40th Anniversary Edition

DATABASE PROCESSING

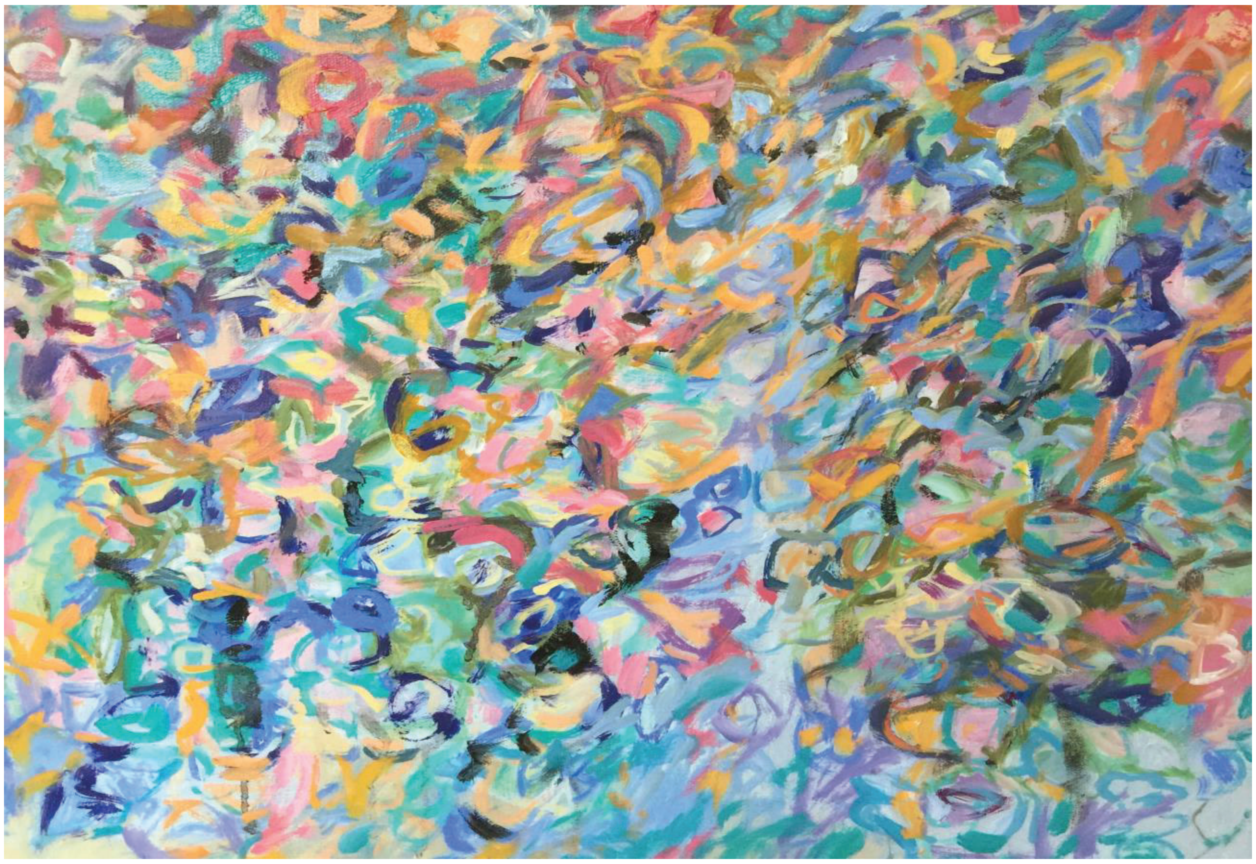
Fundamentals, Design, and Implementation

15th Edition

David M. Kroenke | David J. Auer | Scott L. Vandenberg | Robert C. Yoder

Online Chapter 10A

Managing Databases with Microsoft SQL Server 2017



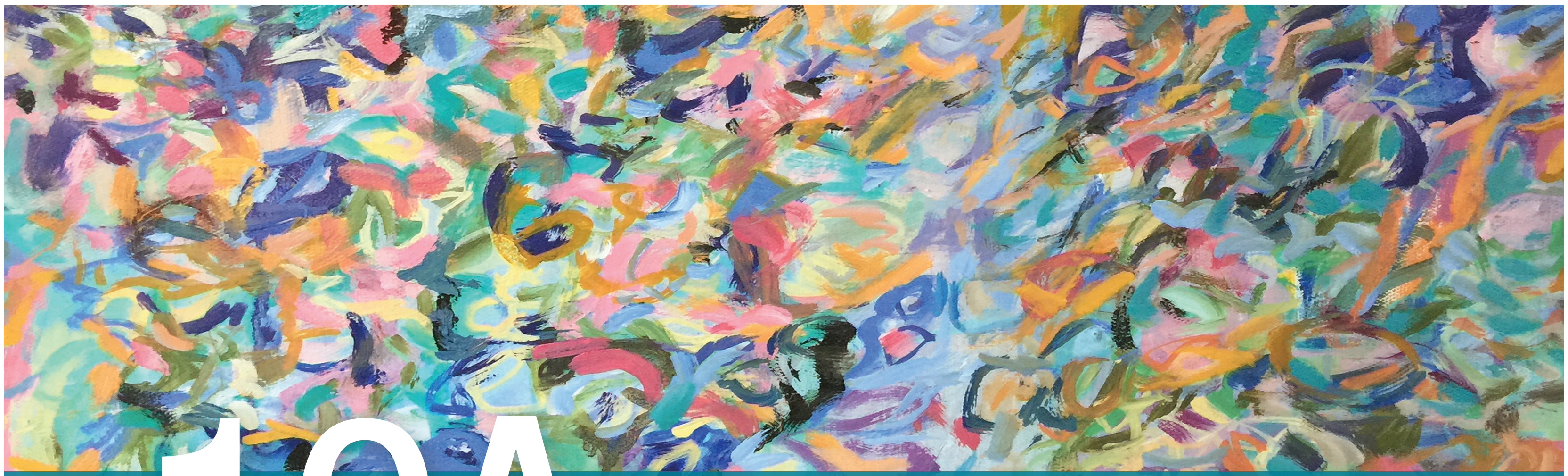
Vice President, IT & Careers: Andrew Gilfillan
Senior Portfolio Manager: Samantha Lewis
Managing Producer: Laura Burgess
Associate Content Producer: Stephany Harrington
Portfolio Management Assistant: Madeline Houpt
Director of Product Marketing: Brad Parkins
Product Marketing Manager: Heather Taylor
Product Marketing Assistant: Jesika Bethea
Field Marketing Manager: Molly Schmidt
Field Marketing Assistant: Kelli Fisher
Cover Image: Cover art "waterfall" by Donna Auer
Vice President, Product Model Management: Jason Fournier
Senior Product Model Manager: Eric Hakanson
Lead, Production and Digital Studio: Heather Darby
Digital Studio Course Producer: Jaimie Noy
Program Monitor: SPi Global
Full-Service Project Management and Composition: Cenveo® Publisher Services
Printer/Binder: LSC Communications
Cover Printer: Phoenix
Text Font: 10/12 Mentor Pro

Library of Congress Cataloging-in-Publication Data

Names: Kroenke, David M., 1948- author. | Auer, David J., author. |
Vandenberg, Scott L., author. | Yoder, Robert C., author.
Title: Database processing : fundamentals, design, and implementation /David
M. Kroenke, David J. Auer, Western Washington University, Scott L.
Vandenberg, Siena College, Robert C. Yoder, Siena College.
Description: 15th edition, 40th anniversary edition. | Boston : Pearson,
[2018] | Includes bibliographical references and index.
Identifiers: LCCN 2017041164| ISBN 9780134802749 | ISBN 0134802748
Subjects: LCSH: Database management.
Classification: LCC QA76.9.D3 K7365 2018 | DDC 005.74—dc23 LC record available
at https://lccn.loc.gov/2017041164

Chapter 10A – 10 9 8 7 6 5 4 3 2 1   ISBN 10: 0-13-480274-8   ISBN 13: 978-0-13-480274-9

Pearson
330 Hudson Street | New York, NY 10013



10A Managing Databases with Microsoft SQL Server 2017

Chapter Objectives

- To install Microsoft SQL Server 2017
- To use SQL Server 2017's graphical utilities
- To create a database in SQL Server 2017
- To submit both SQL DDL and DML via the Microsoft SQL Server Management Studio
- To import Microsoft Excel worksheet data into a database table
- To understand the use of SQL/Persistent Stored Modules (SQL/PSM) in SQL Server T-SQL
- To understand the purpose and role of user-defined functions and to create simple user-defined functions
- To understand the purpose and role of stored procedures and to create simple stored procedures
- To understand the purpose and role of triggers and to create simple triggers
- To understand how SQL Server implements indexes, concurrency control, and cursors
- To understand the SQL Server security model
- To understand the fundamental features of SQL Server backup and recovery facilities

This chapter describes the basic features and functions of Microsoft SQL Server 2017. The discussion uses the View Ridge Gallery *VRG* database from Chapter 7, and it parallels the discussion of the database administration tasks in Chapter 9. The presentation is similar in scope and orientation to that of Oracle Database in Chapter 10B and Oracle MySQL 5.7 in Chapter 10C.

SQL Server 2017 is a large and complicated product. In this one chapter, we will be able to only scratch the surface. Your goal should be to learn sufficient basics so you can continue learning on your own or in other classes.

The topics and techniques discussed here also apply to SQL Server 2016, SQL Server 2014, SQL Server 2012, SQL Server 2008 R2, SQL Server 2008, and the earlier SQL Server 2005, though the exact functions of the earlier versions of the SQL Server Management Studio may vary a bit from the SQL Server 2017 Management Studio. The material you learn in this chapter will be applicable to these older versions.

The Microsoft SQL Server 2017 DBMS

Microsoft SQL Server is an enterprise-class DBMS that has been around for many years. SQL Server 2005, which included the first release of SQL Server Express Edition, was released in November 2005. Subsequently, SQL Server 2008 was released in 2008, SQL Server 2008 R2 in 2010, SQL Server 2012 in 2012, SQL Server 2014 in March 2014, and SQL Server 2016 in 2016. Now SQL Server 2017 has been released and is available for use with this book. SQL Server 2017 is available in several versions, which can be reviewed at the Microsoft SQL Server 2017 Web site (*www.microsoft.com/en-us/sql-server/sql-server-2017-editions*) or in detail at the Microsoft Docs Web site (*https://docs.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-2017*). For our purposes, there are five editions you need to be aware of:

- **Enterprise Edition.** This is a powerful and feature-laden commercial version. It handles up to the maximum number of CPUs or CPU cores allowed by the operating system, the maximum memory supported by the operating system, and a maximum database size of 524 petabytes (PBytes). It includes full data warehouse capabilities and business intelligence capabilities.
- **Standard Edition.** This is the basic commercial version. It does not have the complete feature set of the Enterprise Edition. It handles up to 4 CPUs or 24 cores (lesser of 4 CPUs or 24 cores), 128 GBytes of memory, and a maximum database size of 524 PBytes. It has very limited data warehouse capabilities.
- **Web Edition.** This is a low-cost option for users who are going to use SQL Server only to support the database functions of their Web site. It does not have the complete feature set of the Enterprise Edition. It handles up to 4 CPUs or 16 cores (lesser of 4 CPUs or 16 cores), 64 GBytes of memory, and a maximum database size of 524 PBytes. It has very limited data warehouse capabilities.
- **Developer Edition.** This is a single-user version of the Enterprise Edition, and it has the complete feature set of the Enterprise Edition. It is intended, as the name implies, for use by a single user who is doing database and application development work. It is now available for free download, which makes it the preferred version for use with this book. However, it *cannot* be used in a production environment.
- **Express Edition.** This free, feature-limited version is available for download. It supports 1 CPU or 4 cores (lesser of 1 CPU or 4 cores), 1410 MBytes of memory, and a maximum database size of 10 GBytes. Despite its limitations, it is a great learning tool (and did we mention that it's free?). Express Edition *can* be used in a production environment.

BY THE WAY The SQL Server Express Edition was introduced with SQL Server 2005, and SQL Server 2017 includes the **SQL Server 2017 Express** edition. The SQL Server Express editions seem to be designed to compete with Oracle Corporation's MySQL (see Chapter 10C). Although MySQL does not have as many features as SQL Server, it is an open source database that has had the advantage of being available for download over the Internet at no cost. It has become widely used and very popular as a DBMS, supporting Web sites running the Apache Web server. The SQL Server Express editions, like MySQL, may be used in a production environment.

On March 31, 2016, Microsoft announced that SQL Server Developer Edition would be available for free[1] instead of the approximately $100 price it had previously had.

[1] See *https://blogs.technet.microsoft.com/dataplatforminsider/2016/03/31/microsoft-sql-server-developer-edition-is-now-free/?MC=Vstudio&MC=SQL&MC=IE&MC=HTML5&MC=JavaScript*. This announcement initially applied to SQL Server 2014, but the blog noted that it would also apply to SQL Server 2016 and SQL Server 2017.

This is a significant announcement because it means that ***we can use Developer Edition instead of Express Edition as the foundation for our work in this book***. Keep in mind, however, that you can use SQL Server Developer Edition only for your own personal work. SQL Server Express Edition, on the other hand, can be used in a production environment.

Starting with SQL Server 2016, Microsoft has split the database engine and administrative tools into two separate installations. Previously, all needed components were installed in one installation procedure. And finally, Microsoft SQL Server 2017 will run on the Linux operating system! Given Microsoft's history of having Microsoft products run only on the Microsoft Windows operating system (OS), this is a very important and significant development.

BY THE WAY Whereas earlier versions of Microsoft SQL Server allowed you to install the program on either a 32-bit or 64-bit operating system, SQL Server 2017 requires a 64-bit OS. Windows 10—and earlier versions of the Windows operating system—is available in both 32-bit and 64-bit versions. To determine which version of Windows 10 you have, click the **Start** button (Windows icon on the lower-left corner of the screen), then click **Settings** (the gear icon), then click **System**, and the click **About**. Look at the **System Type** setting.

The 32-bit versions of Microsoft programs at the download sites are designated as x86, whereas the 64-bit versions are designated as x64. The "x86" refers to Intel processors that include "86" in the processor name (for example, the Intel 80486 CPU chip) and related processors such as the Intel Pentium CPU chip (which would have been the 80586 if Intel hadn't switched to names instead of numbers for its product line).

In this book we are using a 64-bit version of Windows 10 Creators Update (Version 1703), updated with current security and operating system updates. All instructions are based on this version of Windows 10 and may vary if you are using a different Windows edition (7 or 8.10) or an earlier version of Windows 10.

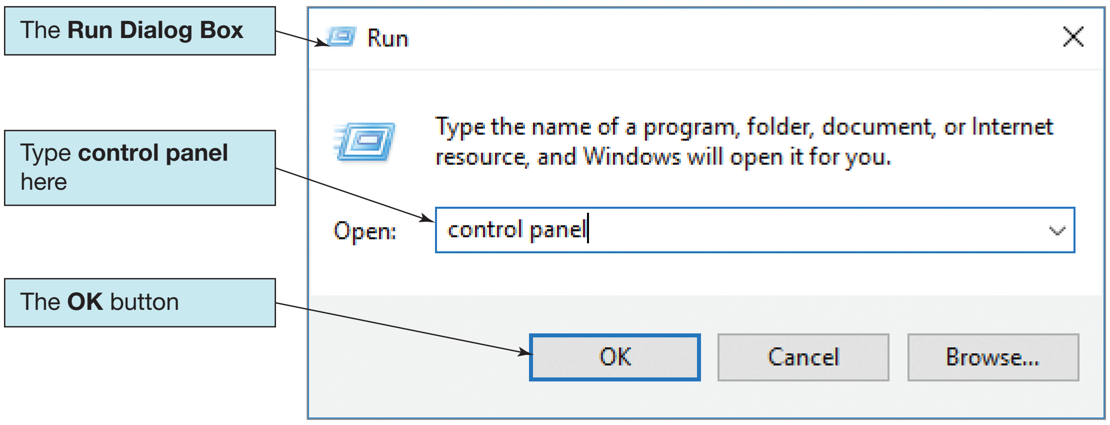
Installing Microsoft SQL Server 2017

Regardless of which version of Microsoft SQL Server you are going to use, you should download and install it now. Before installing Microsoft SQL Server itself, you need to make sure you have installed the software prerequisites. Be aware that SQL Server 2017 is an enterprise-class DBMS and, as such, is much more complex than Microsoft Access. Further, it does not include application development tools, such as form and report generators.

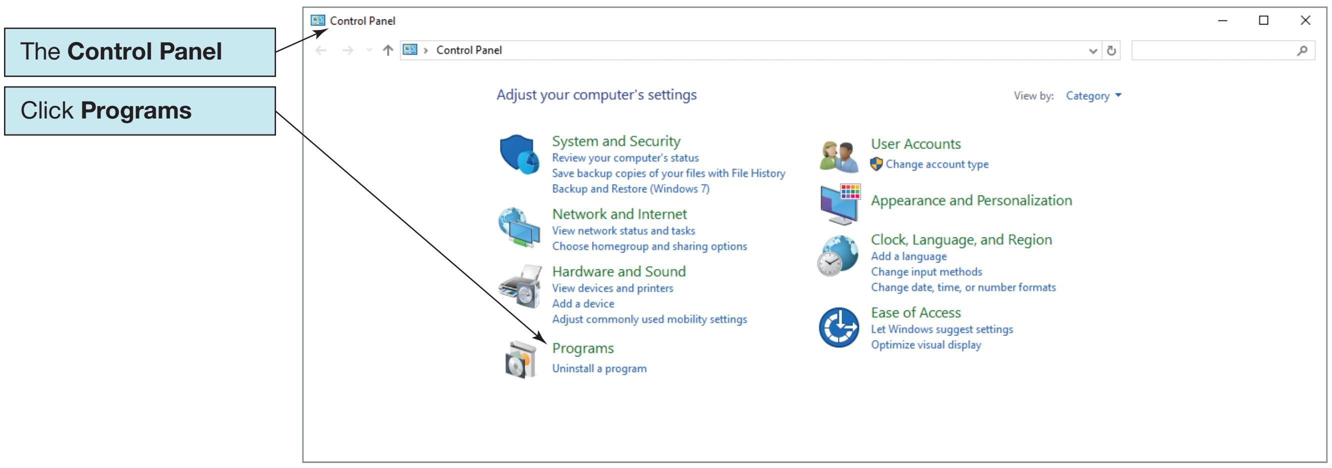
Installing Microsoft SQL Server 2017 Required Software

You should download and install the following Microsoft software:

- **.NET Framework 3.5 Service Pack 1.** This is provided with the Windows 10 OS and is an update to Microsoft .NET Framework 3.5. This version of the .NET Framework is *not* automatically installed. If you are running Windows 10, this can be installed by using the Windows Control Panel. Open Control Panel using the **Windows key + x key combination** to display the shortcut menu, then:
  - If the Control Panel command is shown (it is available in Windows 10 versions prior to the Windows 10 Creator Update), click the **Control Panel** command.
  - If the Control Panel command is *not* shown (it is *not* available in the Windows 10 Creator Update), click the **Run** command. The Run dialog box is displayed. Type the text *Control Panel* in the Open text box, as shown in Figure 10A-1(a) and then click the **OK** button.


The **Run Dialog Box**
Type **control panel** here
The **OK** button
Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open: control panel
OK
Cancel
Browse...

(a) The Run Dialog Box


The **Control Panel**
Click **Programs**
Control Panel
Adjust your computer's settings
View by: Category
System and Security
Review your computer's status
Save backup copies of your files with File History
Backup and Restore (Windows 7)
Network and Internet
View network status and tasks
Choose homegroup and sharing options
Hardware and Sound
View devices and printers
Add a device
Adjust commonly used mobility settings
Programs
Uninstall a program
User Accounts
Change account type
Appearance and Personalization
Clock, Language, and Region
Add a language
Change input methods
Change date, time, or number formats
Ease of Access
Let Windows suggest settings
Optimize visual display

(b) The Control Panel

FIGURE 10A-1

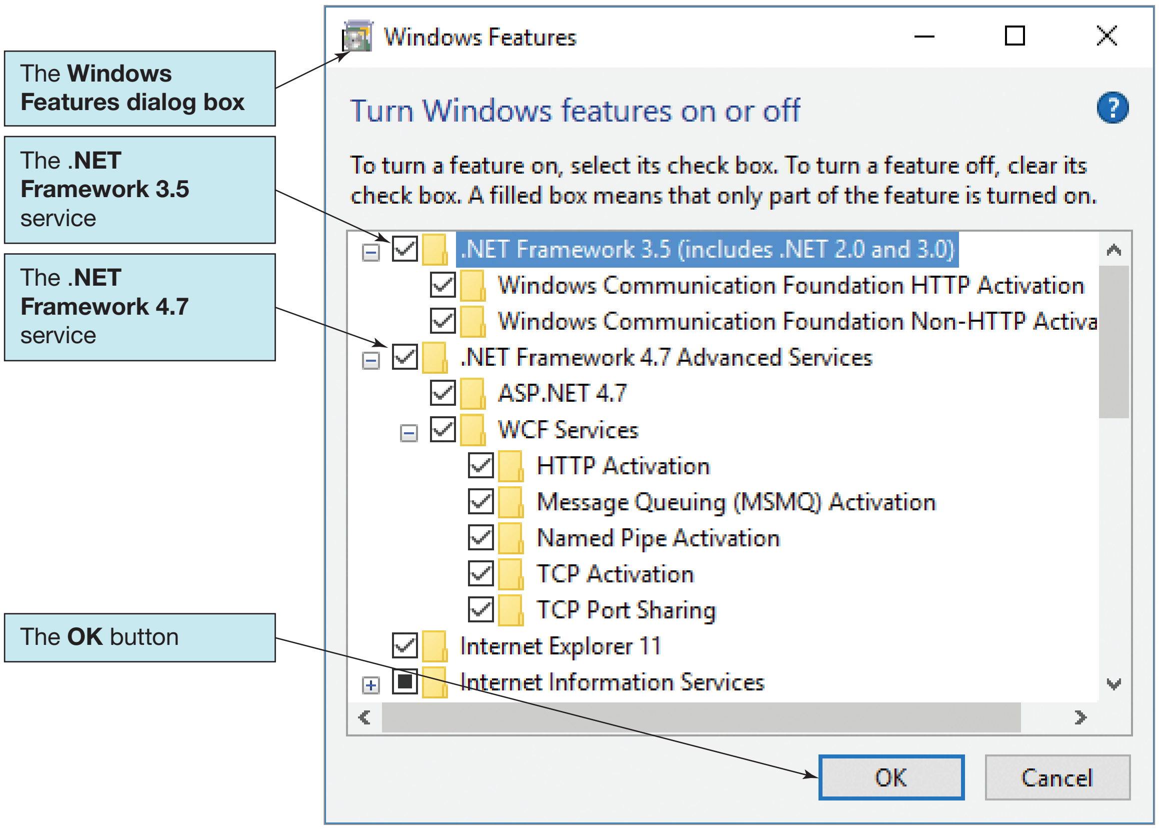
Preparations for Installing Microsoft SQL Server 2017

- The Control Panel is displayed, as shown in Figure 10A-1(b). In Control Panel, click **Programs**. In Programs, click **Turn Windows features on or off**. In the Turn Windows features on or off dialog box, select the .NET Framework 3.5 features shown in Figure 10A-1(c). See the following note about *.NET Framework 4.6.1 or later* before clicking the **OK** button to begin the installation process.
- **.NET Framework 4.6.1 or later.** This is an updated version of the Microsoft .NET Framework (*www.visualstudio.com/downloads/download-visual-studio-vs*). If you are running Windows 10, this is provided with the Windows OS and should be enabled by default. SQL Server 2016 SP1 and later versions require at least .NET Framework 4.6.1. However, .NET Framework 7.0 is available and is installed with the Windows 10 Creator Update. If needed .NET Framework 4.6.1 or later can be installed by using the Windows Control Panel. See the installation instructions in the discussion of .NET Framework 3.5 SP 1 earlier, and use Figure 10A-1(c) as a guide to feature selection. After both .NET Framework 3.5 SP 1 and .NET Framework 4.6.1/4.7 features have been selected, click the **OK** button to begin the installation process.
- **Oracle Java Runtime Environment (JRE)**. This is required in order to install the PolyBase component of Microsoft SQL Server 2017. Note that the Java JRE is included in the Java software development kit (SDK) that we install to support Web database application development and the NetBeans integrated development environment (IDE) in Appendix H, "Getting Started with Web Servers, PHP, and the NetBeans IDE." You may prefer to install the Java SDK and Java JRE together at this time. If so, see the installation steps in Appendix H. Otherwise, download the current version of the JRE (which is JRE 8 update 152 as of January 8, 2018)

which can be found at *www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html*. Click the **Download** button to download the appropriate JRE file into your Downloads folder as shown in Figure 10A-1(d). When the download is finished, click the **Run** button shown in Figure 10A-1(e) and install the JRE.

- **NOTE:** We are using the Microsoft Edge Web browser. If you are using a non-Microsoft Web browser, after the download is complete, find the **jre-8u151-windows-x64.exe** file in the **Downloads** folder. Right-click the file name to display a shortcut menu, and then click **Run as administrator** to install the JRE.
- **NOTE:** The Java JRE is included in the Java software development kit (SDK) that we install to support Web database application development and the NetBeans integrated development environment (IDE) in Appendix H, "Getting Started with Web Servers, PHP, and the NetBeans IDE." You may prefer to install the Java SDK and Java JRE together at this time. If so, see the installation steps in Appendix H.

- **Update for Visual C++ 2013 and Visual C++ Redistributable Package** and **KB3164398**. This is a required patched version of the Visual C++ 2013 redistributable package. It can be found at *https://support.microsoft.com/en-us/kb/3138367*. Download both the 32-bit (x86) and 64-bit versions (x64) into your **Downloads** folder, and then install (right-click the file name, as you will need to use the **Run as Administrator** command). Figure 10A-3 shows the 32-bit version of the update being installed. As of October 2, 2017, Microsoft Visual C++ 2013 Update 5 is available. Reboot your computer after installing (you may be prompted to do so). You may need to accept the license and terms before the install begins for these Microsoft updates. If you already have these patches installed, you may see a "repair" button instead of an "install" button.


The Windows Features dialog box
The .NET Framework 3.5 service
The .NET Framework 4.7 service
The OK button
Windows Features
OK
Cancel

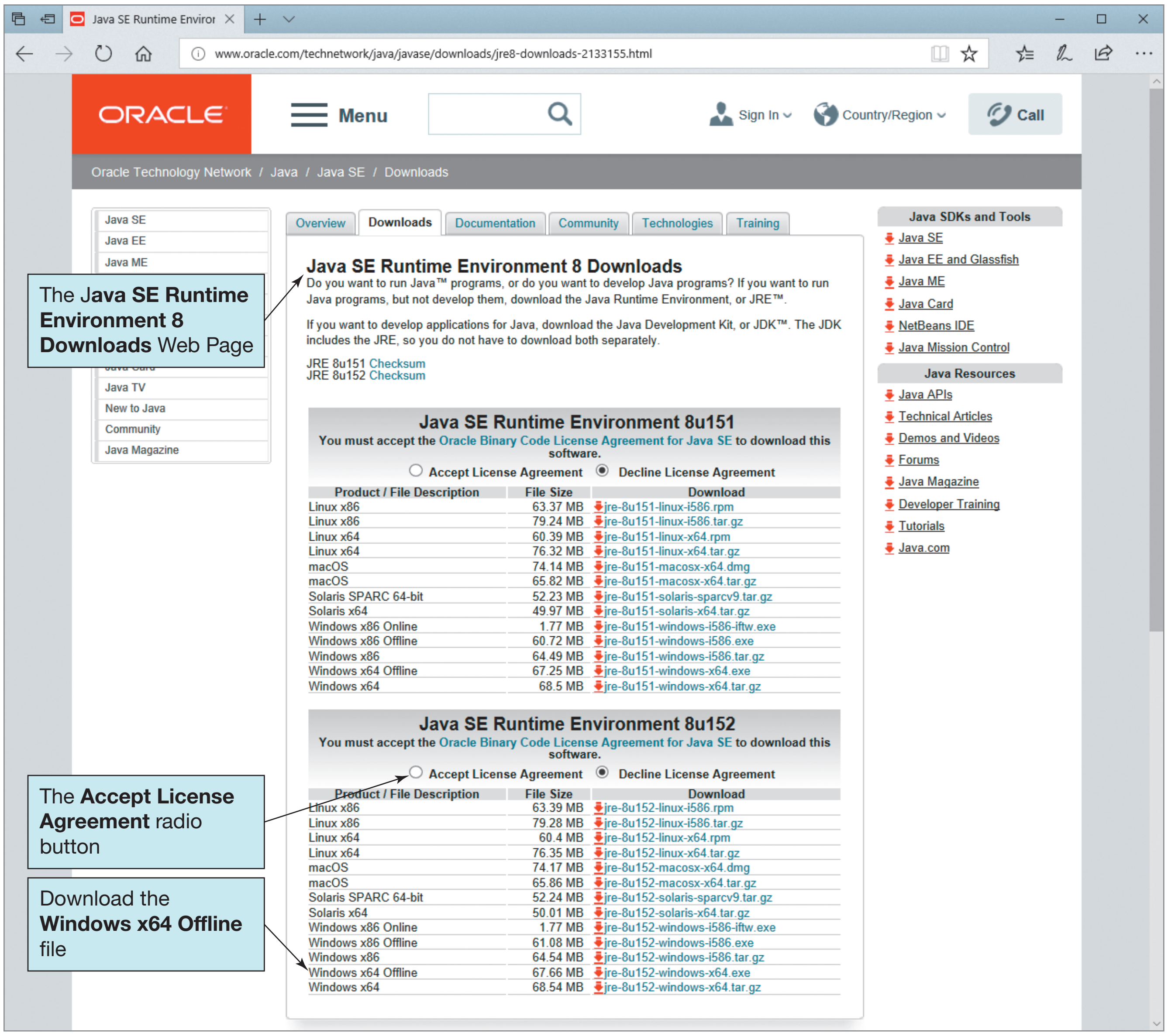
Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- [✓] .NET Framework 3.5 (includes .NET 2.0 and 3.0)
  - [✓] Windows Communication Foundation HTTP Activation
  - [✓] Windows Communication Foundation Non-HTTP Activa
- [✓] .NET Framework 4.7 Advanced Services
  - [✓] ASP.NET 4.7
  - [✓] WCF Services
    - [✓] HTTP Activation
    - [✓] Message Queuing (MSMQ) Activation
    - [✓] Named Pipe Activation
    - [✓] TCP Activation
    - [✓] TCP Port Sharing
- [✓] Internet Explorer 11
- [■] Internet Information Services

(c) The Windows Features Dialog Box

FIGURE 10A-1
Continued



Java SE Runtime Environment 8 Downloads

Do you want to run Java™ programs, or do you want to develop Java programs? If you want to run Java programs, but not develop them, download the Java Runtime Environment, or JRE™.

If you want to develop applications for Java, download the Java Development Kit, or JDK™. The JDK includes the JRE, so you do not have to download both separately.

JRE 8u151 Checksum
JRE 8u152 Checksum

Java SE Runtime Environment 8u151

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

[ ] Accept License Agreement [✓ Decline License Agreement]

Product / File Description	File Size	Download
Linux x86	63.37 MB	jre-8u151-linux-i586.rpm
Linux x86	79.24 MB	jre-8u151-linux-i586.tar.gz
Linux x64	60.39 MB	jre-8u151-linux-x64.rpm
Linux x64	76.32 MB	jre-8u151-linux-x64.tar.gz
macOS	74.14 MB	jre-8u151-macosx-x64.dmg
macOS	65.82 MB	jre-8u151-macosx-x64.tar.gz
Solaris SPARC 64-bit	52.23 MB	jre-8u151-solaris-sparcv9.tar.gz
Solaris x64	49.97 MB	jre-8u151-solaris-x64.tar.gz
Windows x86 Online	1.77 MB	jre-8u151-windows-i586-iftw.exe
Windows x86 Offline	60.72 MB	jre-8u151-windows-i586.exe
Windows x86	64.49 MB	jre-8u151-windows-i586.tar.gz
Windows x64 Offline	67.25 MB	jre-8u151-windows-x64.exe
Windows x64	68.5 MB	jre-8u151-windows-x64.tar.gz

Java SE Runtime Environment 8u152

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

[ ] Accept License Agreement [✓ Decline License Agreement]

Product / File Description	File Size	Download
Linux x86	63.39 MB	jre-8u152-linux-i586.rpm
Linux x86	79.28 MB	jre-8u152-linux-i586.tar.gz
Linux x64	60.4 MB	jre-8u152-linux-x64.rpm
Linux x64	76.35 MB	jre-8u152-linux-x64.tar.gz
macOS	74.17 MB	jre-8u152-macosx-x64.dmg
macOS	65.86 MB	jre-8u152-macosx-x64.tar.gz
Solaris SPARC 64-bit	52.24 MB	jre-8u152-solaris-sparcv9.tar.gz
Solaris x64	50.01 MB	jre-8u152-solaris-x64.tar.gz
Windows x86 Online	1.77 MB	jre-8u152-windows-i586-iftw.exe
Windows x86 Offline	61.08 MB	jre-8u152-windows-i586.exe
Windows x86	64.54 MB	jre-8u152-windows-i586.tar.gz
Windows x64 Offline	67.66 MB	jre-8u152-windows-x64.exe
Windows x64	68.54 MB	jre-8u152-windows-x64.tar.gz

(d) The Java SE Runtime Environment 8 Downloads Web Page

FIGURE 10A-1
Continued

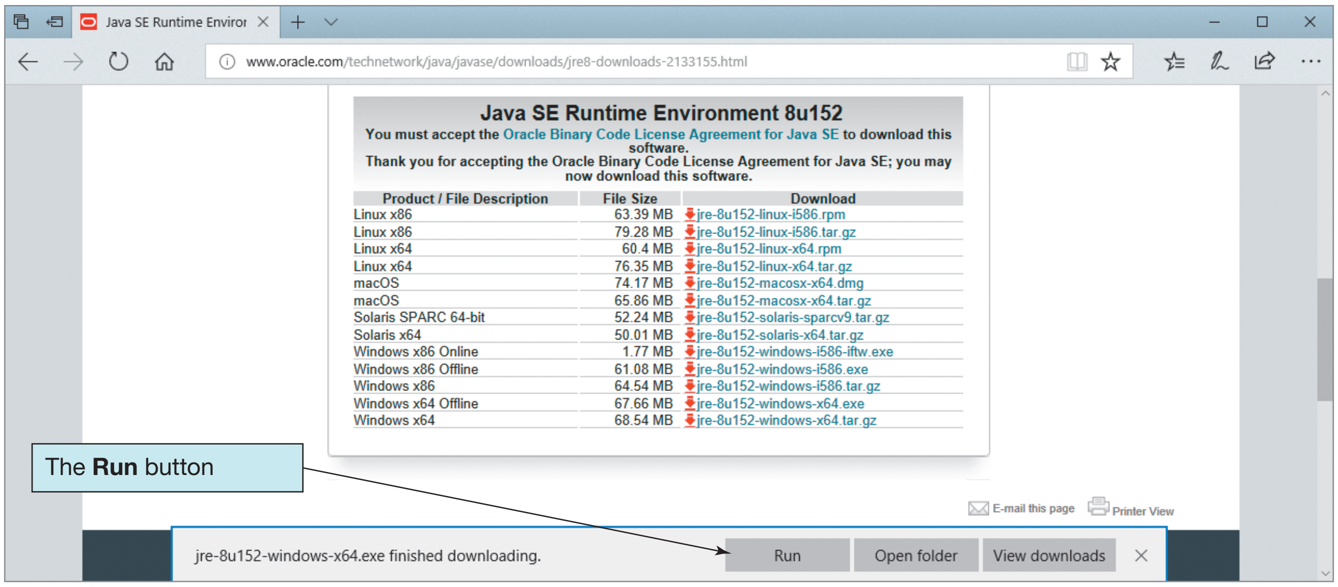
- **Microsoft Visual C++ Redistributable Packages for Visual Studio 2015 Update 3**. Both the 32-bit x86 and 64-bit x64 versions are available at *https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads*. Install both versions if you have a 64-bit OS.
- **Microsoft Visual C++ Redistributable for Visual Studio 2017**. Only the 64-bit x64 version is provided. See *https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads*.

This may seem like a lot of preparation, but sooner or later you will need all of these. Further, if you haven't installed the prerequisites, you may encounter errors when you install SQL Server 2017.

The SQL Server 2017 Developer Edition installation files itself must be downloaded. Although this process is fairly straight forward, there are several steps, which we will now describe.

Downloading the Microsoft SQL Server 2017 Installation File

1. To start the SQL Server 2017 download process, go to the **SQL Server 2017** Web page at *www.microsoft.com/en-us/sql-server/sql-server-2017*, as shown in Figure 10A-1(f).



Java SE Runtime Environment 8u152

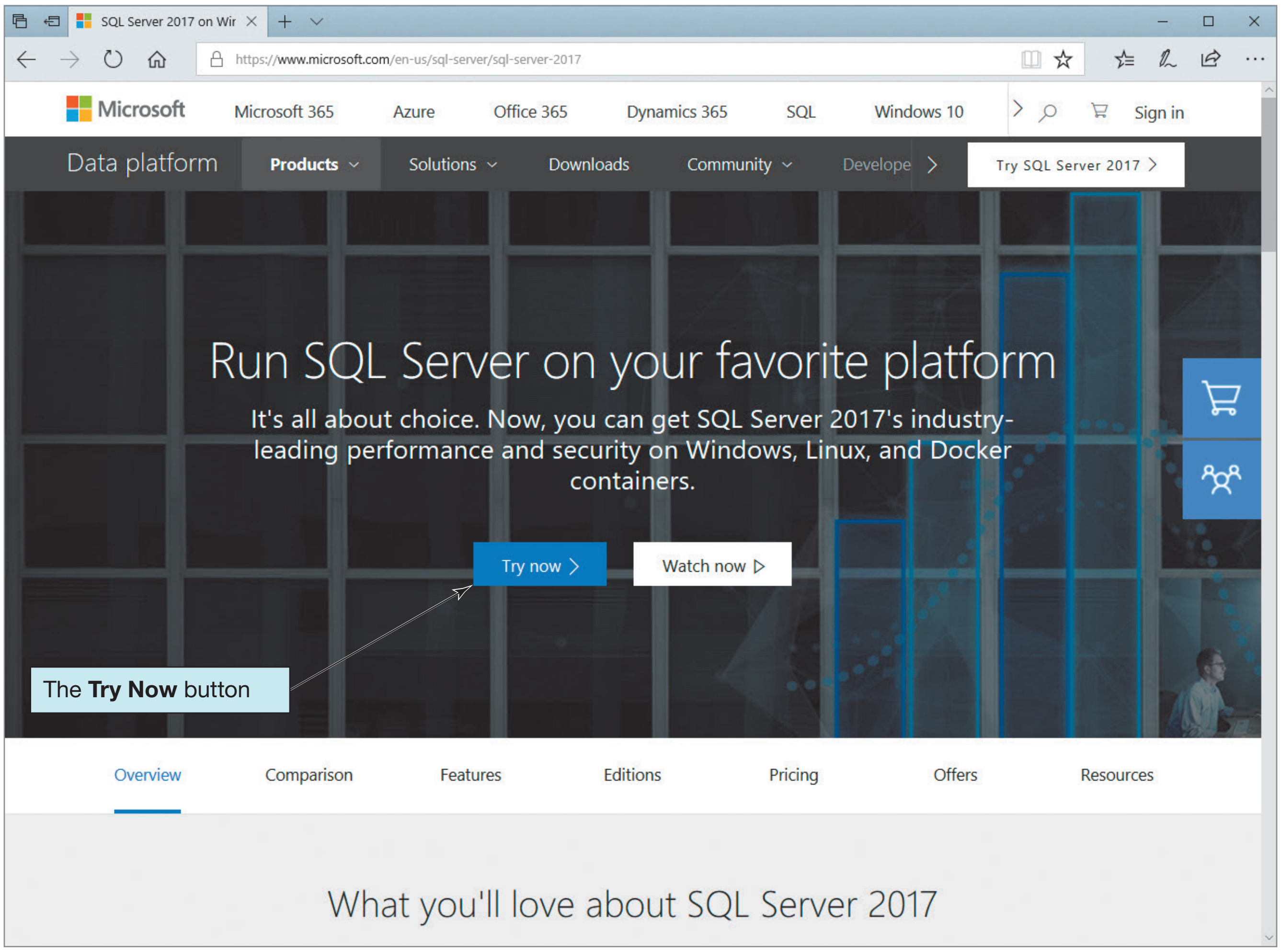
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	63.39 MB	jre-8u152-linux-i586.rpm
Linux x86	79.28 MB	jre-8u152-linux-i586.tar.gz
Linux x64	60.4 MB	jre-8u152-linux-x64.rpm
Linux x64	76.35 MB	jre-8u152-linux-x64.tar.gz
macOS	74.17 MB	jre-8u152-macosx-x64.dmg
macOS	65.86 MB	jre-8u152-macosx-x64.tar.gz
Solaris SPARC 64-bit	52.24 MB	jre-8u152-solaris-sparcv9.tar.gz
Solaris x64	50.01 MB	jre-8u152-solaris-x64.tar.gz
Windows x86 Online	1.77 MB	jre-8u152-windows-i586-iftw.exe
Windows x86 Offline	61.08 MB	jre-8u152-windows-i586.exe
Windows x86	64.54 MB	jre-8u152-windows-i586.tar.gz
Windows x64 Offline	67.66 MB	jre-8u152-windows-x64.exe
Windows x64	68.54 MB	jre-8u152-windows-x64.tar.gz

jre-8u152-windows-x64.exe finished downloading.

(e) The Run Button



Run SQL Server on your favorite platform

It's all about choice. Now, you can get SQL Server 2017's industry-leading performance and security on Windows, Linux, and Docker containers.

What you'll love about SQL Server 2017

(f) The SQL Server 2017 Web Page

FIGURE 10A-1

Continued

2. Click the **Try Now** button. The **Download SQL Server 2017 for Windows** Web page is displayed, as shown in Figure 10A-1(g).
3. As shown in Figure 10A-1(g), we are given a choice of which SQL Server 2017 edition to download. Click the Developer edition **Download Now** button.
4. A dialog box asking "What do you want to do with SQLServer2017-SSEI-Dev .exe (5.1. MB)" is displayed at the bottom of the Web page. Click the **Run** button.
5. A Windows User Account dialog box appears, asking if we want to run SQL Server 2017. Click the **Yes** button.
6. The **SQL Server 2017 Developer Edition Download** dialog box *Select an installation type* page is displayed, as shown in Figure 10A-1(h). The SQLServer-SSEI-Dev.exe program we are running, is not the complete set of SQL Server 2017 installation files. It is the tool we will use to download those files.
7. We want to download the complete set of installation files onto our computer, rather than run on online setup, so click the **Download Media** button. The SQL Server 2017 Developer Edition Download dialog box *Specify SQL Server installer download* page is displayed, as shown in Figure 10A-1(i).
8. This page allows use to set parameters for the file we download. We do want the English language version, which is the default. We will download the ISO disk image version, so select the **ISO** radio button. By default, the file will be downloaded into our Downloads folder, but we can select a different location by clicking the **Browse** button. We have created a specific folder to hold our SQL Server 2017 files, so we browse to *C:\Users\{UserName}\Downloads\Microsoft\MSSQL-2017-Developer*. These settings are shown in Figure 10A-1(i). After completing the complete set of parameters, click the **Download** button.
9. While the files are being downloaded, the *Downloading media* page is displayed showing the download progress. When the download is complete, the *Download successful!* page is displayed, as shown in Figure 10A-1(j). We have completed downloading the SQL Server 2017 installation files, so click the **Close** button. A dialog box will ask if want to close the installation window, so click the **Yes** button.

The Microsoft SQL Server Management Studio and Microsoft SQL Server 2017 Reporting Services are linked to the download of Microsoft SQL Server 2017 Developer Edition, but require a separate installation. **Microsoft SQL Server Management Studio** is the graphical management utility for Microsoft SQL Server 2017. Although Microsoft SQL Server 2017 can be run from a command-line program such as Windows PowerShell, SQL Server Management Studio makes it much easier to work with SQL Server. **Microsoft SQL Server 2017 Reporting Services** provides SQL Server support for business intelligence (BI) systems reporting systems as discussed in Chapter 12 and Appendix J, "Business Intelligence Systems." We will discuss how to install these programs after we complete the installation of SQL Server 2017 itself.

Although not required, you may also want to download and install the following software:

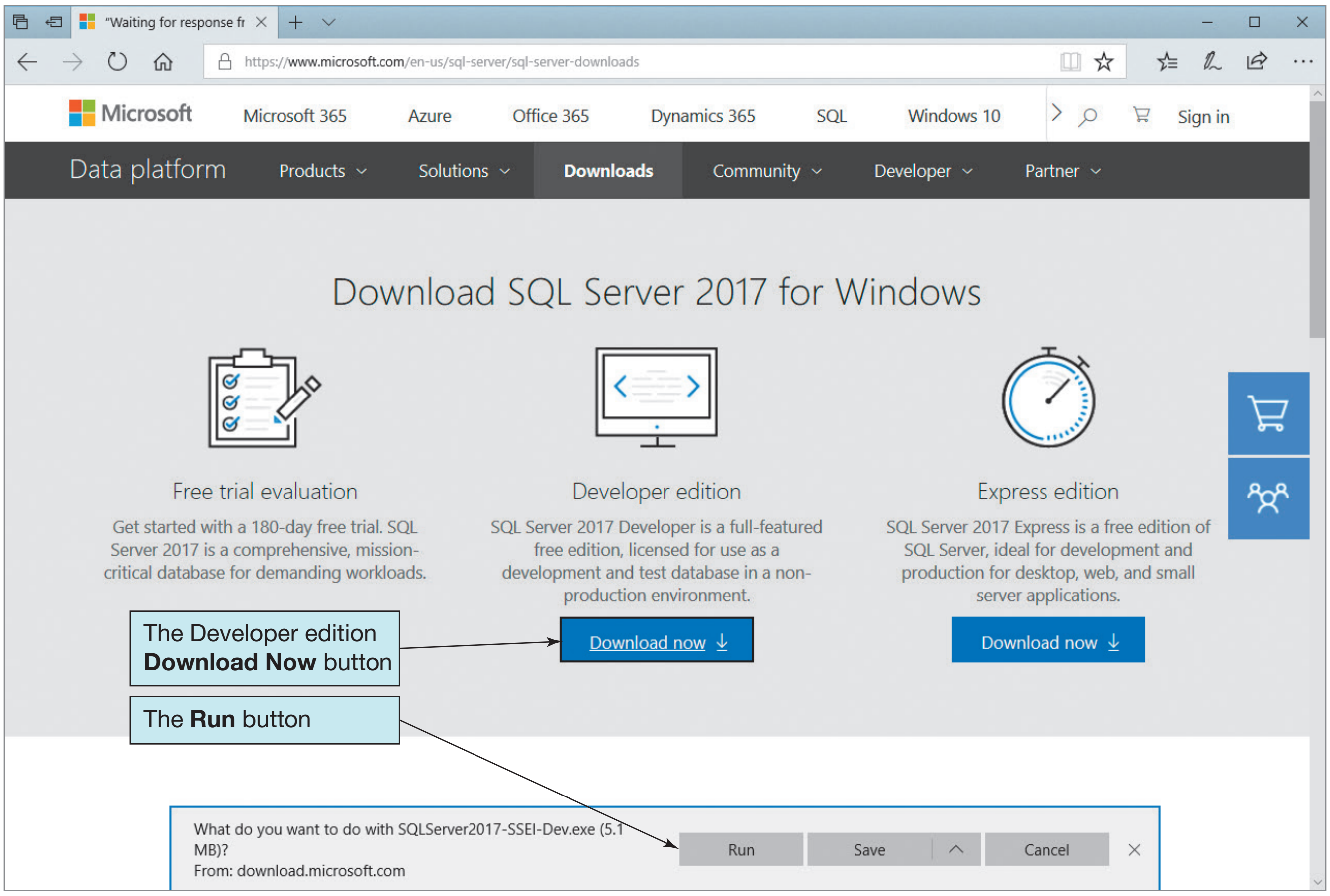
- Microsoft SQL Server 2016 Report Builder (yes, the 2016 version, not 2017 at this time) from *www.microsoft.com/en-us/download/details.aspx?id=53613*
- Microsoft Drivers for PHP for Microsoft Server (select the SQLSRV40.EXE version) from *www.microsoft.com/en-us/download/details.aspx?id=20098*

SQL Server documentation is available online at *https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation*.

Installing the Microsoft SQL Server 2017 DBMS

Now that we have downloaded the SQL Server 2017 installation files, let's install SQL Server 2017 on our computer. Our first task is to determine which of SQL Server's many components we will actually install. Here are our options:

- **SQL Server Database Engine**. This is the basic set of database processing feature. *Yes*, we will install it.



Download SQL Server 2017 for Windows

Free trial evaluation

Get started with a 180-day free trial. SQL Server 2017 is a comprehensive, mission-critical database for demanding workloads.

Developer edition

SQL Server 2017 Developer is a full-featured free edition, licensed for use as a development and test database in a non-production environment.

Express edition

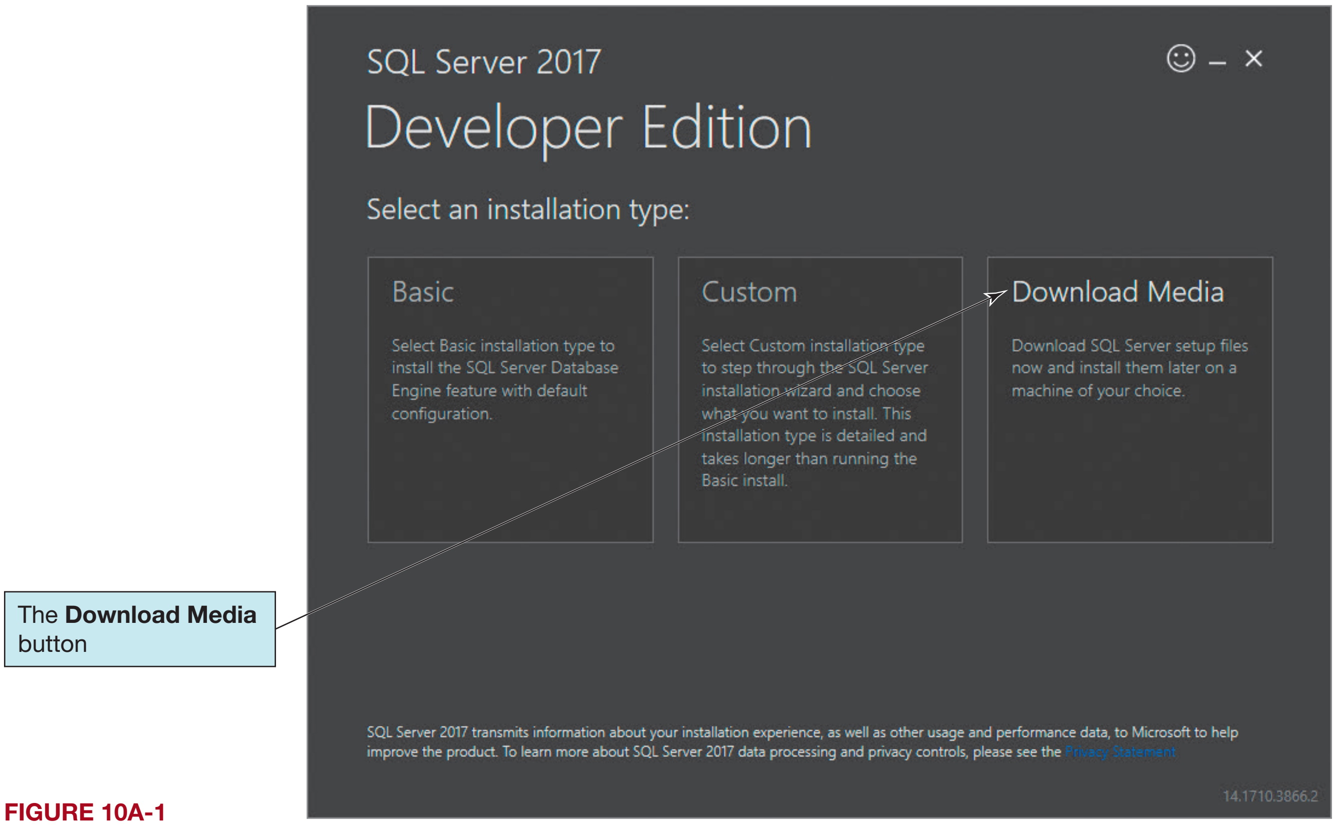
SQL Server 2017 Express is a free edition of SQL Server, ideal for development and production for desktop, web, and small server applications.

The Developer edition **Download Now** button

The **Run** button

What do you want to do with SQLServer2017-SSEI-Dev.exe (5.1 MB)?
From: download.microsoft.com

(g) The Download SQL Server 2017 for Windows Web Page



SQL Server 2017

Developer Edition

Select an installation type:

Basic

Select Basic installation type to install the SQL Server Database Engine feature with default configuration.

Custom

Select Custom installation type to step through the SQL Server installation wizard and choose what you want to install. This installation type is detailed and takes longer than running the Basic install.

Download Media

Download SQL Server setup files now and install them later on a machine of your choice.

