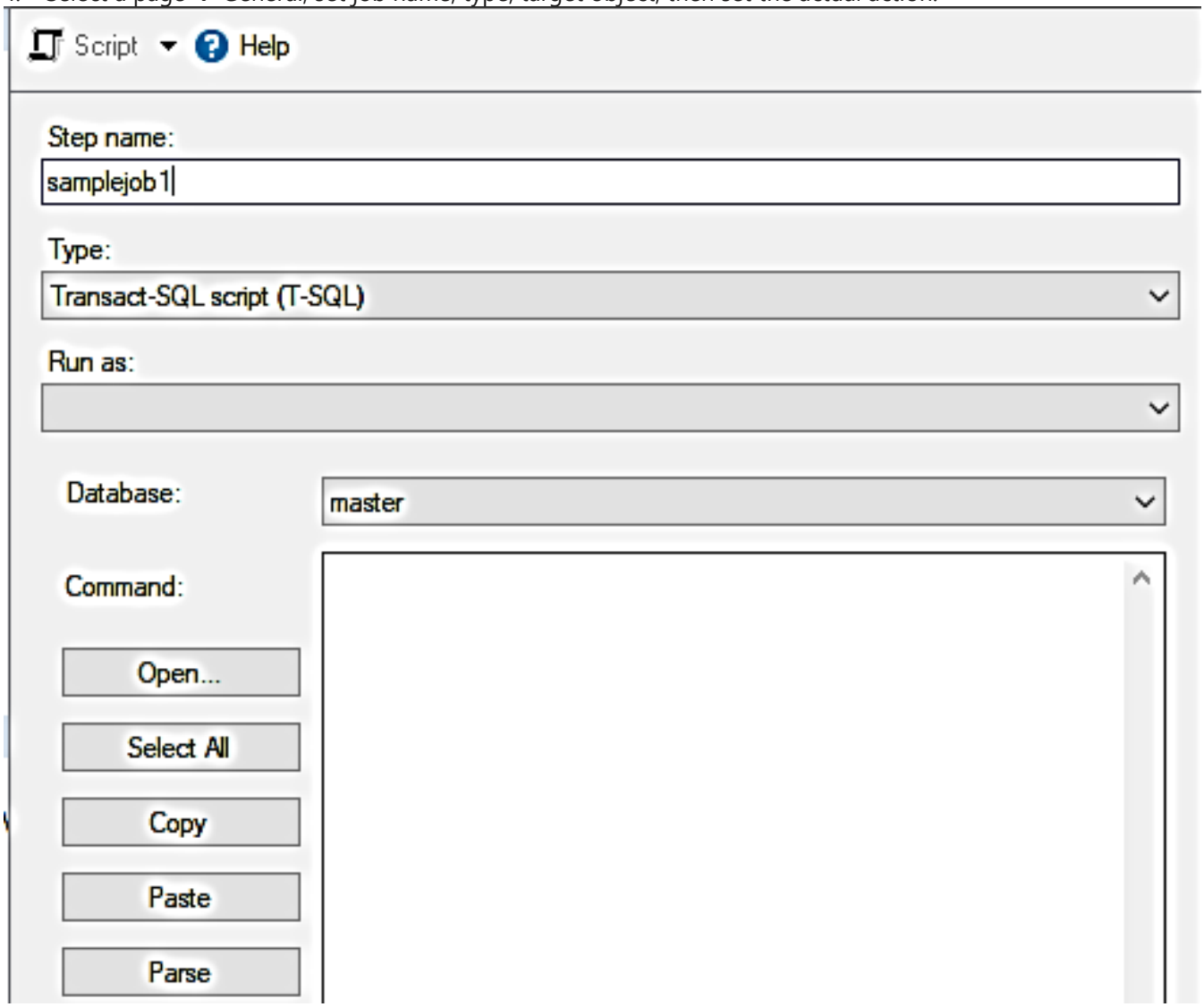


3. On the select a page, click on steps, add steps by clicking the new step button below.

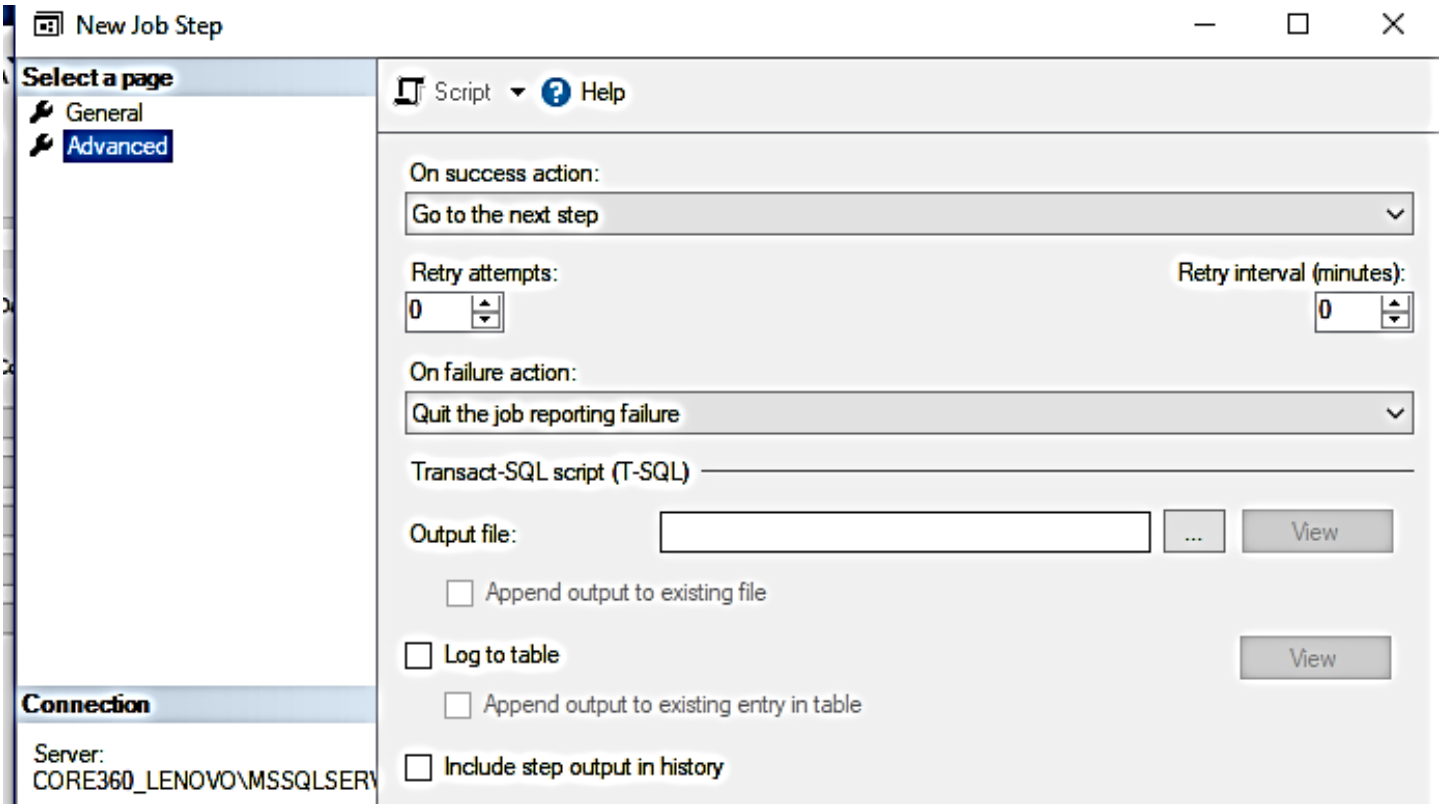