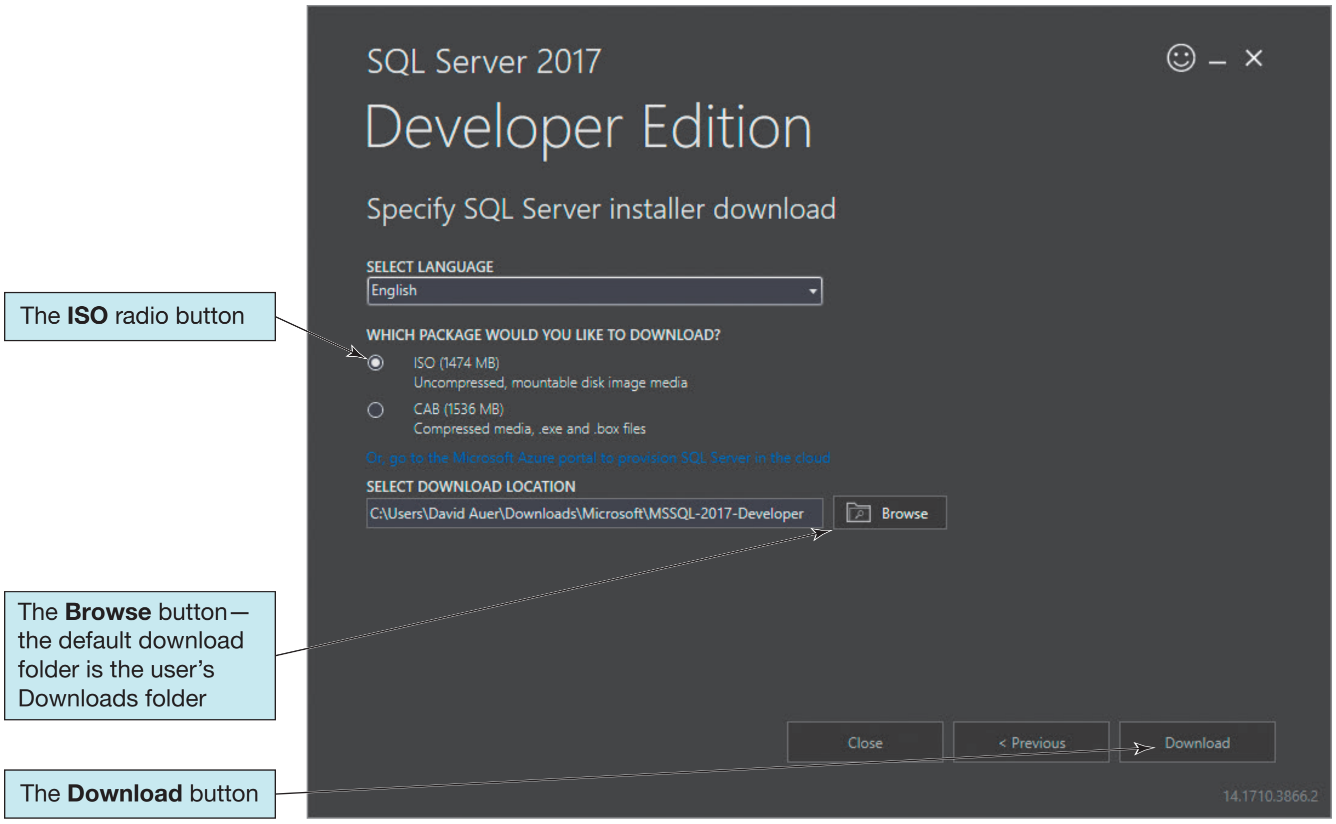
The **Download Media** button

SQL Server 2017 transmits information about your installation experience, as well as other usage and performance data, to Microsoft to help improve the product. To learn more about SQL Server 2017 data processing and privacy controls, please see the Privacy Statement

14.1710.3866.2

FIGURE 10A-1
Continued

(h) The Select an Installation Type Page



SQL Server 2017

Developer Edition

Specify SQL Server installer download

SELECT LANGUAGE
English

WHICH PACKAGE WOULD YOU LIKE TO DOWNLOAD?

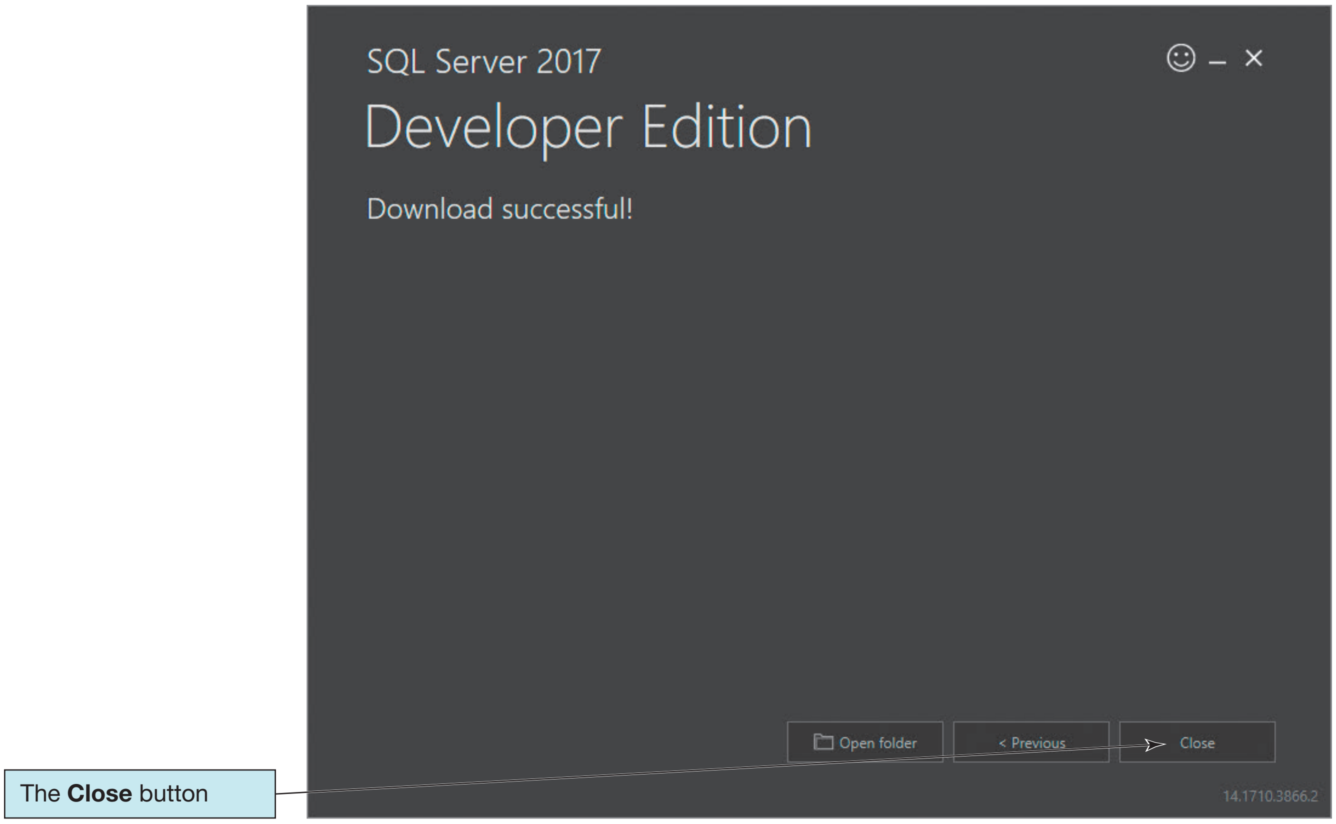
[✓ ISO (1474 MB)
Uncompressed, mountable disk image media]

CAB (1536 MB)
Compressed media, .exe and .box files

Or, go to the Microsoft Azure portal to provision SQL Server in the cloud

SELECT DOWNLOAD LOCATION
C:\Users\David Auer\Downloads\Microsoft\MSSQL-2017-Developer

(i) The Select SQL Server Installer Download Page



SQL Server 2017

Developer Edition

Download successful!

(j) The Download Successful Page

FIGURE 10A-1
Continued

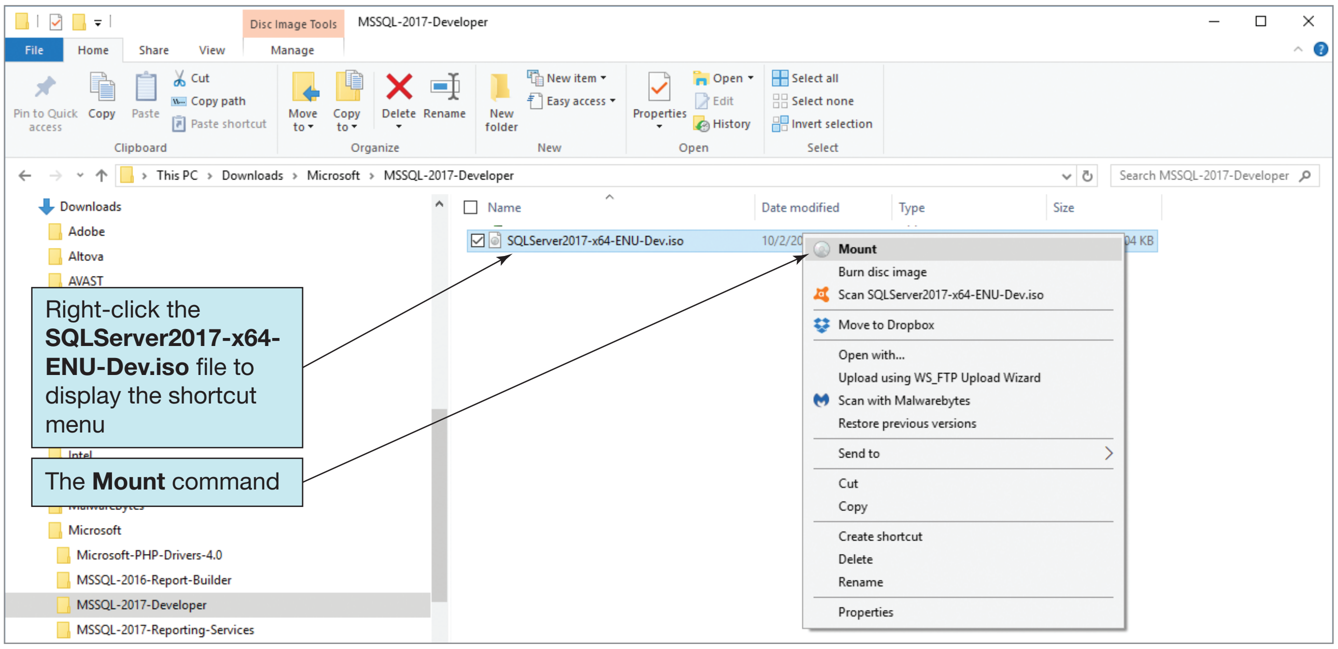
- **Analysis Services**. This feature provides support for online analytical processing (OLAP) and other data mining processes as discussed in Chapter 12 and Appendix J, "Business Intelligence Systems." *Yes*, we will install it.
- **Reporting Services**. This feature provides support for reporting systems as discussed in Chapter 12 and Appendix J, "Business Intelligence Systems." *Yes*, we will install it, but, starting with SQL Server 2017, it requires a separate download and installation that must be done after the basic SQL Server 2017 installation is complete.
- **Integration Services**. This feature provides support for moving, copying and transforming data reporting systems as discussed in Chapter 12 and Appendix J, "Business Intelligence Systems." It includes **Data Quality Services**. *Yes*, we will install it.
- **Master Data Services**. This feature provides support for data management on an organization level, including data replication.[2] This topic is beyond the scope of this book, so *No*, we will not install it. Neither will we install the **Distributed Replay Controller and Client**.
- **Machine Learning Services**. Available for both within databases and on a stand-alone basis, this feature supports both the R and Python languages.[3] It adds important analysis capabilities. *Yes*, we will install it.
- **PolyBase**. This feature provides access to external data beyond that stored in databases using the standard T-SQL language.[4] Specific support for data stored in Hadoop and Azure Blob storage is included. This relates to Big Data topics as discussed in Chapter 12 and Appendix K, "Big Data." *Yes*, we will install it.

With this set of features to be installed, we now install SQL Server 2017. Figure 10A-2 illustrates the installation process of Microsoft SQL Server on a Windows operating system. Figure 10A-2 shows the *SQL Server Installation Center*, which is the first screen to appear when the installation process begins.

Installing Microsoft SQL Server 2017

1. Open Windows File Explorer, and browse to the folder where the downloaded *SQLServer2017-x64-ENU-Dev.iso* file is located. Right-click the filename to display a shortcut menu, as shown in Figure 10A-2(a), and in the shortcut menu click the **Mount** command. An ***.iso file** is a disk image generally intended for burning a physical DVD. However, current operating systems such as Windows 10 allow the file to be mounted as a virtual DVD drive, and the Mount command is used to do this.
2. The *SQLServer2017-x64-ENU-Dev.iso* mounted as a virtual DVD drive named *SqlSetup_x64_ENU*, as shown in Figure 10A-2(b). Right-click the setup.exe file to display a shortcut menu, and in the shortcut menu click the **Run as administrator** command.
3. A User Account Control dialog box appears asking if you want to run SQL Server 2017. Click the **Yes** button.
4. The **SQL Server Installation Center** with the *Planning* page is displayed as shown in Figure 10A-2(c). Click the **Installation** page button.
5. The SQL Server Installation Center *Installation* page is displayed as shown in Figure 10A-2(d). To start the actual installation process, click on **New SQL Server stand-alone installation or add features to an existing installation** button.
6. The **SQL Server 2017 Setup dialog box** appears with the *Product Key* page displayed, as shown in Figure 10A-2(e).

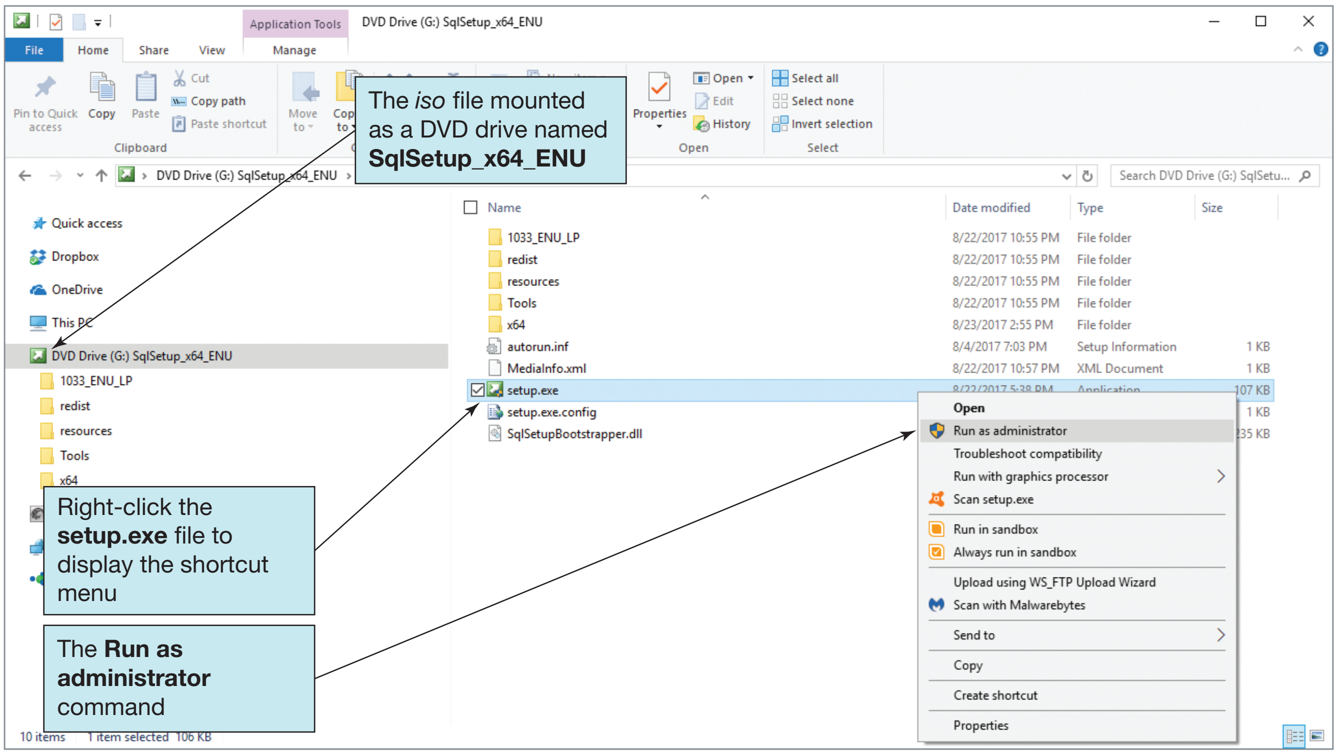
[2] For more information on Master Data Services see *https://docs.microsoft.com/en-us/sql/master-data-services/master-data-services-overview-mds*.
[3] For more information on Machine Learning Services see *https://docs.microsoft.com/en-us/sql/advanced-analytics/r/r-services*.
[4] For more information on PolyBase see *https://docs.microsoft.com/en-us/sql/relational-databases/polybase/polybase-guide*.



Right-click the **SQLServer2017-x64-ENU-Dev.iso** file to display the shortcut menu

The **Mount** command

(a) The SQLServer2017-x64-ENU-Dev.iso The Shortcut Menu



The *iso* file mounted as a DVD drive named **SqlSetup_x64_ENU**

Right-click the **setup.exe** file to display the shortcut menu

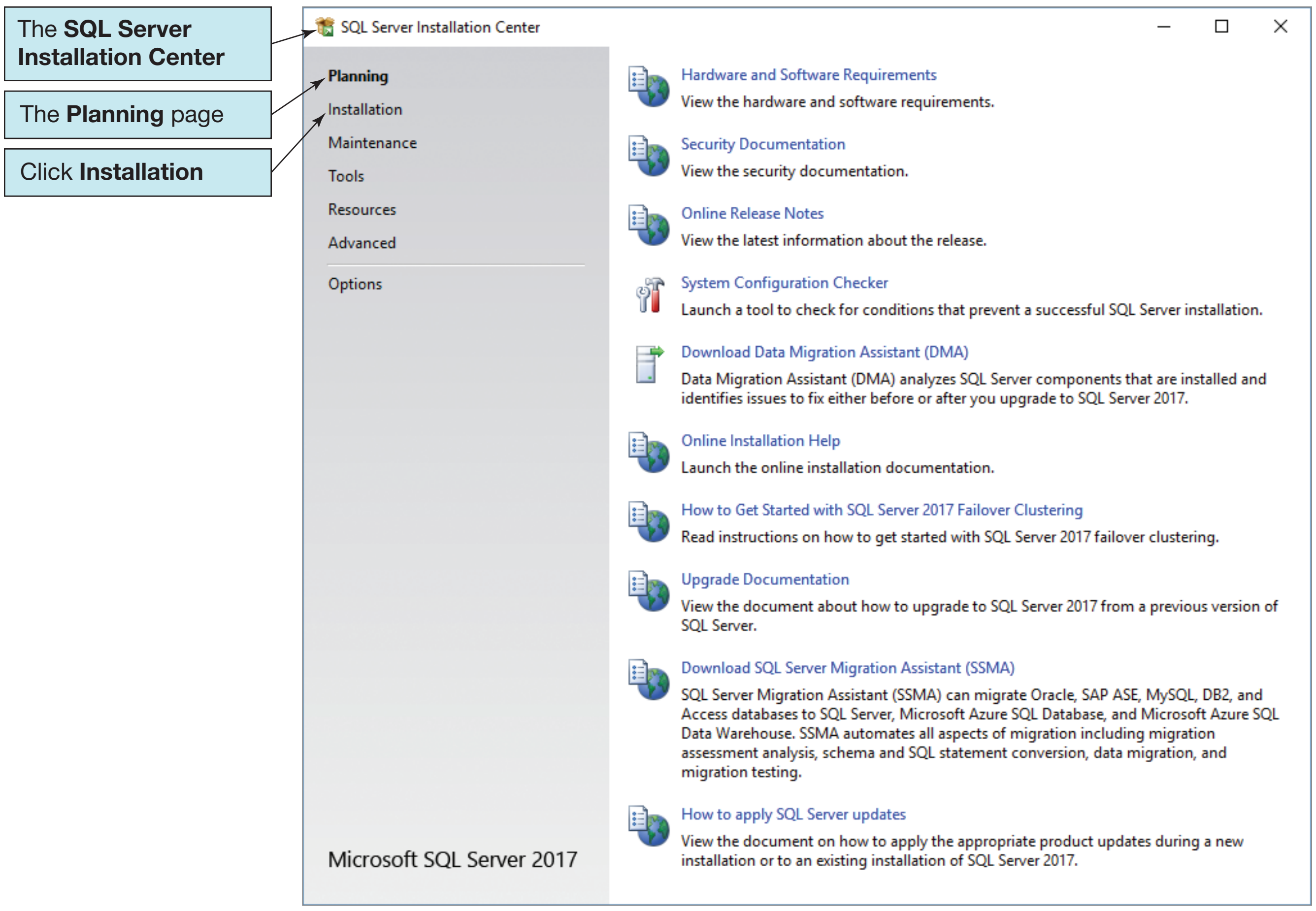
The **Run as administrator** command

Name	Date modified	Type	Size
1033_ENU_LP	8/22/2017 10:55 PM	File folder	
redist	8/22/2017 10:55 PM	File folder	
resources	8/22/2017 10:55 PM	File folder	
Tools	8/22/2017 10:55 PM	File folder	
x64	8/23/2017 2:55 PM	File folder	
autorun.inf	8/4/2017 7:03 PM	Setup Information	1 KB
MediaInfo.xml	8/22/2017 10:57 PM	XML Document	1 KB
setup.exe	8/22/2017 5:38 PM	Application	107 KB
setup.exe.config			1 KB
SqlSetupBootstrapper.dll			35 KB

(b) The SQLServer2017-x64-ENU-Dev.iso The Shortcut Menu

FIGURE 10A-2

Installing Microsoft SQL Server 2017



SQL Server Installation Center

Hardware and Software Requirements
View the hardware and software requirements.

Security Documentation
View the security documentation.

Online Release Notes
View the latest information about the release.

System Configuration Checker
Launch a tool to check for conditions that prevent a successful SQL Server installation.

Download Data Migration Assistant (DMA)
Data Migration Assistant (DMA) analyzes SQL Server components that are installed and identifies issues to fix either before or after you upgrade to SQL Server 2017.

Online Installation Help
Launch the online installation documentation.

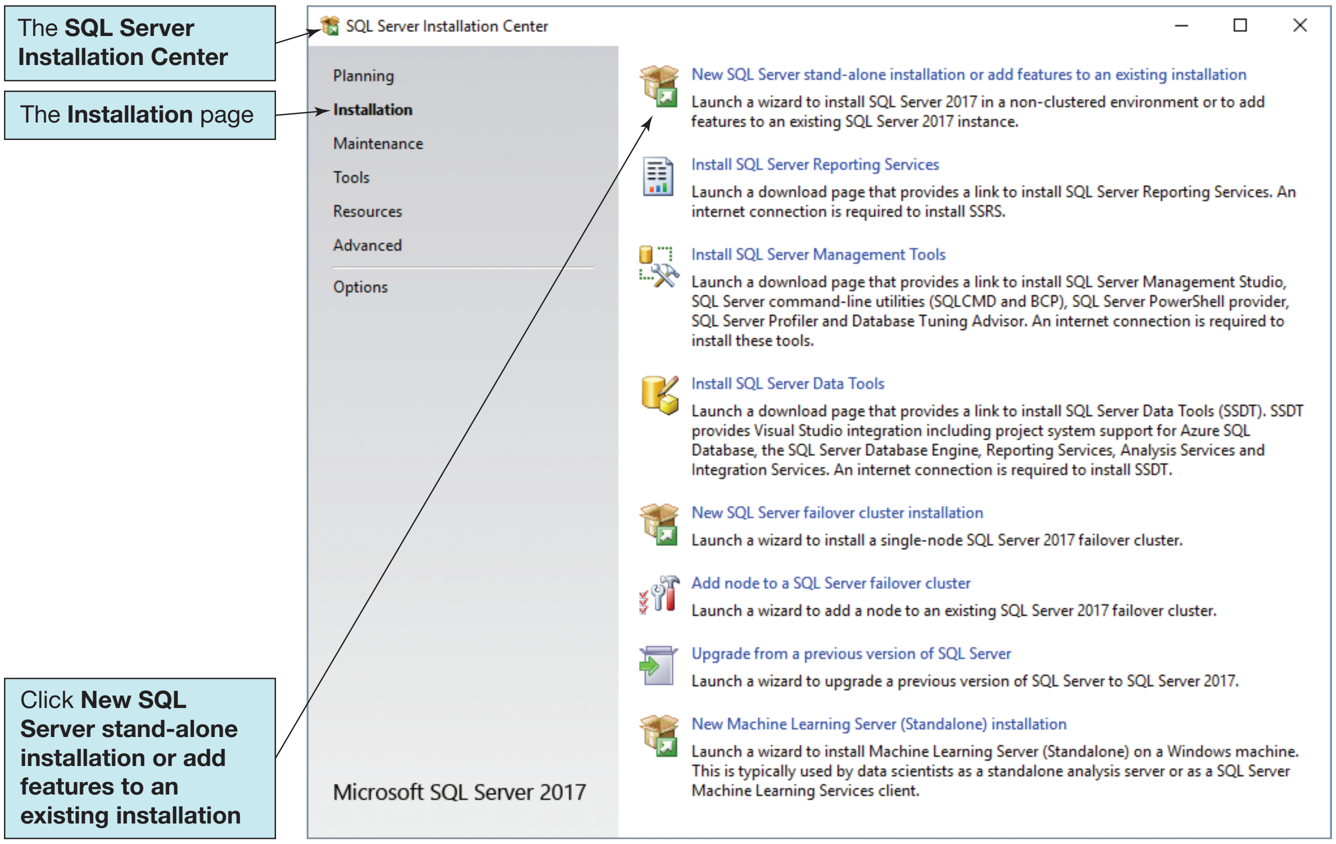
How to Get Started with SQL Server 2017 Failover Clustering
Read instructions on how to get started with SQL Server 2017 failover clustering.

Upgrade Documentation
View the document about how to upgrade to SQL Server 2017 from a previous version of SQL Server.

Download SQL Server Migration Assistant (SSMA)
SQL Server Migration Assistant (SSMA) can migrate Oracle, SAP ASE, MySQL, DB2, and Access databases to SQL Server, Microsoft Azure SQL Database, and Microsoft Azure SQL Data Warehouse. SSMA automates all aspects of migration including migration assessment analysis, schema and SQL statement conversion, data migration, and migration testing.

How to apply SQL Server updates
View the document on how to apply the appropriate product updates during a new installation or to an existing installation of SQL Server 2017.

(c) The SQL Server Installation Center Dialog Box Planning Page


The **SQL Server Installation Center**
The **Installation** page
Click **New SQL Server stand-alone installation or add features to an existing installation**

SQL Server Installation Center

Planning
Installation
Maintenance
Tools
Resources
Advanced
Options
Microsoft SQL Server 2017

New SQL Server stand-alone installation or add features to an existing installation
Launch a wizard to install SQL Server 2017 in a non-clustered environment or to add features to an existing SQL Server 2017 instance.

Install SQL Server Reporting Services
Launch a download page that provides a link to install SQL Server Reporting Services. An internet connection is required to install SSRS.

Install SQL Server Management Tools
Launch a download page that provides a link to install SQL Server Management Studio, SQL Server command-line utilities (SQLCMD and BCP), SQL Server PowerShell provider, SQL Server Profiler and Database Tuning Advisor. An internet connection is required to install these tools.

Install SQL Server Data Tools
Launch a download page that provides a link to install SQL Server Data Tools (SSDT). SSDT provides Visual Studio integration including project system support for Azure SQL Database, the SQL Server Database Engine, Reporting Services, Analysis Services and Integration Services. An internet connection is required to install SSDT.

New SQL Server failover cluster installation
Launch a wizard to install a single-node SQL Server 2017 failover cluster.

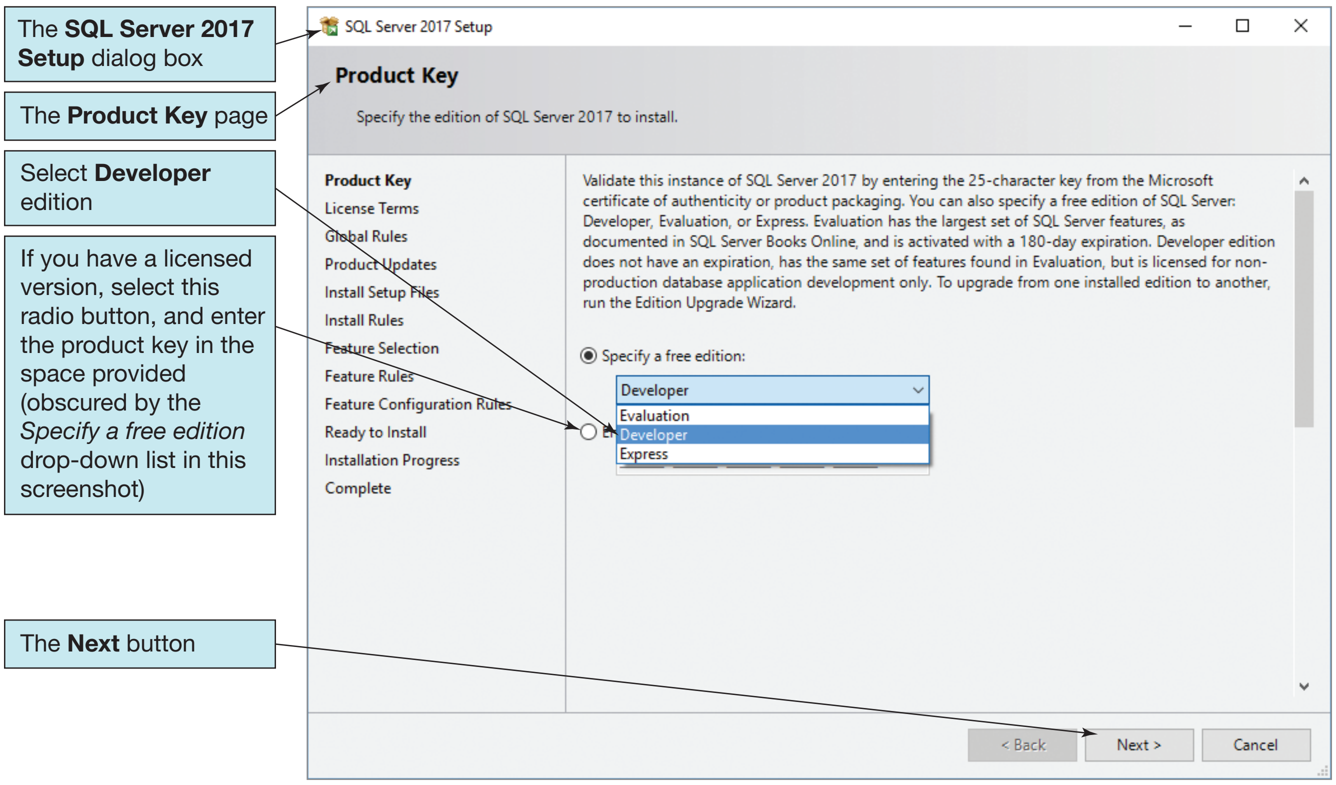
Add node to a SQL Server failover cluster
Launch a wizard to add a node to an existing SQL Server 2017 failover cluster.

Upgrade from a previous version of SQL Server
Launch a wizard to upgrade a previous version of SQL Server to SQL Server 2017.

New Machine Learning Server (Standalone) installation
Launch a wizard to install Machine Learning Server (Standalone) on a Windows machine. This is typically used by data scientists as a standalone analysis server or as a SQL Server Machine Learning Services client.

(d) The SQL Server Installation Center Dialog Box Installation Page

FIGURE 10A-2
Continued



Specify the edition of SQL Server 2017 to install.

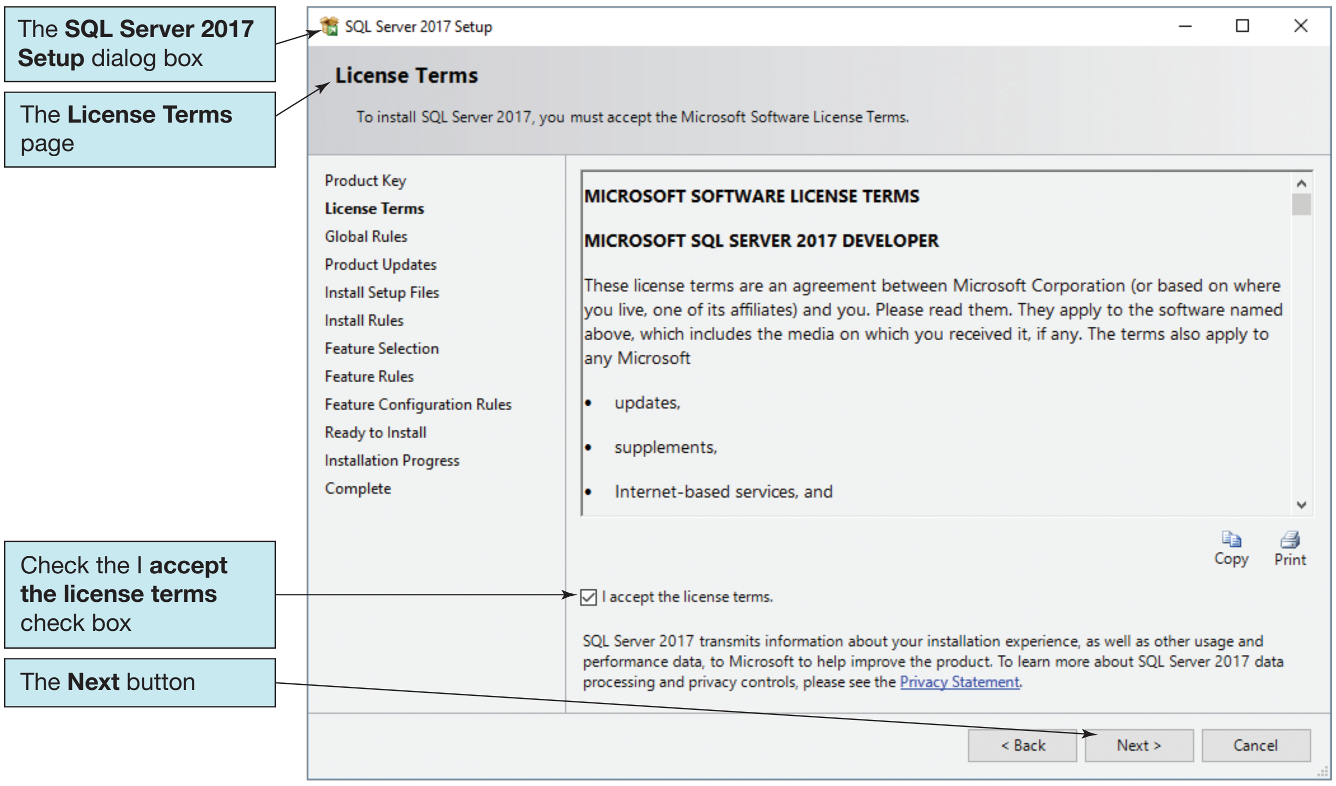
Validate this instance of SQL Server 2017 by entering the 25-character key from the Microsoft certificate of authenticity or product packaging. You can also specify a free edition of SQL Server: Developer, Evaluation, or Express. Evaluation has the largest set of SQL Server features, as documented in SQL Server Books Online, and is activated with a 180-day expiration. Developer edition does not have an expiration, has the same set of features found in Evaluation, but is licensed for non-production database application development only. To upgrade from one installed edition to another, run the Edition Upgrade Wizard.

Specify a free edition:

(e) The SQL Server 2017 Setup Product Key Page

FIGURE 10A-2
Continued

7. Select the edition of SQL Server 2017 that you are installing—we are installing the free SQL Server 2017 Developer edition. SQL Server 2017 Express Edition is also a free edition and does not need a product key. The time-limited evaluation edition may also be installed as a time-limited version. Alternately, if you have purchased a product key, type it in the **Enter the product key** text box, as shown in Figure 10A-2(e) (the text box for entering the product key is obscured by the *Specify a free edition* down-down list).
8. Click the **Next** button. The SQL Server 2017 Setup **License Terms** page is displayed. Accept the license terms, as shown in Figure 10A-2(f).
9. The installation works through the install process. If necessary, the SQL Server 2017 Setup **Global Rules** page is displayed. *This page is displayed only if there are rules that did not pass.*
10. If product updates are available, the SQL Server 2017 Setup **Product Updates** page may be displayed. This page shows the updates that will be installed as part of the installation. *If there are no updates available, this page will not be displayed.*
11. If the *Product Updates* page is displayed, click the **Next** button. The installation then installs needed setup files while displaying the SQL Server 2017 Setup **Install Setup Files** page. *This page is displayed only during the file installation process.* This page is shown in Figure 10A-2(g). The *Back* button is not available to go back to this page!
12. Click the **Next** button. The installation works through the process until the SQL Server 2017 Setup **Install Rules** page is displayed, as shown in Figure 10A-2(h). The Windows Firewall warning simply says that the Windows Firewall is enabled on the computer and is a reminder to set the correct port settings in the firewall depending upon what access you want to allow. The default settings are enough for now.



License Terms

To install SQL Server 2017, you must accept the Microsoft Software License Terms.

MICROSOFT SOFTWARE LICENSE TERMS

MICROSOFT SQL SERVER 2017 DEVELOPER

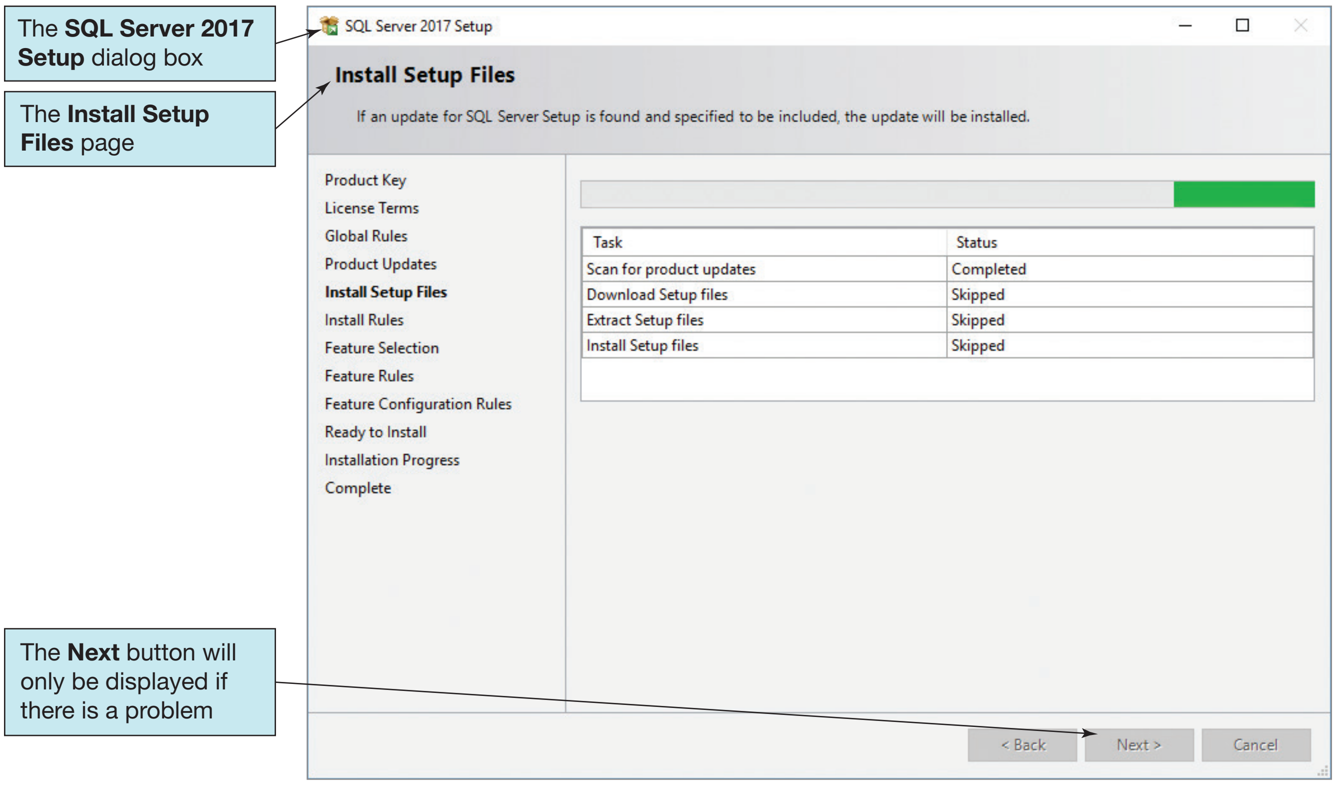
These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to the software named above, which includes the media on which you received it, if any. The terms also apply to any Microsoft

- updates,
- supplements,
- Internet-based services, and

☑ I accept the license terms.

SQL Server 2017 transmits information about your installation experience, as well as other usage and performance data, to Microsoft to help improve the product. To learn more about SQL Server 2017 data processing and privacy controls, please see the Privacy Statement.

(f) The SQL Server 2017 Setup License Terms Page



Install Setup Files

If an update for SQL Server Setup is found and specified to be included, the update will be installed.

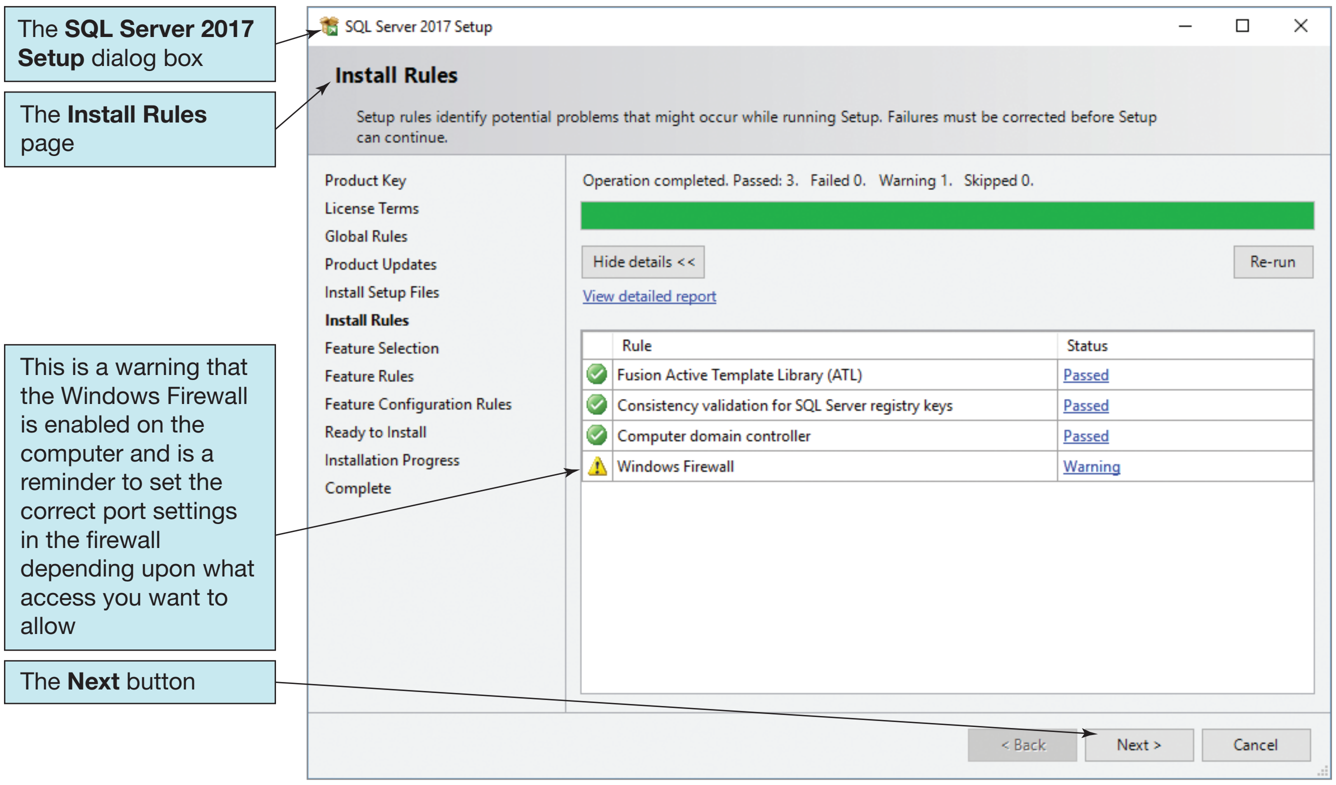
Task	Status
Scan for product updates	Completed
Download Setup files	Skipped
Extract Setup files	Skipped
Install Setup files	Skipped

(g) The SQL Server 2017 Setup Install Setup Files Page

FIGURE 10A-2
Continued

13. Click the **Next** button. The installation works through the process until the SQL Server 2017 Setup **Feature Selection** page is displayed. Click the **Select All** button to install all features, as shown in Figure 10A-2(i), then *uncheck the checkboxes for the following features*:
    - Instance Features: SQL Server Replication
    - Shared Features: Scale Out Master, Scale Out Worker, Distributed Replay Controller, Distributed Replay Client, Master Data Services.

    Note that this selection resets the installation outline in the left-hand window and more steps are displayed!
14. Click the **Next** button. The installation works through the SQL Server 2017 Setup **Feature Rules** page. *Note that if there are no problems, this page will not be displayed.* If you installed all the .NET Framework products, the Java JRE, and the Visual Studio C++ Redistributable files specified earlier, the installation will pass these tests.
    - If you have any failed tests in feature rules, cancel the SQL Server installation, install all the software prerequisites listed earlier, and restart the installation. Note that Polybase can only be installed on one instance of SQL Server, and if you already have another instance installed with Polybase, do *not* install it for SQL Server 2017 Developer!
15. If you are on the **Feature Rules** page, click the **Next** button. The installation works through the process until the SQL Server 2017 Setup **Instance Configuration** page is displayed, as shown in Figure 10A-2(j). The term **instance** refers to the fact that more than one copy of SQL Server 2017 can be installed on one computer, and each separate installation is referred to as an instance. For example, it is possible (and quite easy) to install an instance of the Developer


The **SQL Server 2017 Setup** dialog box
The **Install Rules** page
This is a warning that the Windows Firewall is enabled on the computer and is a reminder to set the correct port settings in the firewall depending upon what access you want to allow
The **Next** button
SQL Server 2017 Setup
Product Key
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Feature Configuration Rules
Ready to Install
Installation Progress
Complete
Hide details <<
Re-run
View detailed report
< Back
Next >
Cancel

Install Rules

Setup rules identify potential problems that might occur while running Setup. Failures must be corrected before Setup can continue.

Operation completed. Passed: 3. Failed 0. Warning 1. Skipped 0.

Rule	Status
Fusion Active Template Library (ATL)	Passed
Consistency validation for SQL Server registry keys	Passed
Computer domain controller	Passed
Windows Firewall	Warning

(h) The SQL Server 2017 Setup Install Rules Page

FIGURE 10A-2

Continued

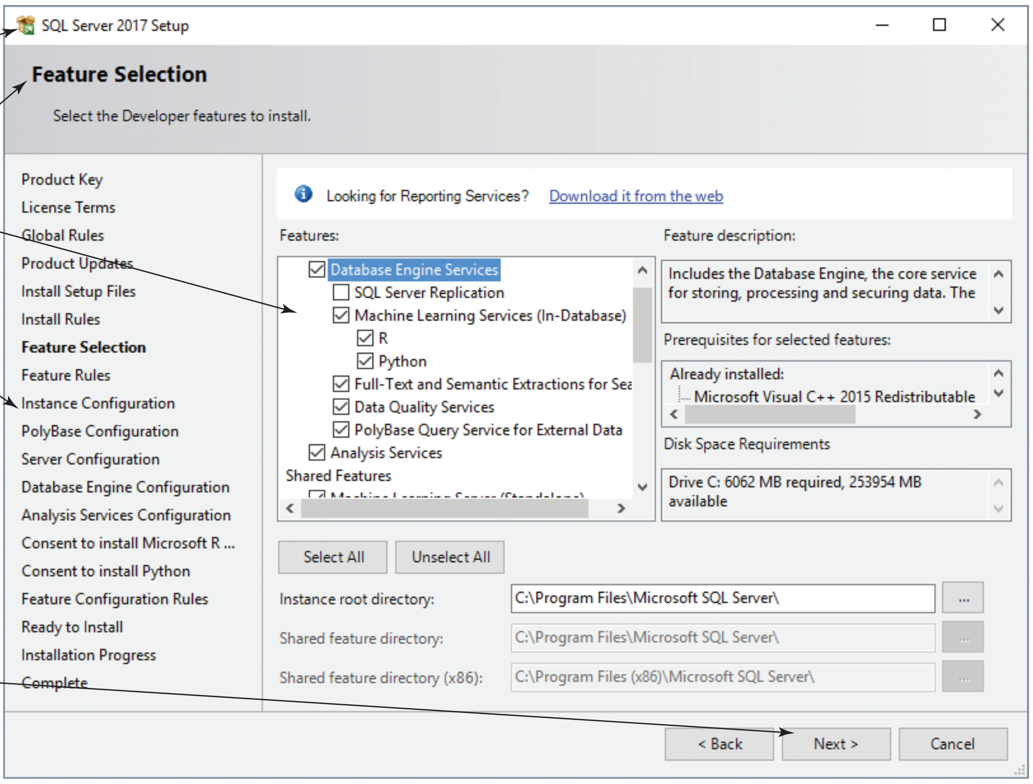
The **SQL Server 2017 Setup** dialog box

The **Feature Selection** page

Note that only some features are selected for installation

Note that additional installation steps now appear depending upon which features are selected for installation

The **Next** button


SQL Server 2017 Setup

Feature Selection

Select the Developer features to install.

Product Key
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Instance Configuration
PolyBase Configuration
Server Configuration
Database Engine Configuration
Analysis Services Configuration
Consent to install Microsoft R ...
Consent to install Python
Feature Configuration Rules
Ready to Install
Installation Progress
Complete

Looking for Reporting Services? Download it from the web

Features:
- [✓] Database Engine Services
  - [ ] SQL Server Replication
  - [✓] Machine Learning Services (In-Database)
    - [✓] R
    - [✓] Python
  - [✓] Full-Text and Semantic Extractions for Sea
  - [✓] Data Quality Services
  - [✓] PolyBase Query Service for External Data
- [✓] Analysis Services

Shared Features

Feature description:
Includes the Database Engine, the core service for storing, processing and securing data. The

Prerequisites for selected features:
Already installed:
Microsoft Visual C++ 2015 Redistributable

Disk Space Requirements
Drive C: 6062 MB required, 253954 MB available

Select All | Unselect All

Instance root directory: C:\Program Files\Microsoft SQL Server\
Shared feature directory: C:\Program Files\Microsoft SQL Server\
Shared feature directory (x86): C:\Program Files (x86)\Microsoft SQL Server\

< Back | Next > | Cancel

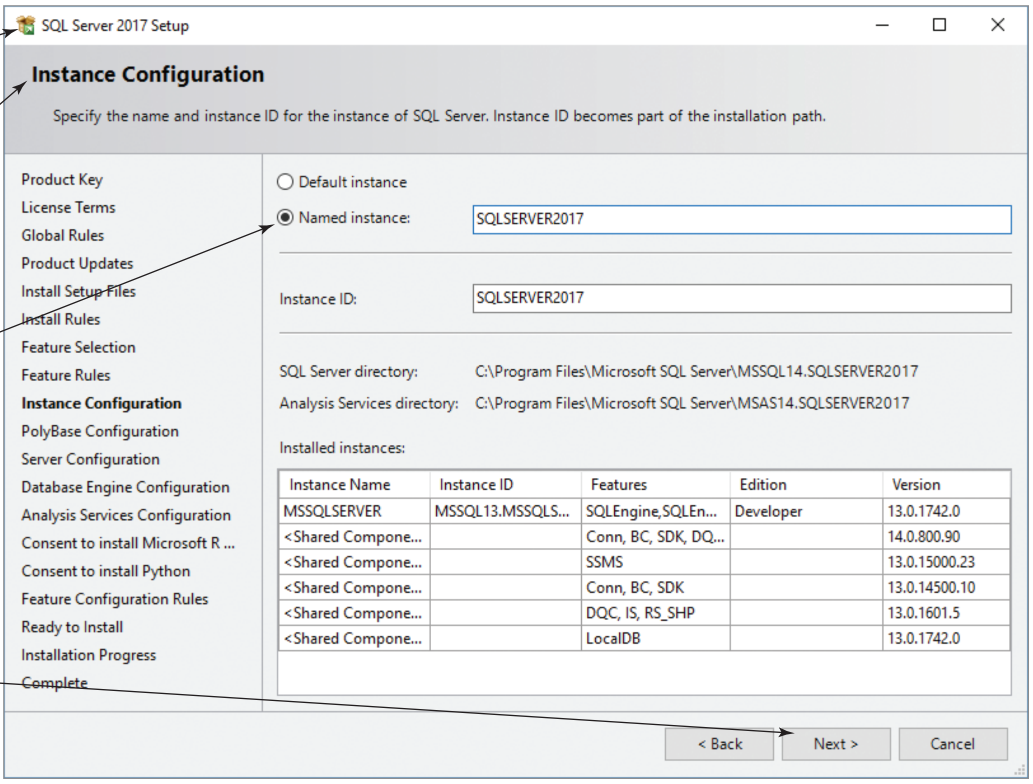
(i) The SQL Server 2017 Setup Feature Selection Page

The **SQL Server 2017 Setup** dialog box

The **Instance Configuration** page

The first instance of SQL Server on a computer is the **default instance** (always named **MSSQLSERVER**)—additional instances are **named instances** (here we use the name **SQLSERVER2017**)

The **Next** button


SQL Server 2017 Setup

Instance Configuration

Specify the name and instance ID for the instance of SQL Server. Instance ID becomes part of the installation path.

Product Key
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Instance Configuration
PolyBase Configuration
Server Configuration
Database Engine Configuration
Analysis Services Configuration
Consent to install Microsoft R ...
Consent to install Python
Feature Configuration Rules
Ready to Install
Installation Progress
Complete

( ) Default instance
(●) Named instance: SQLSERVER2017

Instance ID: SQLSERVER2017

SQL Server directory: C:\Program Files\Microsoft SQL Server\MSSQL14.SQLSERVER2017
Analysis Services directory: C:\Program Files\Microsoft SQL Server\MSAS14.SQLSERVER2017

Installed instances:

Instance Name	Instance ID	Features	Edition	Version
MSSQLSERVER	MSSQL13.MSSQLS...	SQLEngine,SQLEn...	Developer	13.0.1742.0
<Shared Compone...		Conn, BC, SDK, DQ...		14.0.800.90
<Shared Compone...		SSMS		13.0.15000.23
<Shared Compone...		Conn, BC, SDK		13.0.14500.10
<Shared Compone...		DQC, IS, RS_SHP		13.0.1601.5
<Shared Compone...		LocalDB		13.0.1742.0

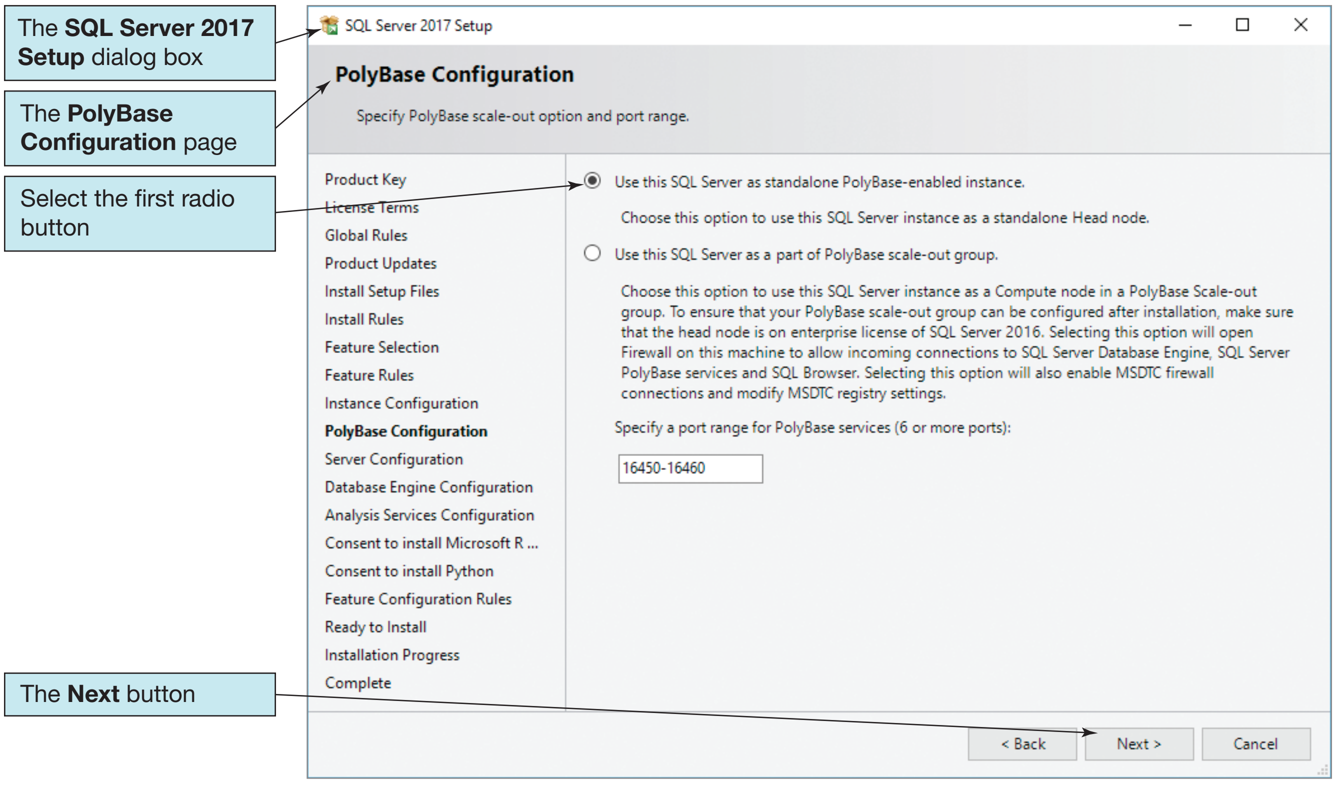
< Back | Next > | Cancel

(j) The SQL Server 2017 Setup Instance Configuration Page

FIGURE 10A-2
Continued

version of SQL Server 2017 (functionally equivalent to the Enterprise version but intended for a single user) and an instance of SQL Server 2017 Express Edition on the same workstation. If you are installing your first instance of the Enterprise, Standard, or Developer edition of SQL Server 2017, you are installing the **default instance**. The Default instance radio button will be selected and the instance name of *MSSQLSERVER* will be used. If you are installing an SQL Server 2017 Express Edition, the Named instance radio button will be selected and the instance name of *SQLEXPRESS* will be used. If you are installing an additional instance of SQL Server, as we are here, this is a **named instance**. We are using the Instance ID *SQLSERVER2017*.

16. Click the **Next** button. The SQL Server 2017 Setup **PolyBase Configuration** page is displayed, as shown in Figure 10A-2(k). The default settings are correct, so click the **Next** button.
17. The installation works through the process until the SQL Server 2017 Setup **Server Configuration** page is displayed, as shown in Figure 10A-2(l). Change (1) the Startup Type for **SQL Server Agent** from Manual to **Automatic** and, if necessary, (2) the **SQL Server Browser** from Disabled to **Automatic**, as shown in Figure 10A-2(l).
18. Click the **Next** button. The installation works through the process until the SQL Server 2017 Setup **Database Engine Configuration** page is displayed, as shown in Figure 10A-2(m). Change the **Authentication Mode** to **Mixed Mode**, and enter and confirm a **password** for the SQL Server system administrator (sa) [Be sure you remember this password!]. Click the **Add Current User** button to add *yourself* as an SQL Server administrator. All the other settings are correct.
19. Click the **Next** button. The installation works through the process until the SQL Server 2017 Setup **Analysis Services Configuration** page is displayed, as shown in Figure 10A-2(n). Select **Multidimensional and Data Mining Mode**, and click the **Add Current User** button to give yourself administrative permissions for Analysis Services. All the other settings are correct.
20. Click the **Next** button. The installation works through the process until the SQL Server 2017 Setup **Consent to install Microsoft R Open** page is displayed, as shown in Figure 10A-2(o). Microsoft R Open is a statistical package that is used for data analysis. This page is displayed only if it is supported by the version of SQL Server 2017 you are installing. If this page is displayed, click the **Accept** button. All the other settings are correct.
21. Click the **Next** button. The installation works through the process until the SQL Server 2017 Setup **Consent to install Python** page is displayed, as shown in Figure 10A-2(p). Python is a programming language that is used for data analysis and other application development. This page is displayed only if it is supported by the version of SQL Server 2017 you are installing. If this page is displayed, click the **Accept** button. All the other settings are correct.
22. Click the **Next** button. The installation works through the process until the SQL Server 2017 Setup **Feature Configuration Rules** page is displayed. *This page is displayed only if there are rules that did not pass.* To see this page, press the *Back* button, and then the *View Details* button. All rules should pass their tests.
23. If the *Feature Configuration Rules* page is displayed, click the **Next** button. The **Ready to Install** page is displayed, as shown in Figure 10A-2(q). This page summarizes the installation and displays the installation settings so that you can check to be sure they are correct. The settings shown here are correct for our installation process.
24. Click the **Install** button. SQL Server 2017 is installed (this will take a while, so be patient), as shown in Figure 10A-2(r).
25. When the installation is complete, the **Complete** page is displayed, together with a **Computer restart required** dialog box, as shown in Figure 10A-2(s). This means



PolyBase Configuration

Specify PolyBase scale-out option and port range.

[✓ ●] Use this SQL Server as standalone PolyBase-enabled instance.

Choose this option to use this SQL Server instance as a standalone Head node.

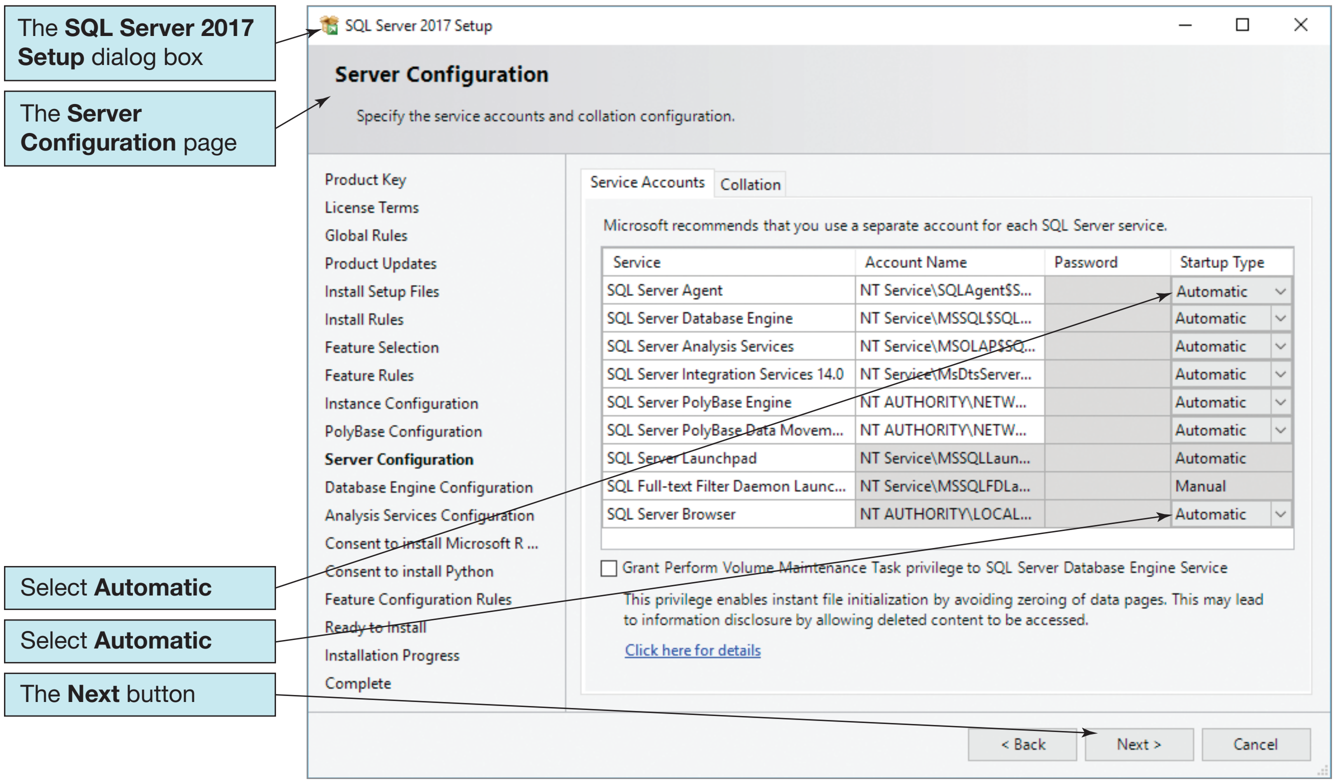
○ Use this SQL Server as a part of PolyBase scale-out group.

Choose this option to use this SQL Server instance as a Compute node in a PolyBase Scale-out group. To ensure that your PolyBase scale-out group can be configured after installation, make sure that the head node is on enterprise license of SQL Server 2016. Selecting this option will open Firewall on this machine to allow incoming connections to SQL Server Database Engine, SQL Server PolyBase services and SQL Browser. Selecting this option will also enable MSDTC firewall connections and modify MSDTC registry settings.

Specify a port range for PolyBase services (6 or more ports):

16450-16460

(k) The SQL Server 2017 Setup PolyBase Configuration Page



Server Configuration

Specify the service accounts and collation configuration.

Microsoft recommends that you use a separate account for each SQL Server service.

Service	Account Name	Password	Startup Type
SQL Server Agent	NT Service\SQLAgent$S...		Automatic
SQL Server Database Engine	NT Service\MSSQL$SQL...		Automatic
SQL Server Analysis Services	NT Service\MSOLAP$SQ...		Automatic
SQL Server Integration Services 14.0	NT Service\MsDtsServer...		Automatic
SQL Server PolyBase Engine	NT AUTHORITY\NETW...		Automatic
SQL Server PolyBase Data Movem...	NT AUTHORITY\NETW...		Automatic
SQL Server Launchpad	NT Service\MSSQLLaun...		Automatic
SQL Full-text Filter Daemon Launc...	NT Service\MSSQLFDLa...		Manual
SQL Server Browser	NT AUTHORITY\LOCAL...		Automatic

☐ Grant Perform Volume Maintenance Task privilege to SQL Server Database Engine Service

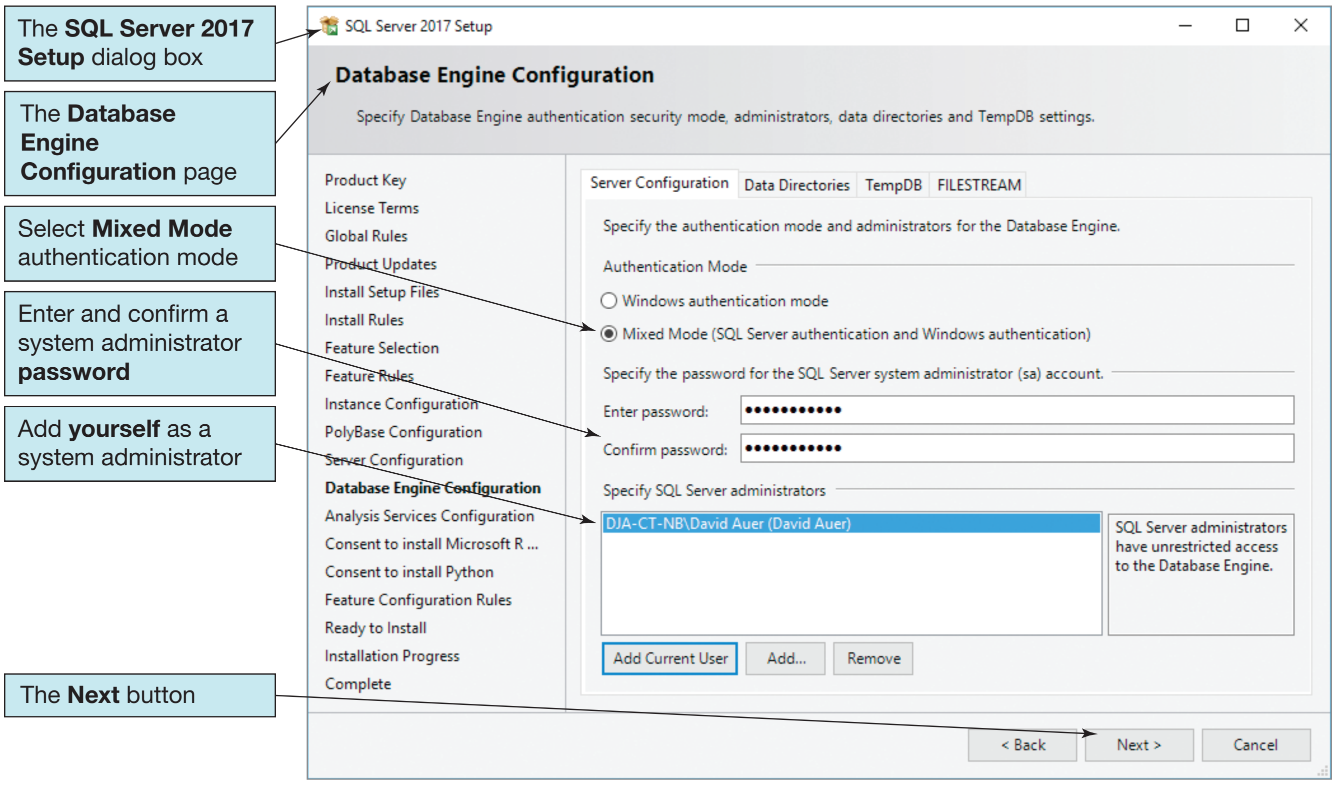
This privilege enables instant file initialization by avoiding zeroing of data pages. This may lead to information disclosure by allowing deleted content to be accessed.

Click here for details

(l) The SQL Server 2017 Setup Server Configuration Page

FIGURE 10A-2

Continued


The **SQL Server 2017 Setup** dialog box
The **Database Engine Configuration** page
Select **Mixed Mode** authentication mode
Enter and confirm a system administrator **password**
Add **yourself** as a system administrator
The **Next** button

SQL Server 2017 Setup

Database Engine Configuration

Specify Database Engine authentication security mode, administrators, data directories and TempDB settings.

Product Key
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Instance Configuration
PolyBase Configuration
Server Configuration
Database Engine Configuration
Analysis Services Configuration
Consent to install Microsoft R ...
Consent to install Python
Feature Configuration Rules
Ready to Install
Installation Progress
Complete

Server Configuration | Data Directories | TempDB | FILESTREAM

Specify the authentication mode and administrators for the Database Engine.

Authentication Mode

( ) Windows authentication mode
(●) Mixed Mode (SQL Server authentication and Windows authentication)

Specify the password for the SQL Server system administrator (sa) account.

Enter password: ●●●●●●●●●●●
Confirm password: ●●●●●●●●●●●

Specify SQL Server administrators

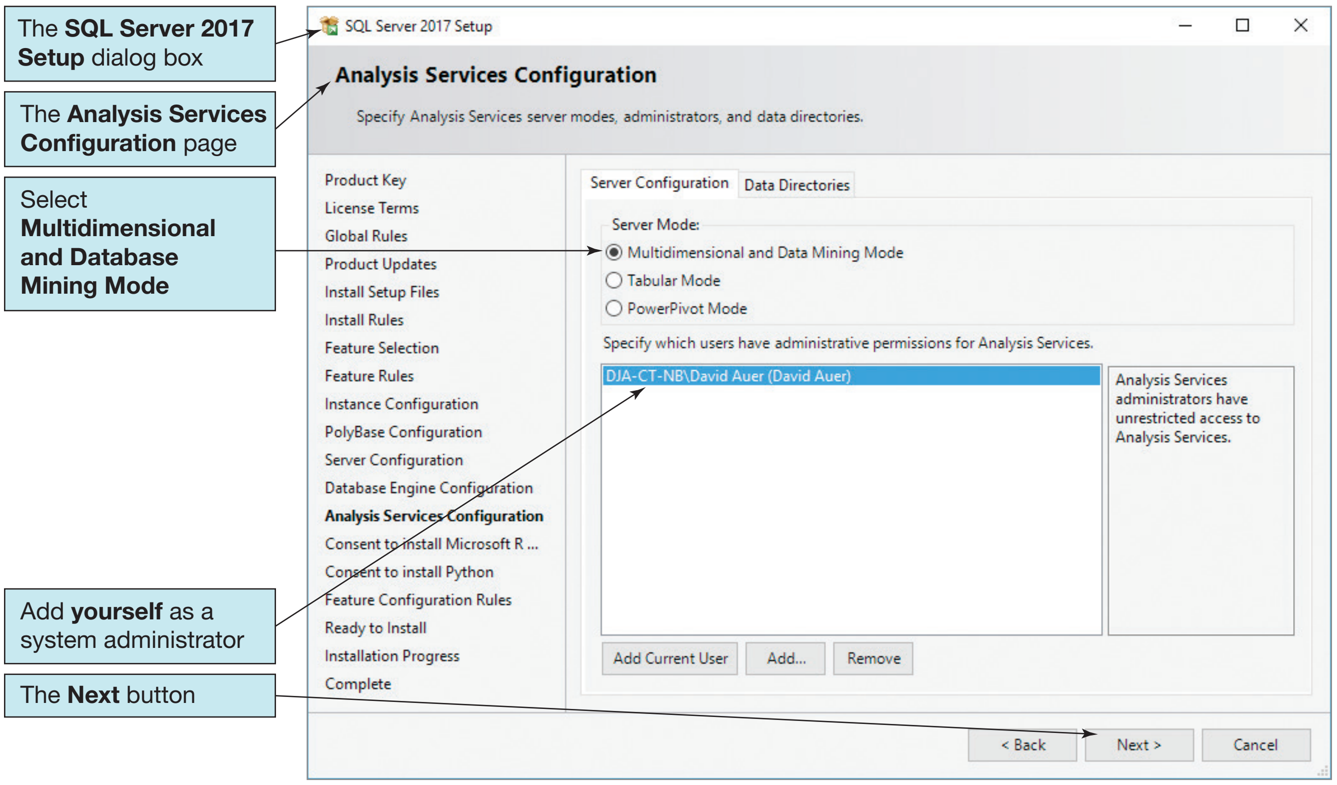
DJA-CT-NB\David Auer (David Auer)

SQL Server administrators have unrestricted access to the Database Engine.

Add Current User | Add... | Remove

< Back | Next > | Cancel

(m) The SQL Server 2017 Setup Database Engine Configuration Page


The **SQL Server 2017 Setup** dialog box
The **Analysis Services Configuration** page
Select **Multidimensional and Database Mining Mode**
Add **yourself** as a system administrator
The **Next** button

SQL Server 2017 Setup

Analysis Services Configuration

Specify Analysis Services server modes, administrators, and data directories.

Product Key
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Instance Configuration
PolyBase Configuration
Server Configuration
Database Engine Configuration
Analysis Services Configuration
Consent to install Microsoft R ...
Consent to install Python
Feature Configuration Rules
Ready to Install
Installation Progress
Complete

Server Configuration | Data Directories

Server Mode:

(●) Multidimensional and Data Mining Mode
( ) Tabular Mode
( ) PowerPivot Mode

Specify which users have administrative permissions for Analysis Services.

DJA-CT-NB\David Auer (David Auer)

Analysis Services administrators have unrestricted access to Analysis Services.

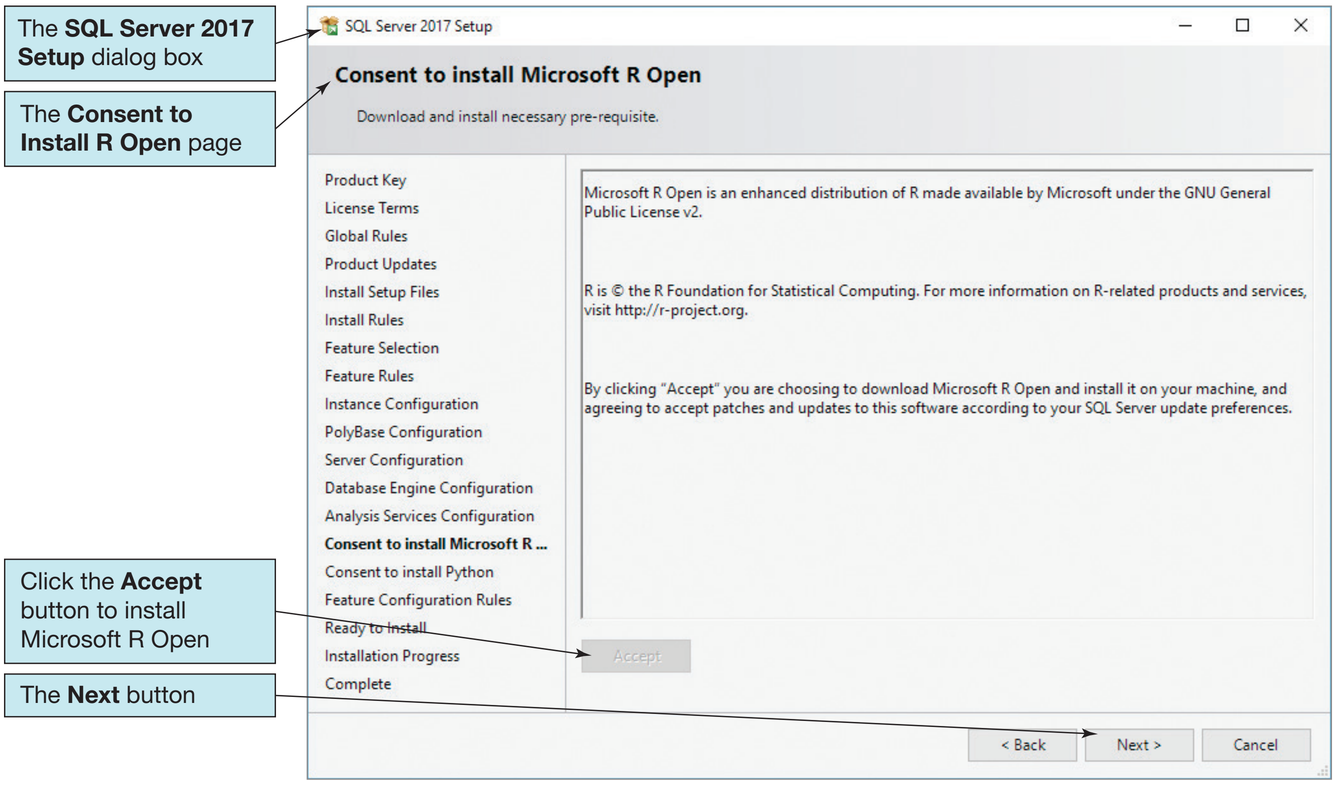
Add Current User | Add... | Remove

< Back | Next > | Cancel

(n) The SQL Server 2017 Setup Analysis Services Configuration Page

FIGURE 10A-2

Continued



Consent to install Microsoft R Open

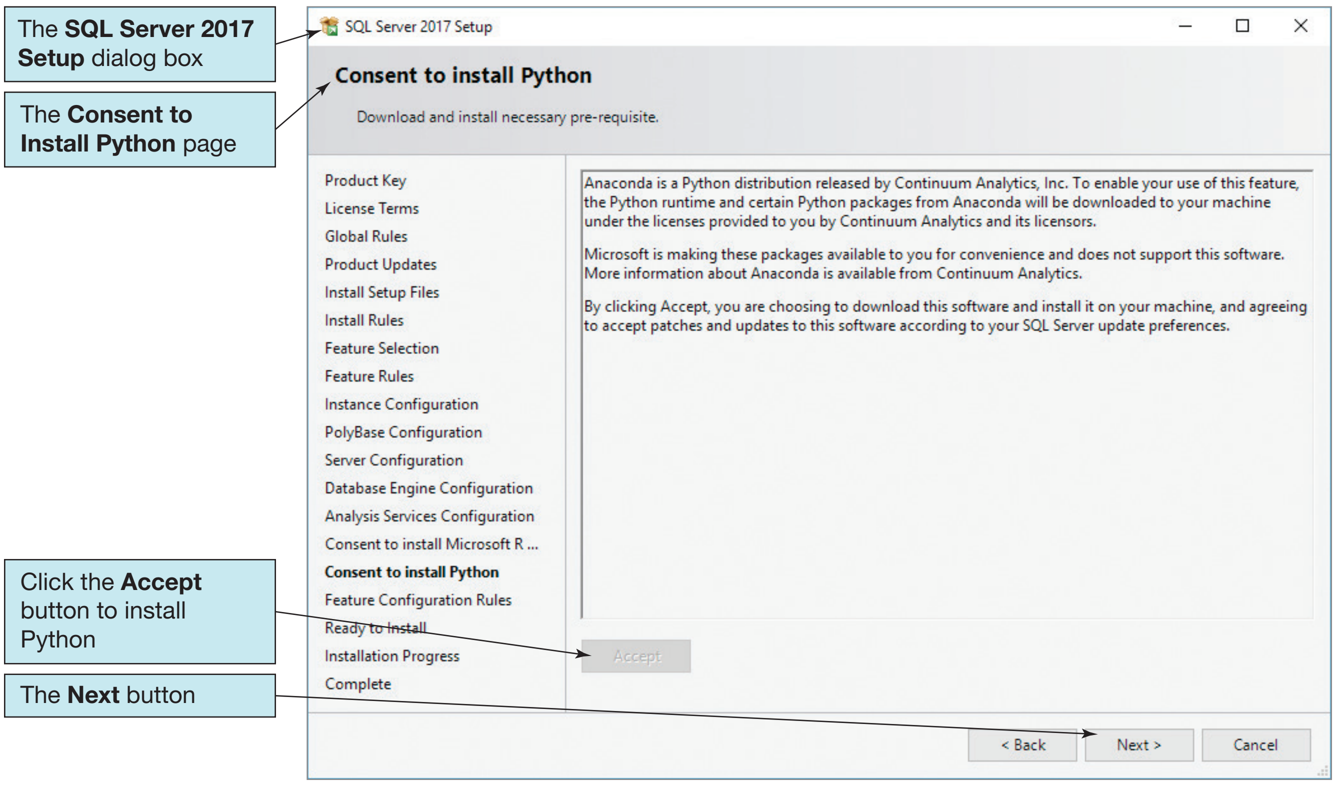
Download and install necessary pre-requisite.

Microsoft R Open is an enhanced distribution of R made available by Microsoft under the GNU General Public License v2.

R is © the R Foundation for Statistical Computing. For more information on R-related products and services, visit http://r-project.org.

By clicking "Accept" you are choosing to download Microsoft R Open and install it on your machine, and agreeing to accept patches and updates to this software according to your SQL Server update preferences.

(o) The SQL Server 2017 Setup Consent to Install Microsoft Open R Page



Consent to install Python

Download and install necessary pre-requisite.

Anaconda is a Python distribution released by Continuum Analytics, Inc. To enable your use of this feature, the Python runtime and certain Python packages from Anaconda will be downloaded to your machine under the licenses provided to you by Continuum Analytics and its licensors.

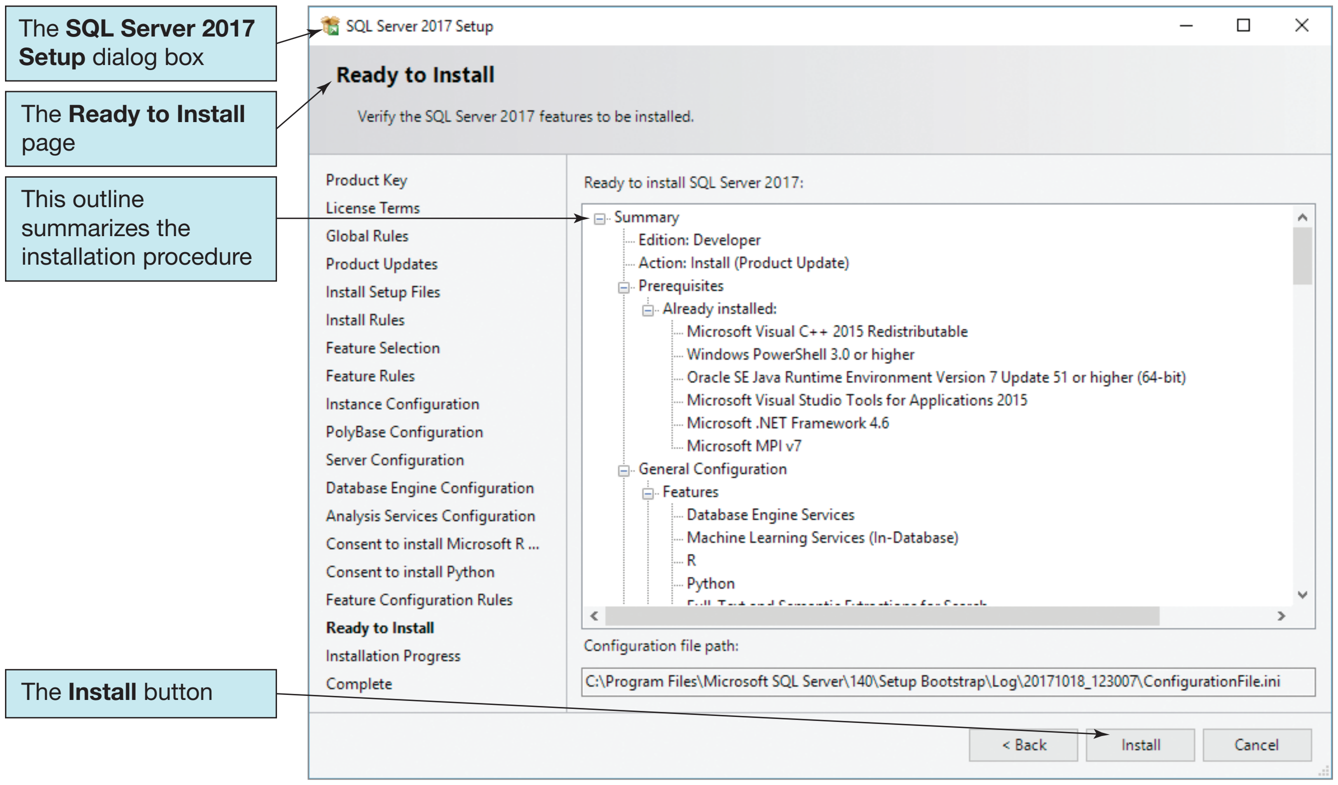
Microsoft is making these packages available to you for convenience and does not support this software. More information about Anaconda is available from Continuum Analytics.

By clicking Accept, you are choosing to download this software and install it on your machine, and agreeing to accept patches and updates to this software according to your SQL Server update preferences.

(p) The SQL Server 2017 Setup Consent to Install Python Page

FIGURE 10A-2

Continued


The SQL Server 2017 Setup dialog box
The Ready to Install page
This outline summarizes the installation procedure
The Install button

SQL Server 2017 Setup

Ready to Install

Verify the SQL Server 2017 features to be installed.

Product Key
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Instance Configuration
PolyBase Configuration
Server Configuration
Database Engine Configuration
Analysis Services Configuration
Consent to install Microsoft R ...
Consent to install Python
Feature Configuration Rules
Ready to Install
Installation Progress
Complete

Ready to install SQL Server 2017:

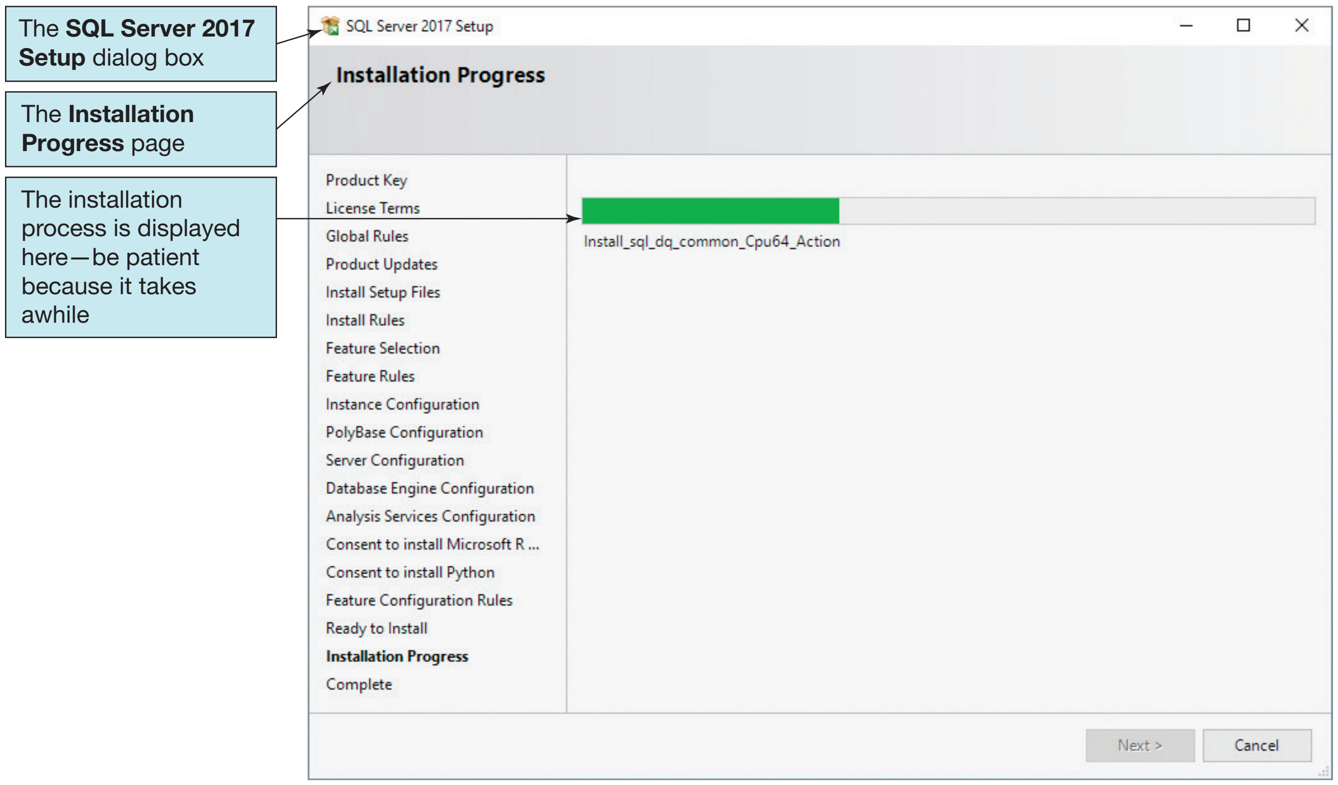
```
Summary
    Edition: Developer
    Action: Install (Product Update)
    Prerequisites
        Already installed:
            Microsoft Visual C++ 2015 Redistributable
            Windows PowerShell 3.0 or higher
            Oracle SE Java Runtime Environment Version 7 Update 51 or higher (64-bit)
            Microsoft Visual Studio Tools for Applications 2015
            Microsoft .NET Framework 4.6
            Microsoft MPI v7
    General Configuration
        Features
            Database Engine Services
            Machine Learning Services (In-Database)
            R
            Python
```

Configuration file path:

C:\Program Files\Microsoft SQL Server\140\Setup Bootstrap\Log\20171018_123007\ConfigurationFile.ini

< Back | Install | Cancel

(q) The SQL Server 2017 Setup Ready to Install Page


The SQL Server 2017 Setup dialog box
The Installation Progress page
The installation process is displayed here—be patient because it takes awhile

SQL Server 2017 Setup

Installation Progress

Product Key
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Instance Configuration
PolyBase Configuration
Server Configuration
Database Engine Configuration
Analysis Services Configuration
Consent to install Microsoft R ...
Consent to install Python
Feature Configuration Rules
Ready to Install
Installation Progress
Complete

Install_sql_dq_common_Cpu64_Action

Next > | Cancel

(r) The SQL Server 2017 Setup Installation Progress Page

FIGURE 10A-2

Continued

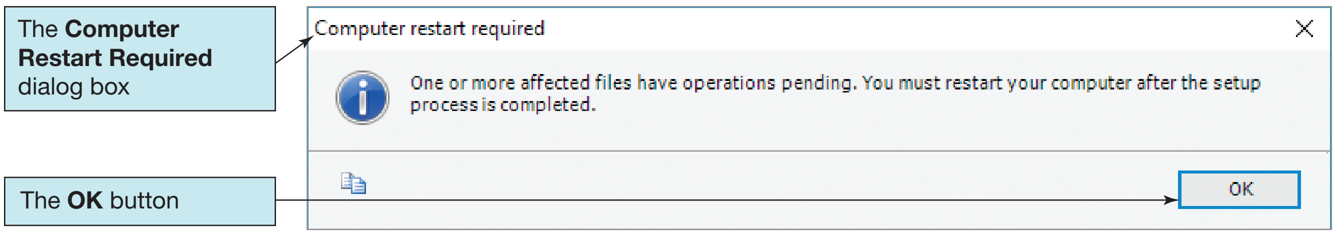
that after you close the SQL Server 2017 Setup dialog box, you will need to close all other programs and restart your computer below proceeding with the installation of additional SQL Server 2017 related programs. Click the **OK** button.

26. The **Complete** page is displayed, as shown in Figure 10A-2(t).

27. Click the **Close** button to close the SQL Server 2017 Setup dialog box.

Close the **SQL Server Installation Center Dialog Box** and any other open programs on your computer, and then restart it. After the restart, reopen the SQL Server Installation Center as described earlier in this chapter. We will need it to install Microsoft SQL Server 2017 Reporting Services and the administrative utilities we use with SQL Server 2017.

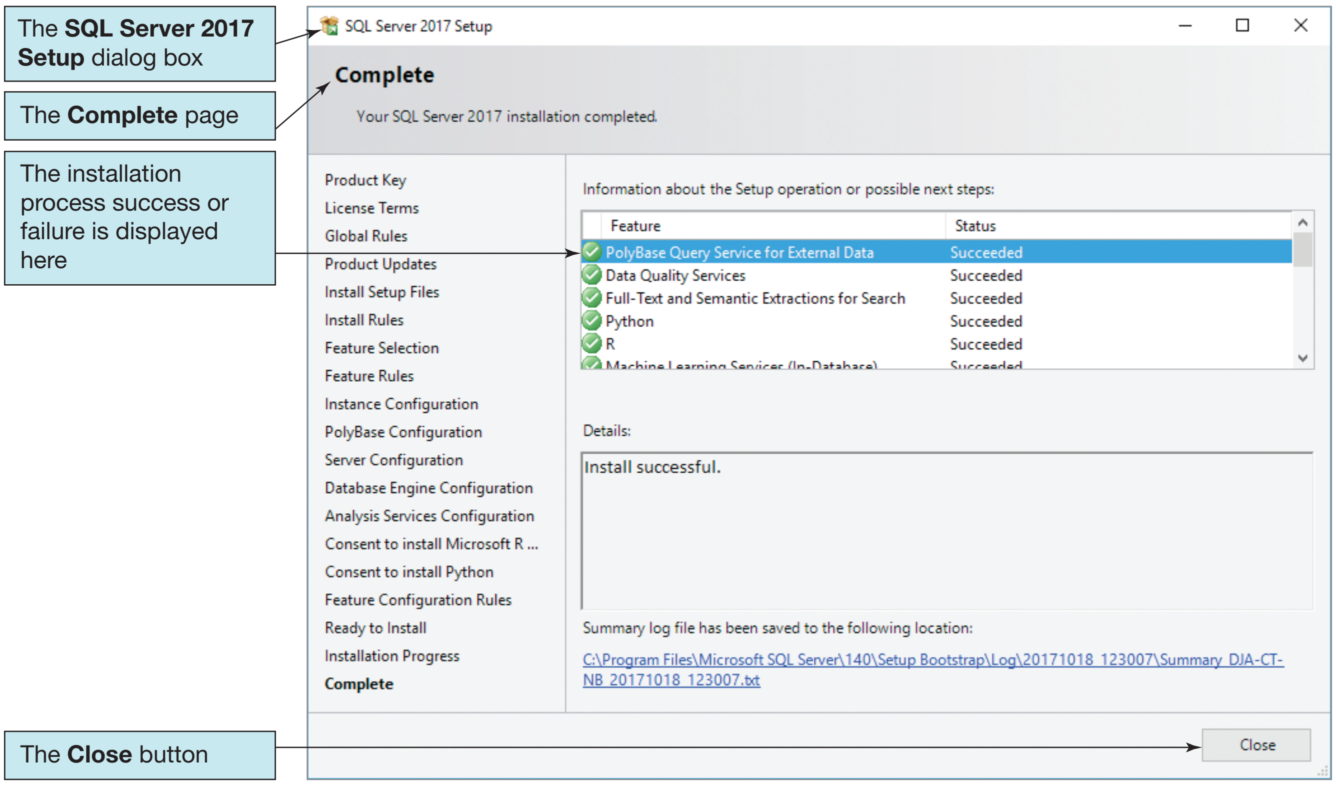
After you have installed SQL Server 2017, you should check for the latest service packs and patches using Windows Update to make sure your installation is as secure as possible. Additional patches may not automatically installed by the Windows 10 Update feature. Specifically, regularly check for, download and install the **SQL Server 2017 for Microsoft Windows Latest Cumulative Update** from *https://www.microsoft.com/en-us/download/details.aspx?id=56128*. As this is being written (January, 2018), **Cumulative**



Computer restart required

One or more affected files have operations pending. You must restart your computer after the setup process is completed.

(s) The Computer Restart Required Dialog Box


The **SQL Server 2017 Setup** dialog box
The **Complete** page
The installation process success or failure is displayed here
The **Close** button
SQL Server 2017 Setup
Close

Complete

Your SQL Server 2017 installation completed.

Product Key
License Terms
Global Rules
Product Updates
Install Setup Files
Install Rules
Feature Selection
Feature Rules
Instance Configuration
PolyBase Configuration
Server Configuration
Database Engine Configuration
Analysis Services Configuration
Consent to install Microsoft R ...
Consent to install Python
Feature Configuration Rules
Ready to Install
Installation Progress
Complete

Information about the Setup operation or possible next steps:

Feature	Status
✓ PolyBase Query Service for External Data	Succeeded
✓ Data Quality Services	Succeeded
✓ Full-Text and Semantic Extractions for Search	Succeeded
✓ Python	Succeeded
✓ R	Succeeded
✓ Machine Learning Services (In-Database)	Succeeded

Details:

Install successful.

Summary log file has been saved to the following location:

C:\Program Files\Microsoft SQL Server\140\Setup Bootstrap\Log\20171018_123007\Summary_DJA-CT-NB_20171018_123007.txt

(t) The SQL Server 2017 Setup Complete Page

FIGURE 10A-2
Continued

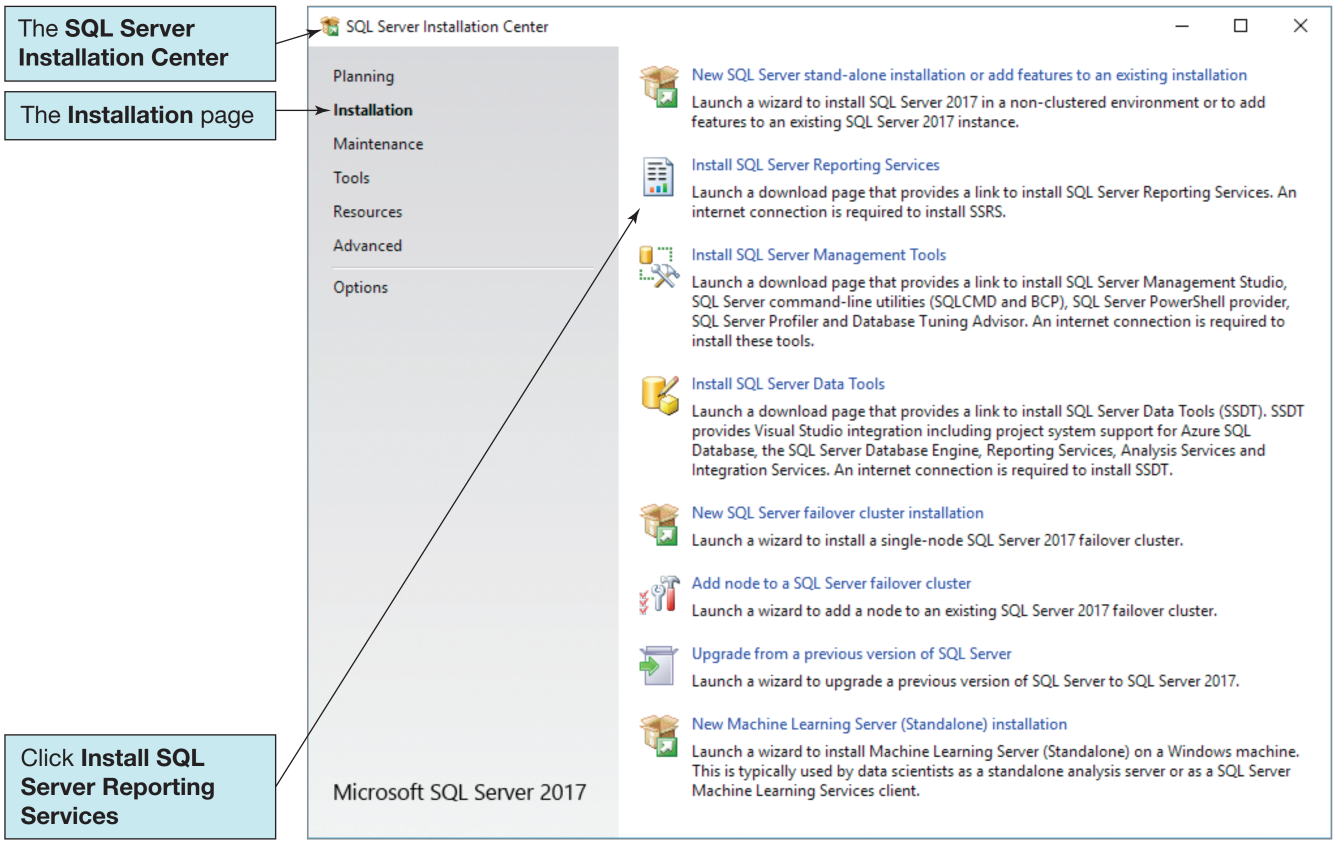
Update Package 3 for SQL Server 2017 is available. For information on protecting SQL Server 2017 from the **Meltdown** and **Spectre** critical bugs, see Microsoft KB 4073225 "SQL Server guidance to protect against speculative execution side-channel vulnerabilities" at *https://support.microsoft.com/en-us/help/4073225/guidance-for-sql-server*.

Installing Microsoft SQL Server 2017 Reporting Services

One tool of business intelligence (BI) systems, **reporting systems** sort, filter, group, and make basic calculations on database data. Microsoft SQL Server 2017 Reporting Services is the tool used in SQL Server in support of reporting systems, adding the capability of producing reports based on database data to SQL Server. Previously included in the SQL Server installation package itself, Microsoft SQL Server 2017 Reporting Services must now be downloaded and installed as a separate package. The installation package, named *SQLServer-ReportingServices.exe*, can be downloaded from *https://www.microsoft.com/en-us/download/details.aspx?id=55252*. This URL is available from a link on the SQL Server Installation Center Dialog box.

Installing Microsoft SQL Server 2017 Reporting Services

1. As noted at the end of the previous section, you need to restart the SQL Server Installation Center, and then open the Installation page after rebooting your computer during the installation of SQL Server 2017 Developer edition. To start the actual Microsoft SQL Server 2017 Reporting Services installation process, click on **Install SQL Server Reporting Services**, as shown in Figure 10A-3(a).
2. A Web browser is launched and displays the **Microsoft SQL Server 2017 Reporting Services** Web page, as shown in Figure 10A-3(b). Click the Download button.


The SQL Server Installation Center
The Installation page
Click Install SQL Server Reporting Services
SQL Server Installation Center
Planning
Installation
Maintenance
Tools
Resources
Advanced
Options
Microsoft SQL Server 2017

New SQL Server stand-alone installation or add features to an existing installation
Launch a wizard to install SQL Server 2017 in a non-clustered environment or to add features to an existing SQL Server 2017 instance.

Install SQL Server Reporting Services
Launch a download page that provides a link to install SQL Server Reporting Services. An internet connection is required to install SSRS.

Install SQL Server Management Tools
Launch a download page that provides a link to install SQL Server Management Studio, SQL Server command-line utilities (SQLCMD and BCP), SQL Server PowerShell provider, SQL Server Profiler and Database Tuning Advisor. An internet connection is required to install these tools.

Install SQL Server Data Tools
Launch a download page that provides a link to install SQL Server Data Tools (SSDT). SSDT provides Visual Studio integration including project system support for Azure SQL Database, the SQL Server Database Engine, Reporting Services, Analysis Services and Integration Services. An internet connection is required to install SSDT.

New SQL Server failover cluster installation
Launch a wizard to install a single-node SQL Server 2017 failover cluster.

Add node to a SQL Server failover cluster
Launch a wizard to add a node to an existing SQL Server 2017 failover cluster.

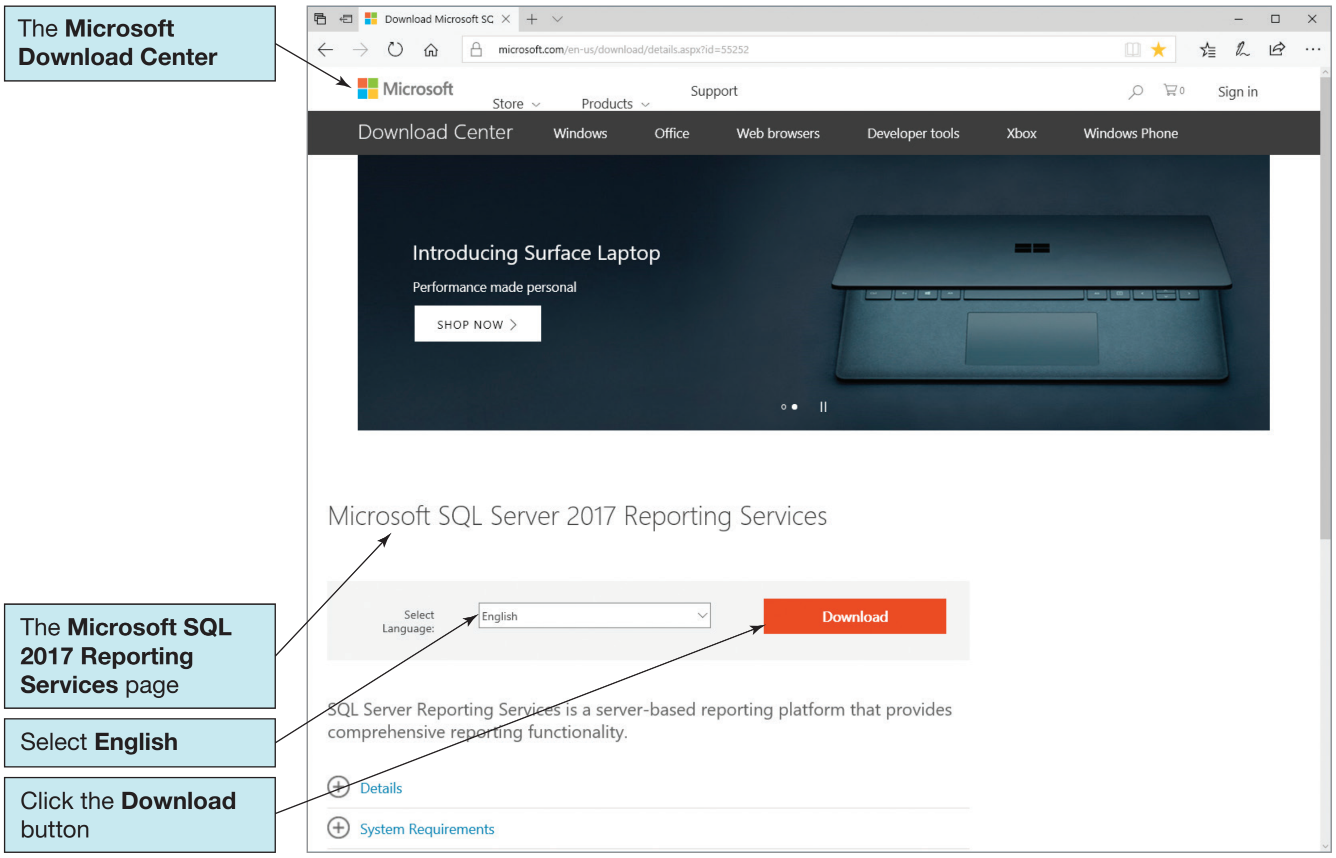
Upgrade from a previous version of SQL Server
Launch a wizard to upgrade a previous version of SQL Server to SQL Server 2017.