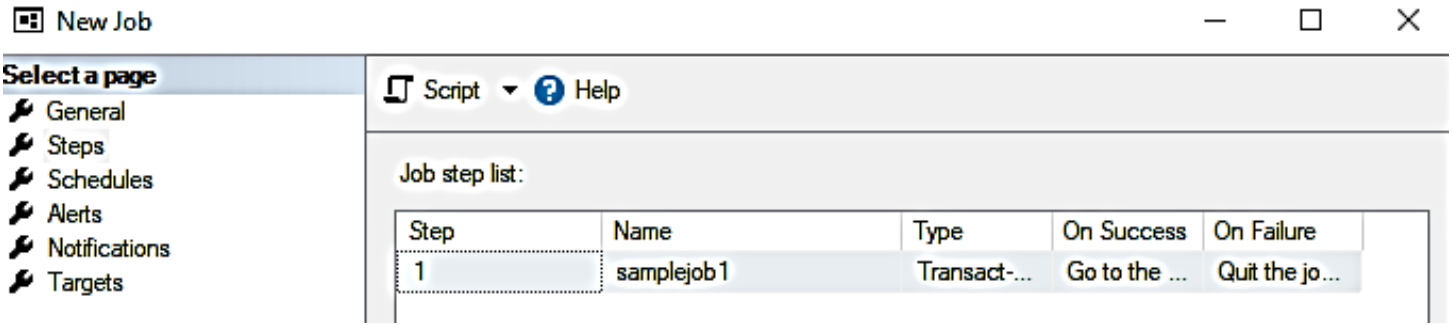
4. Select a page → General, set job name, type, target object, then set the actual action.