New Machine Learning Server (Standalone) installation
Launch a wizard to install Machine Learning Server (Standalone) on a Windows machine. This is typically used by data scientists as a standalone analysis server or as a SQL Server Machine Learning Services client.

(a) The SQL Server Installation Center Dialog Box Installation Page

FIGURE 10A-3

Installing Microsoft SQL Server 2017 Reporting Services


The Microsoft Download Center
The Microsoft SQL 2017 Reporting Services page
Select English
Click the Download button
Download Microsoft SC
microsoft.com/en-us/download/details.aspx?id=55252
Microsoft
Store
Products
Support
Sign in
Download Center
Windows
Office
Web browsers
Developer tools
Xbox
Windows Phone
Introducing Surface Laptop
Performance made personal
SHOP NOW
Details
System Requirements

Microsoft SQL Server 2017 Reporting Services

Select Language: English

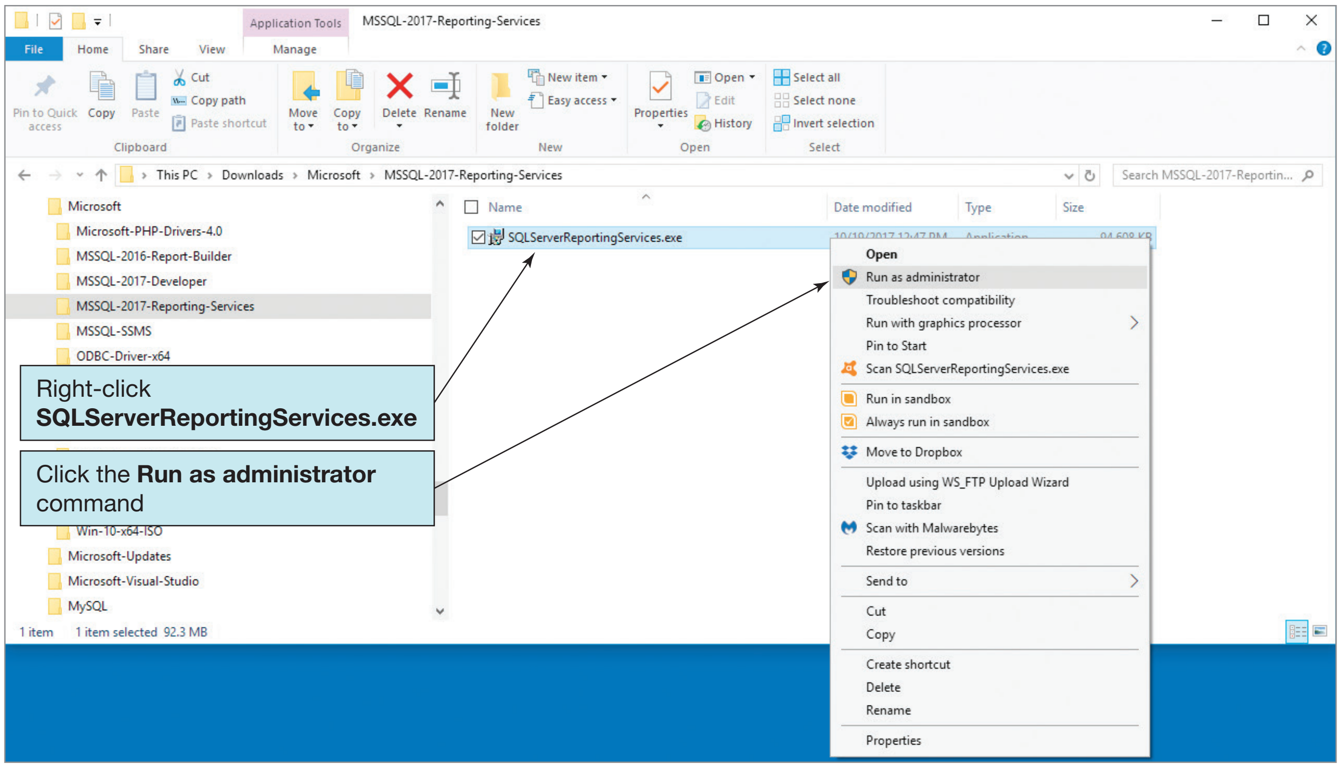
Download

SQL Server Reporting Services is a server-based reporting platform that provides comprehensive reporting functionality.

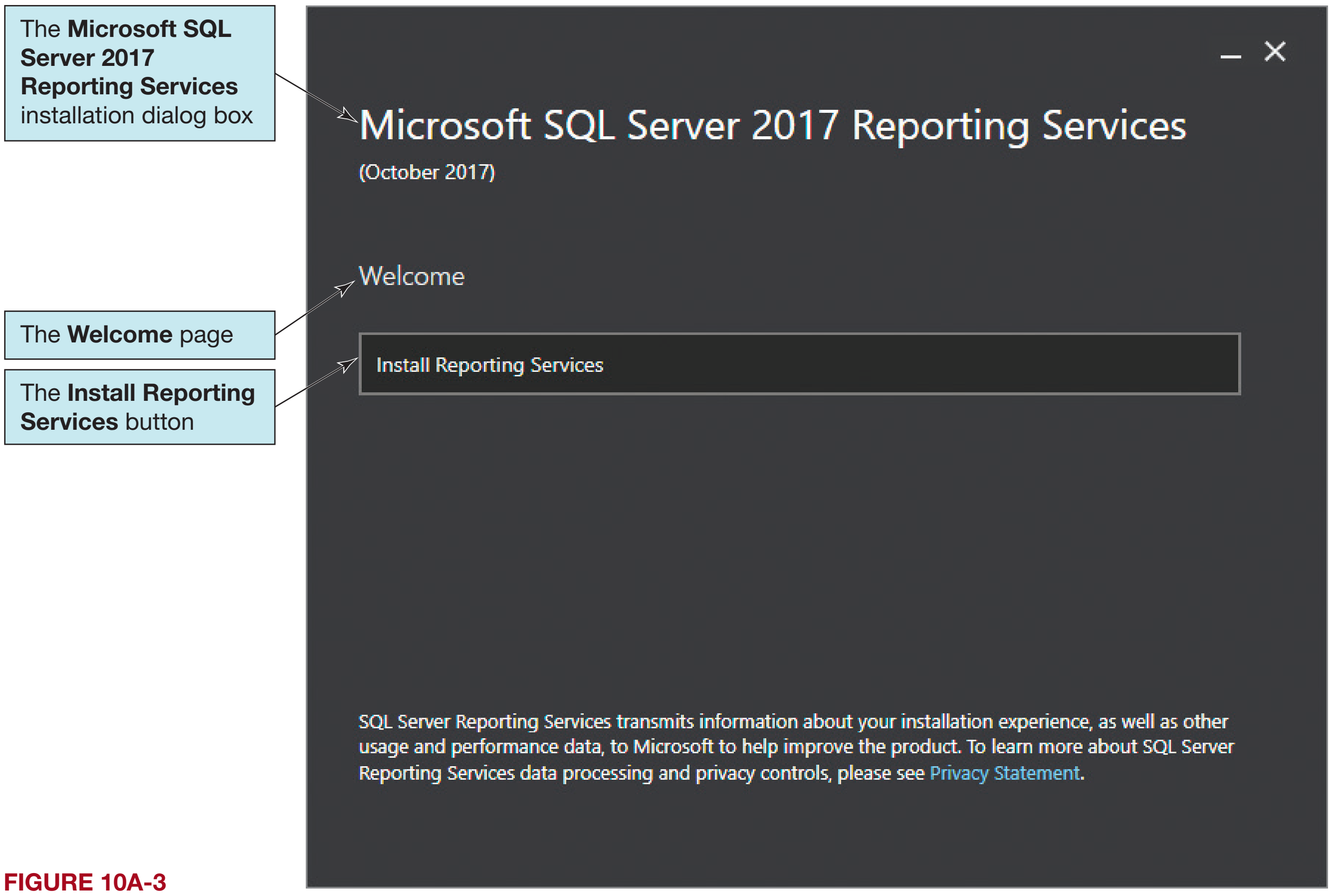
(b) The Microsoft SQL Server 2017 Reporting Services Download Web Page

FIGURE 10A-3
Continued

3. Open File Explorer, and locate the *SQLServerReportingServices.exe* installation file in your Downloads folder. By default, the file will be in the Downloads folder itself. However, we have created a *Microsoft* folder in the Downloads folder and a *MSSQL-2017-Reporting Services* folder in the Microsoft folder, and our file is located there, as shown in Figure 10A-3(c). Right-click the ***SQLServerReportingServices.exe*** file to display the shortcut menu, and then click the **Run as administrator** command.
4. A User Account Control dialog box appears asking if you want to run SQL Server Reporting Services. Click the **Yes** button.
5. The Microsoft SQL Server Reporting Services Installation dialog box is launched, and the *Welcome* page is displayed, as shown in Figure 10A-3(d). Click the **Install Reporting Services** button.
   - **NOTE:** At this point, make sure that Microsoft Server 2017 is actually running on your computer. See the discussion on pages 45-46 below.
6. The *Choose an edition to install* page is displayed, as shown in Figure 10A-3(e). Click the **Choose a free edition** radio button, and then select **Developer** from the drop-down list.
7. Click the **Next** button.
8. The *Review the license terms* page is displayed, as shown in Figure 10A-3(f). Check the **I accept the license terms** checkbox (this should be the only choice), and then click the **Next** button.
9. Click the **Next** button.
10. The *Install Database Engine* page is displayed, as shown in Figure 10A-3(g). Click the **Install Reporting Services Only** radio button (this should be the only choice), and then click the **Next** button.
11. The *Specify an install location* page is displayed, as shown in Figure 10A-3(h). The default location is correct. Click the **Install** button.



(c) The SQLServerReportingServices.exe file and Shortcut Menu


The **Microsoft SQL Server 2017 Reporting Services** installation dialog box
The **Welcome** page
The **Install Reporting Services** button
Microsoft SQL Server 2017 Reporting Services
(October 2017)
Welcome
Install Reporting Services

SQL Server Reporting Services transmits information about your installation experience, as well as other usage and performance data, to Microsoft to help improve the product. To learn more about SQL Server Reporting Services data processing and privacy controls, please see Privacy Statement.

FIGURE 10A-3
Continued

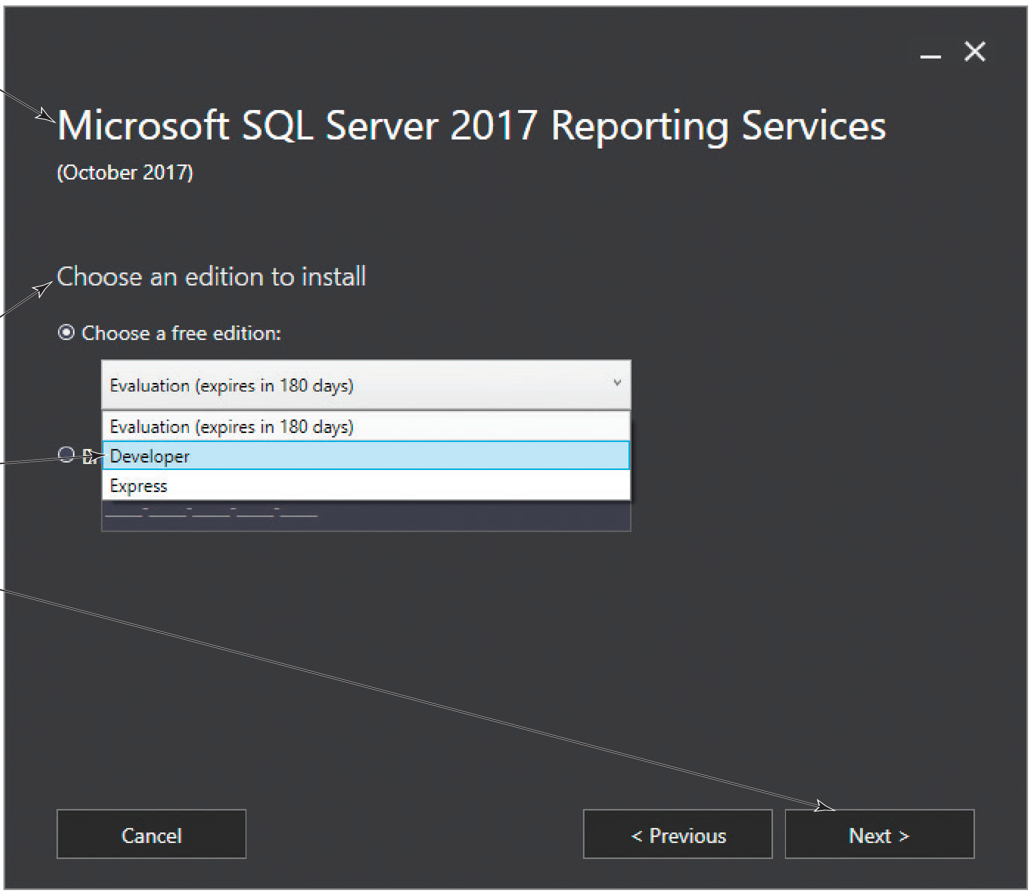
(d) The Microsoft SQL Server Reporting Services Installation Welcome Page

The **Microsoft SQL Server 2017 Reporting Services** installation dialog box

The **Choose an edition to install** page

Select the **Choose a free edition:** radio button and then **Developer** from the drop-down list

The **Next** button



Microsoft SQL Server 2017 Reporting Services
(October 2017)
Choose an edition to install
Choose a free edition:
Evaluation (expires in 180 days)
Evaluation (expires in 180 days)
Developer
Express
Cancel
< Previous
Next >

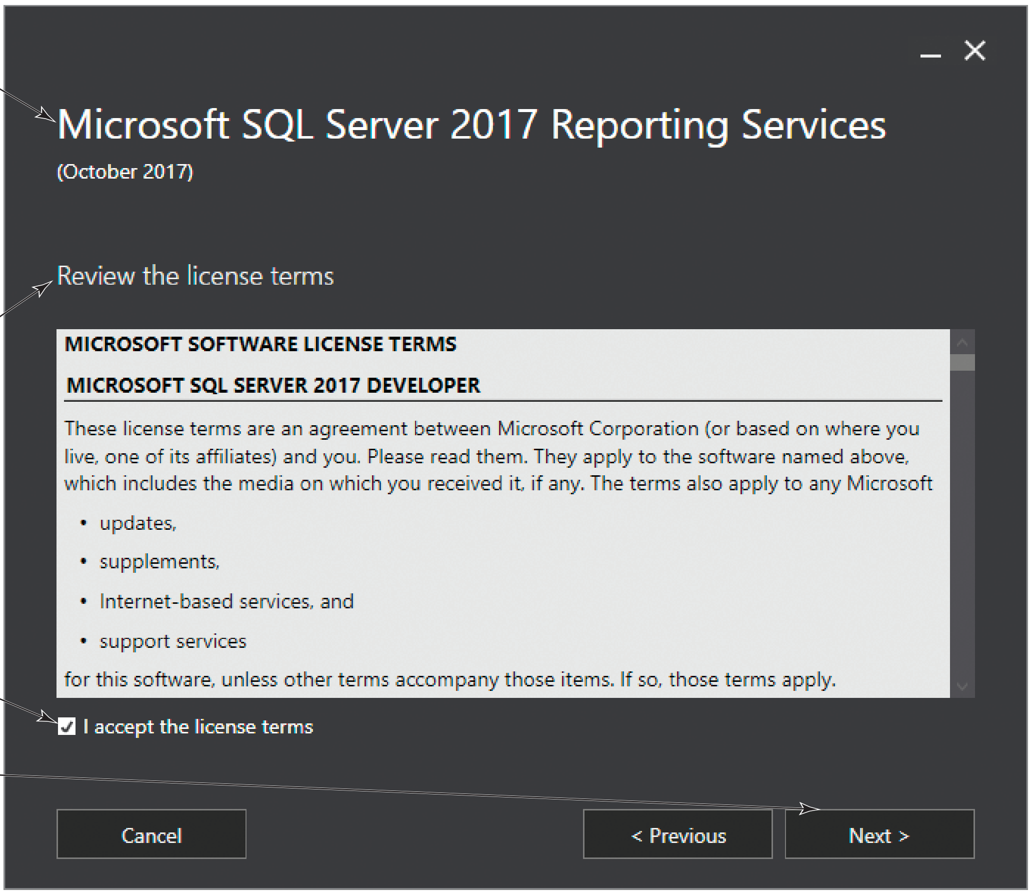
(e) The Choose an Edition to Install Page

The **Microsoft SQL Server 2017 Reporting Services** installation dialog box

The **Review the license terms** page

Check the **I accept the license terms** checkbox

The **Next** button



MICROSOFT SOFTWARE LICENSE TERMS

MICROSOFT SQL SERVER 2017 DEVELOPER

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to the software named above, which includes the media on which you received it, if any. The terms also apply to any Microsoft

- updates,
- supplements,
- Internet-based services, and
- support services

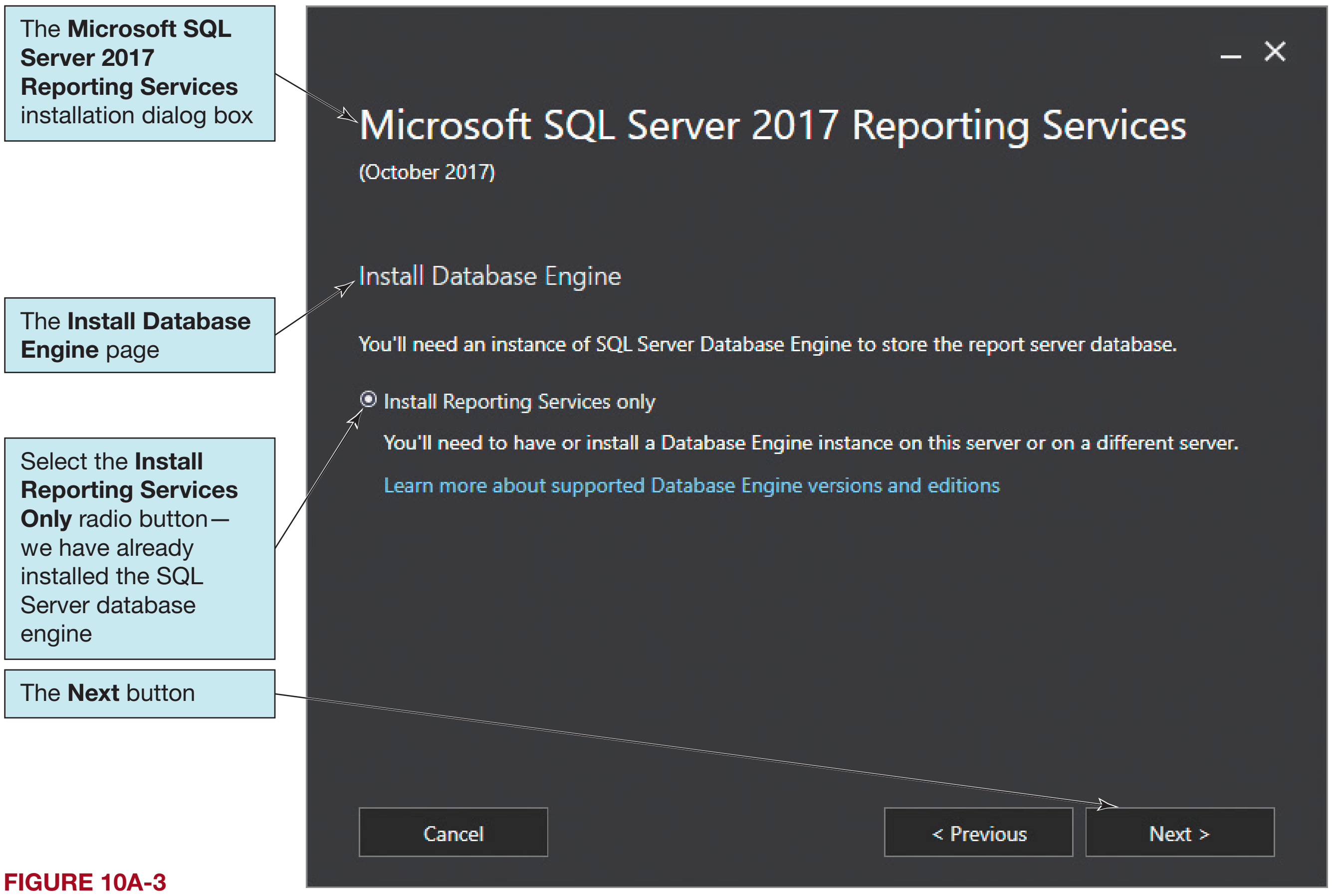
for this software, unless other terms accompany those items. If so, those terms apply.

[✓] I accept the license terms

FIGURE 10A-3

Continued

(f) The Review the License Terms Page


The **Microsoft SQL Server 2017 Reporting Services** installation dialog box
The **Install Database Engine** page
Select the **Install Reporting Services Only** radio button—we have already installed the SQL Server database engine
The **Next** button
Microsoft SQL Server 2017 Reporting Services
(October 2017)
Install Database Engine
Learn more about supported Database Engine versions and editions
Cancel
< Previous
Next >

You'll need an instance of SQL Server Database Engine to store the report server database.

[✓ Install Reporting Services only]

You'll need to have or install a Database Engine instance on this server or on a different server.

FIGURE 10A-3
Continued

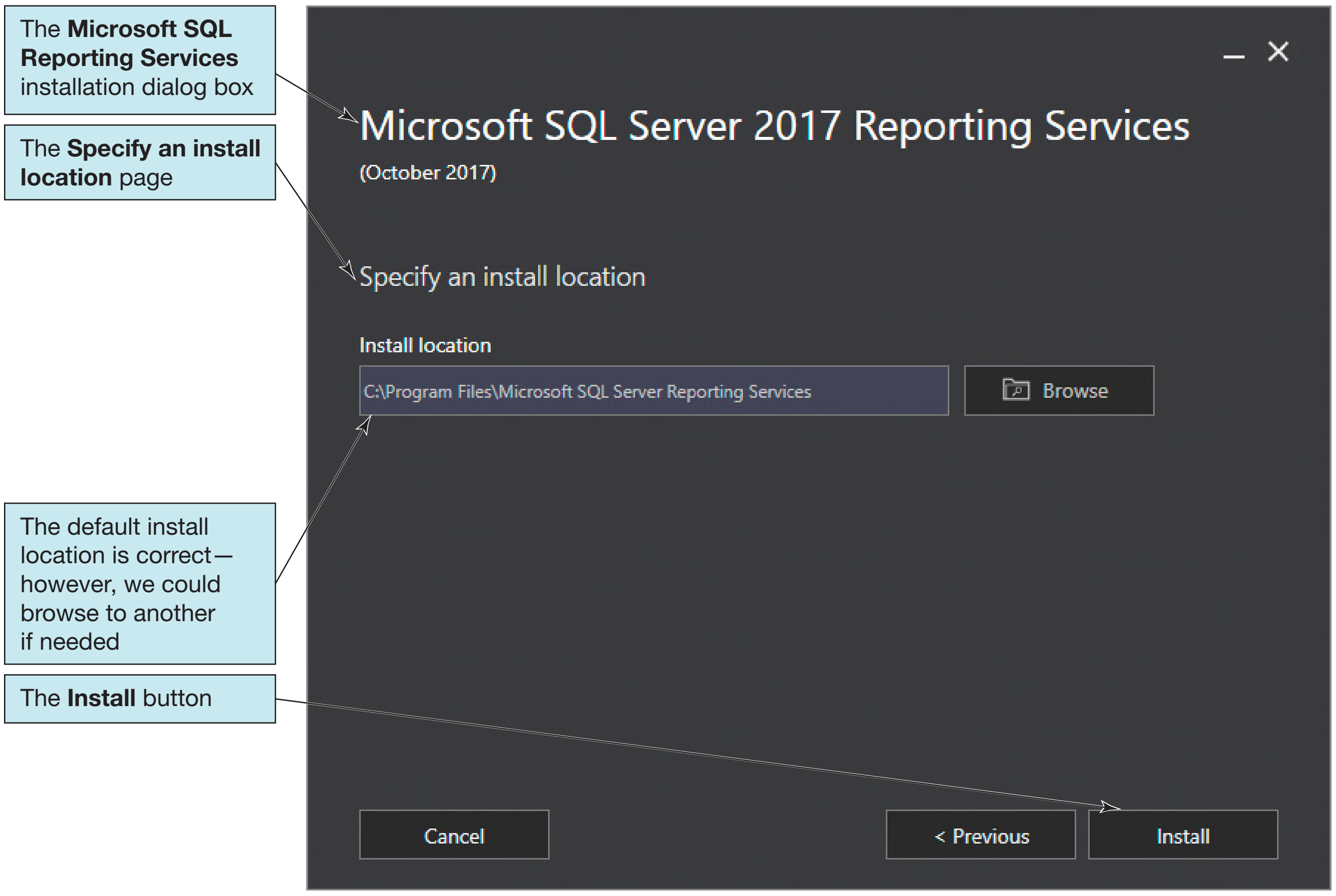
(g) The Install Database Engine Page

12. The *Setup completed* page is displayed, as shown in Figure 10A-3(i). Although we have successfully installed Microsoft SQL Server 2017 Reporting Services, we still need to configure them. Click the **Configure manually and customize setting** radio button, and then click the **Configure report server** button.

When we install Microsoft SQL Server 2017 Reporting Services, the **Report Server Configuration Manager** is installed. We have just launched it in step 12 above, but it is also now available as a separate utility and accessible in the Windows 10 menu. We will now use it to configure our installation of Microsoft SQL Server 2017 Reporting Services.

Configuring Microsoft SQL Server 2017 Reporting Services

1. We have launched the Report Server Configuration Manager in step 9 in the last set of steps. However, we could have also started it using the Windows 10 menu. The **Report Server Configuration Manager** opens with the **Report Server Configuration Connection** dialog box open and ready to use, as shown in Figure 10A-4(a). The default values provided are correct, so click the **Connect** button.
2. Once connected, the Report Server Configuration Manager displays the *Report Server Status* page, as shown in Figure 10A-4(b). The basic report server data is shown, but needed configuration data, such as the Report Server Database Name, is missing because we have not configured the report server yet. Note that the connected report server (DJA-CT-NB\SSRS) is shown in the Connect window on the left-hand side of the Report Server Configuration Manager and that a set of **configuration setting page icons** are shown below the connection. Make sure the **Report Server Status** (right above the Start and Stop buttons) is *Started*–if it isn't, click the **Start** button.


The **Microsoft SQL Reporting Services** installation dialog box
The **Specify an install location** page
The default install location is correct—however, we could browse to another if needed
The **Install** button

Microsoft SQL Server 2017 Reporting Services

(October 2017)

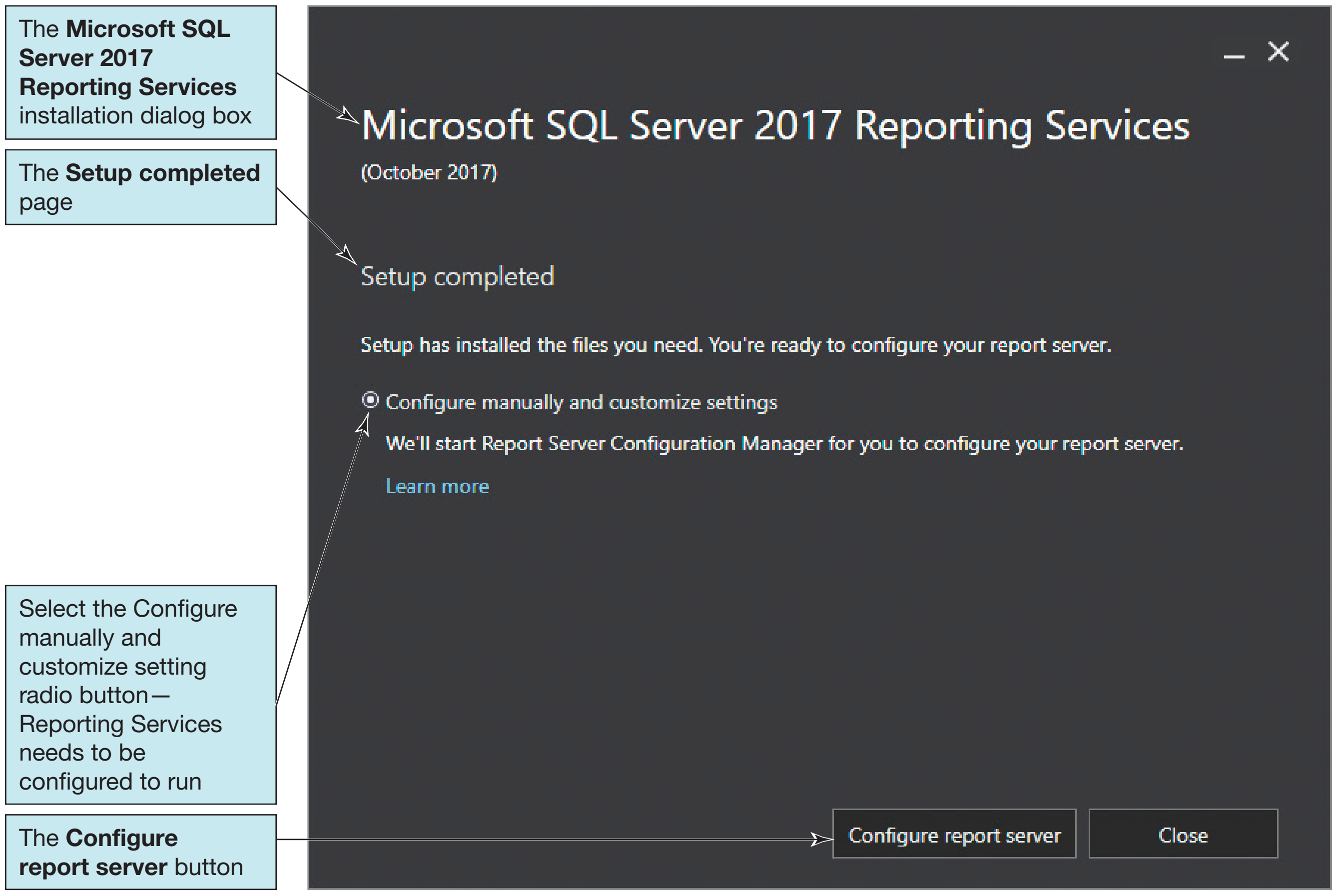
Specify an install location

Install location

C:\Program Files\Microsoft SQL Server Reporting Services

Browse
Cancel
< Previous
Install

(h) The Install Database Engine Page


The **Microsoft SQL Server 2017 Reporting Services** installation dialog box
The **Setup completed** page
Select the Configure manually and customize setting radio button—Reporting Services needs to be configured to run
The **Configure report server** button

Microsoft SQL Server 2017 Reporting Services

(October 2017)

Setup completed

Setup has installed the files you need. You're ready to configure your report server.

● Configure manually and customize settings

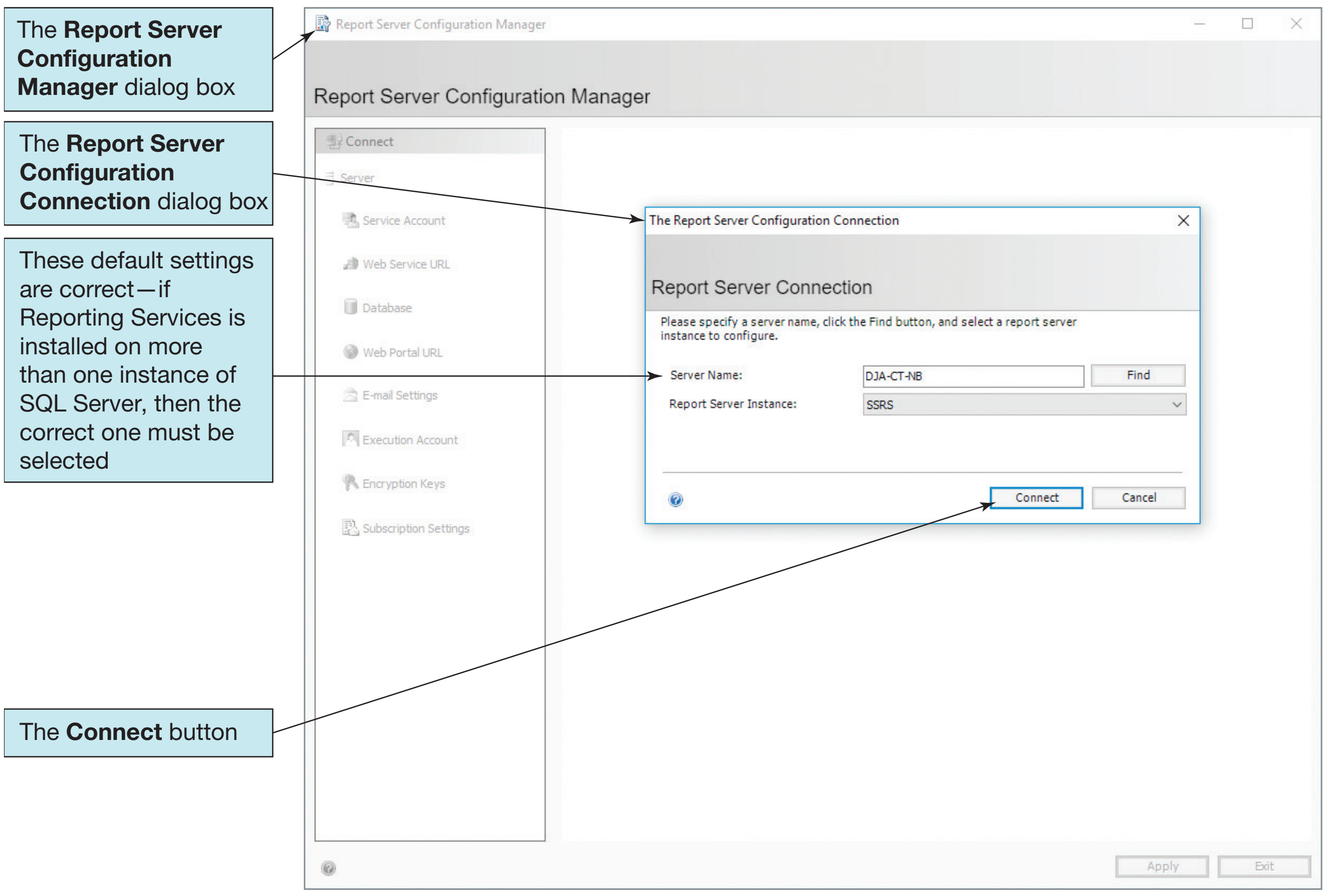
We'll start Report Server Configuration Manager for you to configure your report server.

Learn more

Configure report server
Close

(i) The Setup Completed Page

FIGURE 10A-3
Continued


The **Report Server Configuration Manager** dialog box
The **Report Server Configuration Connection** dialog box
These default settings are correct—if Reporting Services is installed on more than one instance of SQL Server, then the correct one must be selected
The **Connect** button

Report Server Configuration Manager

Report Server Configuration Manager

Connect
Server
Service Account
Web Service URL
Database
Web Portal URL
E-mail Settings
Execution Account
Encryption Keys
Subscription Settings

The Report Server Configuration Connection

Report Server Connection

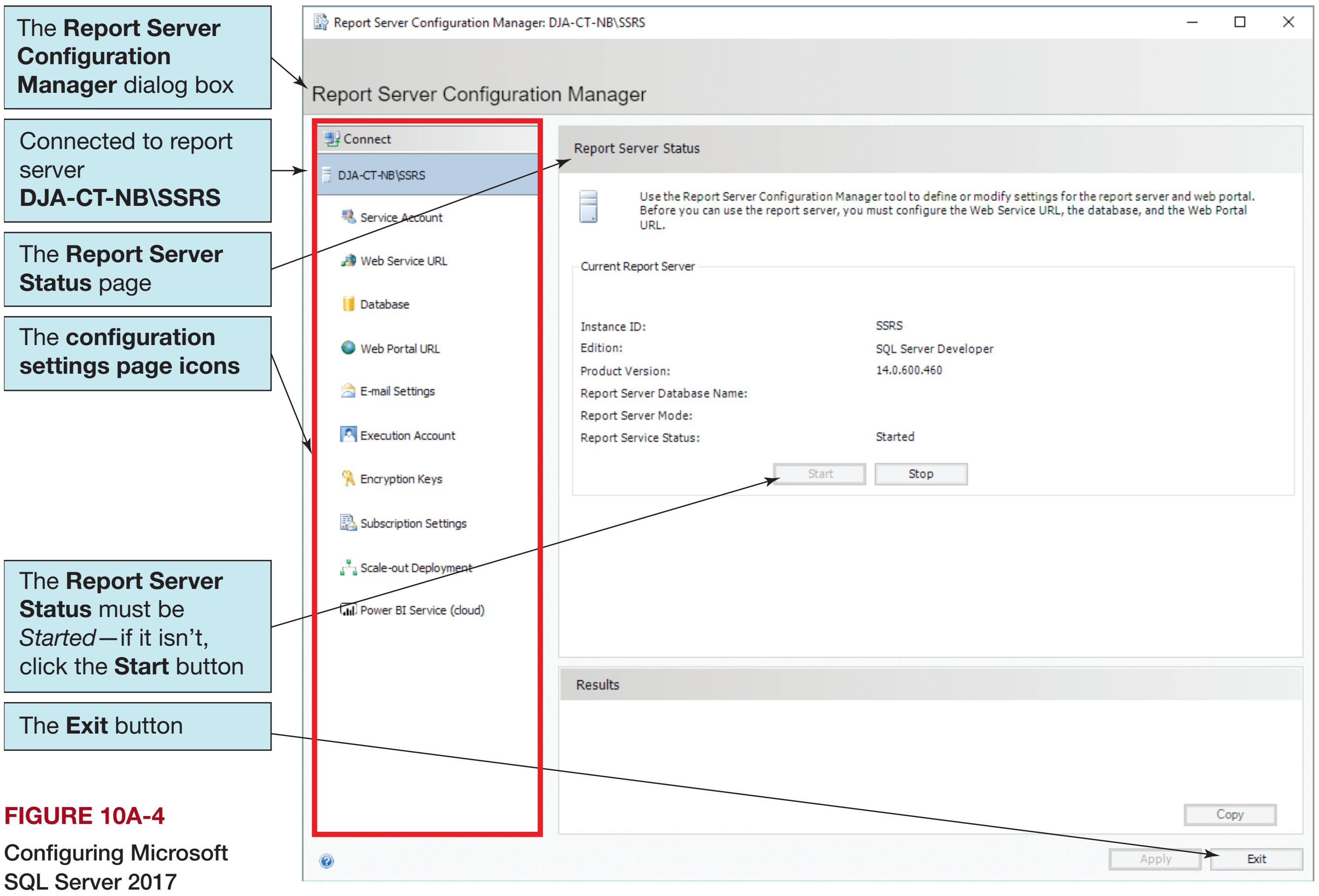
Please specify a server name, click the Find button, and select a report server instance to configure.

Server Name: DJA-CT-NB Find
Report Server Instance: SSRS

Connect Cancel

Apply Exit

(a) The Report Server Configuration Connection Dialog Box


The **Report Server Configuration Manager** dialog box
Connected to report server **DJA-CT-NB\SSRS**
The **Report Server Status** page
The **configuration settings page icons**
The **Report Server Status** must be *Started*—if it isn't, click the **Start** button
The **Exit** button

Report Server Configuration Manager: DJA-CT-NB\SSRS

Report Server Configuration Manager

Connect
DJA-CT-NB\SSRS
Service Account
Web Service URL
Database
Web Portal URL
E-mail Settings
Execution Account
Encryption Keys
Subscription Settings
Scale-out Deployment
Power BI Service (cloud)

Report Server Status

Use the Report Server Configuration Manager tool to define or modify settings for the report server and web portal. Before you can use the report server, you must configure the Web Service URL, the database, and the Web Portal URL.

Current Report Server

Instance ID:	SSRS
Edition:	SQL Server Developer
Product Version:	14.0.600.460
Report Server Database Name:	
Report Server Mode:	
Report Service Status:	Started

Start Stop

Results

Copy

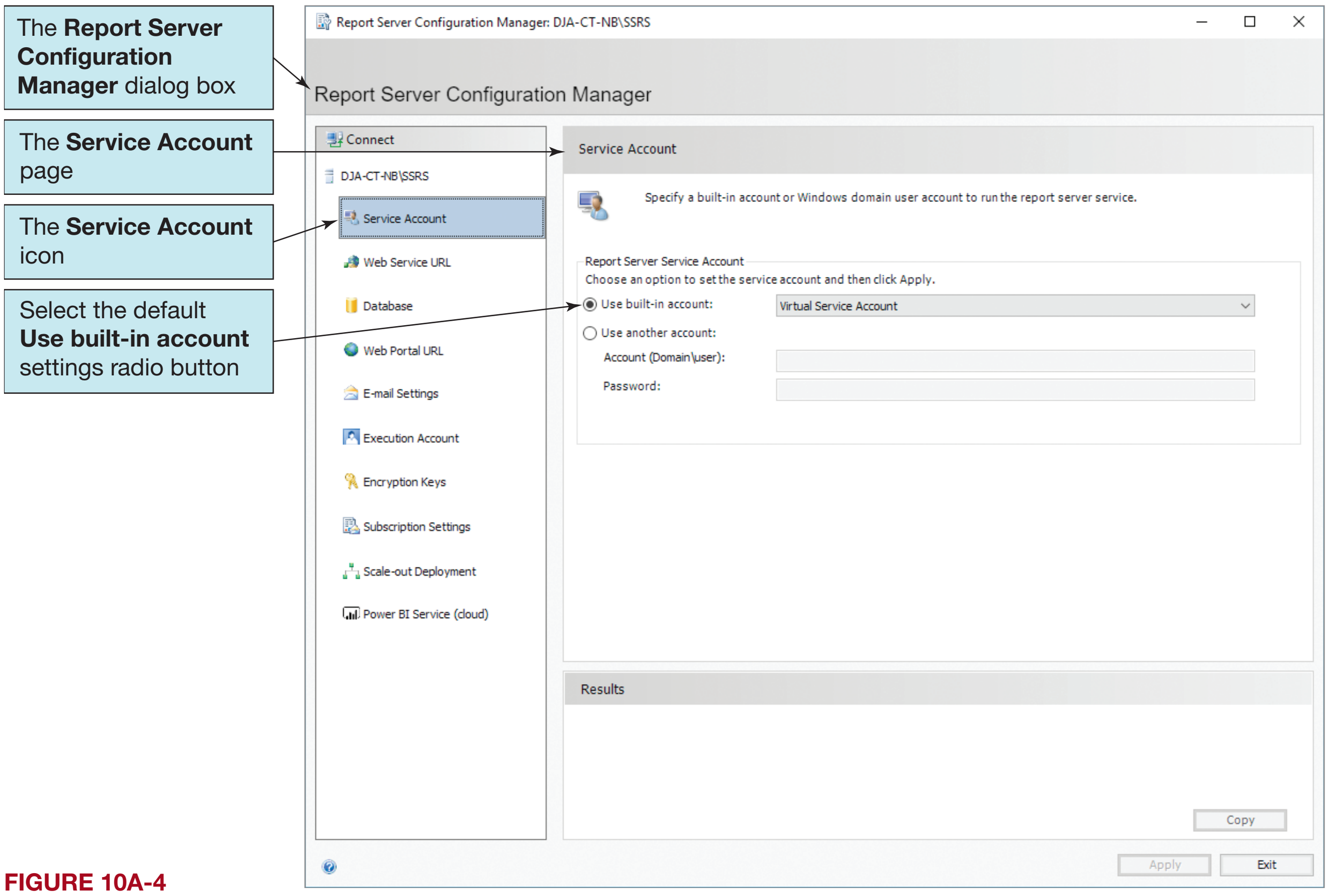
Apply Exit

(b) The Report Server Status Page

FIGURE 10A-4

Configuring Microsoft SQL Server 2017 Reporting Services

3. Click the **Service Account** icon in the Connect window to display the *Service Account* page, as shown in Figure 10A-4(c). If necessary, select the **Use built-in account** radio button to enable the default Virtual Service account. No other action is necessary on this page.
4. Click the **Web Service URL** icon in the Connect window to display the *Web Service URL* page, as shown in Figure 10A-4(d). Note that the default settings are correct but that they have not been applied to the new reporting services configuration. We also must specify an HTTPS Certificate and its associated port from the drop-down list. To configure reporting services with this data, click the **Apply** button.
   - **NOTE:** Your options on the drop-down list may differ, and not include an HTTPS certificate. If one is not available, skip adding the HTTPS certificate.
5. Click the **Database** icon in the Connect window to display the *Report Server Database* page, as shown in Figure 10A-4(e). Note that the data for the *Current Report Server Database* and the *Current Report Server Database Credential* is blank. To configure this data, click the **Change Database** button.
6. The **Report Server Database Configuration Wizard** is displayed, as shown in Figure 10A-4(f). Note the list of steps in the left-hand window, with the *Action* step and page displayed. If necessary, select the **Create a new report server database** radio button, and then click the **Next** button.
7. The Report Server Database Configuration Wizard *Database Server* page is displayed, as shown in Figure 10A-4(g). This page determines which SQL Server 2017 instance the report server will be installed on. The **Server Name** text box must contain a name consisting of the computer name and the SQL Server instance name separated by a backslash (\). In our case, this will be *DJA-CT-NB\SQLSERVER2017*. If you installed SQL Server 2017 using the default SQL instance name, only the computer name


The **Report Server Configuration Manager** dialog box
The **Service Account** page
The **Service Account** icon
Select the default **Use built-in account** settings radio button
Report Server Configuration Manager: DJA-CT-NB\SSRS
Report Server Configuration Manager
Connect
DJA-CT-NB\SSRS
Service Account
Web Service URL
Database
Web Portal URL
E-mail Settings
Execution Account
Encryption Keys
Subscription Settings
Scale-out Deployment
Power BI Service (cloud)
Results
Copy
Apply
Exit

Service Account

Specify a built-in account or Windows domain user account to run the report server service.

Report Server Service Account
Choose an option to set the service account and then click Apply.
[✓ Use built-in account:] Virtual Service Account
Use another account:
Account (Domain\user):
Password:

FIGURE 10A-4
Continued

(c) The Service Account Page

The **Report Server Configuration Manager** dialog box

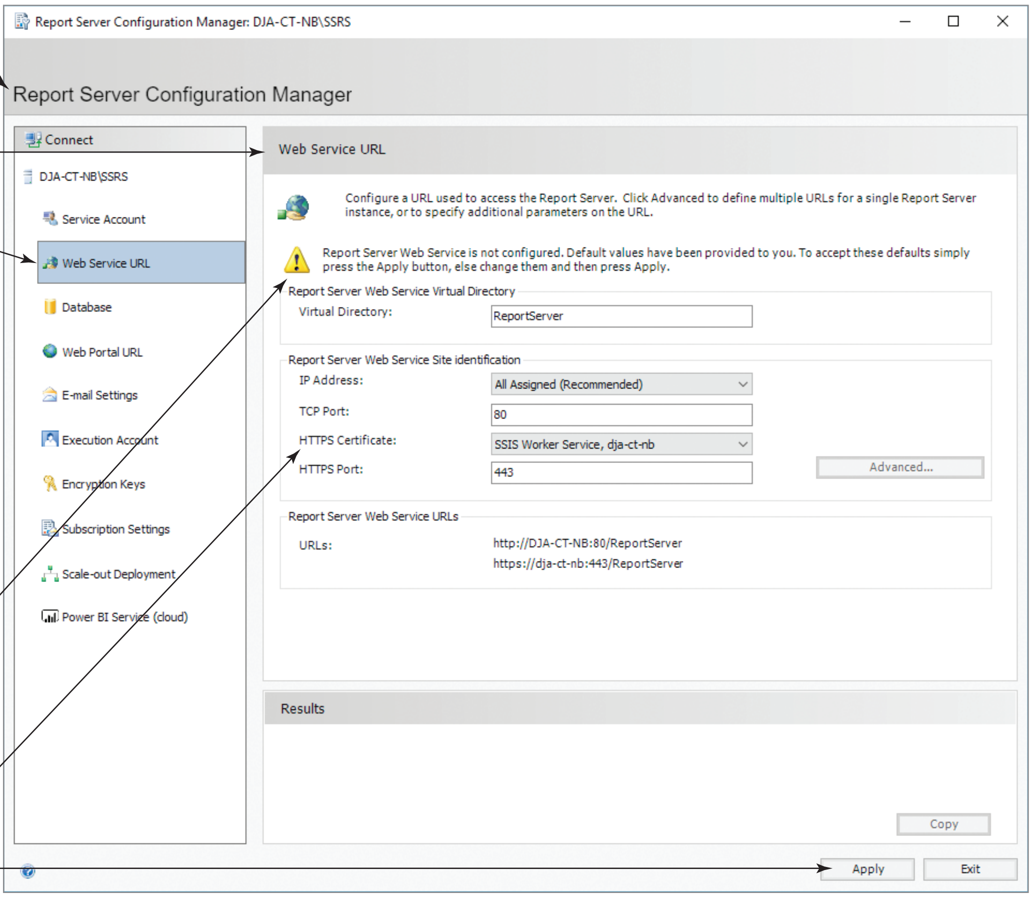
The **Web Service URL page**

The **Web Service URL** icon

The default settings are correct, but have not been applied to the reporting services—click the **Apply** button to configure reporting services with these values

Add an **HTTPS Certificate**

The **Apply** button


Report Server Configuration Manager: DJA-CT-NB\SSRS

Report Server Configuration Manager

Connect
DJA-CT-NB\SSRS
Service Account
Web Service URL
Database
Web Portal URL
E-mail Settings
Execution Account
Encryption Keys
Subscription Settings
Scale-out Deployment
Power BI Service (cloud)

Web Service URL

Configure a URL used to access the Report Server. Click Advanced to define multiple URLs for a single Report Server instance, or to specify additional parameters on the URL.

Report Server Web Service is not configured. Default values have been provided to you. To accept these defaults simply press the Apply button, else change them and then press Apply.

Report Server Web Service Virtual Directory
Virtual Directory: ReportServer

Report Server Web Service Site identification
IP Address: All Assigned (Recommended)
TCP Port: 80
HTTPS Certificate: SSIS Worker Service, dja-ct-nb
HTTPS Port: 443
Advanced...

Report Server Web Service URLs
URLs: http://DJA-CT-NB:80/ReportServer
https://dja-ct-nb:443/ReportServer

Results

Copy
Apply
Exit

(d) The Web Service URL Page

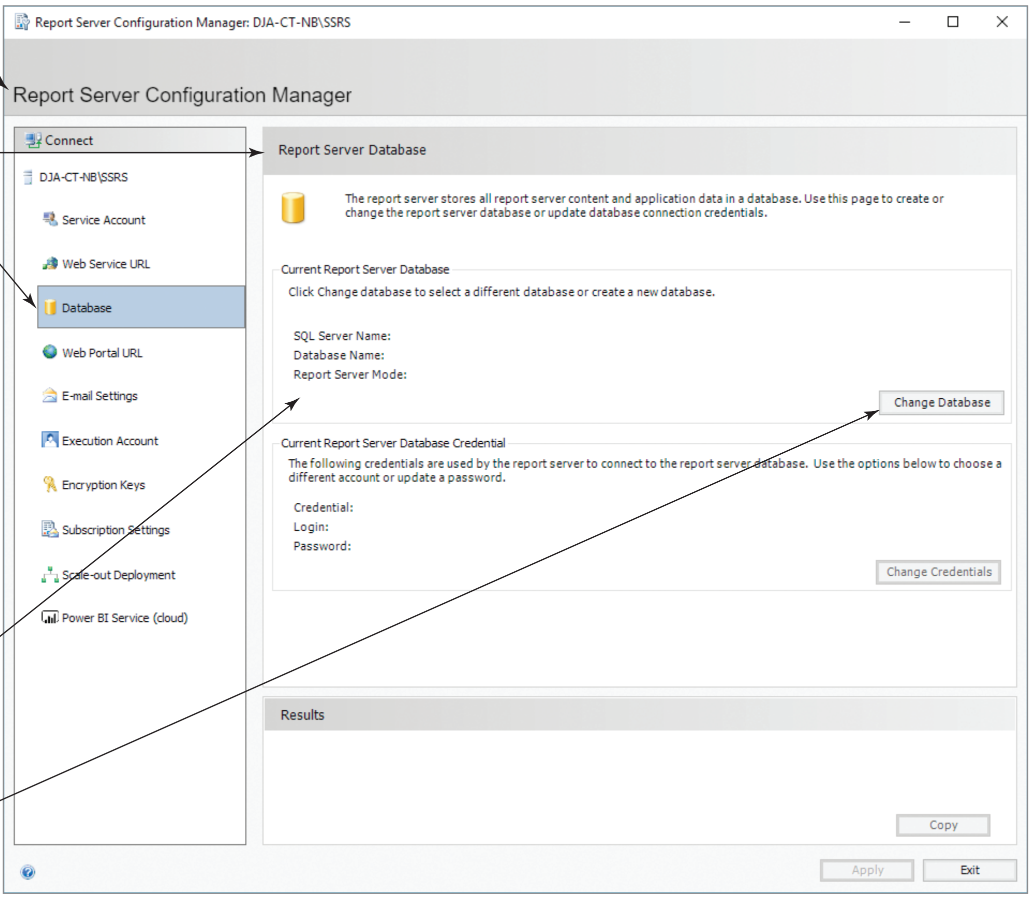
The **Report Server Configuration Manager** dialog box

The **Report Server Database page**

The **Database** icon

There are currently no report server database or database credentials assigned

Click the **Change Database** button to run the **Report Server Database Configuration Wizard**


Report Server Configuration Manager: DJA-CT-NB\SSRS

Report Server Configuration Manager

Connect
DJA-CT-NB\SSRS
Service Account
Web Service URL
Database
Web Portal URL
E-mail Settings
Execution Account
Encryption Keys
Subscription Settings
Scale-out Deployment
Power BI Service (cloud)

Report Server Database

The report server stores all report server content and application data in a database. Use this page to create or change the report server database or update database connection credentials.

Current Report Server Database
Click Change database to select a different database or create a new database.

SQL Server Name:
Database Name:
Report Server Mode:

Change Database

Current Report Server Database Credential
The following credentials are used by the report server to connect to the report server database. Use the options below to choose a different account or update a password.

Credential:
Login:
Password:

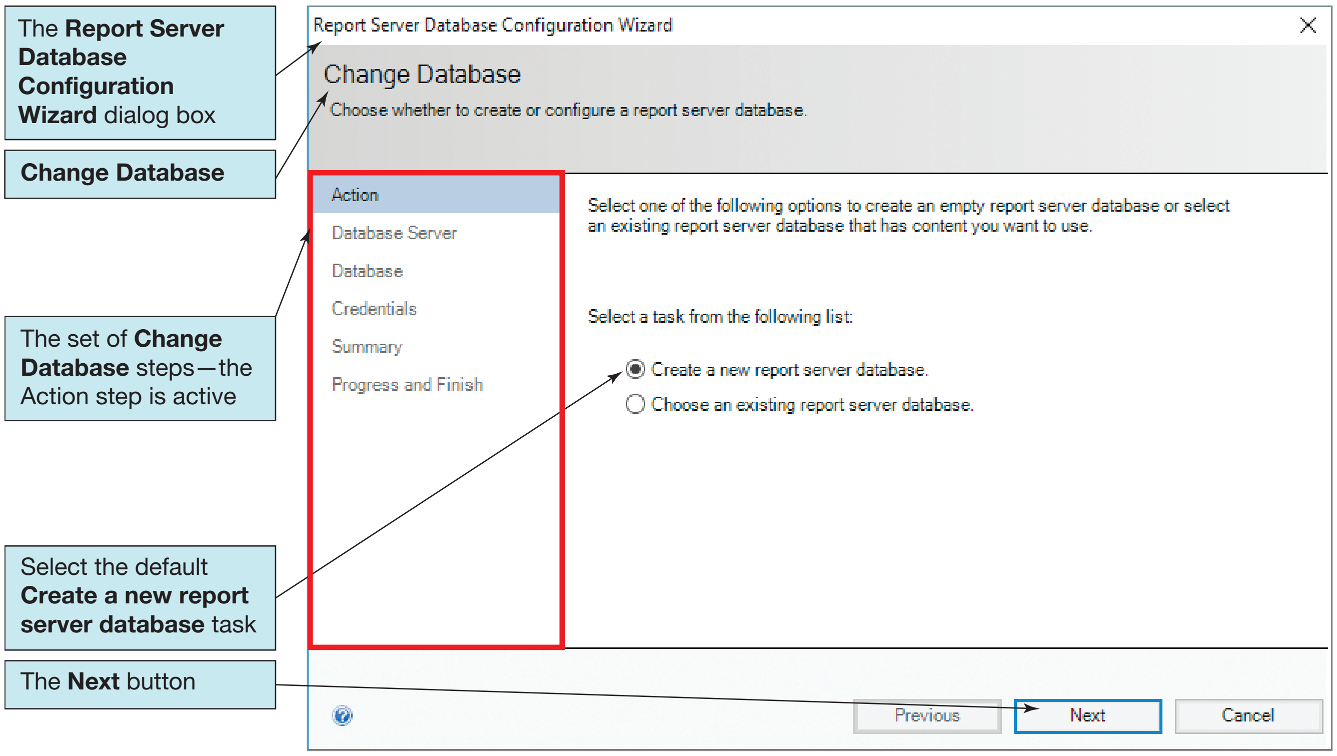
Change Credentials

Results

Copy
Apply
Exit

(e) The Initial Report Server Database Page

FIGURE 10A-4
Continued


The **Report Server Database Configuration Wizard** dialog box
Change Database
The set of **Change Database** steps—the Action step is active
Select the default **Create a new report server database** task
The **Next** button
Action
Database Server
Database
Credentials
Summary
Progress and Finish
Previous
Next
Cancel

Report Server Database Configuration Wizard

Change Database

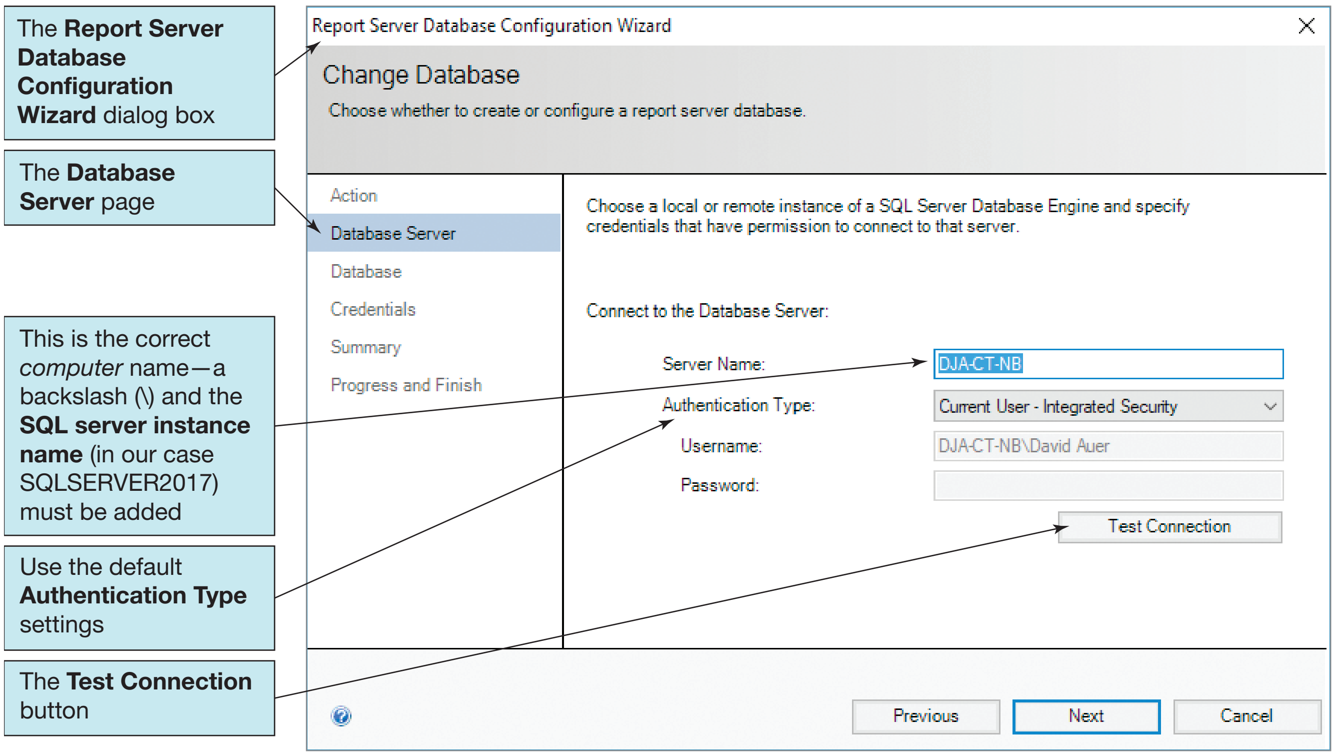
Choose whether to create or configure a report server database.

Select one of the following options to create an empty report server database or select an existing report server database that has content you want to use.

Select a task from the following list:

- [✓ Create a new report server database.]
- Choose an existing report server database.

(f) The Action Page


The **Report Server Database Configuration Wizard** dialog box
The **Database Server** page
This is the correct *computer* name—a backslash (\) and the **SQL server instance name** (in our case SQLSERVER2017) must be added
Use the default **Authentication Type** settings
The **Test Connection** button
Action
Database Server
Database
Credentials
Summary
Progress and Finish
Test Connection
Previous
Next
Cancel

Report Server Database Configuration Wizard

Change Database

Choose whether to create or configure a report server database.

Choose a local or remote instance of a SQL Server Database Engine and specify credentials that have permission to connect to that server.

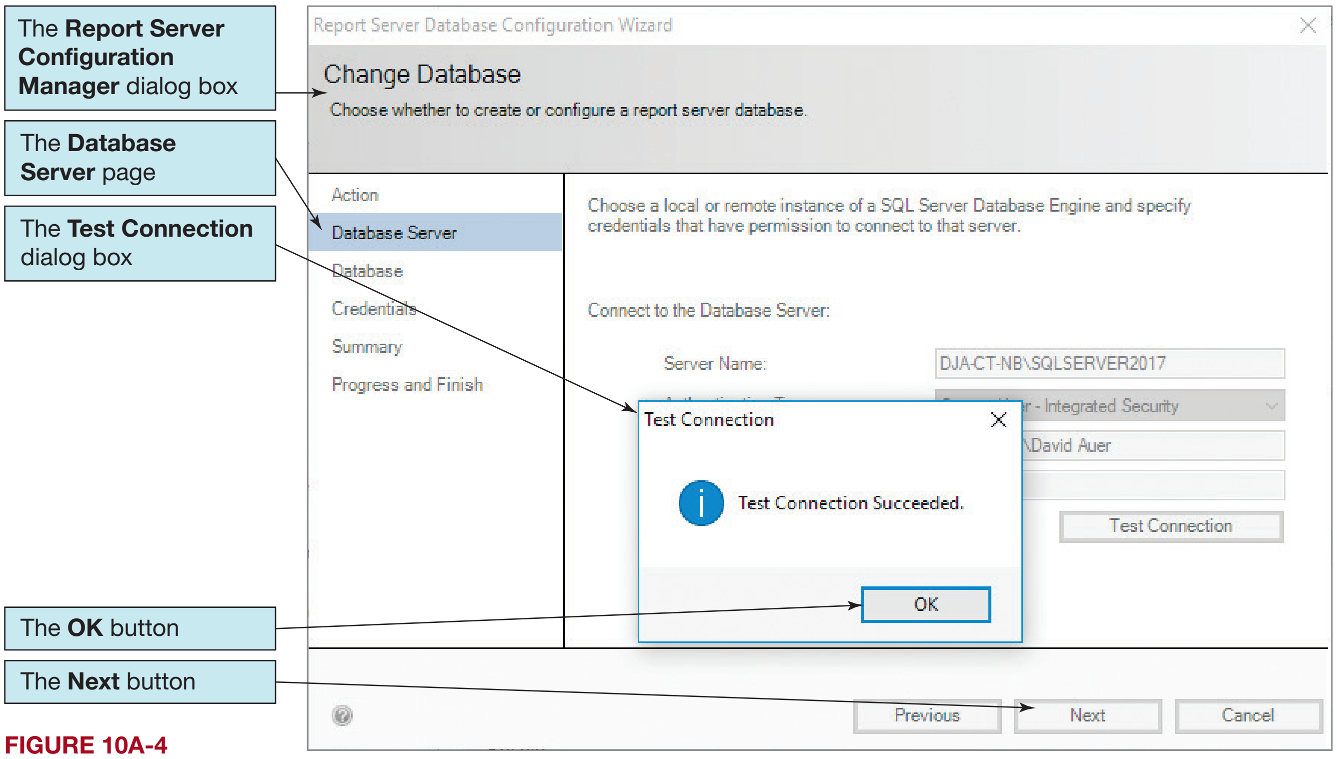
Connect to the Database Server:

Server Name:	DJA-CT-NB
Authentication Type:	Current User - Integrated Security
Username:	DJA-CT-NB\David Auer
Password:	

(g) The Database Server Page

FIGURE 10A-4

Continued


The **Report Server Configuration Manager** dialog box
The **Database Server** page
The **Test Connection** dialog box
The **OK** button
The **Next** button
Report Server Database Configuration Wizard
Action
Database Server
Database
Credentials
Summary
Progress and Finish
Test Connection
OK
Previous
Next
Cancel

Change Database

Choose whether to create or configure a report server database.

Choose a local or remote instance of a SQL Server Database Engine and specify credentials that have permission to connect to that server.

Connect to the Database Server:

Server Name: DJA-CT-NB\SQLSERVER2017

r - Integrated Security

\David Auer

Test Connection Succeeded.

FIGURE 10A-4

Continued

(h) The Test Connection Dialog Box

needs to appear. You computer name will be different than the example. Note that the default **Authentication Type** settings are correct—we want to use the current username, and Integrated Security will automatically provide that username and the associated password. Type in the appropriate Server Name, and then click the **Test Connection** button.

8. The **Test Server** dialog box is displayed showing that the database server connection works, as shown in Figure 10A-4(h). Click the Test Server dialog box **OK** button, and then click the Report Server Database Configuration Wizard **Next** button. Remember or write down your *ComputerName\UserName*, as you will need it for step 10.

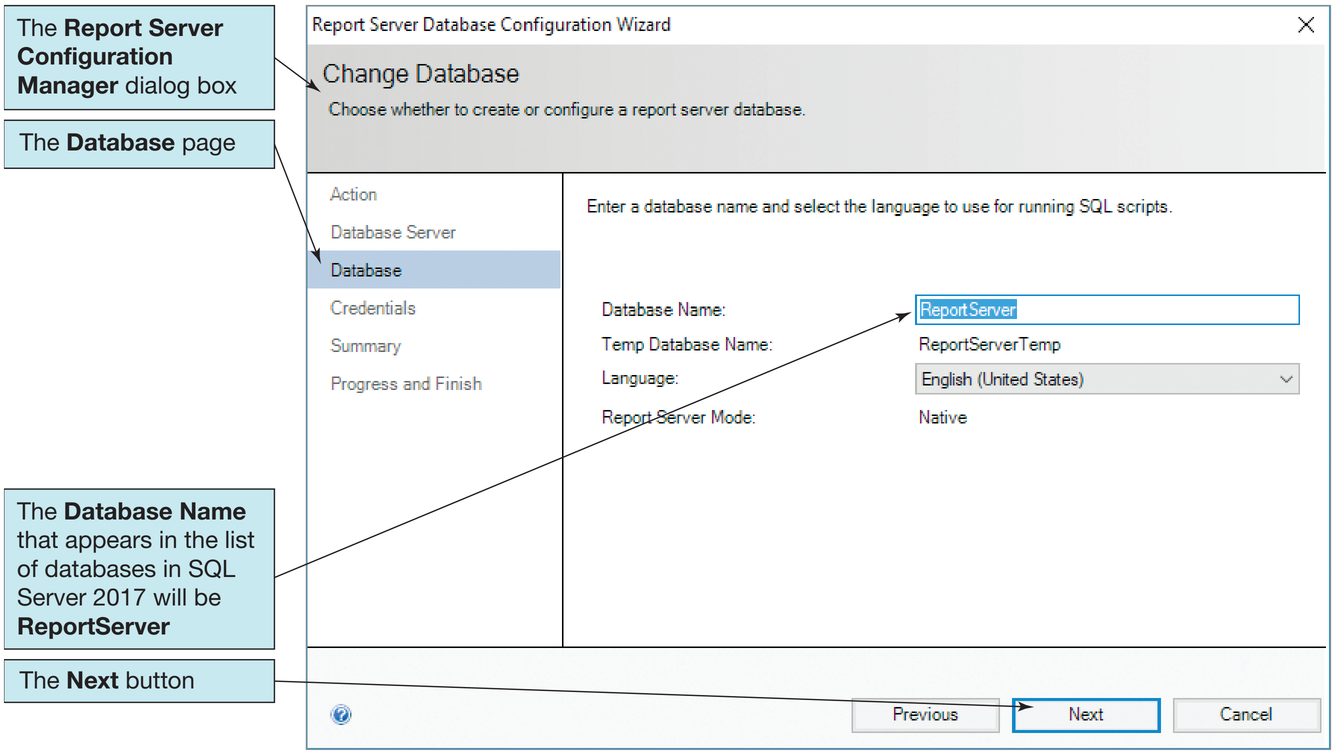
9. The Report Server Database Configuration Wizard *Database* page is displayed, as shown in Figure 10A-4(i). This page determines the name of the report services database as it will appear in SQL Server 2017. The default database name of *ReportServer* and the other settings are correct. Click the **Next** button.

10. The Report Server Database Configuration Wizard *Credentials* page is displayed, as shown in Figure 10A-4(j). This page determines the username and password of the user who will have permission to use the report server database. Here we simply use our own name and password with automatic Windows authentication. Click the **Next** button.

11. The Report Server Database Configuration Wizard *Summary* page is displayed, as shown in Figure 10A-4(k). This page lists all the settings we have selected on the previous pages and summarizes them so that we can check for correctness before the final configuration is set. Review the setting, and then click the **Next** button.

12. The Report Server Database Configuration Wizard *Progress and Finish* page is displayed, as shown in Figure 10A-4(l). After the configuration process is complete, click the **Finish** button.

13. The Report Server Configuration Manager *Database* page is displayed again, but now all the needed settings for the *Current Report Server Database* and *Current Report Server Database Credential* are present, as shown in Figure 10A-4(m).

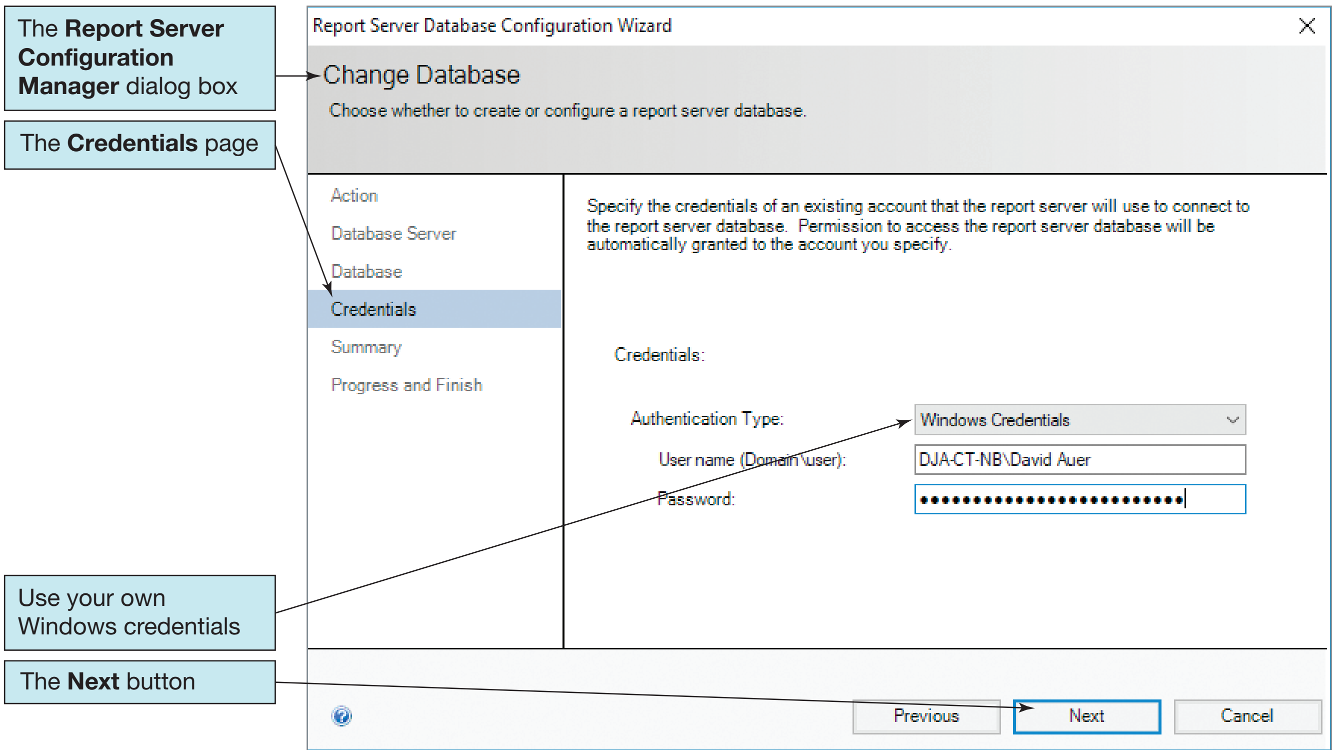

The **Report Server Configuration Manager** dialog box
The **Database** page
The **Database Name** that appears in the list of databases in SQL Server 2017 will be **ReportServer**
The **Next** button
Report Server Database Configuration Wizard
Change Database
Action
Database Server
Database
Credentials
Summary
Progress and Finish
Previous
Next
Cancel

Choose whether to create or configure a report server database.

Enter a database name and select the language to use for running SQL scripts.

Database Name:	ReportServer
Temp Database Name:	ReportServerTemp
Language:	English (United States)
Report Server Mode:	Native

(i) The Database Page



Choose whether to create or configure a report server database.

Specify the credentials of an existing account that the report server will use to connect to the report server database. Permission to access the report server database will be automatically granted to the account you specify.

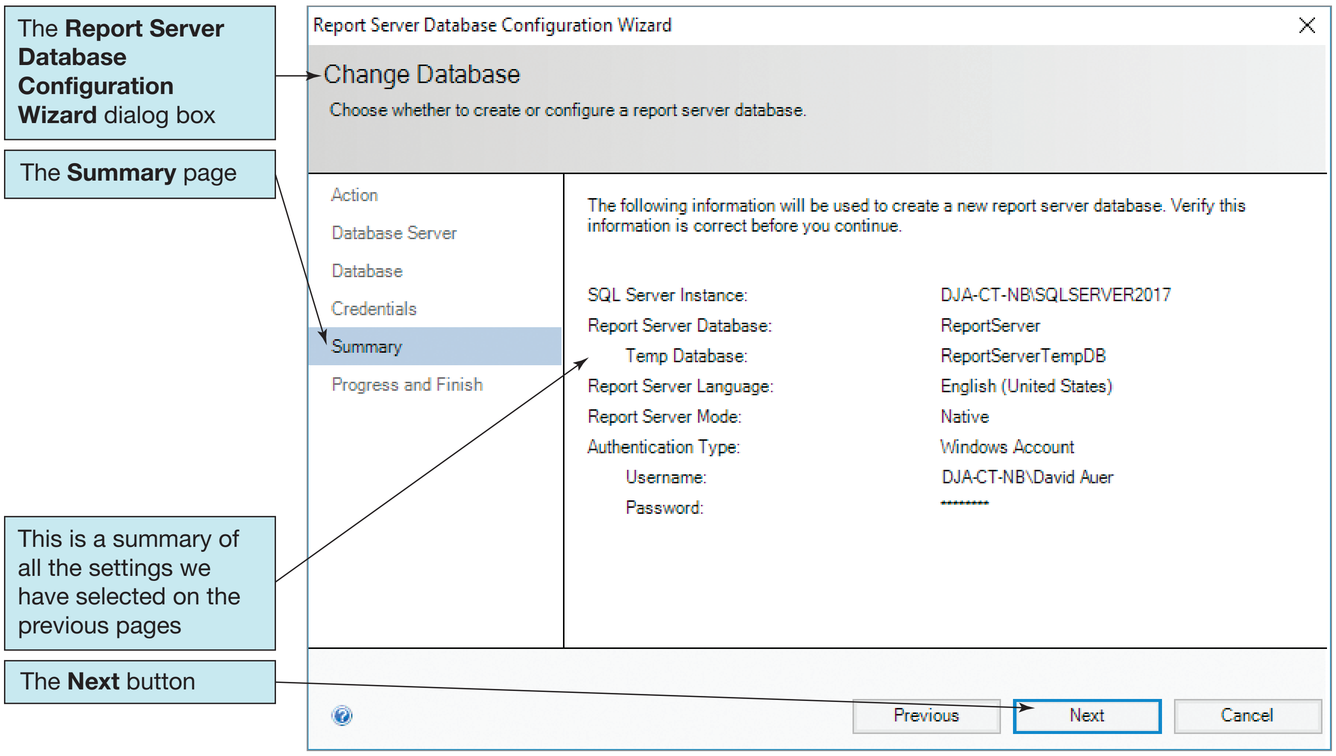
Credentials:

Authentication Type:	Windows Credentials
User name (Domain\user):	DJA-CT-NB\David Auer
Password:	•••••••••••••••••••••••••

(j) The Credentials Page

FIGURE 10A-4

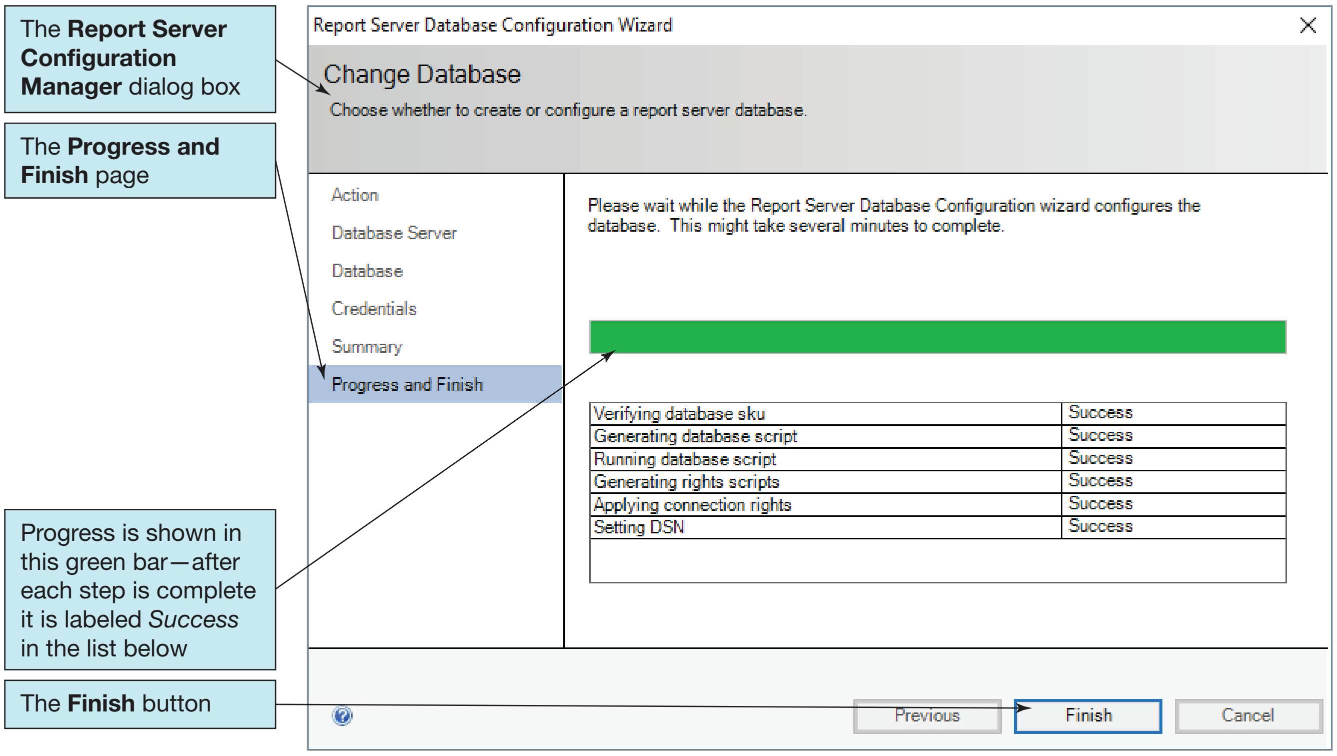
Continued


The **Report Server Database Configuration Wizard** dialog box
The **Summary** page
This is a summary of all the settings we have selected on the previous pages
The **Next** button
Report Server Database Configuration Wizard
Change Database
Choose whether to create or configure a report server database.
Action
Database Server
Database
Credentials
Summary
Progress and Finish
Previous
Next
Cancel

The following information will be used to create a new report server database. Verify this information is correct before you continue.

SQL Server Instance:	DJA-CT-NB\SQLSERVER2017
Report Server Database:	ReportServer
Temp Database:	ReportServerTempDB
Report Server Language:	English (United States)
Report Server Mode:	Native
Authentication Type:	Windows Account
Username:	DJA-CT-NB\David Auer
Password:	********

(k) The Summary Page

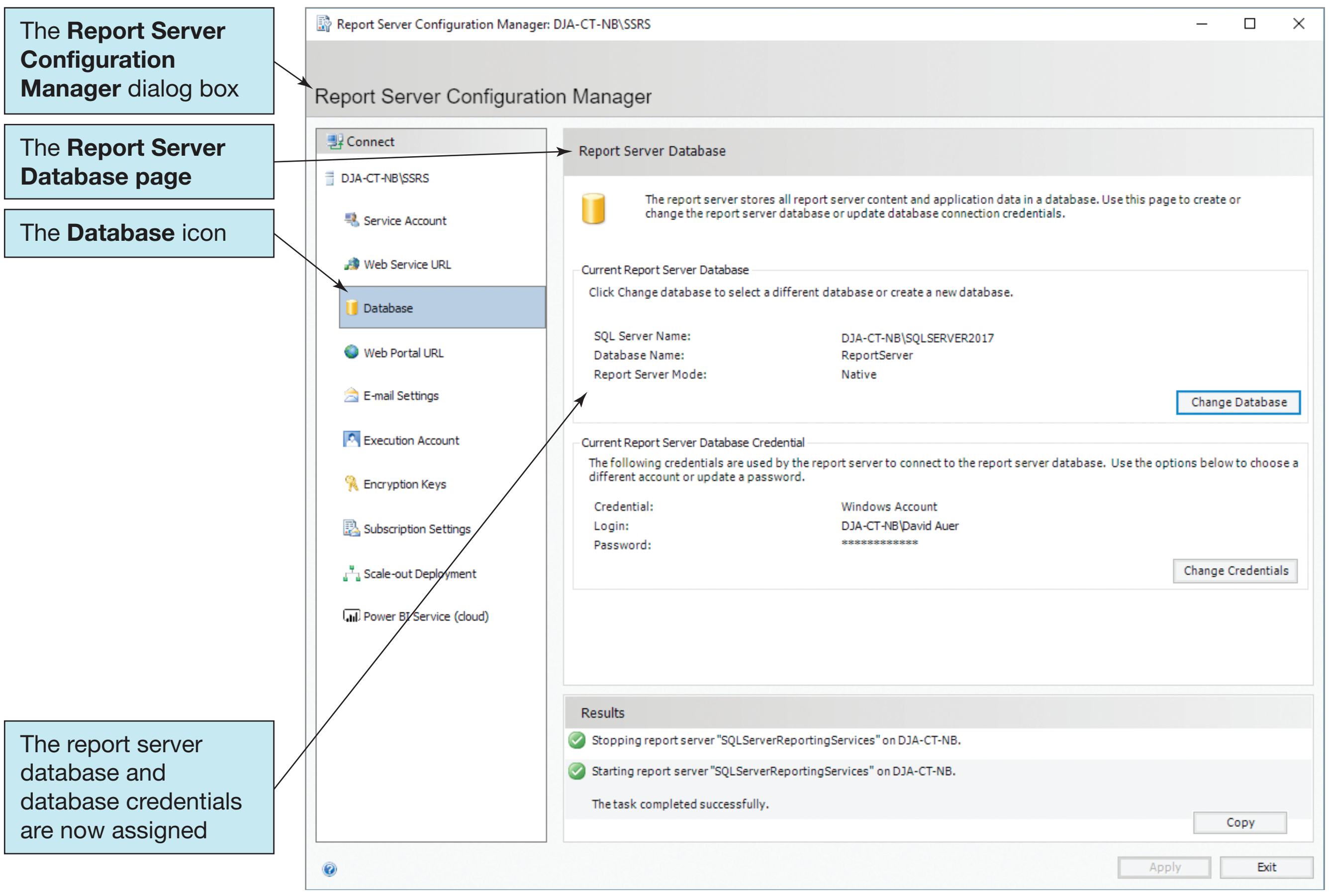

The **Report Server Configuration Manager** dialog box
The **Progress and Finish** page
Progress is shown in this green bar—after each step is complete it is labeled *Success* in the list below
The **Finish** button
Report Server Database Configuration Wizard
Change Database
Choose whether to create or configure a report server database.
Action
Database Server
Database
Credentials
Summary
Progress and Finish
Previous
Finish
Cancel

Please wait while the Report Server Database Configuration wizard configures the database. This might take several minutes to complete.

Verifying database sku	Success
Generating database script	Success
Running database script	Success
Generating rights scripts	Success
Applying connection rights	Success
Setting DSN	Success

(l) The Progress and Finish Page

FIGURE 10A-4
Continued


The **Report Server Configuration Manager** dialog box
The **Report Server Database page**
The **Database** icon
The report server database and database credentials are now assigned

Report Server Configuration Manager: DJA-CT-NB\SSRS

Report Server Configuration Manager

Connect
DJA-CT-NB\SSRS
Service Account
Web Service URL
Database
Web Portal URL
E-mail Settings
Execution Account
Encryption Keys
Subscription Settings
Scale-out Deployment
Power BI Service (cloud)

Report Server Database

The report server stores all report server content and application data in a database. Use this page to create or change the report server database or update database connection credentials.

Current Report Server Database

Click Change database to select a different database or create a new database.

SQL Server Name: DJA-CT-NB\SQLSERVER2017
Database Name: ReportServer
Report Server Mode: Native

Change Database

Current Report Server Database Credential

The following credentials are used by the report server to connect to the report server database. Use the options below to choose a different account or update a password.

Credential: Windows Account
Login: DJA-CT-NB\David Auer
Password: ************

Change Credentials

Results

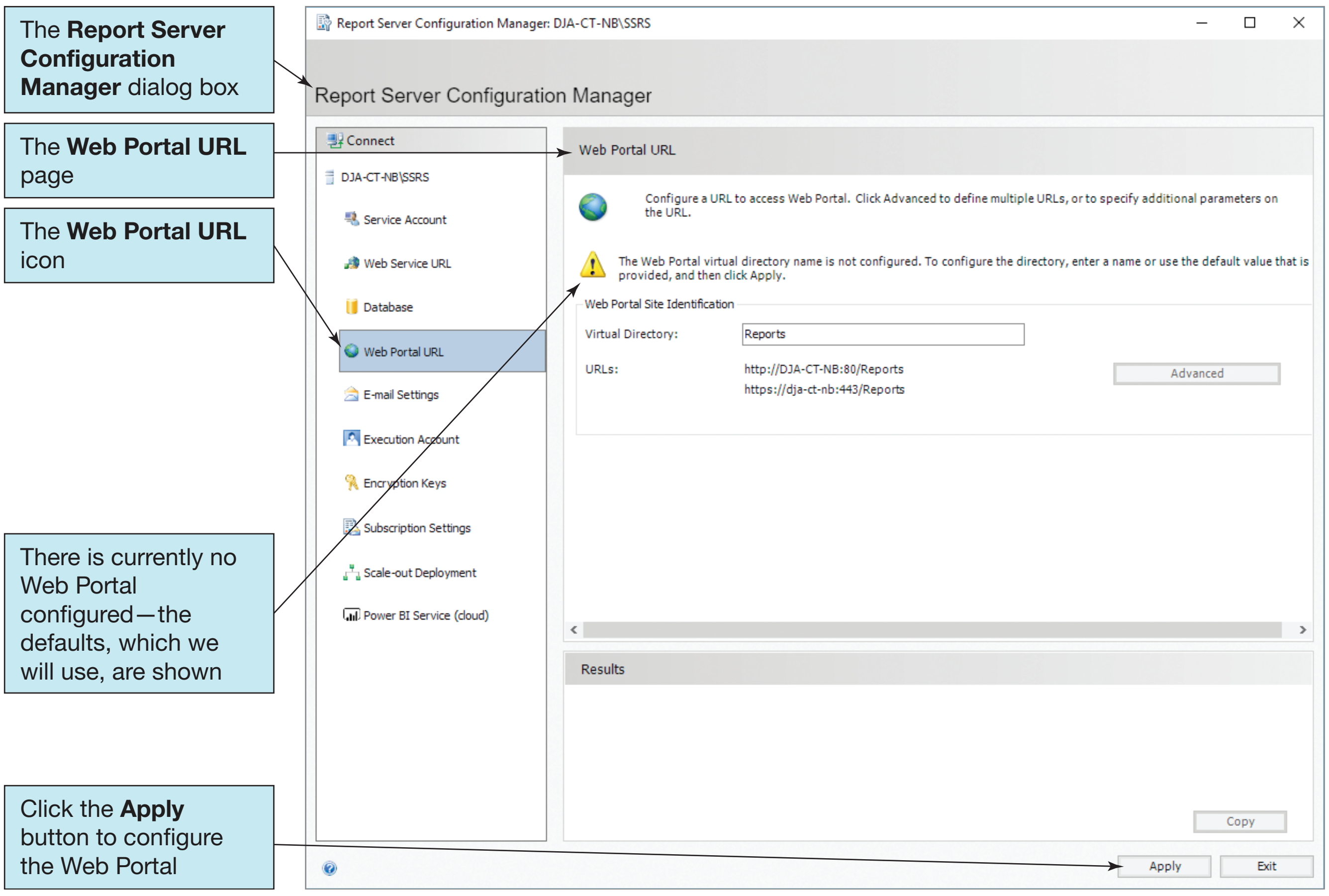
Stopping report server "SQLServerReportingServices" on DJA-CT-NB.
Starting report server "SQLServerReportingServices" on DJA-CT-NB.

The task completed successfully.

Copy

Apply Exit

(m) The Completed Report Server Database Page


The **Report Server Configuration Manager** dialog box
The **Web Portal URL** page
The **Web Portal URL** icon
There is currently no Web Portal configured—the defaults, which we will use, are shown
Click the **Apply** button to configure the Web Portal

Report Server Configuration Manager: DJA-CT-NB\SSRS

Report Server Configuration Manager

Connect
DJA-CT-NB\SSRS
Service Account
Web Service URL
Database
Web Portal URL
E-mail Settings
Execution Account
Encryption Keys
Subscription Settings
Scale-out Deployment
Power BI Service (cloud)

Web Portal URL

Configure a URL to access Web Portal. Click Advanced to define multiple URLs, or to specify additional parameters on the URL.

The Web Portal virtual directory name is not configured. To configure the directory, enter a name or use the default value that is provided, and then click Apply.

Web Portal Site Identification

Virtual Directory: Reports
URLs: http://DJA-CT-NB:80/Reports
https://dja-ct-nb:443/Reports

Advanced

Results

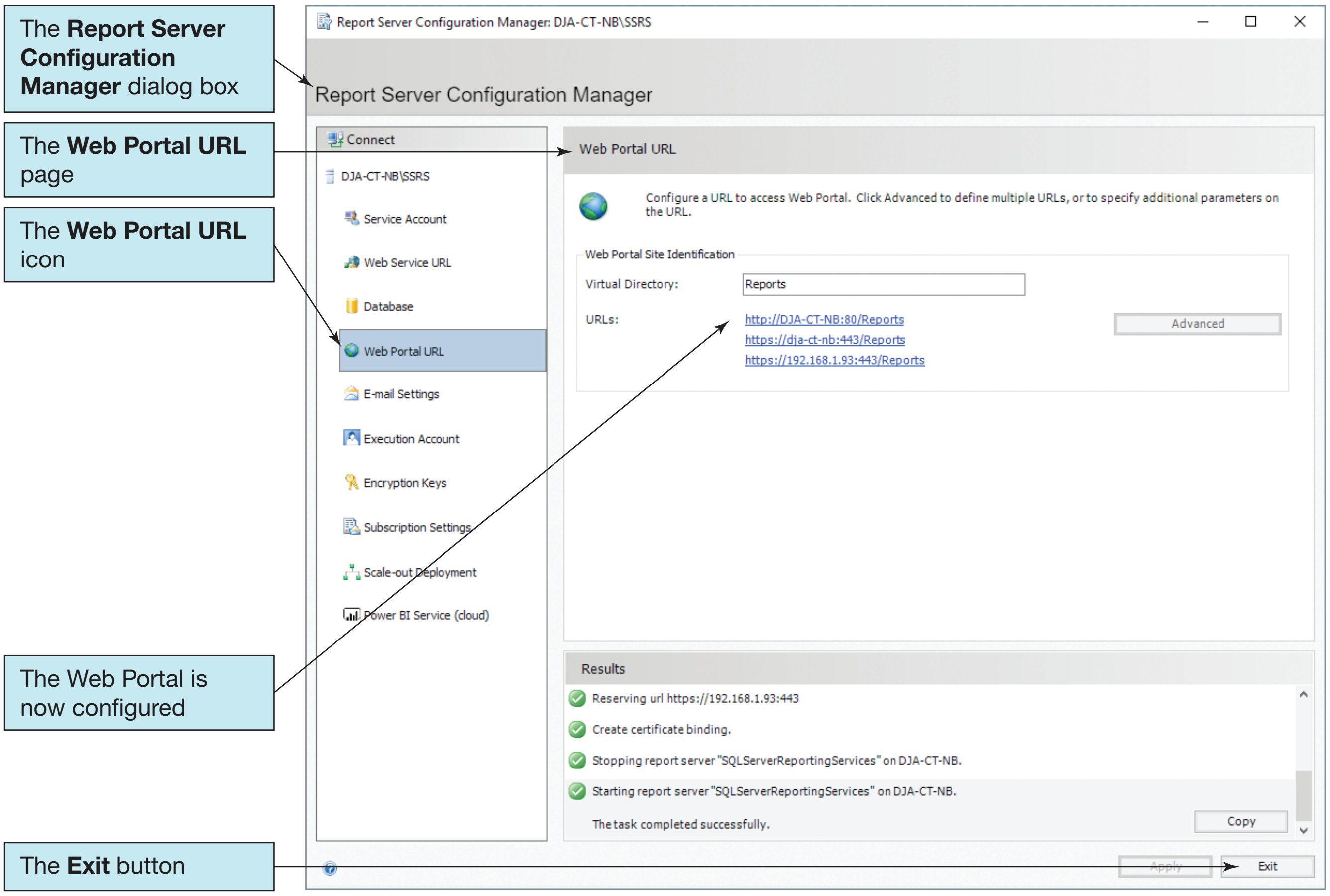
Copy

Apply Exit

(n) The Completed Web Portal URL Page

FIGURE 10A-4

Continued



Web Portal URL

Configure a URL to access Web Portal. Click Advanced to define multiple URLs, or to specify additional parameters on the URL.

Web Portal Site Identification

Virtual Directory: Reports

URLs: http://DJA-CT-NB:80/Reports
https://dja-ct-nb:443/Reports
https://192.168.1.93:443/Reports

Results

```
Reserving url https://192.168.1.93:443
Create certificate binding.
Stopping report server "SQLServerReportingServices" on DJA-CT-NB.
Starting report server "SQLServerReportingServices" on DJA-CT-NB.
The task completed successfully.
```

(o) The Completed Web Portal URL Page

FIGURE 10A-4
Continued

14. Click the **Web Portal URL** icon in the Connect window to display the *Web Portal URL* page, as shown in Figure 10A-4(n). Note that the default settings are correct, but that they have not been applied to the new reporting services configuration. To configure reporting services with this data, click the **Apply** button.
    - **NOTE:** Your list of URLs may differ. Use your list as it appears to you.
15. Figure 10A-4(o) shows the Web Portal URL page with the configured settings. Although other settings can be configured, we do not need to configure them at this time, and we have completed configuring the report services on our SQLSERVER2017 instance of Microsoft SQL Server 2017. Click the **Exit** button.

Do NOT close the **SQL Server Installation Center Dialog Box at this time**. We will need it to install the SQL Server Management Studio GUI utility we use with SQL Server 2017.

Microsoft SQL Server 2017 Utilities

Microsoft SQL Server 2017 provides both **command-line utilities** and **graphical user interface (GUI) utilities** for use with the Microsoft SQL Server DBMS. Although some DBAs prefer command-line utilities, the current GUI utilities are quite comprehensive and user friendly, and after introducing both types, we will use only the GUI utilities for the remainder of this chapter.

SQL CMD and Microsoft PowerShell

In the beginning, there were the command-line utilities. A *command-line utility* is strictly text based. You are presented with a symbolic prompt to show you where to enter your commands. You type in a command (only one at a time) and press the Enter key to execute it. The results are displayed as plaintext (with some rudimentary character-based line- and box-drawing capabilities) in response. All major computer operating systems have their version of a command-line utility. For personal computer users using a Microsoft operating system, the classic example is the MS-DOS command line, which still exists in Windows as the CMD program.

For SQL Server, the classic command-line tool is the **SQL CMD utility**, which is still available as part of the **Microsoft Command Line Utilities 13.1 for SQL Server** package and available for downloading at *https://www.microsoft.com/en-us/download/details.aspx?id=53591*.

However, the functionality of the SQL CMD utility has now been made available in the newer **Microsoft Windows PowerShell** utility. Microsoft PowerShell is a powerful command-line and scripting utility that runs command equivalents called **cmdlets**. The older **sqlps cmdlet** has now been replaced by the **xSQLServer cmdlet**, which makes the capabilities of the SQL CMD utility available in PowerShell. Windows PowerShell can be loaded directly from the Microsoft Windows 10 menu–right-click the **Windows PowerShell** icon in the Windows PowerShell folder, and then click **Run as administrator** to ensure that the utility has all necessary permissions. Or, search for PowerShell using the search icon, then right-click on the desired version (32-bit or 64-bit) to display a shortcut menu, and click **Run as administrator**. Thus, PowerShell carries on and improves upon the command-line tradition. For more information on Microsoft Windows PowerShell, see *https://msdn.microsoft.com/powershell.*

Microsoft SQL CLR

If you are an applications developer using Microsoft Visual Studio as your **integrated development environment (IDE)** for developing applications that use SQL Server, you will probably be using the **SQL Common Language Runtime (CLR)** technology that is built into SQL Server. SQL CLR enables SQL Server to provide Microsoft .NET CLR support, which allows application components written in programming languages such as Visual Basic.NET (VB.NET) to be stored and run in SQL Server. Now database triggers and stored procedures (discussed in Chapter 7 and later in this chapter) can be written in VB.NET or C#.NET instead of standard SQL. SQL CLR is beyond the scope of this book, but you may use it in a class on applications development.

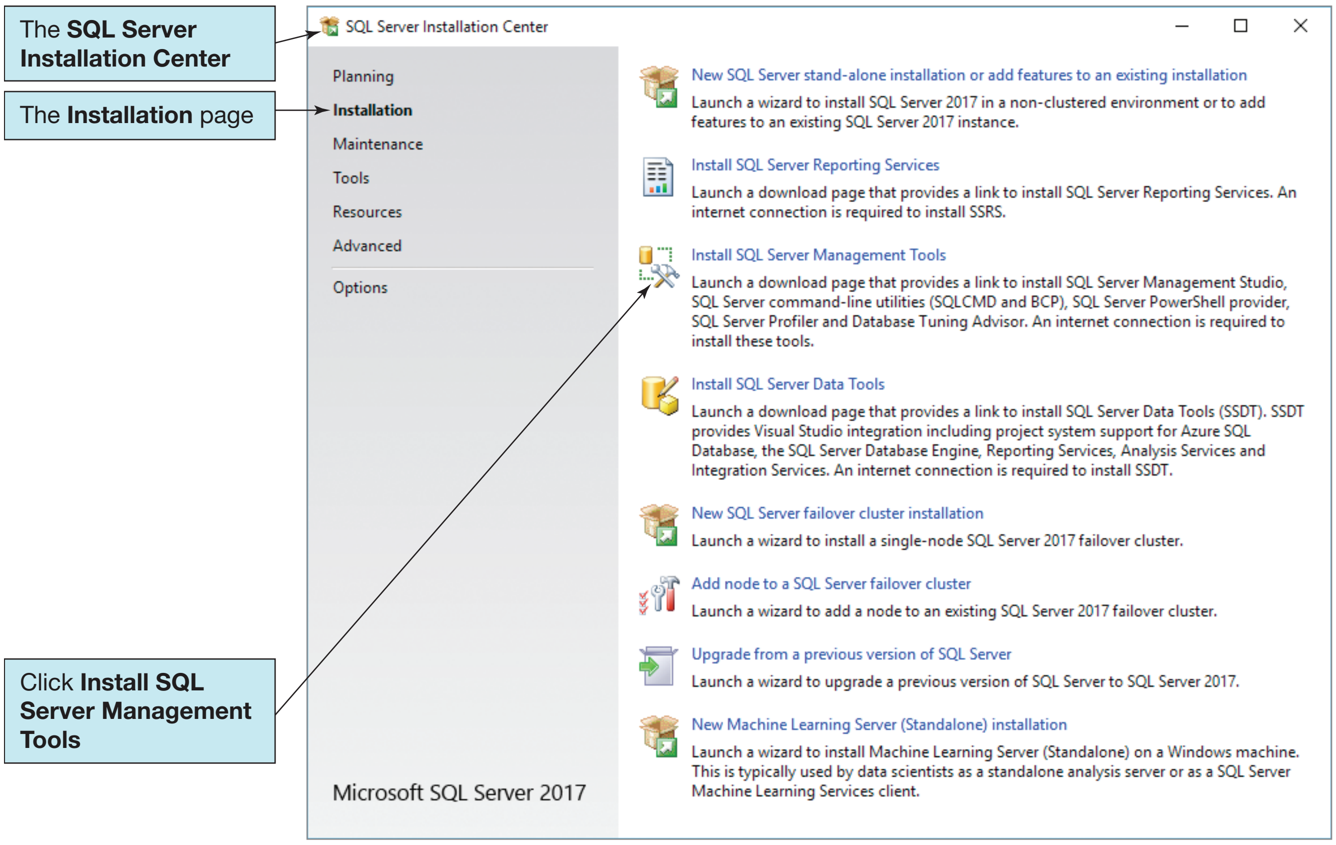
The Microsoft SQL Server Management Studio

Although command-line utilities can be powerful, they can also be tedious and ugly. That's why GUI applications such as Windows were created in the first place. And popular personal databases such as Microsoft Access have certainly put GUI features to good use. Therefore, we have to ask, "Is there a GUI display for SQL Server 2017?" The answer, of course, is "yes, there is," and it is called the *Microsoft SQL Server Management Studio*.

The Microsoft SQL Server Management Studio (SMSS) provides data modeling, database development, and SQL Server 2017 administration tools. Like Microsoft SQL Server Reporting Services, SQL Server Management Studio is now a separate installation. The advantage of this is that the SQL Server Management Studio is a separate program, is not tied to a specific version of SQL Server, and may be updated on a schedule of its own. The installation package, named *SMSS-Setup_ENU.exe* can be downloaded from *https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms*. This Web page is accessed when you click the *Install SQL Server Management Tools* link in the SQL Server Installation Center.

Installing Microsoft SQL Server Management Studio

1. To start the actual installation process, click on **Install SQL Server Management Tools**, as shown in Figure 10A-5(a). Despite what the text for this option states, the only tool that will be installed is SQL Server Management Studio.
2. A Web browser is launched and displays the **Download Microsoft SQL Server Management Studio (SSMS)** Web page, as shown in Figure 10A-5(b).


The SQL Server Installation Center
The Installation page
Click Install SQL Server Management Tools
SQL Server Installation Center
Planning
Installation
Maintenance
Tools
Resources
Advanced
Options
Microsoft SQL Server 2017

New SQL Server stand-alone installation or add features to an existing installation
Launch a wizard to install SQL Server 2017 in a non-clustered environment or to add features to an existing SQL Server 2017 instance.

Install SQL Server Reporting Services
Launch a download page that provides a link to install SQL Server Reporting Services. An internet connection is required to install SSRS.

Install SQL Server Management Tools
Launch a download page that provides a link to install SQL Server Management Studio, SQL Server command-line utilities (SQLCMD and BCP), SQL Server PowerShell provider, SQL Server Profiler and Database Tuning Advisor. An internet connection is required to install these tools.

Install SQL Server Data Tools
Launch a download page that provides a link to install SQL Server Data Tools (SSDT). SSDT provides Visual Studio integration including project system support for Azure SQL Database, the SQL Server Database Engine, Reporting Services, Analysis Services and Integration Services. An internet connection is required to install SSDT.

New SQL Server failover cluster installation
Launch a wizard to install a single-node SQL Server 2017 failover cluster.

Add node to a SQL Server failover cluster
Launch a wizard to add a node to an existing SQL Server 2017 failover cluster.

Upgrade from a previous version of SQL Server
Launch a wizard to upgrade a previous version of SQL Server to SQL Server 2017.

New Machine Learning Server (Standalone) installation
Launch a wizard to install Machine Learning Server (Standalone) on a Windows machine. This is typically used by data scientists as a standalone analysis server or as a SQL Server Machine Learning Services client.