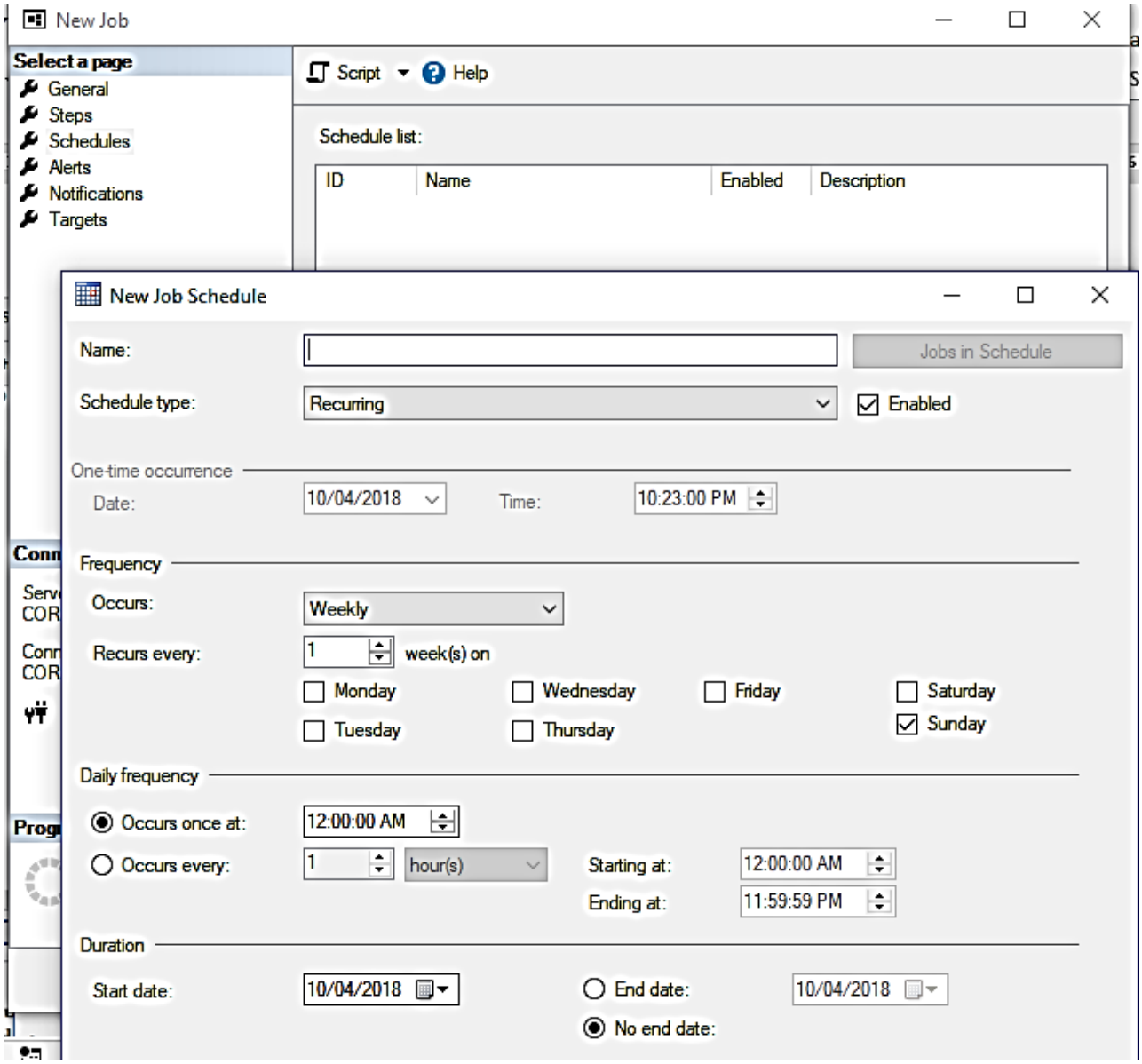
5. You can also set advanced settings for your steps.



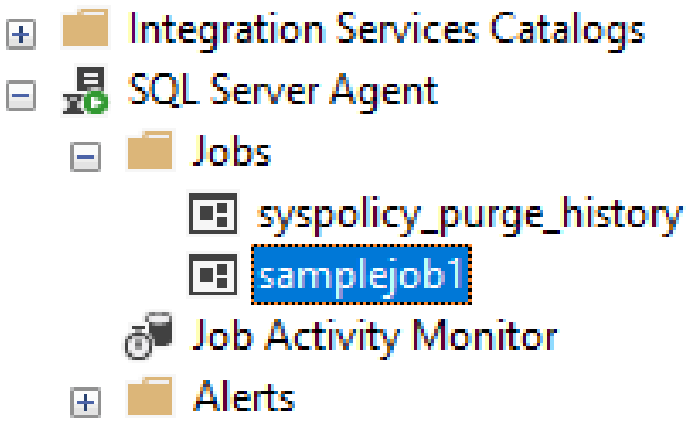
6. You should see your created job on the list. You can organize job order here.



7. The schedule for the job can also be set.



8. Once the new job had been configures, click ok to return to smss. You should see your job on the jobs list.



9. You can also perform other actions for the new job by rclicking the job and opening its context menu.