(a) The SQL Server Installation Center Dialog Box Installation Page

FIGURE 10A-5

Installing Microsoft SQL Server Management Studio

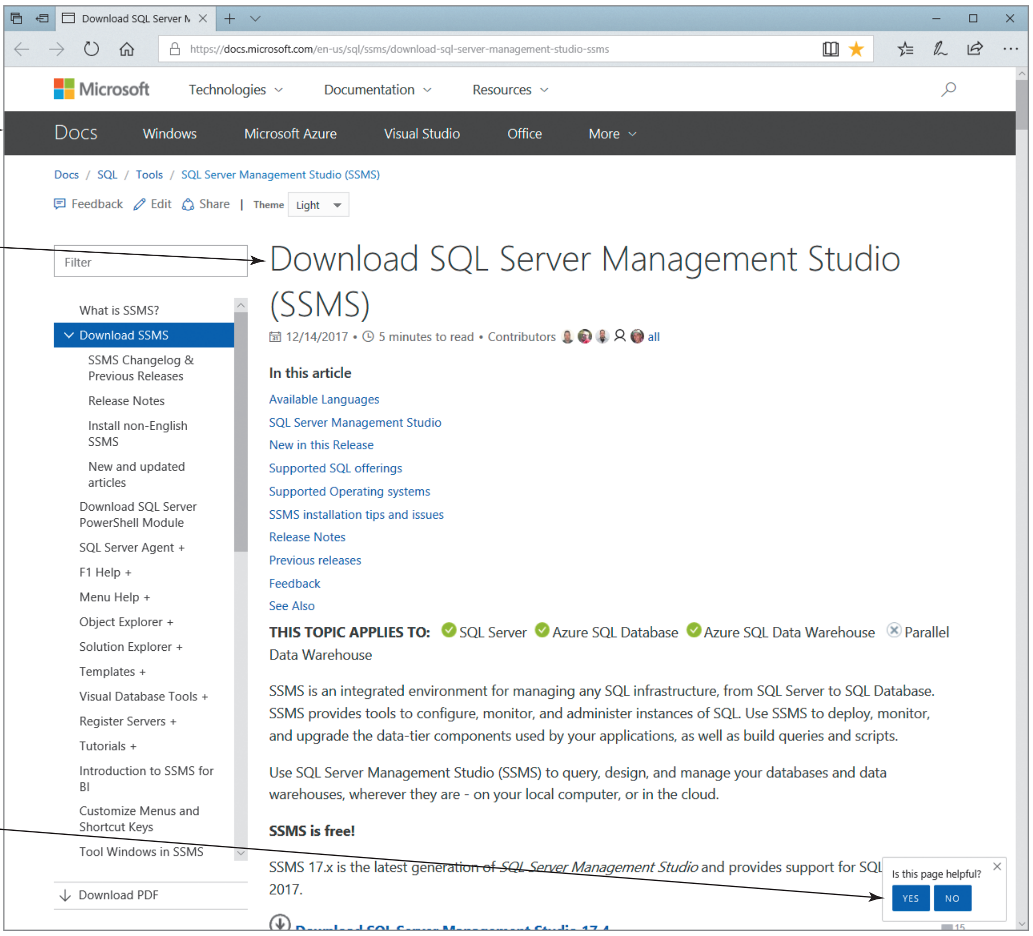
3. Close the **Is this page helpful?** box, and scroll down the Web page until you can see the list of downloadable files, as shown in Figure 10A-5(c). Because this is the initial installation of SQL Server Management Studio, we will choose the option to download the complete installation file. When we need to update SQL Server Management Studio, we will choose the Upgrade Package option. Click the **Download SQL Server Management Studio 17.4** button. A dialog box is displayed at the bottom of the browser. Click **Save** to download the file to the Downloads folder or **Save as** to select which folder the file will be saved in. We are saving our file in *Downloads\Microsoft\MSSQL-SMSS*.
   - NOTE: As this is being written, SSMS 17.4 is the current version. You may see and download a later version.
4. Open File Explorer, and locate the *SMSS-Setup-ENU.exe* installation file in the folder you downloaded it to. By default, the file will be in the Downloads folder itself. However, we have created a *Microsoft* folder in the Downloads folder and a *MSSQL-SMSS* folder in the Microsoft folder, and our file is located there, as shown in Figure 10A-5(d). Right-click the ***SSMS-Setup-ENU.exe*** file to display the shortcut menu, and then click the **Run as administrator** command.
5. A User Access Control dialog box appears asking if you want to run the SMSS-Setup-ENU.exe program. Click the **Yes** button.
6. The Microsoft SQL Server Management Studio Installation dialog box is launched, and the *Welcome* page (actually labeled as the *Welcome. Click "Install" to begin.* page) is displayed, as shown in Figure 10A-5(e). Click the **Install** button.

   - NOTE: If you have more than one version of Microsoft SQL Server installed, you will have components that are shared between these installed versions. In that case, the first page the installation dialog box displays will be the *Setup Warnings*

The **Microsoft Docs** Web site

The **Download SQL Server Management Studio (SSMS)** Web page

Close the **Is this page helpful?** box, and then scroll down to see the list of downloadable programs



Download SQL Server Management Studio (SSMS)

12/14/2017 • 5 minutes to read • Contributors all

In this article

Available Languages
SQL Server Management Studio
New in this Release
Supported SQL offerings
Supported Operating systems
SSMS installation tips and issues
Release Notes
Previous releases
Feedback
See Also

THIS TOPIC APPLIES TO: ✓ SQL Server ✓ Azure SQL Database ✓ Azure SQL Data Warehouse ✗ Parallel Data Warehouse

SSMS is an integrated environment for managing any SQL infrastructure, from SQL Server to SQL Database. SSMS provides tools to configure, monitor, and administer instances of SQL. Use SSMS to deploy, monitor, and upgrade the data-tier components used by your applications, as well as build queries and scripts.

Use SQL Server Management Studio (SSMS) to query, design, and manage your databases and data warehouses, wherever they are - on your local computer, or in the cloud.

SSMS is free!

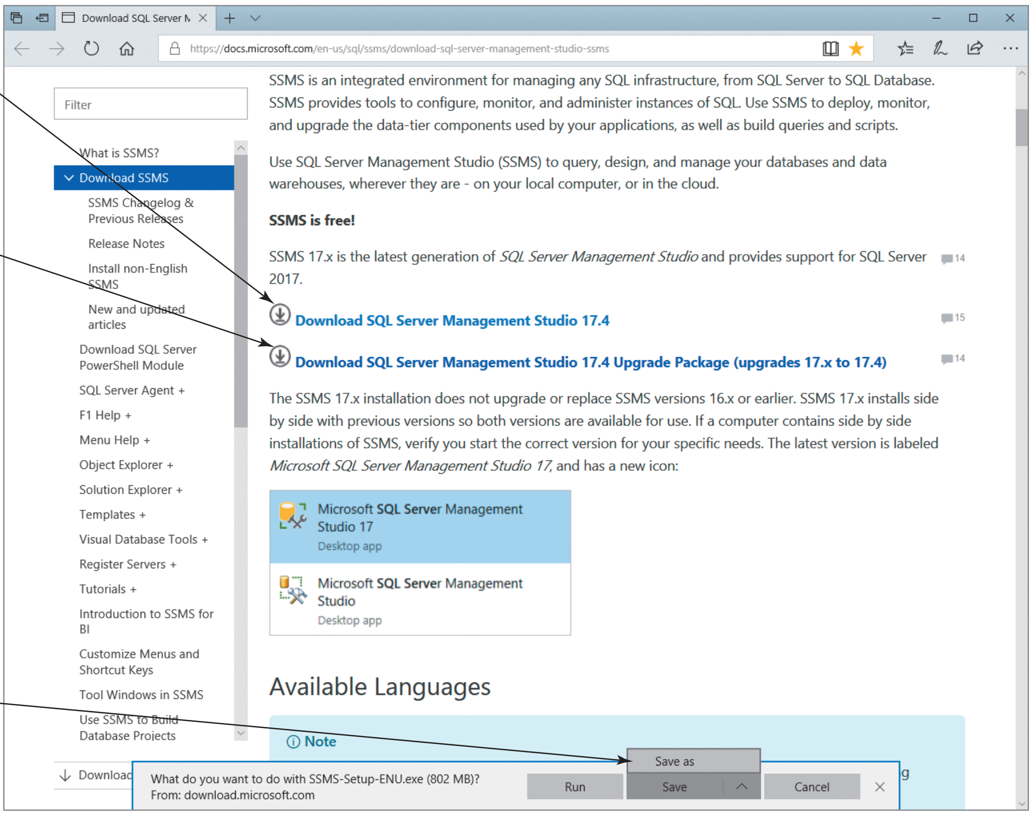
SSMS 17.x is the latest generation of *SQL Server Management Studio* and provides support for SQL 2017.

(b) The Microsoft SQL Server Management Studio (SSMS) Download Web Page

Click **Download SQL Server Management Studio 17.3** for the initial installation

Download the **Upgrade Package** when updating your version of **SQL Server Management Studio**

Click **Save** to download the file to the *Documents* folder, or click **Save as** to select a specific folder for the file download



SSMS is an integrated environment for managing any SQL infrastructure, from SQL Server to SQL Database. SSMS provides tools to configure, monitor, and administer instances of SQL. Use SSMS to deploy, monitor, and upgrade the data-tier components used by your applications, as well as build queries and scripts.

Use SQL Server Management Studio (SSMS) to query, design, and manage your databases and data warehouses, wherever they are - on your local computer, or in the cloud.

SSMS is free!

SSMS 17.x is the latest generation of *SQL Server Management Studio* and provides support for SQL Server 2017.

Download SQL Server Management Studio 17.4

Download SQL Server Management Studio 17.4 Upgrade Package (upgrades 17.x to 17.4)

The SSMS 17.x installation does not upgrade or replace SSMS versions 16.x or earlier. SSMS 17.x installs side by side with previous versions so both versions are available for use. If a computer contains side by side installations of SSMS, verify you start the correct version for your specific needs. The latest version is labeled *Microsoft SQL Server Management Studio 17*, and has a new icon:

Microsoft SQL Server Management Studio 17
Desktop app

Microsoft SQL Server Management Studio
Desktop app

Available Languages

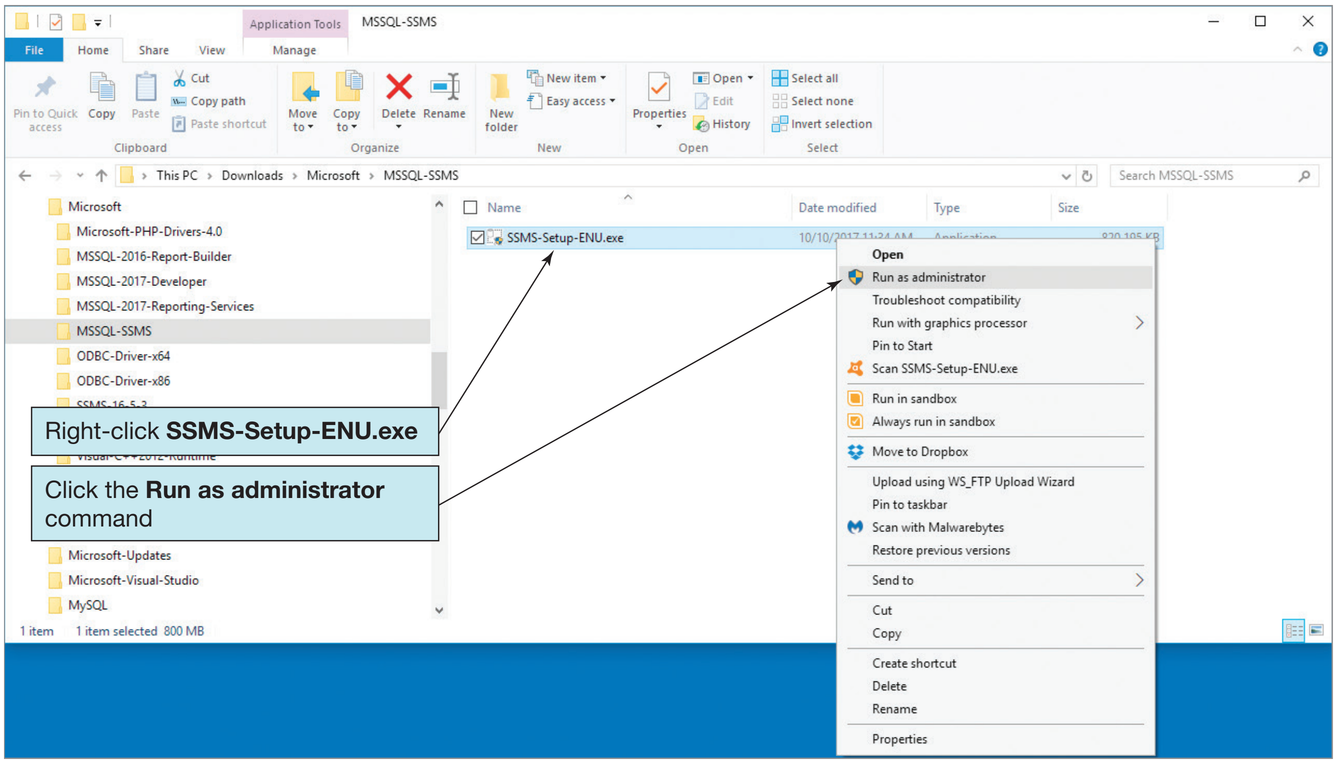
Note

What do you want to do with SSMS-Setup-ENU.exe (802 MB)?
From: download.microsoft.com

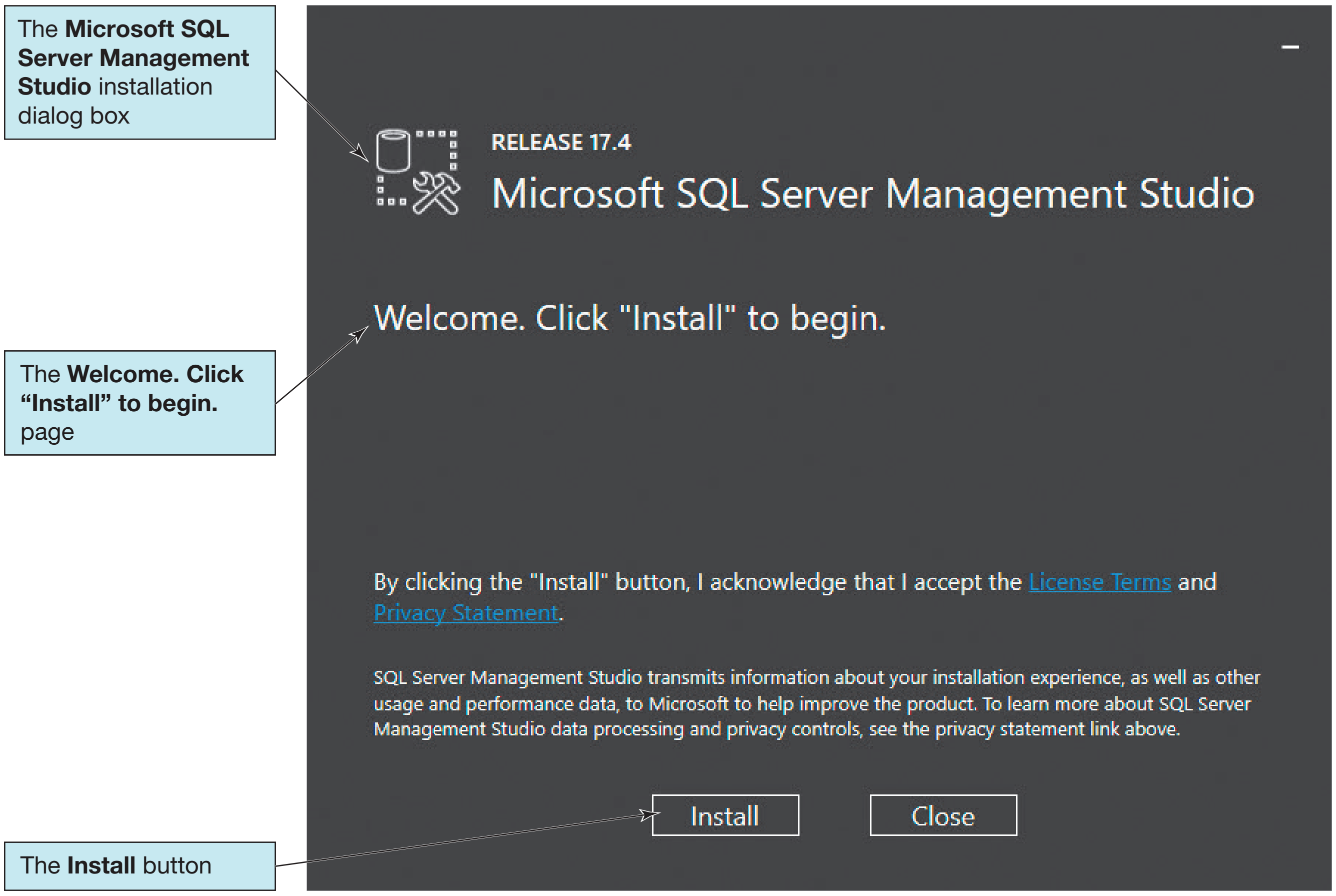
(c) Saving the SSMS-Setup-ENU.exe File

FIGURE 10A-5

Continued



(d) The SMSS-Setup-ENU.exe file and Shortcut Menu


The Microsoft SQL Server Management Studio installation dialog box
The Welcome. Click "Install" to begin. page
The Install button
RELEASE 17.4
Microsoft SQL Server Management Studio
Install
Close

Welcome. Click "Install" to begin.

By clicking the "Install" button, I acknowledge that I accept the License Terms and Privacy Statement.

SQL Server Management Studio transmits information about your installation experience, as well as other usage and performance data, to Microsoft to help improve the product. To learn more about SQL Server Management Studio data processing and privacy controls, see the privacy statement link above.

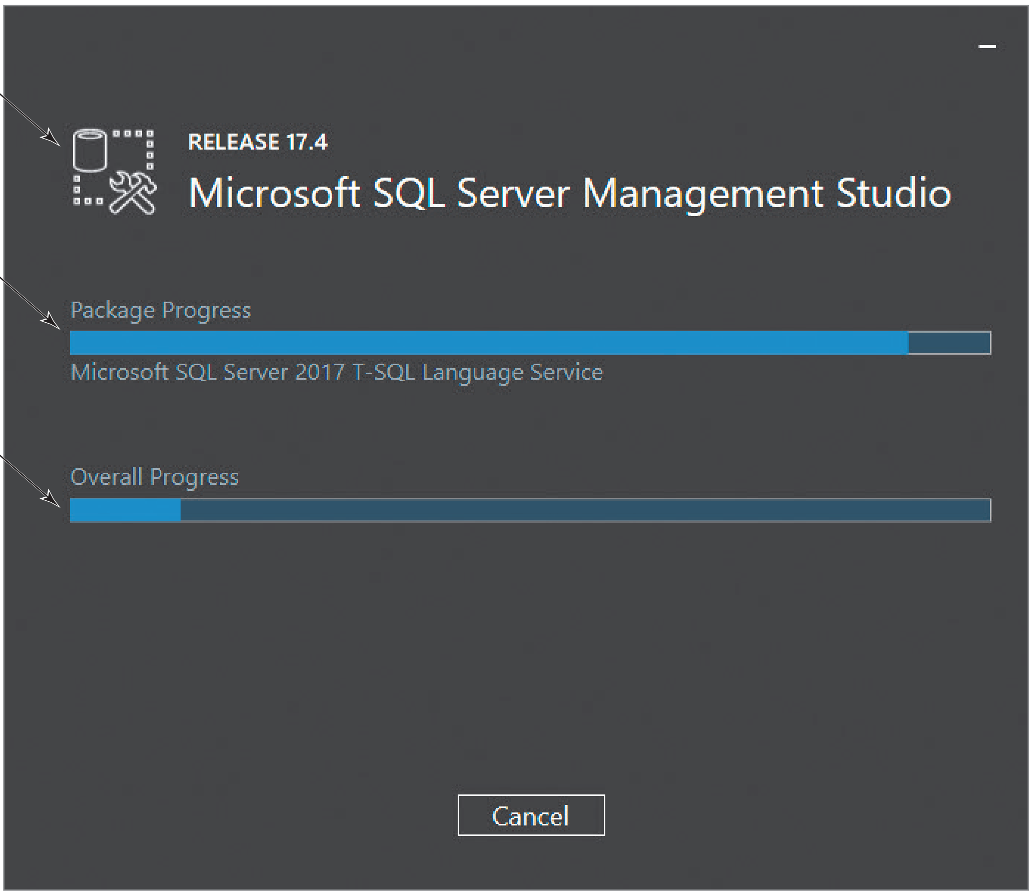
(e) The Microsoft SQL Server Management Studio Installation Welcome Page

FIGURE 10A-5
Continued

The **Microsoft SQL Server Management Studio** installation dialog box

The **Package Progress** bar shows the progress of installing a particular component

The **Overall Progress** bar shows the progress of the entire installation of SQL Server Management Studio



RELEASE 17.4

Microsoft SQL Server Management Studio

Package Progress

Microsoft SQL Server 2017 T-SQL Language Service

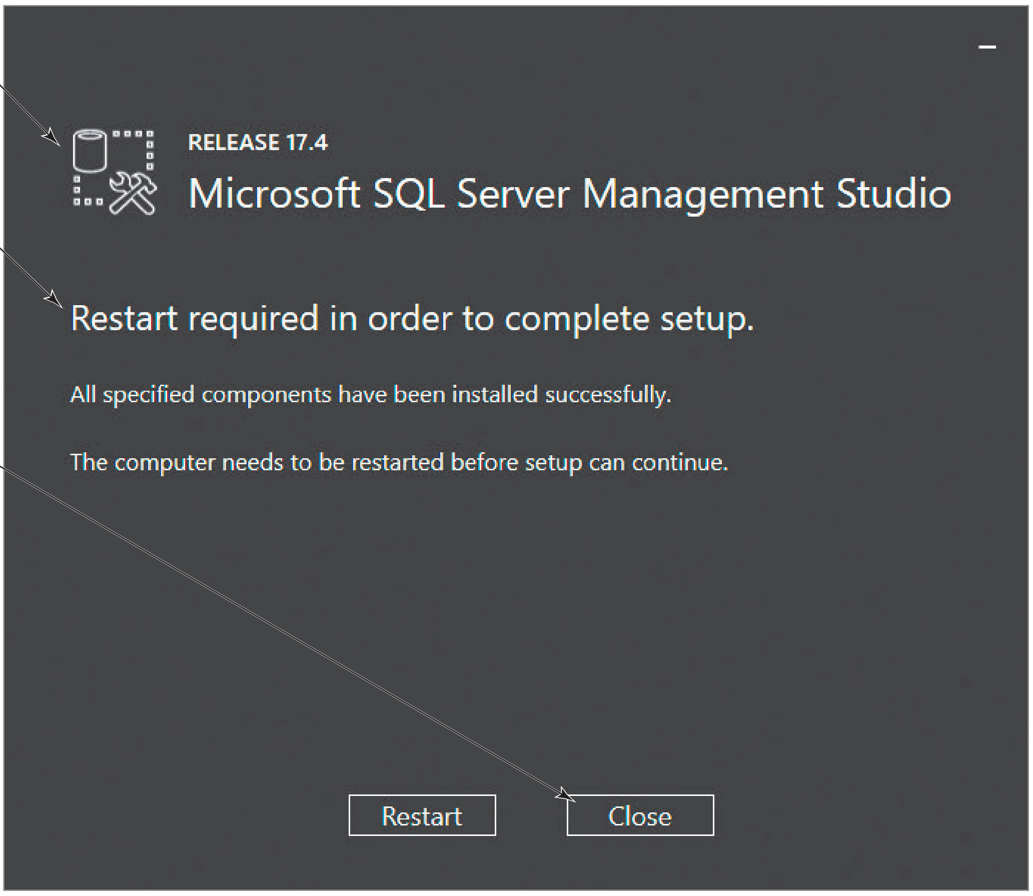
Overall Progress

(f) The Microsoft SQL Server Management Studio Installation Progress Page

The **Microsoft SQL Reporting Services** installation dialog box

The **Restart required in order to complete setup** page

The **close** button



RELEASE 17.4

Microsoft SQL Server Management Studio

Restart required in order to complete setup.

All specified components have been installed successfully.

The computer needs to be restarted before setup can continue.

FIGURE 10A-5

Continued

(g) The Microsoft SQL Server Management Studio Restart Required Page

page, which tells you that these components may be updated, and asking if you wish to continue with the installation. Click the **Yes** button, and Welcome page will be displayed.

7. The installation begins, and package and overall progress bars are displayed, as shown in Figure 10A-5(f). The installation will take a while, so be patient!
8. When the installation is complete, the *Restart required in order to complete setup* page is displayed, as shown in Figure 10A-5(g). Click the **Close** button, then close the SQL Server Installation Center dialog box, and then close all other programs that are running as restart your computer.

This completes our work installing and configuring Microsoft SQL Server 2017, Microsoft SQL Server Reporting Services, and the SQL Server Management Studio.

Using Microsoft SQL Server 2017

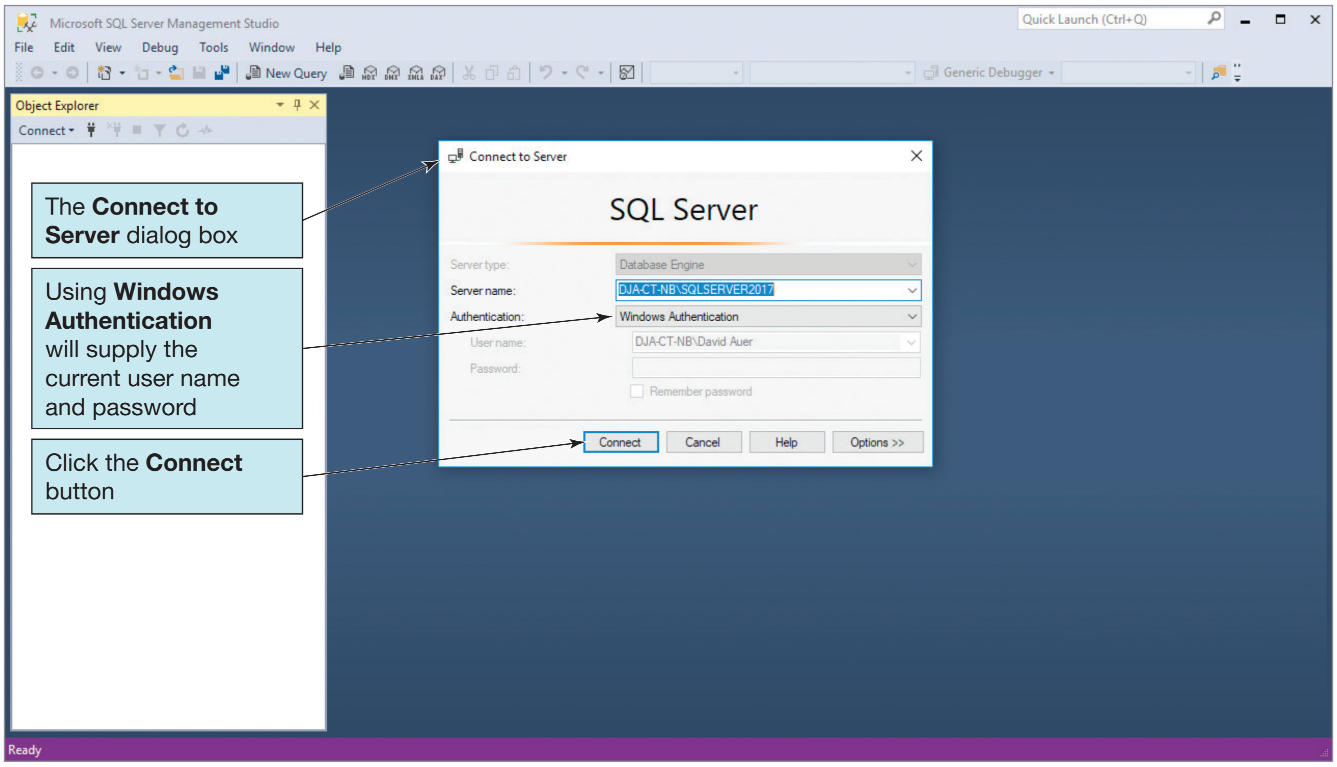
In this book, we are running SQL Server 2017 Developer Edition in Microsoft Windows 10. To open the Microsoft SQL Server Management Studio in Windows 10, click the Windows button, and then select **Microsoft SQL Server Management Studio 17** in the Microsoft SQL Server Tools 17 folder. The Microsoft SQL Server Management Studio is displayed, along with the **Connect to Server dialog box**, as shown in Figure 10A-6.

BY THE WAY You can pin the Microsoft SQL Server Management Studio Express program icon to either or both the Start Menu and the Taskbar. This makes it a lot easier to start the program if you are using it a lot.

To pin the program icon to either the Start menu or the taskbar, follow the command instructions earlier to locate the **Microsoft SQL Server Management Studio** program icon, and then right-click the program icon. A shortcut menu will be displayed. Select either **Pin to Start** or **More | Pin to Taskbar**.

FIGURE 10A-6

Connecting to SQL Server 2017 with the SQL Server Management Studio

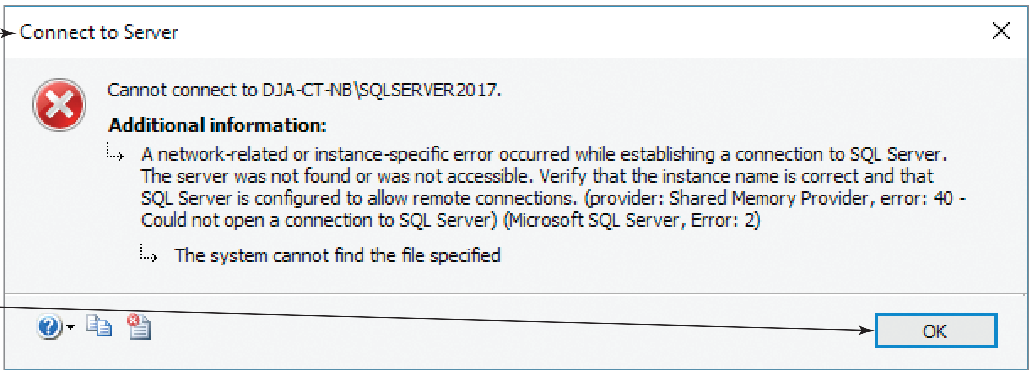

Microsoft SQL Server Management Studio
File Edit View Debug Tools Window Help
New Query
Generic Debugger
Quick Launch (Ctrl+Q)
Object Explorer
Connect
The Connect to Server dialog box
Using Windows Authentication will supply the current user name and password
Click the Connect button
Connect to Server
SQL Server
Connect
Cancel
Help
Options >>
Ready

Server type: Database Engine
Server name: DJA-CT-NB\SQLSERVER2017
Authentication: Windows Authentication
User name: DJA-CT-NB\David Auer
Password:
Remember password

(a) The Connect to Server Dialog Box

The **Connect to Server** error message

The **OK** button



Cannot connect to DJA-CT-NB\SQLSERVER2017.

Additional information:

A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Shared Memory Provider, error: 40 - Could not open a connection to SQL Server) (Microsoft SQL Server, Error: 2)

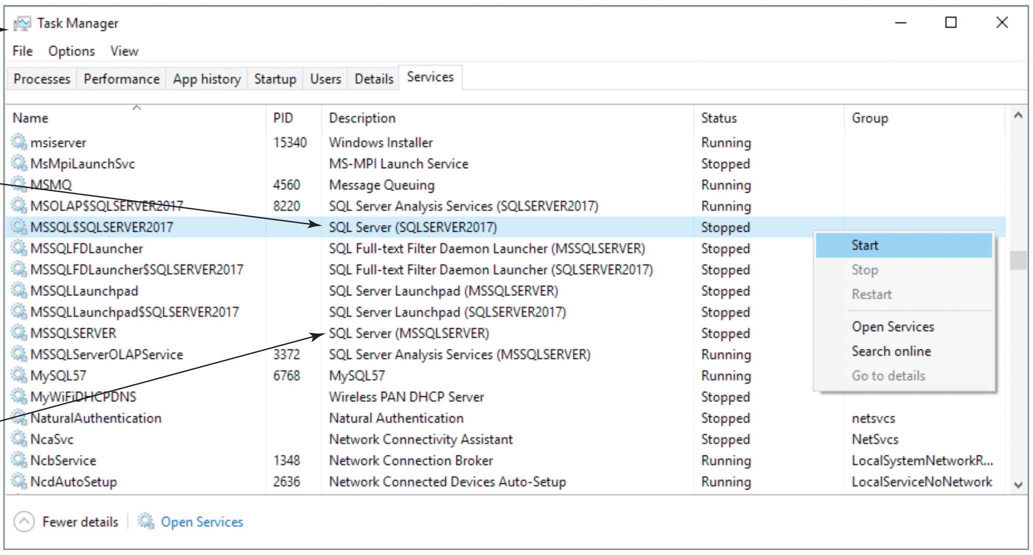
The system cannot find the file specified

(b) The Connect to Server Error Message

Task Manager

This is SQL Server 2017 with instance name **SQLSERVER2017** —right-click to open the shortcut menu and click the **Start** command

This is SQL Server 2016 with instance name **MSSQLSERVER**—we had previously installed SQL Server 2016, and it was assigned the default instance name of MSSQLSEVER



Name	PID	Description	Status	Group
msiserver	15340	Windows Installer	Running	
MsMpiLaunchSvc		MS-MPI Launch Service	Stopped	
MSMQ	4560	Message Queuing	Running	
MSOLAP$SQLSERVER2017	8220	SQL Server Analysis Services (SQLSERVER2017)	Running	
MSSQL$SQLSERVER2017		SQL Server (SQLSERVER2017)	Stopped	
MSSQLFDLauncher		SQL Full-text Filter Daemon Launcher (MSSQLSERVER)	Stopped	
MSSQLFDLauncher$SQLSERVER2017		SQL Full-text Filter Daemon Launcher (SQLSERVER2017)	Stopped	
MSSQLLaunchpad		SQL Server Launchpad (MSSQLSERVER)	Stopped	
MSSQLLaunchpad$SQLSERVER2017		SQL Server Launchpad (SQLSERVER2017)	Stopped	
MSSQLSERVER		SQL Server (MSSQLSERVER)	Stopped	
MSSQLServerOLAPService	3372	SQL Server Analysis Services (MSSQLSERVER)	Running	
MySQL57	6768	MySQL57	Running	
MyWiFiDHCPDNS		Wireless PAN DHCP Server	Stopped	
NaturalAuthentication		Natural Authentication	Stopped	netsvcs
NcaSvc		Network Connectivity Assistant	Stopped	NetSvcs
NcbService	1348	Network Connection Broker	Running	LocalSystemNetworkR...
NcdAutoSetup	2636	Network Connected Devices Auto-Setup	Running	LocalServiceNoNetwork

(c)

FIGURE 10A-6

Continued

If you use the default *Windows Authentication* method or *Mixed* authentication method (which we set up during Microsoft SQL Server 2017 installation), you will be authenticated using your current Windows username and password, so just click the **Connect button.** Note that the Server name used in the *Connect to Server* dialog box is *{ComputerName}/{SQLServerInstanceName}*. If you are using the default SQL Server instance name of MSSQLSERVER, then you can connect with just the {ComputerName} as the Server name.

Note that we have repeatedly experienced a problem when starting the Microsoft SQL Server Management Studio, and when this occurs, the error message shown in Figure 10A-6(b) is displayed. The message indicates that, for some reason, the main SQL Server 2017 database engine has not started. Because the database engine is set to start automatically, we really don't understand why this error is happening–sometimes it starts and sometimes it doesn't (we started noticing this behavior with SQL Server 2016, but have not seen it prior to that).

To fix this problem if it occurs, click the **OK** button to close the error message, and then use the **Windows key** + **X key** combination on the keyboard to display a shortcut menu. In the menu, click the **Task Manager** command to display the Task Manager, as shown in Figure 10A-6(c). In Task Manager, click the **Services** tab and scroll down until you see the *SQLSERVER2017* service (note that we have previously installed SQL Server 2016 on our computer, and it used the default instance *MSSQLSERVER* instance name for the associated service). Right-click the **SQLSERVER2017** service to bring up a shortcut menu, and then click the **Start** command.

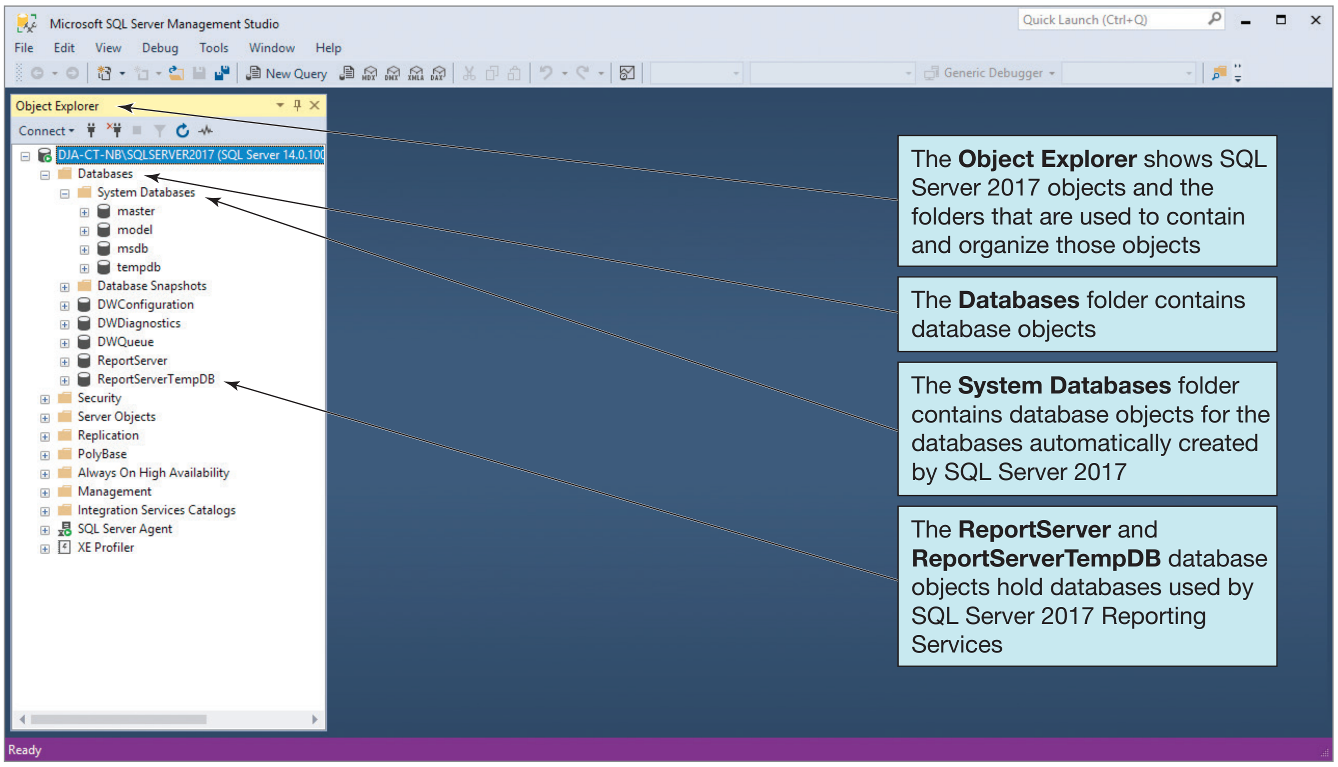
Also start the associated SQL Server Launchpad (labeled *MSSQLLauchpad$-SQLSERVER2017*), SQL Full-text Filter Daemon (labeled *MSSQLFDLauncher$-SQLSERVER2017*), SQL Server Agent (labeled *SQLAgent$SQLSERVER2017*), and SQLServerReportingServices services. After these services are started, close the Task Manager and then log into Microsoft SQL Server 2017.

When you complete your login, the Microsoft SQL Server Management Studio window appears. Find the folder icon labeled Databases in the Object Explorer, as shown in Figure 10A-7. Click the plus sign to open it, and then open the System Database folder the same way. As shown in Figure 10A-7(a), objects representing the databases managed by the SQL Server 2017 DBMS are displayed in these folders. For example, you can see the ReportServer and ReportServerTempDB databases (used by SQL Server 2017 Reporting Services, which we installed earlier in this chapter and which we will discuss in Appendix J, "Business Intelligence Systems") in the Databases folder. There are also three databases prefixed with *DW*, which are used by PolyBase. The master database (which holds the DBMS and database metadata) is in the System Databases folder. We normally keep the System Databases folder closed because we usually do not work with these databases.

Note that when you open the Microsoft SQL Server Management Studio, it will check for updates, notify you when an update is available, and ask if you want to get the update. Alternatively, you can manually check for updates by using the **Tools | Check for Updates** command. If you click to accept the update or manually check for updates, the **SQL Server Management Studio Updates** dialog box will be displayed, as shown in Figure 10A-7(b). If updates are available, an Update button will be displayed, and clicking on the Update button takes you to the same Download SQL Server Management Studio (SSMS) Web page shown in Figure 10A-5(b). Download the update, close the SQL Server Management Studio, and install the updated version using the same steps previously discussed.

FIGURE 10A-7

The Microsoft SQL Server Management Studio



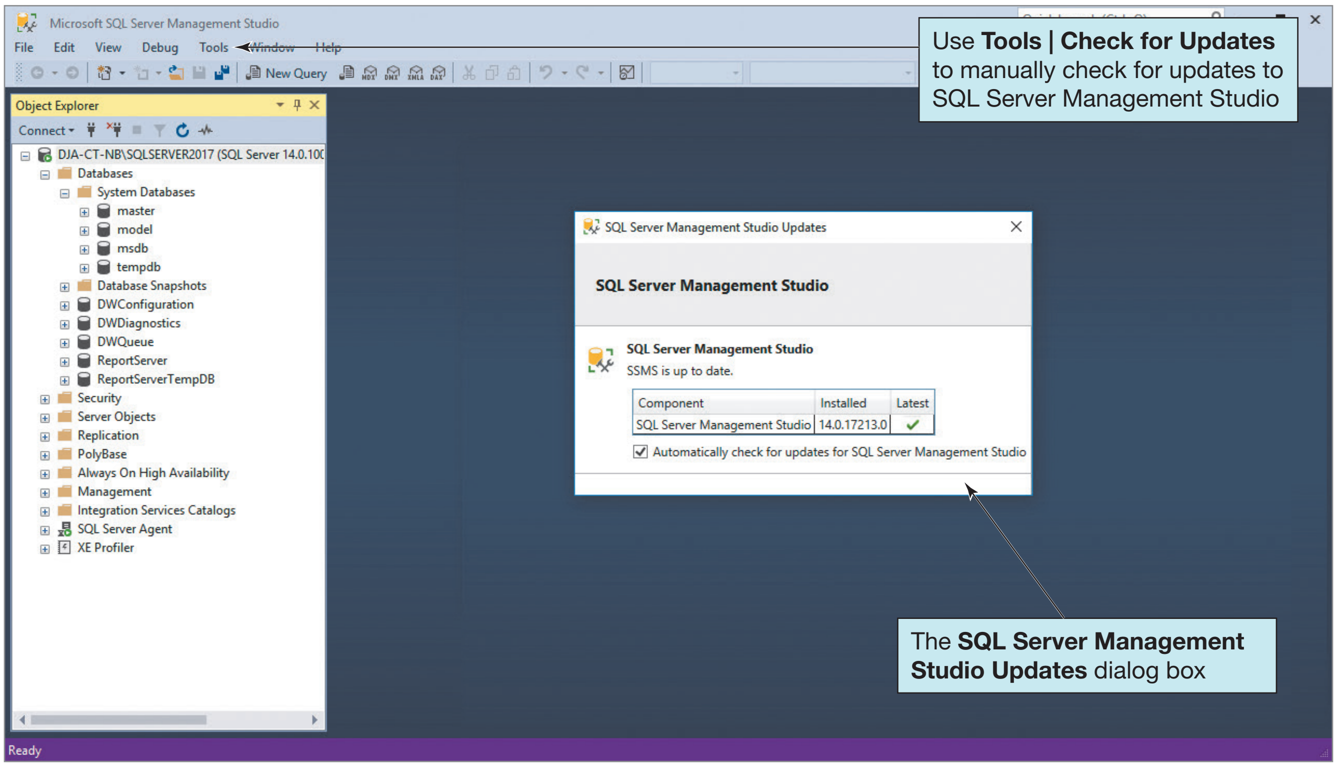
The **Object Explorer** shows SQL Server 2017 objects and the folders that are used to contain and organize those objects

The **Databases** folder contains database objects

The **System Databases** folder contains database objects for the databases automatically created by SQL Server 2017

The **ReportServer** and **ReportServerTempDB** database objects hold databases used by SQL Server 2017 Reporting Services

(a) Database Objects in Object Explorer



SQL Server Management Studio
SSMS is up to date.

Component	Installed	Latest
SQL Server Management Studio	14.0.17213.0	✓

☑ Automatically check for updates for SQL Server Management Studio

Use **Tools | Check for Updates** to manually check for updates to SQL Server Management Studio

The **SQL Server Management Studio Updates** dialog box

FIGURE 10A-7 (b) The SQL Server Management Studio Updates Dialog Box
Continued

Creating a Microsoft SQL Server 2017 Database

Now that Microsoft SQL Server 2017 is installed and the Microsoft SQL Server Management Studio is open, we can create a new database. We will create a database named *Cape_Codd* for the Cape Codd Outdoor Sports database we use in Chapter 2 to discuss SQL Query statements.

Creating an SQL Server 2017 Database

1. Right-click the **Databases** folder in the Object Explorer to display a shortcut menu, as shown in Figure 10A-8.
2. Click the **New Database** command to display the New Database dialog box, as shown in Figure 10A-9.
3. Type the database name **Cape_Codd** in the Database Name text box, and then click the **OK** button. The database is created.
4. Click the **Refresh** button—the databases are sorted into alphabetical order and the Cape_Codd database object is displayed in the Object Explorer, as shown in Figure 10A-10. Click the plus (+) button to display the Cape_Codd database folders, also shown in Figure 10A-10.
5. Right-click the **Cape_Codd** database object to display a shortcut menu, and then click the **Properties** command. The Database Properties–Cape_Codd dialog box is displayed, as shown in Figure 10A-11.
6. In the **Database Properties–Cape_Codd** dialog box, click the **Files** page object, as shown in Figure 10A-11. The database files associated with the Cape_Codd database are displayed.
7. Click the **OK** button in the Database Properties–Cape_Codd dialog box to close the dialog box.

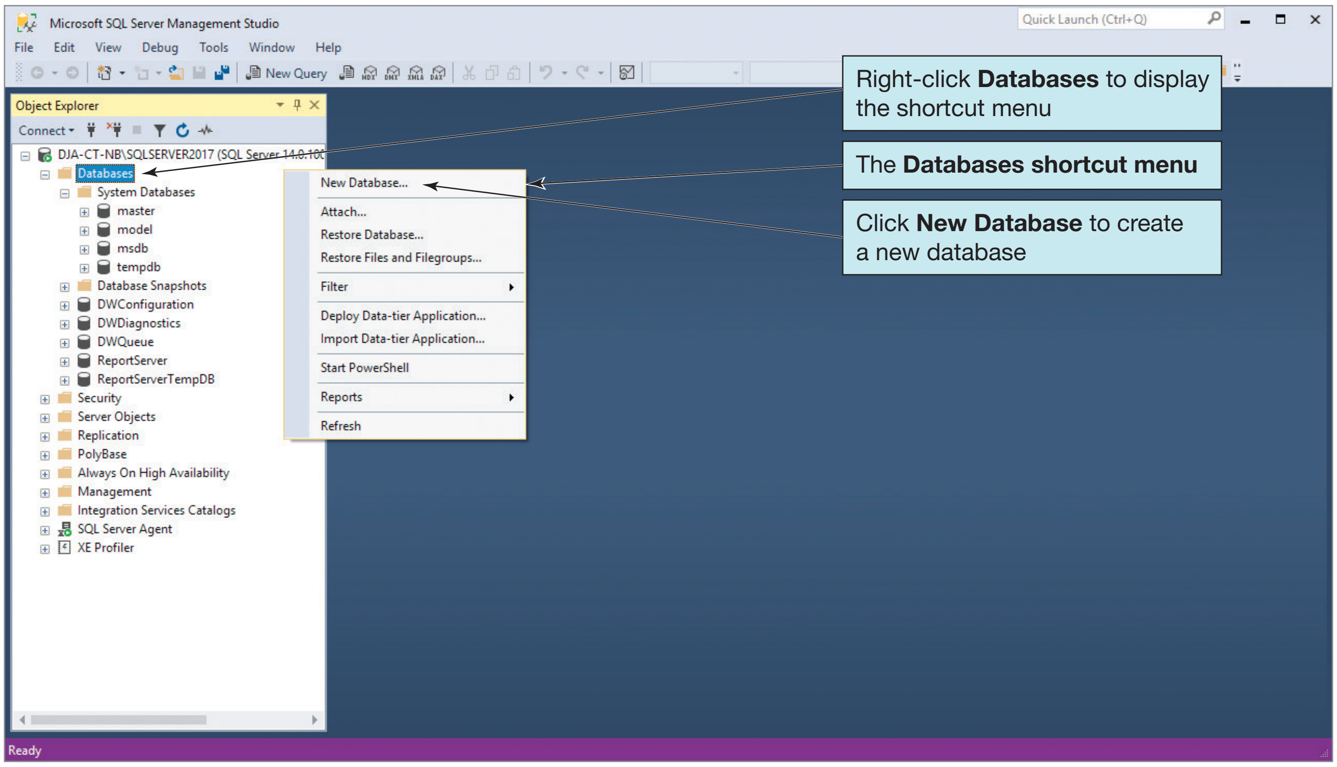
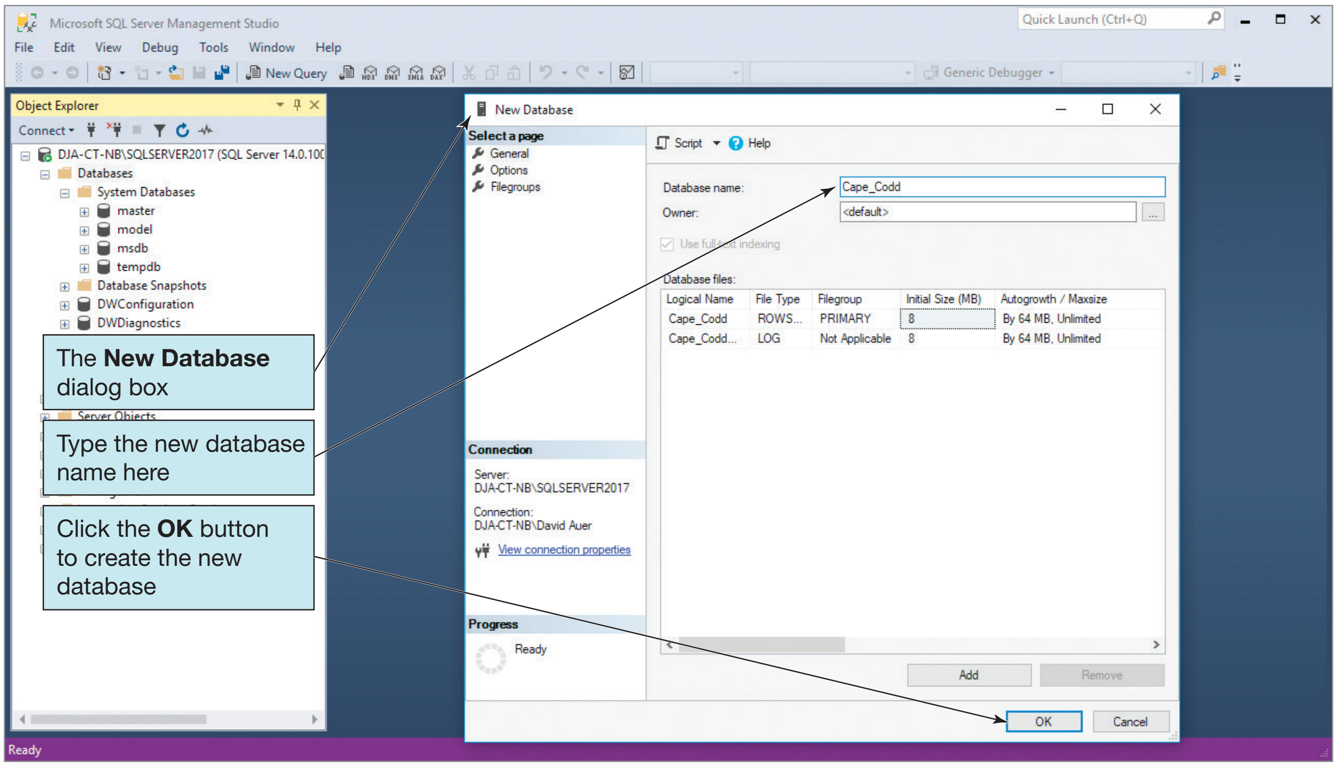

Microsoft SQL Server Management Studio
Object Explorer
DJA-CT-NB\SQLSERVER2017 (SQL Server 14.0.100
Databases
New Database...
Attach...
Restore Database...
Restore Files and Filegroups...
Filter
Deploy Data-tier Application...
Import Data-tier Application...
Start PowerShell
Reports
Refresh
Right-click **Databases** to display the shortcut menu
The **Databases shortcut menu**
Click **New Database** to create a new database

FIGURE 10A-8

The New Database Command


New Database
Select a page
General
Options
Filegroups
Script
Help
The **New Database** dialog box
Type the new database name here
Click the **OK** button to create the new database
OK
Cancel

Database name: Cape_Codd

Owner: <default>

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize
Cape_Codd	ROWS...	PRIMARY	8	By 64 MB, Unlimited
Cape_Codd...	LOG	Not Applicable	8	By 64 MB, Unlimited

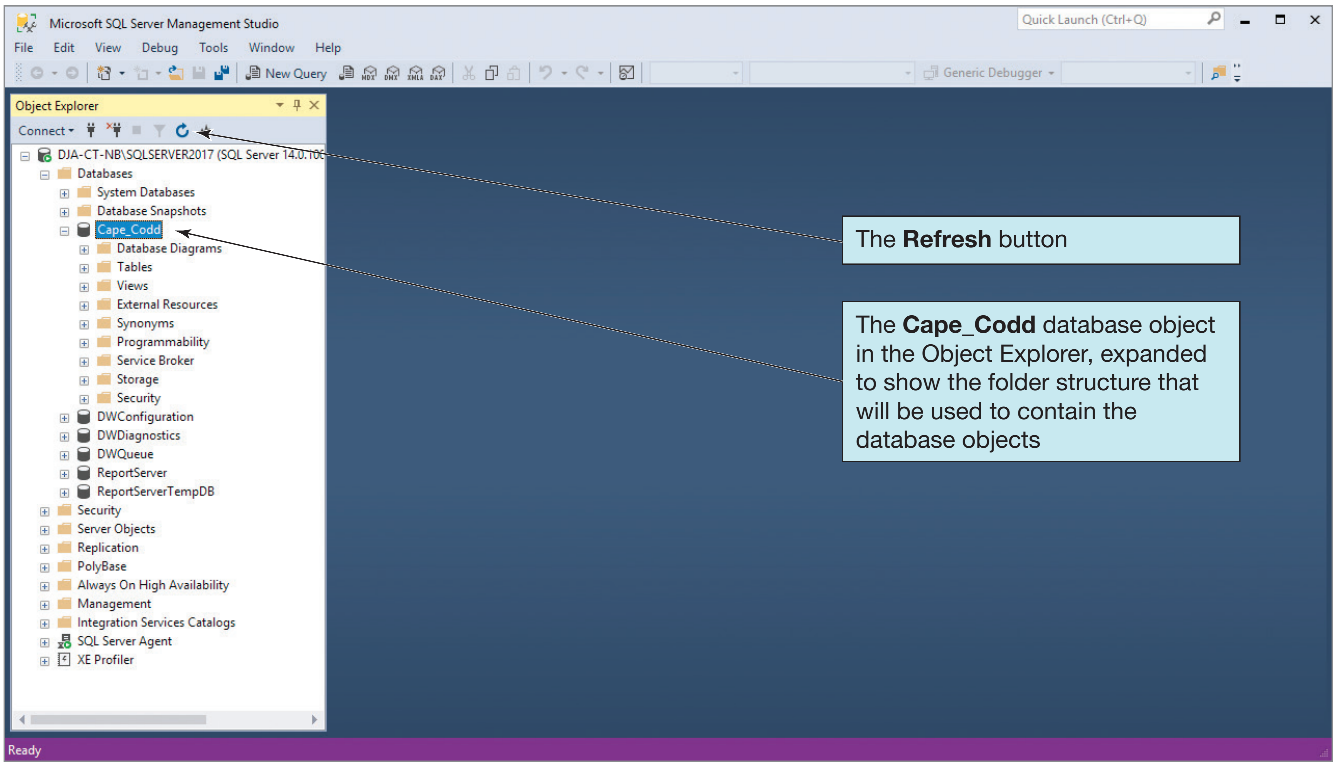
Connection

Server: DJA-CT-NB\SQLSERVER2017

Connection: DJA-CT-NB\David Auer

FIGURE 10A-9

Naming the New Database

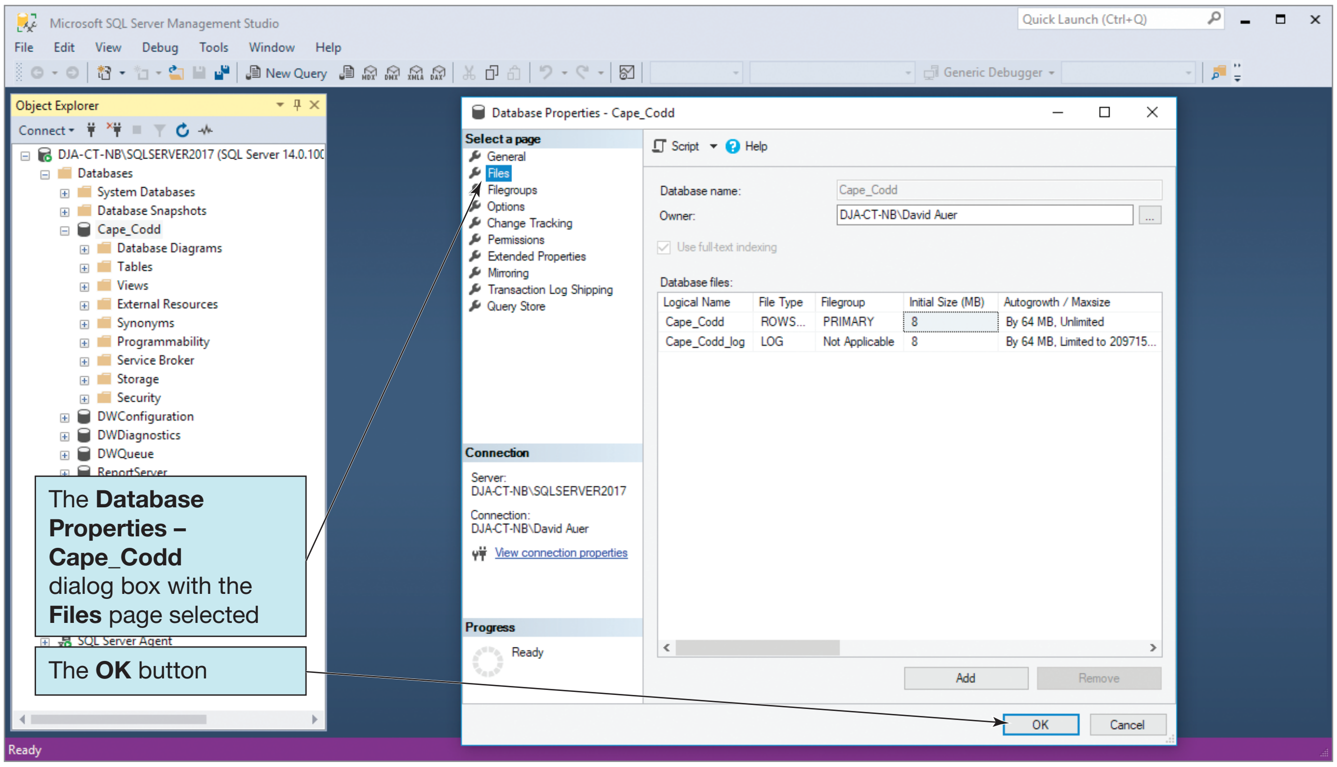


The **Refresh** button

The **Cape_Codd** database object in the Object Explorer, expanded to show the folder structure that will be used to contain the database objects

FIGURE 10A-10

The Cape_Codd Database in the Object Explorer



Database name: Cape_Codd

Owner: DJA-CT-NB\David Auer

Use full-text indexing

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize
Cape_Codd	ROWS...	PRIMARY	8	By 64 MB, Unlimited
Cape_Codd_log	LOG	Not Applicable	8	By 64 MB, Limited to 209715...

Server: DJA-CT-NB\SQLSERVER2017

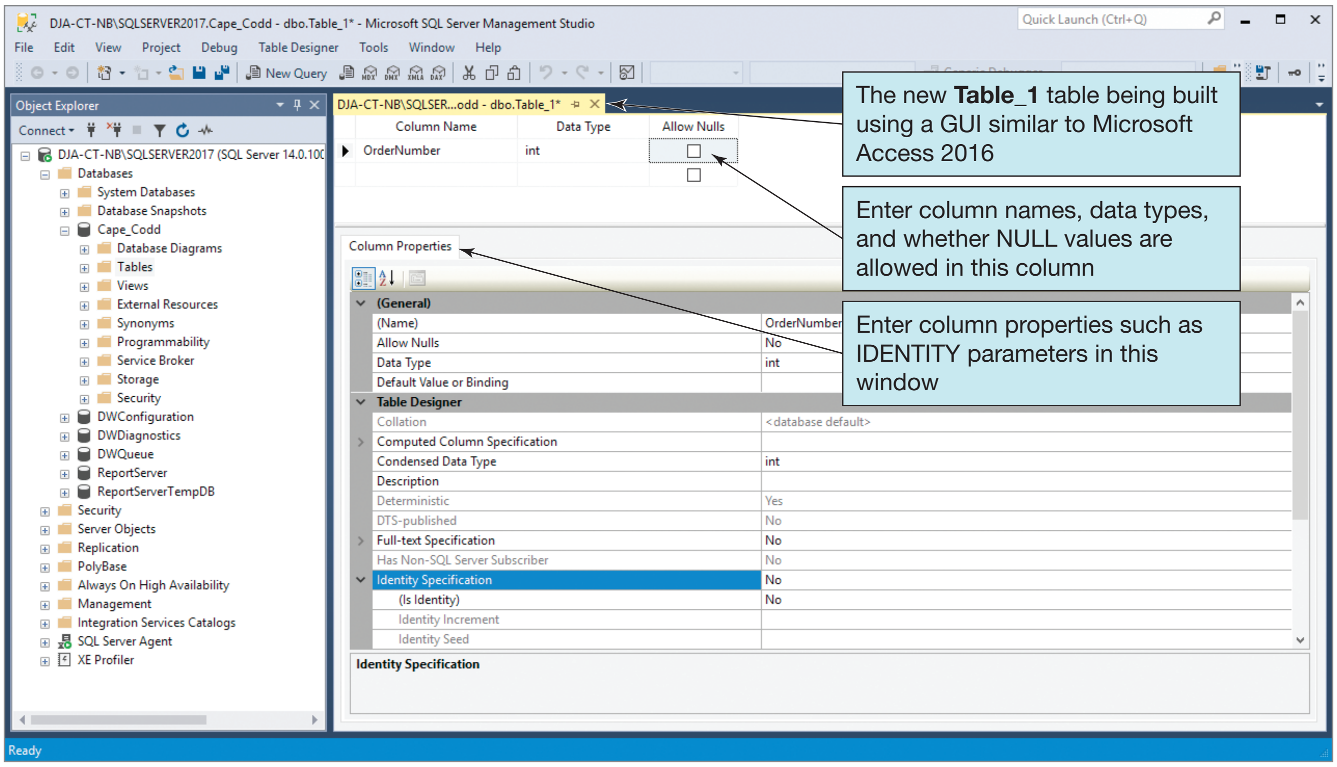
Connection: DJA-CT-NB\David Auer

The **Database Properties – Cape_Codd** dialog box with the **Files** page selected

The **OK** button

FIGURE 10A-11

The Cape_Codd Database Properties Dialog Box



The new **Table_1** table being built using a GUI similar to Microsoft Access 2016

Enter column names, data types, and whether NULL values are allowed in this column

Enter column properties such as IDENTITY parameters in this window

Column Name	Data Type	Allow Nulls
OrderNumber	int	☐
		☐

(General)	
(Name)	OrderNumber
Allow Nulls	No
Data Type	int
Default Value or Binding	
Table Designer	
Collation	<database default>
Computed Column Specification	
Condensed Data Type	int
Description	
Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	No
(Is Identity)	No
Identity Increment	
Identity Seed	

Identity Specification

FIGURE 10A-12

Building a New Table in SQL Server Management Studio

If you look at the database files listed for the Cape_Codd database in the Files page of the Database Properties–Cape_Codd dialog box, as shown in Figure 10A-11, you will see that by default SQL Server creates one data file (logical name Cape_Codd) and one log file (logical name Cape_Codd_log) for each database. You can create multiple files for both data and logs and assign particular tables and logs to particular files and file groups. However, all of this is beyond the scope of our current discussion. To learn more about it on your own, use the SQL Server 2017 help system.

Can we use the Microsoft SQL Server Management Studio the same way we use Microsoft Access to build tables using the GUI? Again, the answer is "yes." By right-clicking the Tables object to display a shortcut menu and then clicking New | Table, a tabbed window is displayed that starts the creation of a new table, currently named Table_1, as shown in Figure 10A-12. The column data shown are for the first column (OrderNumber) in the Cape_Codd database RETAIL_ORDER table described in Chapter 2. The design interface shown is similar to Microsoft Access–basic column specifications are entered in a row in the top pane, and specific column properties are set in the bottom pane. And, like Microsoft Access, the GUI will actually create an SQL statement and run that statement in the DBMS. However, we do not want to use this tool to create tables in the Cape_Codd database, so we will close the window without saving the table.

BY THE WAY

Books on systems analysis and design often identify three design stages:

- Conceptual design (conceptual schema)
- Logical design (logical schema)
- Physical design (physical schema)

The creation and use of file structure and file organization (including physical storage placement and file characteristics) to store database components and physical records of data are a part of the *physical design*, which is defined in these books as the aspects of the database that are actually implemented in the DBMS. Besides physical record and file structure and organization, this includes indexes and query optimization.

Microsoft SQL Server 2017 SQL Statements and SQL Scripts

Because we have already argued that you need to know how to write and use SQL statements instead of relying on GUI tools, we come back to using SQL as the basis of our work. But we do not want to use a command-line utility, and we are not going to use the GUI tool in GUI mode, so what's left?

The answer is that the Microsoft SQL Server Management Studio provides us with an excellent SQL editing environment. This lets us take advantage of GUI capabilities while still working with text-based SQL statements. We do this by opening an SQL Query window but using it more generally as an "SQL statement editor" window.

Opening an SQL Server 2017 SQL Statement Editor Window

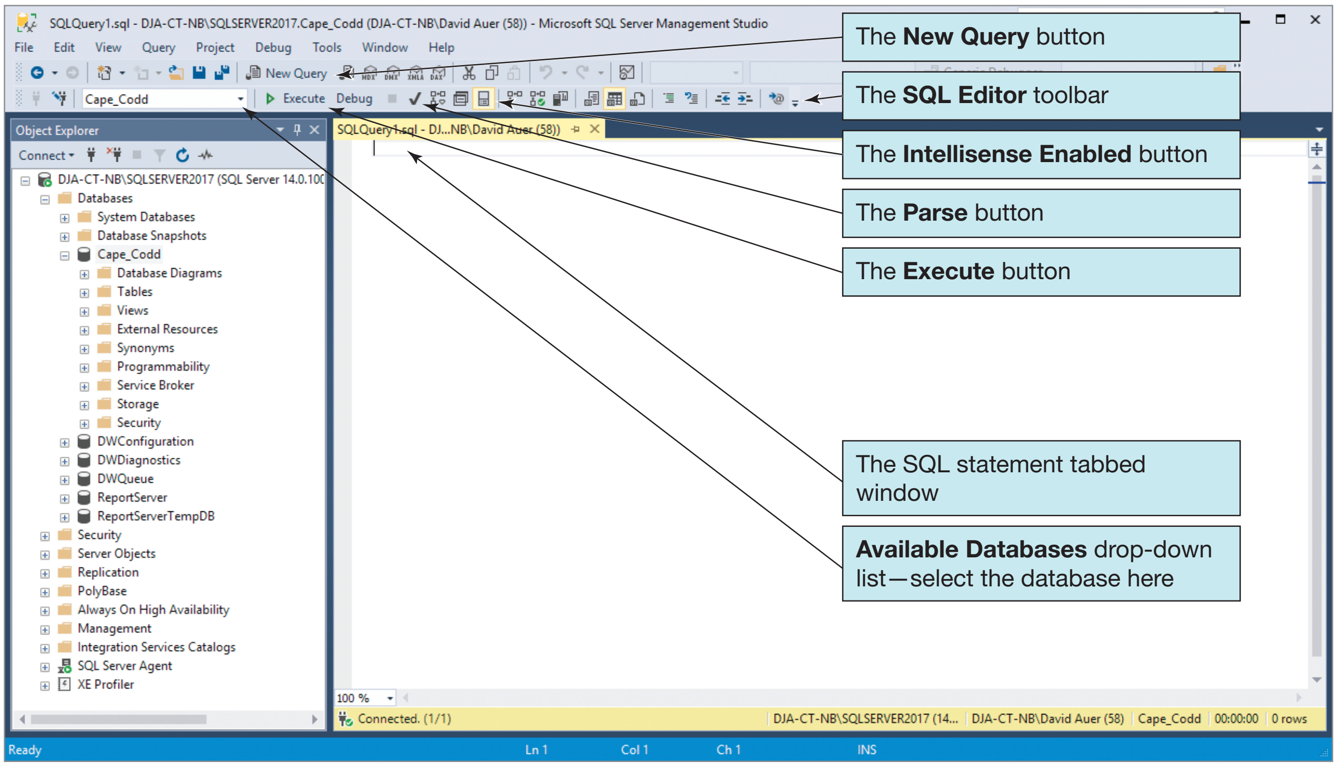
1. Click the **Cape_Codd** database object in the Object Browser to select it.
2. Click the **New Query** button. A new tabbed SQL Query window is displayed, and the SQL Editor toolbar appears, as shown in Figure 10A-13.
3. Click the **IntelliSense Enabled** button to *disable* IntelliSense. **IntelliSense** is an object-search feature that, though useful for experienced developers, can be more confusing than helpful if you do not know how to use it. In this book, we will not use IntelliSense, but if you want to learn more about it, you can read about it in the SQL Server 2017 Books Online.

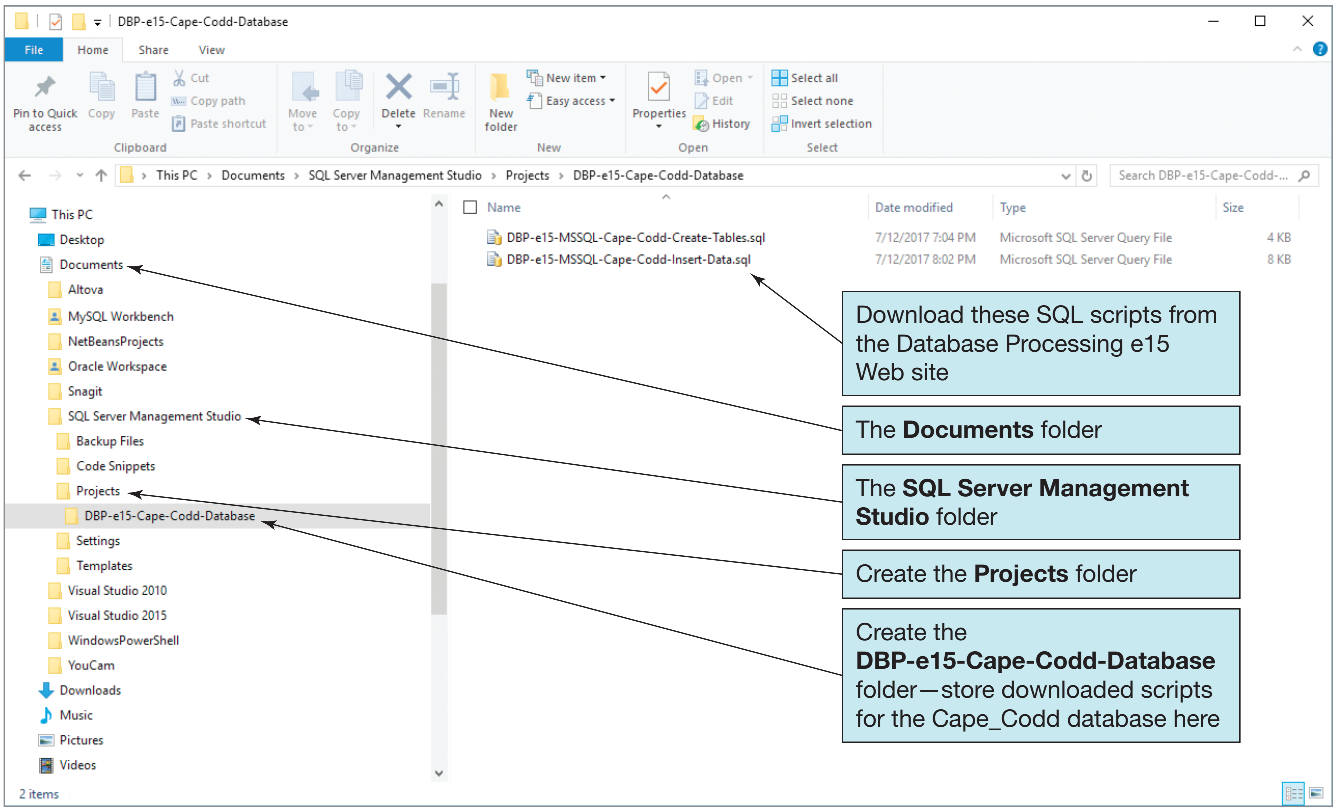
One advantage of using this SQL editor is the ability to save and reuse SQL scripts. For SQL Server, an **SQL script** is a plaintext file labeled with the *.sql* file extension. An SQL script is composed of one or more SQL statements, which can include SQL script comments. **SQL script comments** are lines of text that do not run when the script is executed but are used to document the purpose and contents of the script. Each comment begins with the characters /* **(slash asterisk)** and ends with the characters */ **(asterisk slash)**. Alternatively, for a single-line comment (or a comment at the end of a line), we use -- **(two dashes)** at the start of the comment.

We can save, open, and run (and rerun) SQL scripts. As shown in Figure 10A-14, when the SQL Server Management Studio is installed, the installation process creates a set of folders in the user's Documents folder. In this structure, we need to add a Projects folder to

FIGURE 10A-13

The Microsoft SQL Server Management Studio SQL Editor


SQLQuery1.sql - DJA-CT-NB\SQLSERVER2017.Cape_Codd (DJA-CT-NB\David Auer (58)) - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
Cape_Codd
Execute Debug
Object Explorer
Connect
DJA-CT-NB\SQLSERVER2017 (SQL Server 14.0.100
Databases
System Databases
Database Snapshots
Cape_Codd
Database Diagrams
Tables
Views
External Resources
Synonyms
Programmability
Service Broker
Storage
Security
DWConfiguration
DWDiagnostics
DWQueue
ReportServer
ReportServerTempDB
Security
Server Objects
Replication
PolyBase
Always On High Availability
Management
Integration Services Catalogs
SQL Server Agent
XE Profiler
SQLQuery1.sql - DJ...NB\David Auer (58))
100 %
Connected. (1/1)
DJA-CT-NB\SQLSERVER2017 (14... DJA-CT-NB\David Auer (58) Cape_Codd 00:00:00 0 rows
Ready Ln 1 Col 1 Ch 1 INS
The **New Query** button
The **SQL Editor** toolbar
The **Intellisense Enabled** button
The **Parse** button
The **Execute** button
The SQL statement tabbed window
Available Databases drop-down list—select the database here



Name	Date modified	Type	Size
DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql	7/12/2017 7:04 PM	Microsoft SQL Server Query File	4 KB
DBP-e15-MSSQL-Cape-Codd-Insert-Data.sql	7/12/2017 8:02 PM	Microsoft SQL Server Query File	8 KB

Download these SQL scripts from the Database Processing e15 Web site

The **Documents** folder

The **SQL Server Management Studio** folder

Create the **Projects** folder

Create the **DBP-e15-Cape-Codd-Database** folder—store downloaded scripts for the Cape_Codd database here

FIGURE 10A-14

The SQL Server Management Studio Folder Structure

store our SQL scripts. To further organize our work, we will create subfolders for each of our database projects. This is illustrated by the *DBP-e15-Cape-Codd-Database* folder that was created to store the scripts (including scripts containing SQL queries) that we use with the Cape Codd Outdoor Sports database in Chapter 2.

Using Existing SQL Scripts

Another advantage of SQL scripts is that we can use scripts written by others. To do this we simply store the script in an appropriate location on our computer, open it in an SQL script tabbed window, and execute it.

To illustrate this, we will build the Cape_Codd database we use in Chapter 2 to run our example SQL query statements. This will then allow you to run the Chapter 2 SQL statements in an actual database and to work the Exercises at the end of the chapter.

Opening and Running an Existing SQL Script

1. The SQL scripts needed to build the Cape Codd database are available at *www.pearsonhighered.com/kroenke*. Go to the *Database Processing* 15/e Companion Web site, and download the Student Data Files to your *Downloads* folder. There is a ZIP archive file named **DBP-e15-IM-SRC.zip**, so you will need to extract the files. After you have done this, copy the two files in the *Downloads/SQL Server Management Studio/Projects/DBP-e15-Cape-Codd-Database* folder to your *Documents/SQL Server Management Studio/Projects/DBP-e15-Cape-Codd-Database* folder (if you have not already created the *DBP-e15-Cape-Codd-Database* folder in *Projects*, then copy the entire *Downloads/SQL Server Management Studio/Projects/DBP-e15-Cape-Codd-Database* folder to your *Documents/SQL Server Management Studio/Projects* folder.)
2. Click the **Open File** button shown in Figure 10A-15 to display the Open File dialog box (alternatively, we can use the *File* | *Open* | *File* menu command to open the dialog box). The advantage of this button is that the script will be opened in a new query tabbed window, not the *SQLQuery1* tabbed window that we currently have open.

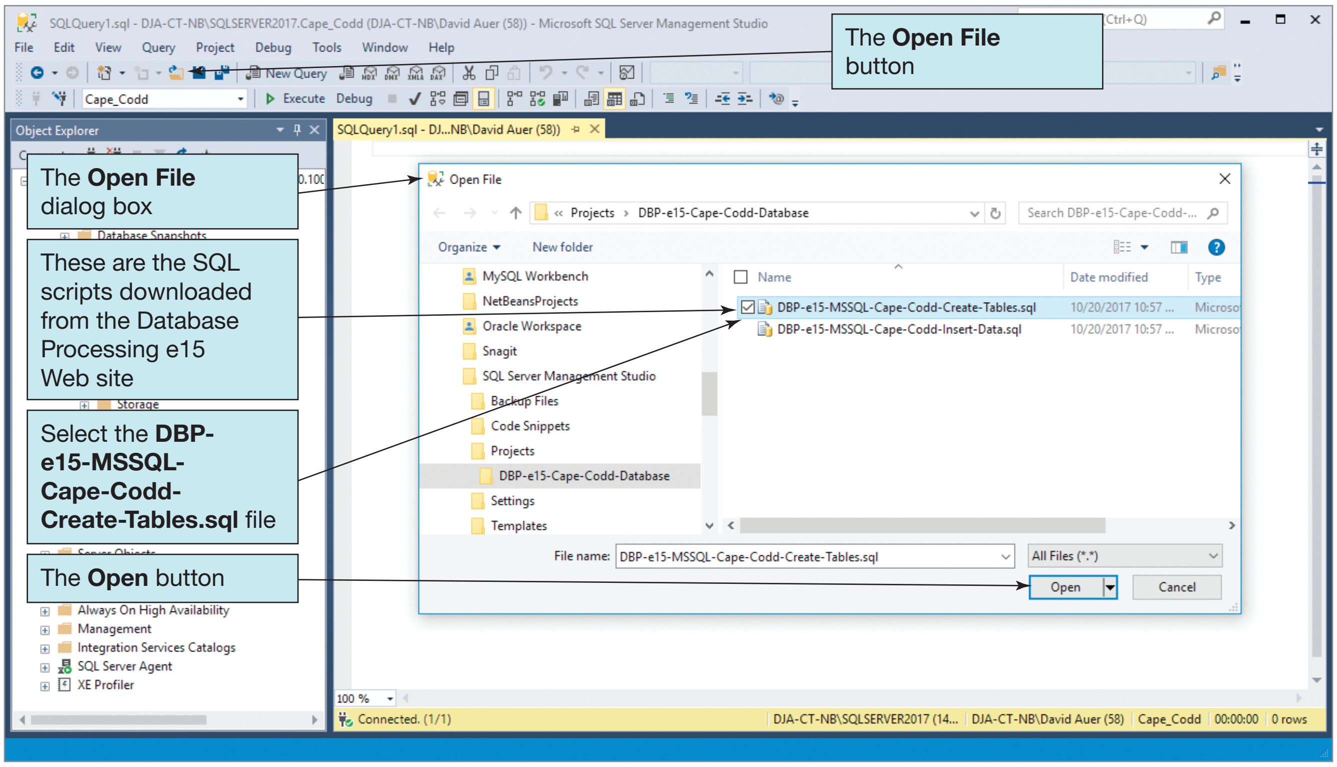

The **Open File** button
The **Open File** dialog box
These are the SQL scripts downloaded from the Database Processing e15 Web site
Select the **DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql** file
The **Open** button
Open File
Projects › DBP-e15-Cape-Codd-Database
DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql
DBP-e15-MSSQL-Cape-Codd-Insert-Data.sql
File name: DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql
All Files (*.*)
Open
Cancel

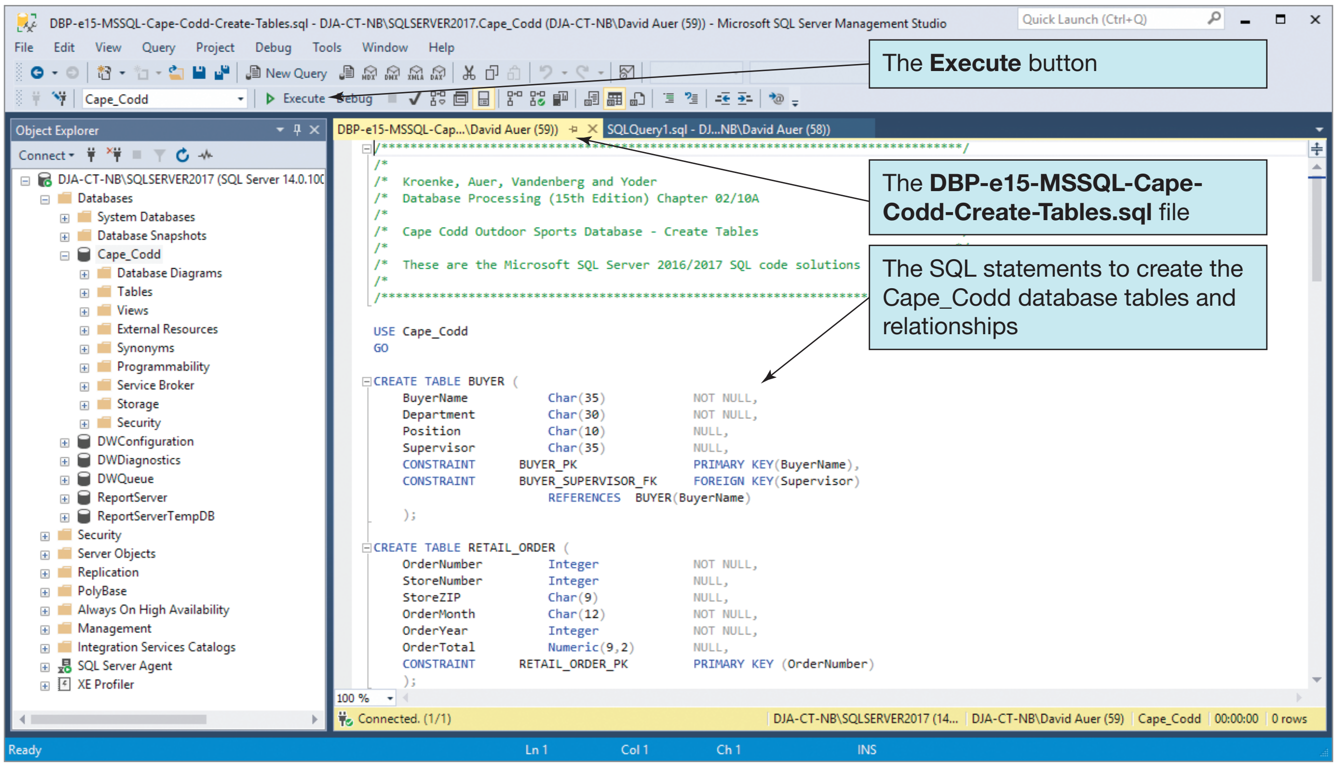
FIGURE 10A-15

The Open File Dialog Box

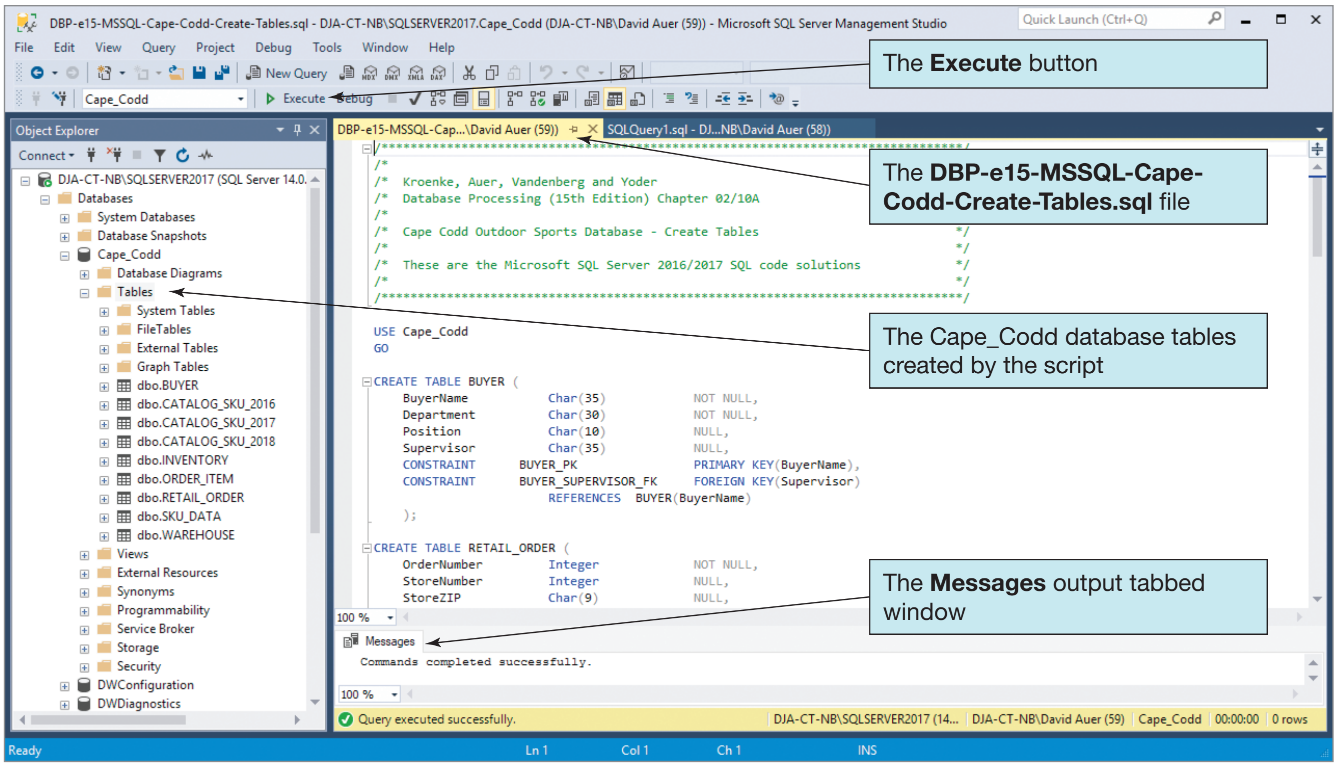
3. Browse to the **DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql** SQL script as shown in Figure 10A-15.
4. Click the **Open** button. The DBP-e15-MySQL-Cape-Codd-Create-Tables SQL script is displayed in a new SQL query tabbed window, as shown in Figure 10A-16.
5. Click the **Execute** button. The SQL script is run, and the Cape Codd database tables are created, as shown in Figure 10A-17.

FIGURE 10A-16

The Cape Codd Database Create Tables Script


The **Execute** button
The **DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql** file
The SQL statements to create the Cape_Codd database tables and relationships
DBP-e15-MSSQL-Cap...\David Auer (59))
SQLQuery1.sql - DJ...NB\David Auer (58))

```
/*****************************************************************************
/*
/*  Kroenke, Auer, Vandenberg and Yoder
/*  Database Processing (15th Edition) Chapter 02/10A
/*
/*  Cape Codd Outdoor Sports Database - Create Tables
/*
/*  These are the Microsoft SQL Server 2016/2017 SQL code solutions
/*
/*****************************************************************************

USE Cape_Codd
GO

CREATE TABLE BUYER (
    BuyerName          Char(35)          NOT NULL,
    Department         Char(30)          NOT NULL,
    Position           Char(10)          NULL,
    Supervisor         Char(35)          NULL,
    CONSTRAINT      BUYER_PK             PRIMARY KEY(BuyerName),
    CONSTRAINT      BUYER_SUPERVISOR_FK  FOREIGN KEY(Supervisor)
                        REFERENCES  BUYER(BuyerName)
    );

CREATE TABLE RETAIL_ORDER (
    OrderNumber        Integer           NOT NULL,
    StoreNumber        Integer           NULL,
    StoreZIP           Char(9)           NULL,
    OrderMonth         Char(12)          NOT NULL,
    OrderYear          Integer           NOT NULL,
    OrderTotal         Numeric(9,2)      NULL,
    CONSTRAINT      RETAIL_ORDER_PK      PRIMARY KEY (OrderNumber)
    );
```


DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql - DJA-CT-NB\SQLSERVER2017.Cape_Codd (DJA-CT-NB\David Auer (59)) - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
Cape_Codd
Execute
Debug
Object Explorer
Connect
DJA-CT-NB\SQLSERVER2017 (SQL Server 14.0.
Databases
System Databases
Database Snapshots
Cape_Codd
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.BUYER
dbo.CATALOG_SKU_2016
dbo.CATALOG_SKU_2017
dbo.CATALOG_SKU_2018
dbo.INVENTORY
dbo.ORDER_ITEM
dbo.RETAIL_ORDER
dbo.SKU_DATA
dbo.WAREHOUSE
Views
External Resources
Synonyms
Programmability
Service Broker
Storage
Security
DWConfiguration
DWDiagnostics
DBP-e15-MSSQL-Cap...\David Auer (59))
SQLQuery1.sql - DJ...NB\David Auer (58))
The **Execute** button
The **DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql** file
The Cape_Codd database tables created by the script
The **Messages** output tabbed window
100 %
Messages
Query executed successfully.
DJA-CT-NB\SQLSERVER2017 (14...
DJA-CT-NB\David Auer (59)
Cape_Codd
00:00:00
0 rows
Ready
Ln 1
Col 1
Ch 1
INS

```
/**************************************************************************/
/*
/*  Kroenke, Auer, Vandenberg and Yoder
/*  Database Processing (15th Edition) Chapter 02/10A
/*
/*  Cape Codd Outdoor Sports Database - Create Tables                    */
/*                                                                       */
/*  These are the Microsoft SQL Server 2016/2017 SQL code solutions      */
/*                                                                       */
/**************************************************************************/

USE Cape_Codd
GO

CREATE TABLE BUYER (
    BuyerName           Char(35)            NOT NULL,
    Department          Char(30)            NOT NULL,
    Position            Char(10)            NULL,
    Supervisor          Char(35)            NULL,
    CONSTRAINT      BUYER_PK                PRIMARY KEY(BuyerName),
    CONSTRAINT      BUYER_SUPERVISOR_FK     FOREIGN KEY(Supervisor)
                        REFERENCES  BUYER(BuyerName)
    );

CREATE TABLE RETAIL_ORDER (
    OrderNumber         Integer             NOT NULL,
    StoreNumber         Integer             NULL,
    StoreZIP            Char(9)             NULL,
```

```
Commands completed successfully.
```

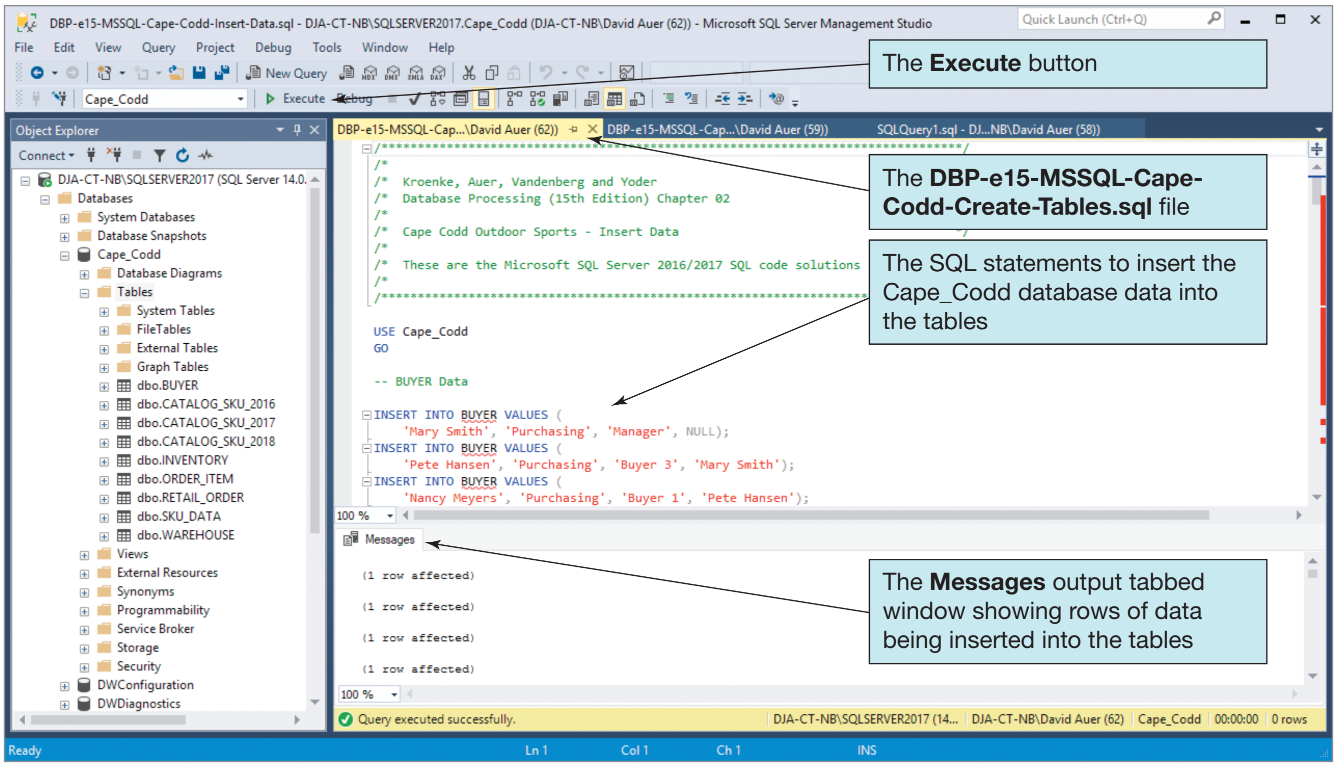
FIGURE 10A-17

The Cape Codd Database Table Objects in Object Explorer

6. Click the **Open File** button to display the Open SQL Script dialog box.
7. Browse to the **DBP-e15-MSSQL-Cape-Codd-Insert-Data.sql** SQL script.
8. Click the **Open** button. The DBP-e15-MSSQL-Cape-Codd-Insert-Data.sql SQL script is displayed in a new SQL query tabbed window.
9. Click the **Execute** button. The SQL script is run, and the Cape Codd database tables are populated with data created, as shown in Figure 10A-18.
10. In the DBP-e15-MSSQL-Cape-Codd-Insert-Data.sql SQL script tabbed window, click the **X [Close]** button on the tab to close the window.
11. In the DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql SQL script tabbed window, click the **X [Close]** button on the tab to close the window.
12. We will now test the Cape Codd database by running a query against the database. As discussed in Chapter 2, we do this by using an SQL SELECT statement. We will run the first SQL query demonstrated in Chapter 2, which is:

```
SELECT    SKU, SKU_Description, Department, Buyer
FROM      SKU_DATA;
```

13. Click the **Intellisense Enabled** button to *disable* Intellisense. Note that by default, Intellisense is enabled whenever you open a new Query or SQL script window, and it must be disabled in each window individually.
14. As shown in Figure 10A-19, enter the SQL statement for the SQL SELECT statement into the existing SQLQuery1 tabbed window, and then click the **Execute** button. The SQL SELECT statement is run, and the query results are displayed in the Result grid window. These results confirm that the Cape Codd data was successfully entered into the Cape Codd database.
15. As we also discuss in Chapter 2, we can save this query as an SQL script for later use. Click the **Save** button to open the Save SQL Script dialog box as shown in Figure 10A-20. Browse to the *Documents/SQL Server Management Studio/Projects/DBP-e15-Cape-Codd-Database* folder, and save this SQL query as SQL-Query-CH02-01.sql.


The **Execute** button
The **DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql** file
The SQL statements to insert the Cape_Codd database data into the tables
The **Messages** output tabbed window showing rows of data being inserted into the tables

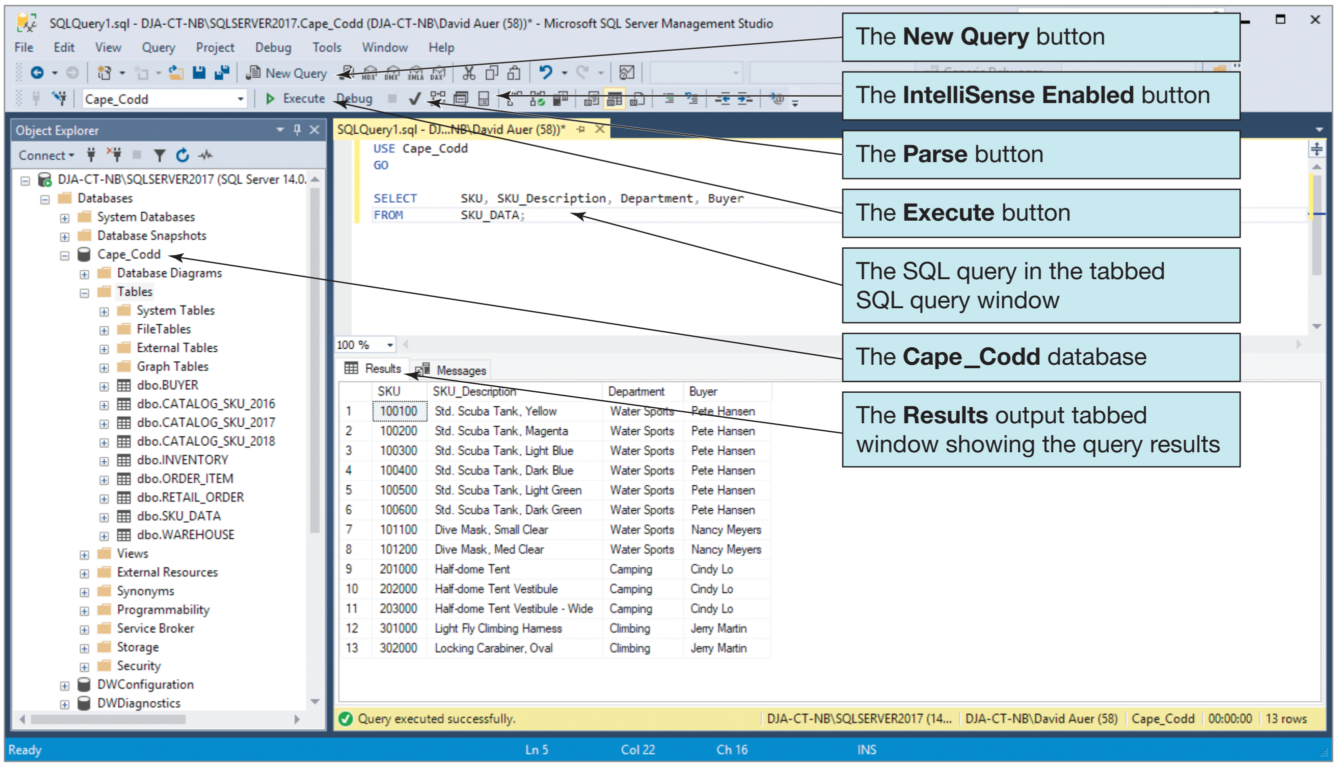
```
/*****************************************************************************
/*
/*  Kroenke, Auer, Vandenberg and Yoder
/*  Database Processing (15th Edition) Chapter 02
/*
/*  Cape Codd Outdoor Sports - Insert Data
/*
/*  These are the Microsoft SQL Server 2016/2017 SQL code solutions
/*
/*****************************************************************************

USE Cape_Codd
GO

-- BUYER Data

INSERT INTO BUYER VALUES (
    'Mary Smith', 'Purchasing', 'Manager', NULL);
INSERT INTO BUYER VALUES (
    'Pete Hansen', 'Purchasing', 'Buyer 3', 'Mary Smith');
INSERT INTO BUYER VALUES (
    'Nancy Meyers', 'Purchasing', 'Buyer 1', 'Pete Hansen');
```

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

Query executed successfully.

FIGURE 10A-18

Populating the Cape Codd Database Tables


The **New Query** button
The **IntelliSense Enabled** button
The **Parse** button
The **Execute** button
The SQL query in the tabbed SQL query window
The **Cape_Codd** database
The **Results** output tabbed window showing the query results

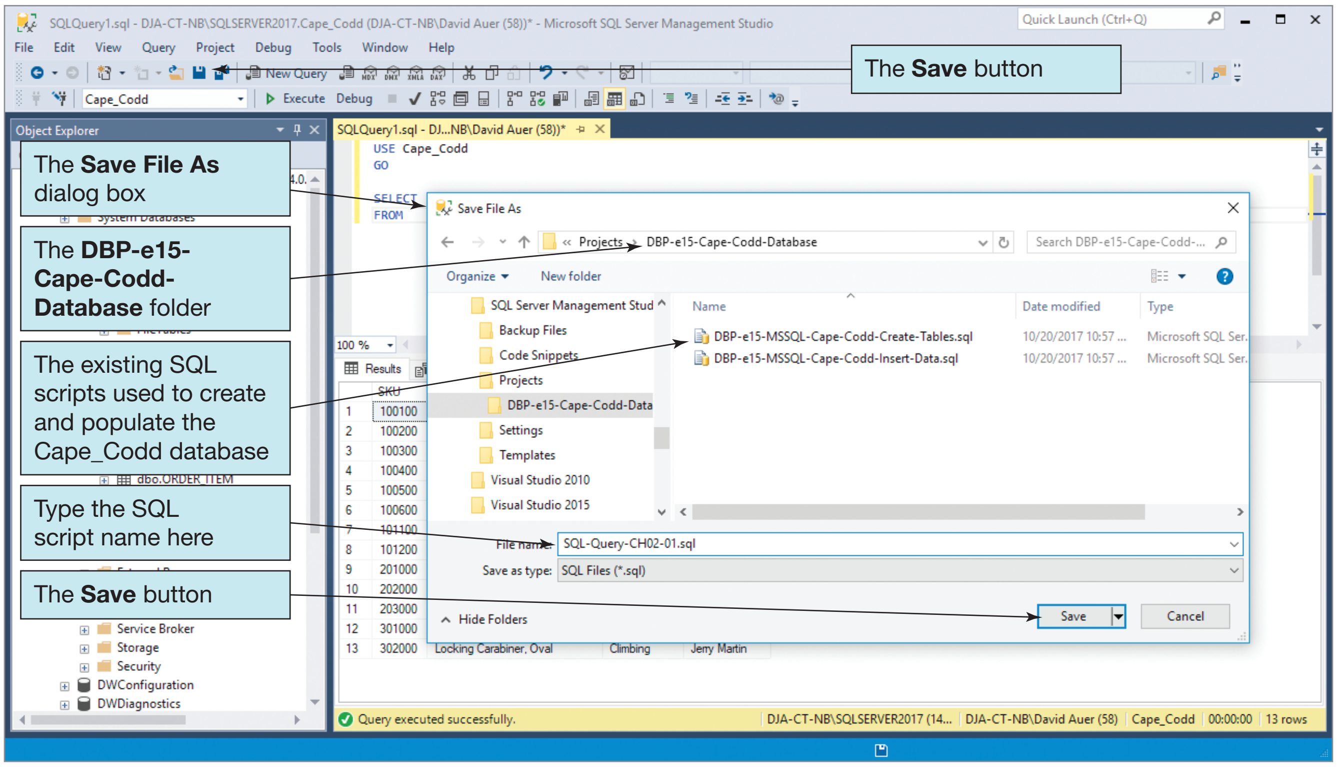
```
USE Cape_Codd
GO

SELECT     SKU, SKU_Description, Department, Buyer
FROM       SKU_DATA;
```

	SKU	SKU_Description	Department	Buyer
1	100100	Std. Scuba Tank, Yellow	Water Sports	Pete Hansen
2	100200	Std. Scuba Tank, Magenta	Water Sports	Pete Hansen
3	100300	Std. Scuba Tank, Light Blue	Water Sports	Pete Hansen
4	100400	Std. Scuba Tank, Dark Blue	Water Sports	Pete Hansen
5	100500	Std. Scuba Tank, Light Green	Water Sports	Pete Hansen
6	100600	Std. Scuba Tank, Dark Green	Water Sports	Pete Hansen
7	101100	Dive Mask, Small Clear	Water Sports	Nancy Meyers
8	101200	Dive Mask, Med Clear	Water Sports	Nancy Meyers
9	201000	Half-dome Tent	Camping	Cindy Lo
10	202000	Half-dome Tent Vestibule	Camping	Cindy Lo
11	203000	Half-dome Tent Vestibule - Wide	Camping	Cindy Lo
12	301000	Light Fly Climbing Harness	Climbing	Jerry Martin
13	302000	Locking Carabiner, Oval	Climbing	Jerry Martin

Query executed successfully.

FIGURE 10A-19

Querying the Cape Codd Database


The Save button
The Save File As dialog box
The DBP-e15-Cape-Codd-Database folder
The existing SQL scripts used to create and populate the Cape_Codd database
Type the SQL script name here
The Save button
Save File As
Projects › DBP-e15-Cape-Codd-Database
Organize
New folder
Name
Date modified
Type
SQL Server Management Stud
Backup Files
Code Snippets
Projects
DBP-e15-Cape-Codd-Data
Settings
Templates
Visual Studio 2010
Visual Studio 2015
File name:
Save as type:
Hide Folders
Save
Cancel
Query executed successfully.

```
USE Cape_Codd
GO

SELECT
FROM
```

Name	Date modified	Type
DBP-e15-MSSQL-Cape-Codd-Create-Tables.sql	10/20/2017 10:57 ...	Microsoft SQL Ser...
DBP-e15-MSSQL-Cape-Codd-Insert-Data.sql	10/20/2017 10:57 ...	Microsoft SQL Ser...

File name: SQL-Query-CH02-01.sql
Save as type: SQL Files (*.sql)

FIGURE 10A-20

Saving the SQL Query as an SQL Script

16. Note that the SQL Query tabbed window is now labeled *SQL-Query-CH02-01* (plus some other data), which is the name of the saved SQL script containing the query.
17. Click the **X [Close]** button to close this tabbed window.

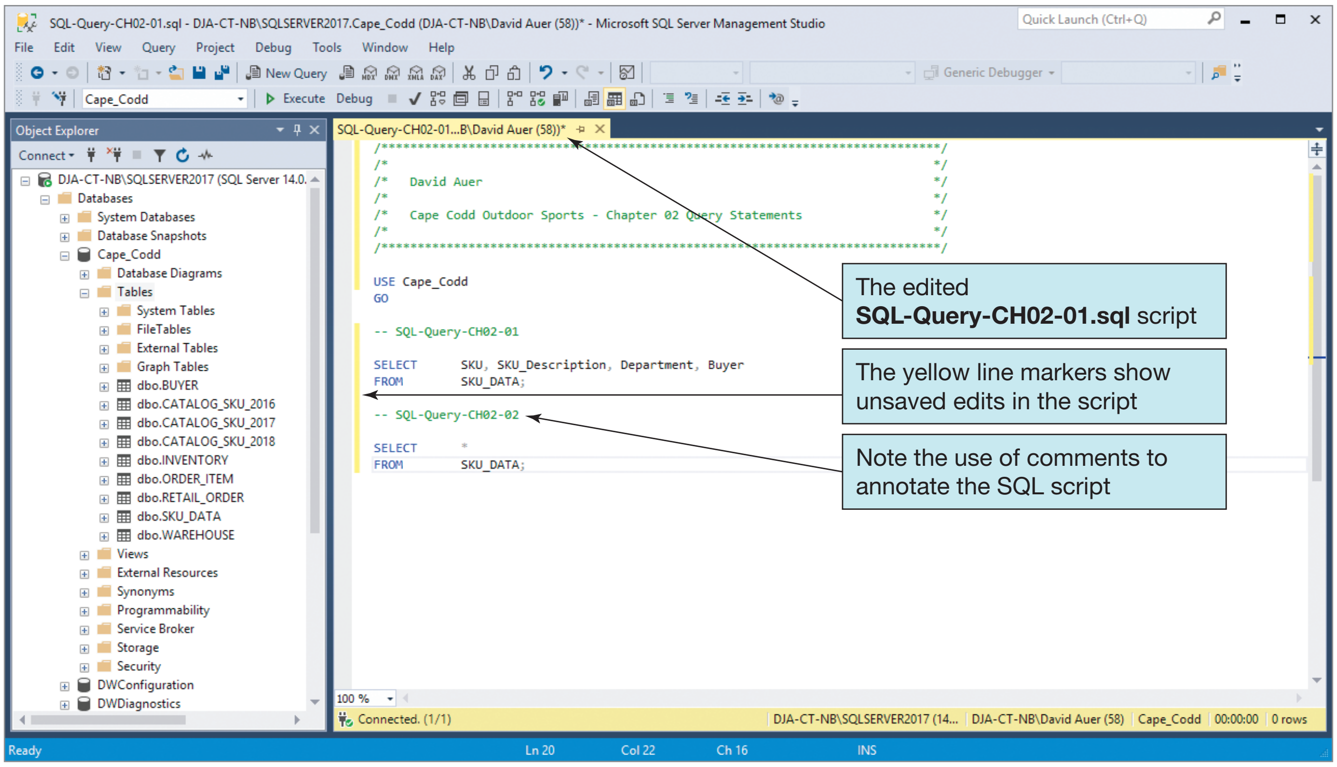
Using a Single SQL Script to Store Multiple SQL Commands

We have now created and queried the Cape Codd Outdoor Sports database used in Chapter 2 for our discussion of SQL query statements. We could save each of the Chapter 2 SQL queries as a separate SQL script, but a more efficient way to store these SQL statements is to combine them in a single SQL script.

We can annotate the SQL script with comments, and, as we will demonstrate, we can run a single SQL statement in the script by selecting that statement and then executing it.

Creating and Using an SQL Script to Store SQL Queries

1. Click the **Open File** button to display the Open SQL Script dialog box.
2. Browse to the SQL-Query-CH02-01.sql SQL script in the *Documents/ SQL Server Management Studio/Projects/ DBP-e15-Cape-Codd-Database* folder.
3. Click the **Open** button. The **SQL-Query-CH02-01.sql** SQL script is displayed in a new SQL query tabbed window (although this is a new tabbed window, it will appear nearly identical to the tabbed window shown in Figure 10A-19—the only difference will be that the tab is labeled *SQL-Query-CH02-01*.
4. Click the **Intellisense Enabled** button to *disable* Intellisense.
5. Edit the SQL script as shown in Figure 10A-21. Note that we are adding an SQL comment header to identify the script, one new query (SQL-Query-CH02-02 from Chapter 2), and individual comments to identify each query.
6. Note that there is a vertical yellow band on the left-hand side of the tabbed window. Lines marked in yellow in this manner are *unsaved* work. Although we


SQL-Query-CH02-01.sql - DJA-CT-NB\SQLSERVER2017.Cape_Codd (DJA-CT-NB\David Auer (58))* - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
Execute Debug
Cape_Codd
Object Explorer
SQL-Query-CH02-01...B\David Auer (58))*
The edited **SQL-Query-CH02-01.sql** script
The yellow line markers show unsaved edits in the script
Note the use of comments to annotate the SQL script
Connected. (1/1)
DJA-CT-NB\SQLSERVER2017 (14... DJA-CT-NB\David Auer (58) Cape_Codd 00:00:00 0 rows
Ready Ln 20 Col 22 Ch 16 INS

```
/*****************************************************************************/
/*                                                                           */
/*   David Auer                                                              */
/*                                                                           */
/*   Cape Codd Outdoor Sports - Chapter 02 Query Statements                  */
/*                                                                           */
/*****************************************************************************/

USE Cape_Codd
GO

-- SQL-Query-CH02-01

SELECT      SKU, SKU_Description, Department, Buyer
FROM        SKU_DATA;

-- SQL-Query-CH02-02

SELECT      *
FROM        SKU_DATA;
```

FIGURE 10A-21

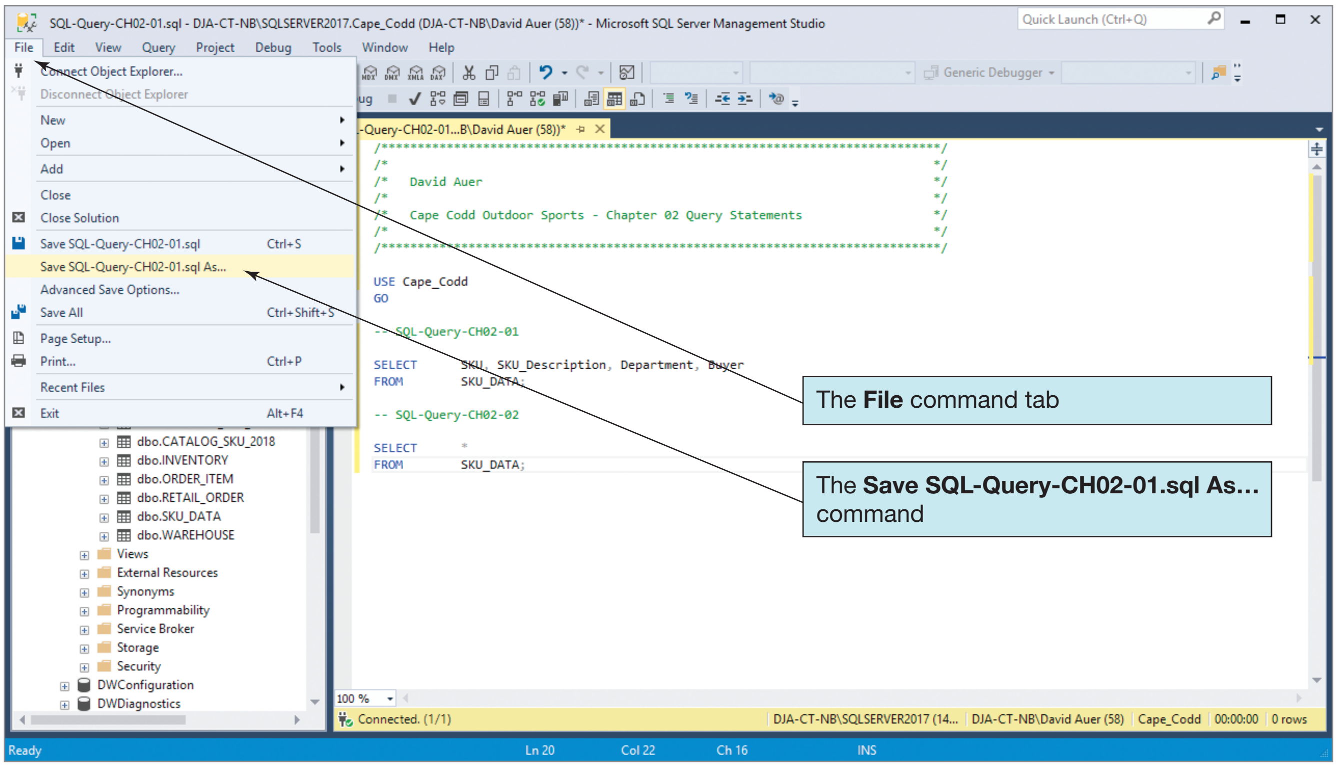
The Edited SQL Script

could save our work under the same file name, the current name really doesn't describe the SQL script with the changes we have made. We will save it under a new, more descriptive name: *Cape-Codd-Chapter-02-SQL-Queries.sql*.

7. Use the File | Save SQL-Query-CH02-01.sql As command in the File menu as shown in Figure 10A-22 to display the **Save File As** dialog box.
8. Type in the new SQL script name **Cape-Codd-Chapter-02-SQL-Queries.sql**.
9. Click the **Save** button to save the SQL script under the new file name.
10. As shown in Figure 10A-23, the tabbed window now displays the new file name and the vertical yellow line is now green. Lines marked in green in this manner are *saved* work.
11. As shown in Figure 10A-23, use the mouse cursor to select (highlight) **SQL-Query-CH02-02** (see how convenient comment labels are?).
12. Click the **Execute** button. Only the selected SQL command is executed, and the results are displayed as shown in the tabbed Results grid in Figure 10A-23. This illustrates how to select and run an individual SQL statement in an SQL script that contains many SQL statements.
13. Click the **X [Close]** button to close this tabbed window.

BY THE WAY At this point you have covered all the material about Microsoft SQL Server 2017 that you need to work with the SQL query statements in Chapter 2. If you worked through this material because of the directions in the "Using SQL in Microsoft SQL Server 2017" section on pages 58–60, you should return to that section at this time and continue your work on SQL query statements.

Use the new **Cape-Codd-Chapter-02-SQL-Queries.sql** script to store all your work in Chapter 02—that will be much easier and more efficient than storing a separate SQL script for each query!

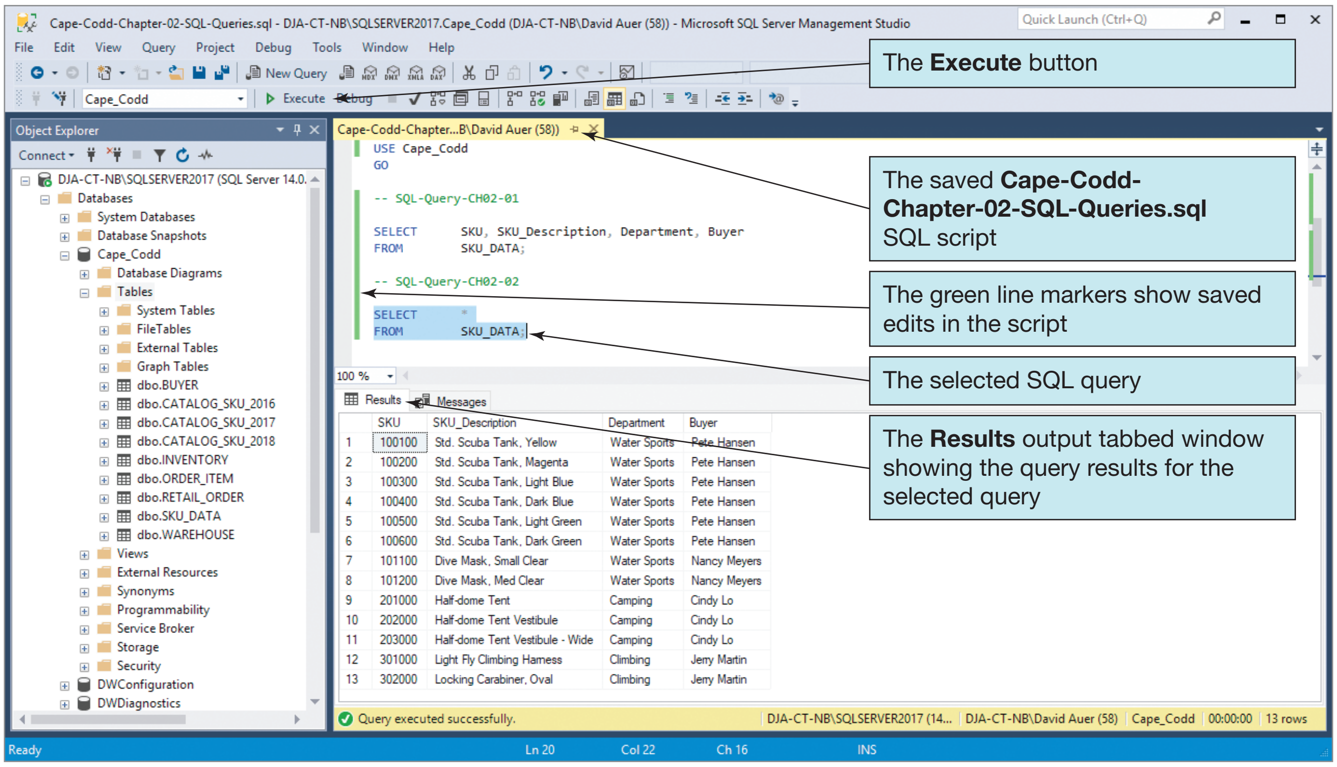

SQL-Query-CH02-01.sql - DJA-CT-NB\SQLSERVER2017.Cape_Codd (DJA-CT-NB\David Auer (58))* - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
Connect Object Explorer...
Disconnect Object Explorer
New
Open
Add
Close
Close Solution
Save SQL-Query-CH02-01.sql Ctrl+S
Save SQL-Query-CH02-01.sql As...
Advanced Save Options...
Save All Ctrl+Shift+S
Page Setup...
Print... Ctrl+P
Recent Files
Exit Alt+F4
The **File** command tab
The **Save SQL-Query-CH02-01.sql As...** command
Ready Ln 20 Col 22 Ch 16 INS

```
/*******************************************************************************/
/*                                                                             */
/*   David Auer                                                                */
/*                                                                             */
/*   Cape Codd Outdoor Sports - Chapter 02 Query Statements                    */
/*                                                                             */
/*******************************************************************************/

USE Cape_Codd
GO

-- SQL-Query-CH02-01

SELECT      SKU, SKU_Description, Department, Buyer
FROM        SKU_DATA;

-- SQL-Query-CH02-02

SELECT      *
FROM        SKU_DATA;
```

```
Connected. (1/1)    DJA-CT-NB\SQLSERVER2017 (14...  DJA-CT-NB\David Auer (58)  Cape_Codd  00:00:00  0 rows
```

FIGURE 10A-22

The File | Save SQL-Query-CH02-01.sql As . . . Command



```
USE Cape_Codd
GO

-- SQL-Query-CH02-01

SELECT      SKU, SKU_Description, Department, Buyer
FROM        SKU_DATA;

-- SQL-Query-CH02-02

SELECT      *
FROM        SKU_DATA;
```

```
    SKU     SKU_Description                  Department    Buyer
1   100100  Std. Scuba Tank, Yellow          Water Sports  Pete Hansen
2   100200  Std. Scuba Tank, Magenta         Water Sports  Pete Hansen
3   100300  Std. Scuba Tank, Light Blue      Water Sports  Pete Hansen
4   100400  Std. Scuba Tank, Dark Blue       Water Sports  Pete Hansen
5   100500  Std. Scuba Tank, Light Green     Water Sports  Pete Hansen
6   100600  Std. Scuba Tank, Dark Green      Water Sports  Pete Hansen
7   101100  Dive Mask, Small Clear           Water Sports  Nancy Meyers
8   101200  Dive Mask, Med Clear             Water Sports  Nancy Meyers
9   201000  Half-dome Tent                   Camping       Cindy Lo
10  202000  Half-dome Tent Vestibule         Camping       Cindy Lo
11  203000  Half-dome Tent Vestibule - Wide  Camping       Cindy Lo
12  301000  Light Fly Climbing Harness       Climbing      Jerry Martin
13  302000  Locking Carabiner, Oval          Climbing      Jerry Martin
```

```
Query executed successfully.    DJA-CT-NB\SQLSERVER2017 (14...  DJA-CT-NB\David Auer (58)  Cape_Codd  00:00:00  13 rows
```

FIGURE 10A-23

The Renamed SQL Script

Implementing the View Ridge Gallery *VRG* Database in Microsoft SQL Server 2017

Now that we know how to use existing SQL scripts and how to create and save SQL scripts for SQL query statements, we will discuss how to use SQL scripts to create and populate database tables of our own. To illustrate this, we will use the View Ridge Gallery *VRG* database introduced in Chapter 6 in our discussion of *database designs* and used as our example of database *implementation* in Chapter 7. In this chapter, we will discuss the specific implementation of the *VRG* database in Microsoft SQL Server 2017 and use that implementation to introduce some topics not covered in Chapter 7.

As we have seen, tables and other SQL Server structures can be created and modified in two ways. The first is to write SQL code using either the CREATE or ALTER SQL statements we discussed in Chapter 7. The second is to use the SQL Server 2017 GUI display tools discussed earlier in this chapter. Although either method will work, CREATE statements are preferred for the reasons described in Chapter 7. Some professionals choose to create structures via SQL but then modify them with the GUI tools.

As discussed in Chapter 7, each DBMS product has its own variant or extension of SQL, where the extensions are usually **procedural language extensions**, which are additions that allow SQL to function similarly to a procedural programming language (e.g., IF . . . THEN . . . ELSE structures). Microsoft's SQL Server version is called **Transact-SQL (T-SQL)**. We will point out specific Transact-SQL syntax as we encounter it in our discussion. For more on Transact-SQL, see the SQL Server 2017 Books Online article "Transact-SQL Reference" at *http://msdn.microsoft.com/en-us/library/bb510741.aspx*.

First, we need to create the *VRG* database itself.

Creating the VRG Database

1. Contract (minimize) the **System Databases** folder by clicking the minus (-) button.
2. Right-click the **Databases** folder in the Object Explorer to display the shortcut menu.
3. Click the **New Database** command to display the New Database dialog box.
4. Type the database name **VRG** in the Database Name text box, and then click the **OK** button. The database is created.
5. Click the **Refresh** button—the databases are sorted into alphabetical order and the *VRG* database object is displayed in the Object Explorer, as shown in Figure 10A-24. Click the plus (+) button next to the VRG database object to display the VRG database folders, also shown in Figure 10A-24.

Using SQL Scripts to Create and Populate Database Tables

Now that we have created the VRG database, we will set up a folder in the SQL Server Management Studio/Project folder to store our SQL scripts and review creating and saving an SQL script.

Creating and Saving an SQL Script to Create the VRG Tables

1. Click the **New Query** button to open a tabbed SQL Query window.
2. In the open tabbed SQL Query window, type the SQL comments shown in Figure 10A-25.
3. Click the **Save** button. The Save File As dialog box is displayed, as shown in Figure 10A-26.
4. Browse to the **Projects** folder. Click the **New Folder** button in the Save File As dialog box. A new folder is displayed in the Save File As dialog box, as shown in Figure 10A-26.
5. Type the folder name **DBP-e15-View-Ridge-Gallery-Database**, and then press the **Enter** key to create the folder.
6. Double-click the new folder name to move to the new folder.

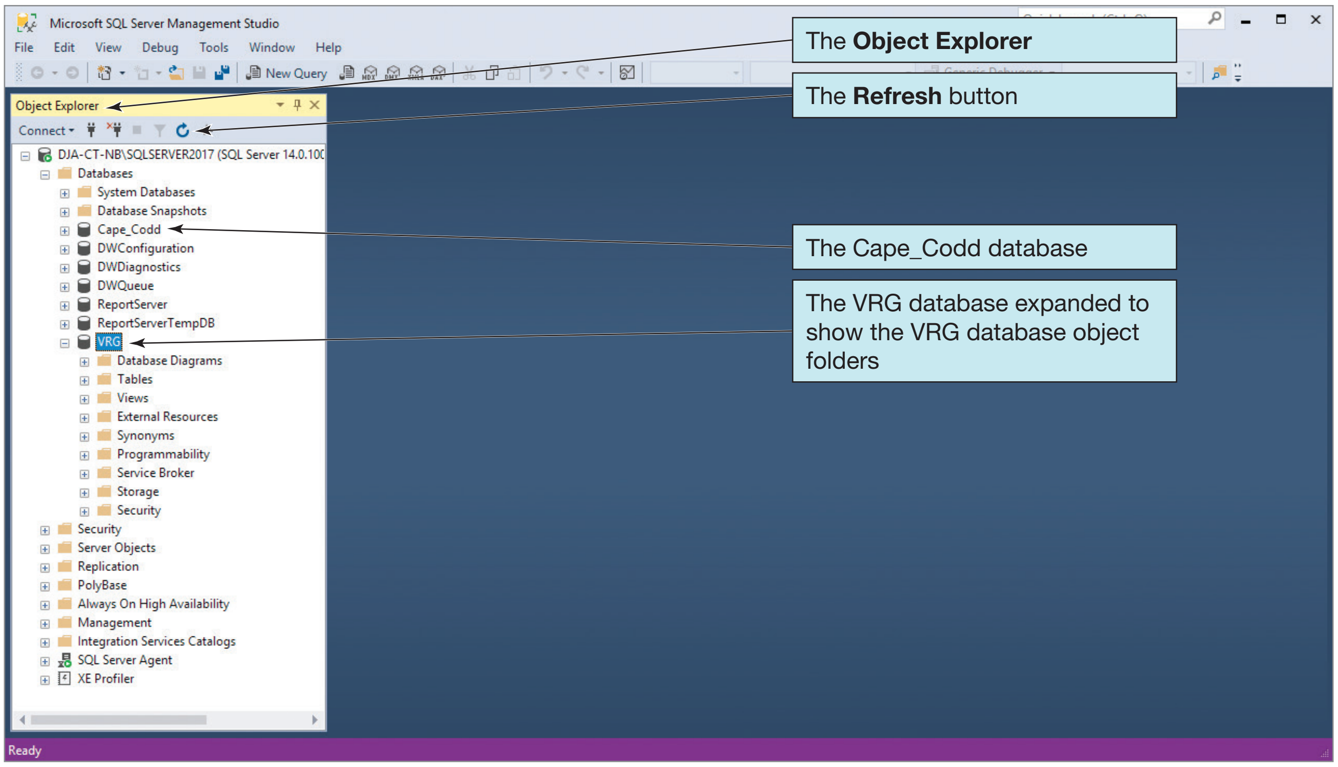
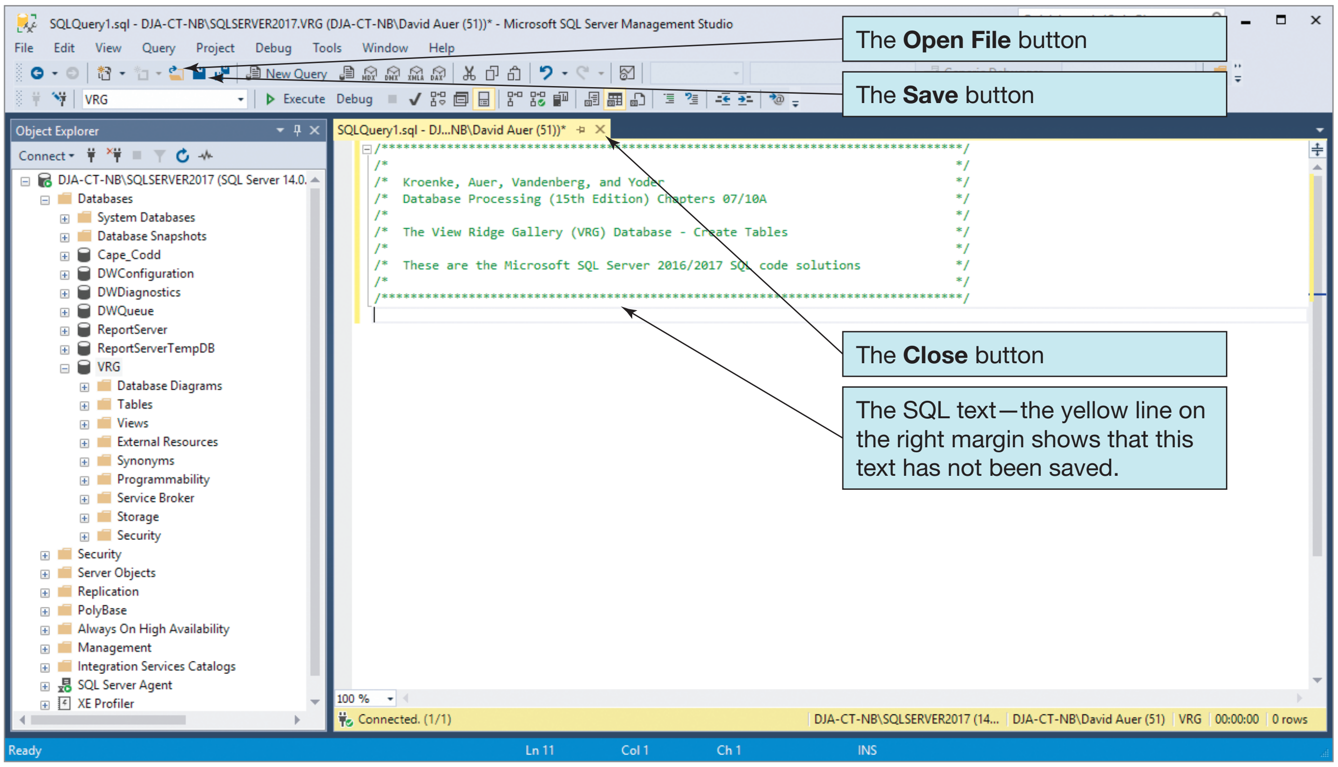


FIGURE 10A-24

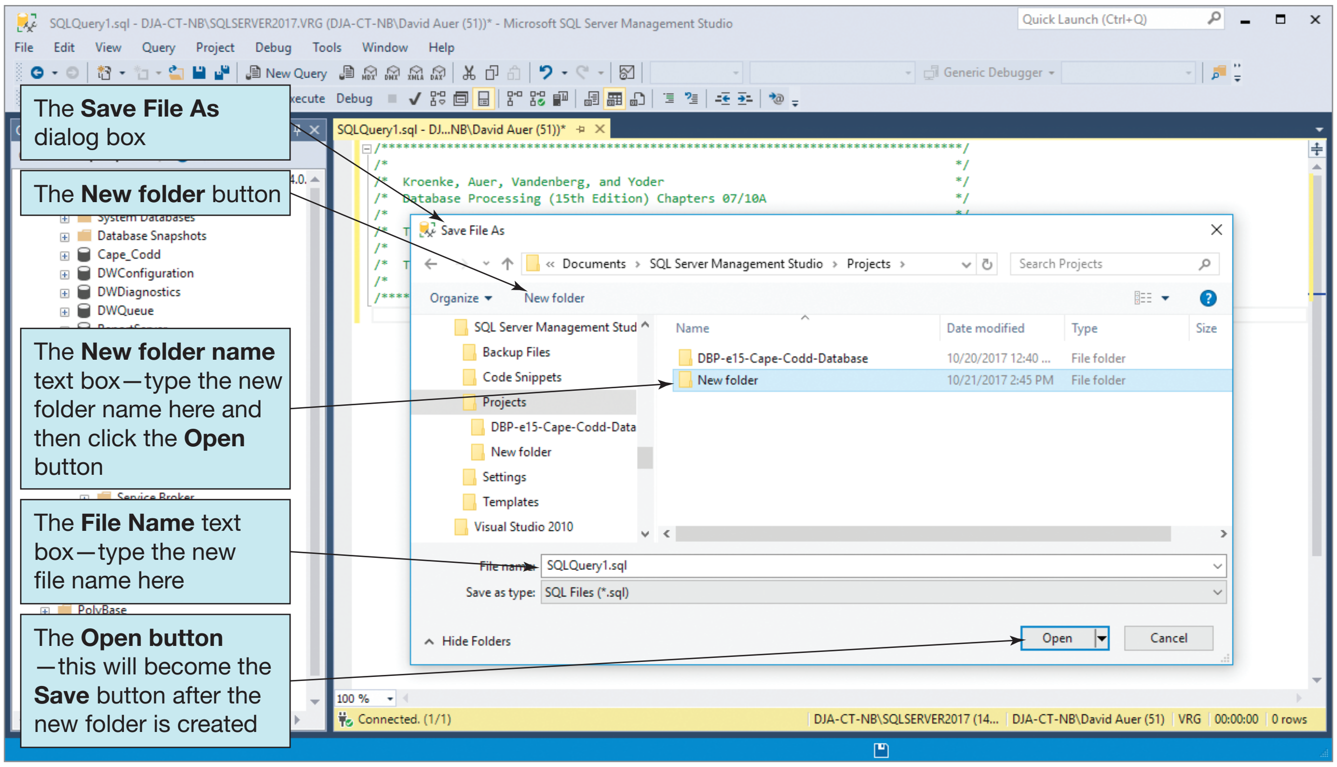
The VRG Database in the Object Explorer


SQLQuery1.sql - DJA-CT-NB\SQLSERVER2017.VRG (DJA-CT-NB\David Auer (51))* - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
VRG
Execute Debug
Object Explorer
Connect
SQLQuery1.sql - DJ...NB\David Auer (51))*
DJA-CT-NB\SQLSERVER2017 (SQL Server 14.0.
Databases
System Databases
Database Snapshots
Cape_Codd
DWConfiguration
DWDiagnostics
DWQueue
ReportServer
ReportServerTempDB
VRG
Database Diagrams
Tables
Views
External Resources
Synonyms
Programmability
Service Broker
Storage
Security
Security
Server Objects
Replication
PolyBase
Always On High Availability
Management
Integration Services Catalogs
SQL Server Agent
XE Profiler
100 %
Connected. (1/1)
DJA-CT-NB\SQLSERVER2017 (14... DJA-CT-NB\David Auer (51) VRG 00:00:00 0 rows
Ready Ln 11 Col 1 Ch 1 INS
The **Open File** button
The **Save** button
The **Close** button
The SQL text—the yellow line on the right margin shows that this text has not been saved.

```
/*****************************************************************************/
/*                                                                           */
/*  Kroenke, Auer, Vandenberg, and Yoder                                     */
/*  Database Processing (15th Edition) Chapters 07/10A                       */
/*                                                                           */
/*  The View Ridge Gallery (VRG) Database - Create Tables                    */
/*                                                                           */
/*  These are the Microsoft SQL Server 2016/2017 SQL code solutions          */
/*                                                                           */
/*****************************************************************************/
```

FIGURE 10A-25

Entering SQL Statements in the SQL Editor


The Save File As dialog box
The New folder button
The New folder name text box—type the new folder name here and then click the Open button
The File Name text box—type the new file name here
The Open button—this will become the Save button after the new folder is created
Save File As
Organize
New folder
Open
Cancel
Hide Folders

```
/*****************************************************************************/
/*                                                                           */
/*  Kroenke, Auer, Vandenberg, and Yoder                                     */
/*  Database Processing (15th Edition) Chapters 07/10A                       */
```

Documents › SQL Server Management Studio › Projects ›

Name	Date modified	Type
DBP-e15-Cape-Codd-Database	10/20/2017 12:40 ...	File folder
New folder	10/21/2017 2:45 PM	File folder

File name: SQLQuery1.sql
Save as type: SQL Files (*.sql)

FIGURE 10A-26

Saving the SQL Script in a New Folder

7. Type the file name **DBP-e15-VRG-Create-Tables** in the File Name text box of the Save File As dialog box.
8. Click the **Save** Button on the Save File As dialog box. The script is saved, the colored line on the left-hand margin of the text is changed to green, and the tab is renamed with the new file name.
9. Click the tabbed window **X [Close]** button shown in Figure 10A-25 to close the script window.

BY THE WAY Because the VRG database example we use in Chapter 7 and this chapter is fairly complex, complete SQL scripts to create the VRG tables and populate them with data are available at the book's Web site at *www.pearsonhighered.com/kroenke*. These scripts will allow you to build the basic VRG database and then actually try out the VRG database SQL code examples in the chapters. You will still need to read and understand the discussions of the SQL code for these two scripts to be sure you understand all the underlying concepts.

Creating the View Ridge Gallery VRG Database Table Structure

The SQL Server 2017 version of the SQL CREATE TABLE statements for the View Ridge Gallery VRG database in Chapter 7 is shown in Figure 10A-27.

Note that we are using the table name TRANS rather than TRANSACTION in Figure 10A-27. This was done because TRANSACTION is a **reserved word** in SQL Server 2017.[5] Even if you make TRANSACTION a **delimited identifier** by placing the name in square brackets, as in [TRANSACTION], SQL Server still becomes confused when executing the logic of stored procedures and triggers. Life became much simpler for applications using this database when the table TRANSACTION was renamed to TRANS (which is not a

[5]For a complete list of SQL Server 2017 reserved keywords, ODBC reserved keywords (ODBC is discussed in Chapter 11), and potential future SQL Server keywords, see the SQL Server 2017 Books Online article on "Reserved Keywords (Transact-SQL)" at *http://msdn.microsoft.com/en-us/library/ms189822.aspx*.

```
/*******************************************************************************/
/*                                                                             */
/*     Kroenke, Auer, Vandenberg, and Yoder                                    */
/*     Database Processing (15th Edition) Chapters 07/10A                      */
/*                                                                             */
/*     The View Ridge Gallery (VRG) Database - Create Tables                   */
/*                                                                             */
/*     These are the Microsoft SQL Server 2016/2017 SQL code solutions         */
/*                                                                             */
/*******************************************************************************/

USE VRG
GO

CREATE TABLE ARTIST (
        ArtistID                Int                 NOT NULL IDENTITY(1,1),
        LastName                Char(25)            NOT NULL,
        FirstName               Char(25)            NOT NULL,
        Nationality             Char(30)            NULL,
        DateOfBirth             Numeric(4)          NULL,
        DateDeceased            Numeric(4)          NULL,
        CONSTRAINT          ArtistPK                PRIMARY KEY(ArtistID),
        CONSTRAINT          ArtistAK1               UNIQUE(LastName, FirstName),
        CONSTRAINT          NationalityValues       CHECK (Nationality IN
                                ('Canadian', 'English', 'French',
                                'German', 'Mexican', 'Russian', 'Spanish',
                                'United States')),
        CONSTRAINT          BirthValuesCheck        CHECK (DateOfBirth < DateDeceased),
        CONSTRAINT          ValidBirthYear          CHECK
                               (DateOfBirth LIKE '[1-2][0-9][0-9][0-9]'),
        CONSTRAINT          ValidDeathYear          CHECK
                               (DateDeceased LIKE '[1-2][0-9][0-9][0-9]')
        );

CREATE TABLE WORK (
        WorkID                  Int                 NOT NULL IDENTITY(500,1),
        Title                   Char(35)            NOT NULL,
        [Copy]                  Char(12)            NOT NULL,
        Medium                  Char(35)            NULL,
        [Description]           Varchar(1000)       NULL DEFAULT 'Unknown provenance',
        ArtistID                Int                 NOT NULL,
        CONSTRAINT          WorkPK                  PRIMARY KEY(WorkID),
        CONSTRAINT          WorkAK1                 UNIQUE(Title, Copy),
        CONSTRAINT          ArtistFK                FOREIGN KEY(ArtistID)
                                REFERENCES ARTIST(ArtistID)
                                    ON UPDATE NO ACTION
                                    ON DELETE NO ACTION
        );

CREATE TABLE CUSTOMER (
        CustomerID              Int                 NOT NULL IDENTITY(1000,1),
        LastName                Char(25)            NOT NULL,
        FirstName               Char(25)            NOT NULL,
        EmailAddress            Varchar(100)        NULL,
        EncryptedPassword       VarChar(50)         NULL,
        Street                  Char(30)            NULL,
        City                    Char(35)            NULL,
        [State]                 Char(2)             NULL,
        ZIPorPostalCode         Char(9)             NULL,
        Country                 Char(50)            NULL,
        AreaCode                Char(3)             NULL,
        PhoneNumber             Char(8)             NULL,
        CONSTRAINT          CustomerPK              PRIMARY KEY(CustomerID),
        CONSTRAINT          EmailAK1                UNIQUE(EmailAddress)
        );
```

FIGURE 10A-27

The SQL Statements to Create the VRG Table Structure

```
CREATE TABLE TRANS (
        TransactionID          Int                 NOT NULL IDENTITY(100,1),
        DateAcquired           Date                NOT NULL,
        AcquisitionPrice       Numeric(8,2)        NOT NULL,
        AskingPrice            Numeric(8,2)        NULL,
        DateSold               Date                NULL,
        SalesPrice             Numeric(8,2)        NULL,
        CustomerID             Int                 NULL,
        WorkID                 Int                 NOT NULL,
        CONSTRAINT          TransPK                PRIMARY KEY(TransactionID),
        CONSTRAINT          TransWorkFK            FOREIGN KEY(WorkID)
                               REFERENCES WORK(WorkID)
                                   ON UPDATE NO ACTION
                                   ON DELETE NO ACTION,
        CONSTRAINT          TransCustomerFK      FOREIGN KEY(CustomerID)
                               REFERENCES CUSTOMER(CustomerID)
                                   ON UPDATE NO ACTION
                                   ON DELETE NO ACTION,
        CONSTRAINT          SalesPriceRange     CHECK
                               ((SalesPrice > 0) AND (SalesPrice <=500000)),
        CONSTRAINT             ValidTransDate    CHECK (DateAcquired <= DateSold)
        );

CREATE TABLE CUSTOMER_ARTIST_INT(
        ArtistID               Int                 NOT NULL,
        CustomerID             Int                 NOT NULL,
        CONSTRAINT          CAIntPK                PRIMARY KEY(ArtistID, CustomerID),
        CONSTRAINT          CAInt_ArtistFK         FOREIGN KEY(ArtistID)
                               REFERENCES ARTIST(ArtistID)
                                   ON UPDATE NO ACTION
                                   ON DELETE CASCADE,
        CONSTRAINT          CAInt_CustomerFK       FOREIGN KEY(CustomerID)
                               REFERENCES CUSTOMER(CustomerID)
                                   ON UPDATE NO ACTION
                                   ON DELETE CASCADE
        );
```

FIGURE 10A-27

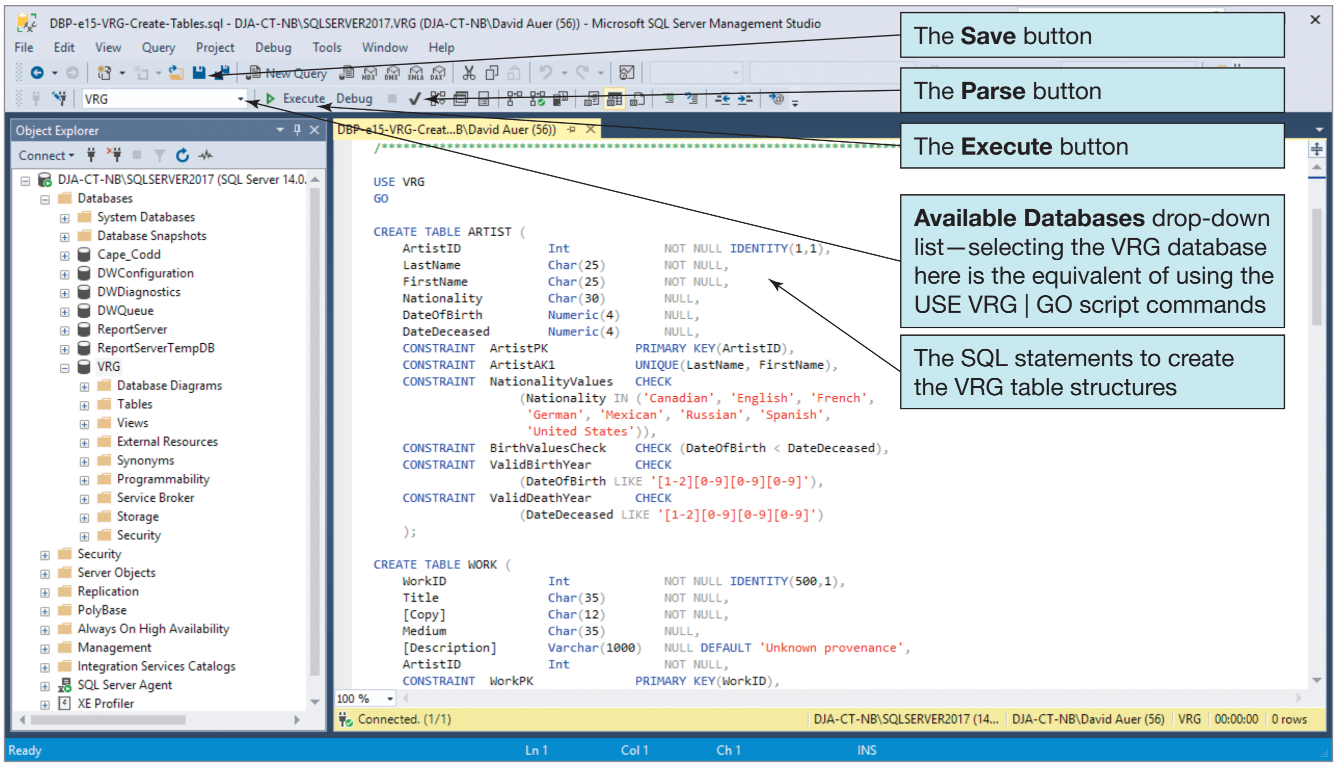
Continued

Transact-SQL keyword, although TRAN is). WORK is not currently a Transact-SQL reserved word, but it is an ODBC reserved word (ODBC will be discussed in Chapter 11) and reserved as a future keyword. Still, SQL Server is currently less sensitive to it, and therefore we can use it in delimited identifier form, enclosed in square brackets as [WORK].

SQL Server supports surrogate keys, and the surrogate key columns are created using the **Transact-SQL IDENTITY ({StartValue}, {Increment}) property** with the primary key of the ARTIST, [WORK], CUSTOMER, and TRANS tables. The IDENTITY property has the syntax IDENTITY ({StartValue}, {Increment}), where *startvalue* is the starting value for the surrogate key and *increment* is the value added to the previous surrogate key value each time a new value is created.

Creating the VRG Table Structure Using SQL Statements

1. In the SQL Server Management Studio, click the **Open File** button shown in Figure 10A-25 to open the Open File dialog box.
2. Click the file name **DBP-e15-VRG-Create-Tables.sql** to select the file.
3. Click the **Open** button in the Open File dialog box. The script is opened for use in a tabbed document window in the SQL editor.
4. Click the **Intellisense Enabled** button to *disable* Intellisense.
5. Type in the SQL statements shown in Figure 10A-27. Be sure to save the script often, and save the script a final time after you have finished entering all the SQL statements.


DBP-e15-VRG-Create-Tables.sql - DJA-CT-NB\SQLSERVER2017.VRG (DJA-CT-NB\David Auer (56)) - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
VRG
Execute Debug
Object Explorer
Connect
DJA-CT-NB\SQLSERVER2017 (SQL Server 14.0.
Databases
System Databases
Database Snapshots
Cape_Codd
DWConfiguration
DWDiagnostics
DWQueue
ReportServer
ReportServerTempDB
VRG
Database Diagrams
Tables
Views
External Resources
Synonyms
Programmability
Service Broker
Storage
Security
Security
Server Objects
Replication
PolyBase
Always On High Availability
Management
Integration Services Catalogs
SQL Server Agent
XE Profiler
DBP-e15-VRG-Creat...B\David Auer (56))
100 %
Connected. (1/1)
DJA-CT-NB\SQLSERVER2017 (14... DJA-CT-NB\David Auer (56) VRG 00:00:00 0 rows
Ready
Ln 1 Col 1 Ch 1 INS
The **Save** button
The **Parse** button
The **Execute** button
Available Databases drop-down list—selecting the VRG database here is the equivalent of using the USE VRG | GO script commands
The SQL statements to create the VRG table structures

```
/*****************************************************************

USE VRG
GO

CREATE TABLE ARTIST (
    ArtistID            Int             NOT NULL IDENTITY(1,1),
    LastName            Char(25)        NOT NULL,
    FirstName           Char(25)        NOT NULL,
    Nationality         Char(30)        NULL,
    DateOfBirth         Numeric(4)      NULL,
    DateDeceased        Numeric(4)      NULL,
    CONSTRAINT  ArtistPK            PRIMARY KEY(ArtistID),
    CONSTRAINT  ArtistAK1           UNIQUE(LastName, FirstName),
    CONSTRAINT  NationalityValues   CHECK
                    (Nationality IN ('Canadian', 'English', 'French',
                     'German', 'Mexican', 'Russian', 'Spanish',
                     'United States')),
    CONSTRAINT  BirthValuesCheck    CHECK (DateOfBirth < DateDeceased),
    CONSTRAINT  ValidBirthYear      CHECK
                    (DateOfBirth LIKE '[1-2][0-9][0-9][0-9]'),
    CONSTRAINT  ValidDeathYear      CHECK
                    (DateDeceased LIKE '[1-2][0-9][0-9][0-9]')
    );

CREATE TABLE WORK (
    WorkID              Int             NOT NULL IDENTITY(500,1),
    Title               Char(35)        NOT NULL,
    [Copy]              Char(12)        NOT NULL,
    Medium              Char(35)        NULL,
    [Description]       Varchar(1000)   NULL DEFAULT 'Unknown provenance',
    ArtistID            Int             NOT NULL,
    CONSTRAINT  WorkPK              PRIMARY KEY(WorkID),
```

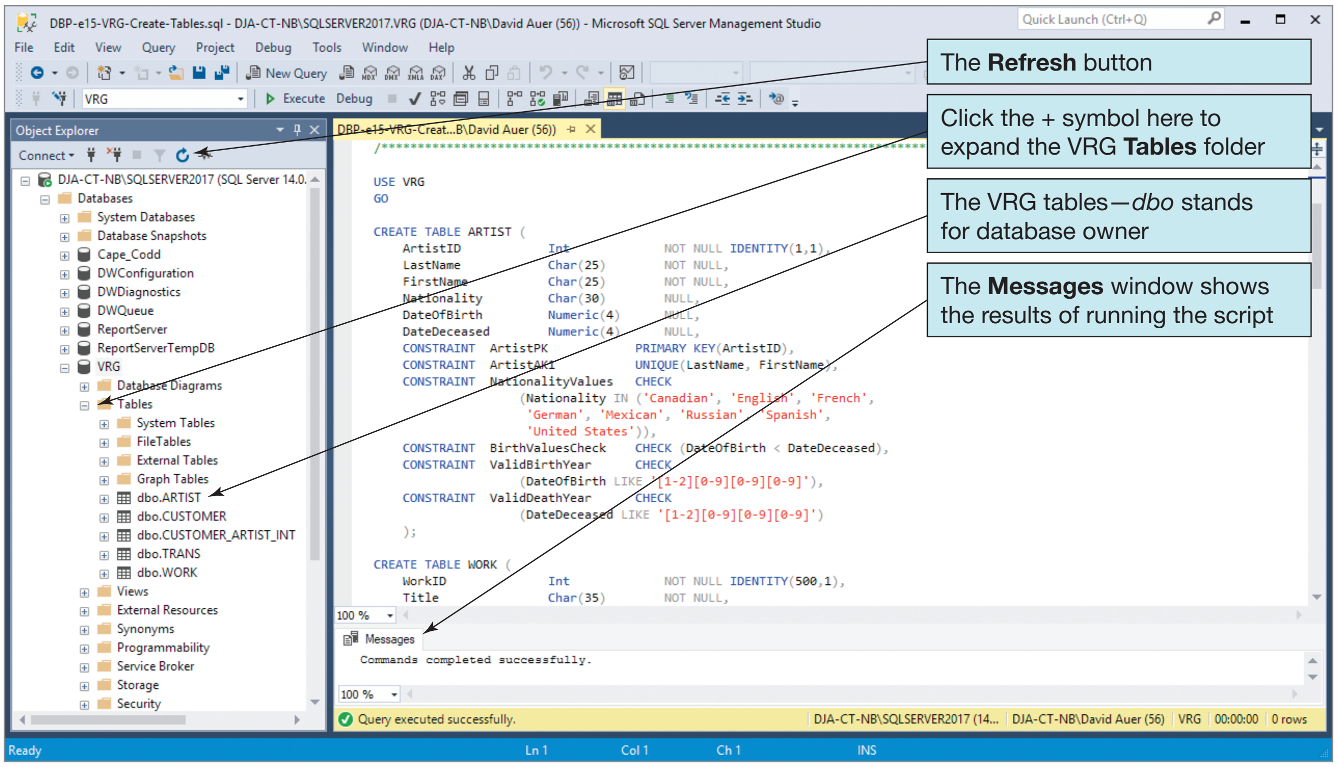
FIGURE 10A-28

The SQL Script to Create the VRG Table Structure

6. Scroll to the first CREATE TABLE statement at the top of the script. The completed SQL script to create the VRG table structure appears as shown in Figure 10A-28.
7. The **USE VRG | GO** commands in the SQL script are used to apply the SQL statements to the VRG database. Alternatively, you can select the database name **VRG** in the Available Databases drop-down list, as shown in Figure 10A-28 by clicking the Available Databases drop-down list arrow and selecting VRG. It is important that VRG be the active database, because the SQL statement will be run for that database.
8. Click the **Parse** button shown in Figure 10A-28. A message window appears below the tabbed document window. If this window contains the message "Command(s) completed successfully," there are probably no errors in the script. However, if any error messages are displayed, then you have errors in your SQL statements. Correct any errors. Repeat this step until you see the message "Command(s) completed successfully."
9. Click the **Save** button to save your debugged SQL script.
10. Click the **Execute** button shown in Figure 10A-28. The tables are created, and the message "Command(s) completed successfully" appears in the tabbed Messages window, as shown in Figure 10A-29.
11. Expand the VRG Tables folder to see the VRG tables, as shown in Figure 10A-29.
12. Click the document window **Close** button to close the SQL script.

BY THE WAY

The *dbo* in Figure 10A-29 stands for **database owner**. This is the name of the default **SQL Server schema**, which is a named collection of database objects—think of a schema as a container that holds database objects such as tables and views. SQL Server schemas are part of the SQL Server security model, which we discuss in detail later in this chapter. SQL Server treats the schema name as part of the object name, which is why the ARTIST table appears in Figure 10A-29 as dbo.ARTIST. For now, just understand that if you installed SQL Server 2017 and created the VRG database, you are the owner of the dbo schema and of any database objects that are contained in the dbo schema.


DBP-e15-VRG-Create-Tables.sql - DJA-CT-NB\SQLSERVER2017.VRG (DJA-CT-NB\David Auer (56)) - Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File Edit View Query Project Debug Tools Window Help
New Query
VRG
Execute Debug
Object Explorer
Connect
DJA-CT-NB\SQLSERVER2017 (SQL Server 14.0.
Databases
System Databases
Database Snapshots
Cape_Codd
DWConfiguration
DWDiagnostics
DWQueue
ReportServer
ReportServerTempDB
VRG
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.ARTIST
dbo.CUSTOMER
dbo.CUSTOMER_ARTIST_INT
dbo.TRANS
dbo.WORK
Views
External Resources
Synonyms
Programmability
Service Broker
Storage
Security
DBP-e15-VRG-Creat...B\David Auer (56))
The **Refresh** button
Click the + symbol here to expand the VRG **Tables** folder
The VRG tables—*dbo* stands for database owner
The **Messages** window shows the results of running the script
100 %
Messages
Query executed successfully.
DJA-CT-NB\SQLSERVER2017 (14... DJA-CT-NB\David Auer (56) VRG 00:00:00 0 rows
Ready Ln 1 Col 1 Ch 1 INS

```
/**********************************************************************

USE VRG
GO

CREATE TABLE ARTIST (
    ArtistID            Int             NOT NULL IDENTITY(1,1),
    LastName            Char(25)        NOT NULL,
    FirstName           Char(25)        NOT NULL,
    Nationality         Char(30)        NULL,
    DateOfBirth         Numeric(4)      NULL,
    DateDeceased        Numeric(4)      NULL,
    CONSTRAINT  ArtistPK            PRIMARY KEY(ArtistID),
    CONSTRAINT  ArtistAK1           UNIQUE(LastName, FirstName),
    CONSTRAINT  NationalityValues   CHECK
                (Nationality IN ('Canadian', 'English', 'French',
                 'German', 'Mexican', 'Russian', 'Spanish',
                 'United States')),
    CONSTRAINT  BirthValuesCheck    CHECK (DateOfBirth < DateDeceased),
    CONSTRAINT  ValidBirthYear      CHECK
                (DateOfBirth LIKE '[1-2][0-9][0-9][0-9]'),
    CONSTRAINT  ValidDeathYear      CHECK
                (DateDeceased LIKE '[1-2][0-9][0-9][0-9]')
    );

CREATE TABLE WORK (
    WorkID              Int             NOT NULL IDENTITY(500,1),
    Title               Char(35)        NOT NULL,
```

```
Commands completed successfully.
```

FIGURE 10A-29

The VRG Database Tables

Reviewing Database Structures in the SQL Server GUI Display

Now we have created the VRG table and relationship structure. After building the table structure using SQL statements, we can inspect the results using the SQL Server GUI tools. Let's take a look at the ARTIST table, particularly at the properties of the ArtistID primary key.

Viewing the ARTIST Table Structure in the GUI Display

1. In the SQL Server Management Studio Object Browser, right-click the **dbo.ARTIST** table object to open the table shortcut menu, as shown in Figure 10A-30.
2. In the table shortcut menu, click the **Design** command. The ARTIST table design is displayed in a tabbed document window, as shown in Figure 10A-31, with the Identity Specification properties expanded. You might need to increase the width of the bottom pane to see it.
3. Click the document window **Close** button to close the ARTIST table's columns and column properties tabbed window.

We can also inspect the constraints on the ARTIST table. We'll take a look at the ValidBirthYear constraint we coded into our SQL CREATE TABLE statements.

Viewing the ARTIST Table Constraints in the GUI Display

1. In the SQL Server Management Studio Object Browser, expand the **dbo.ARTIST** table object.
2. Expand the **Constraints** folder. The ARTIST table constraint objects are displayed in the Object Browser.
3. Right-click the **ValidBirthYear** constraint object to display the shortcut menu.
4. Click the **Modify** command in the shortcut menu. The Check Constraints dialog box appears, as shown in Figure 10A-32.
5. Note that the check constraint itself is located in the Expression text box of the (General) properties group, and it can be edited there if needed. At this point, however, there is no need to change the constraint.
6. Click the **Close** button to close the Check Constraints dialog box.

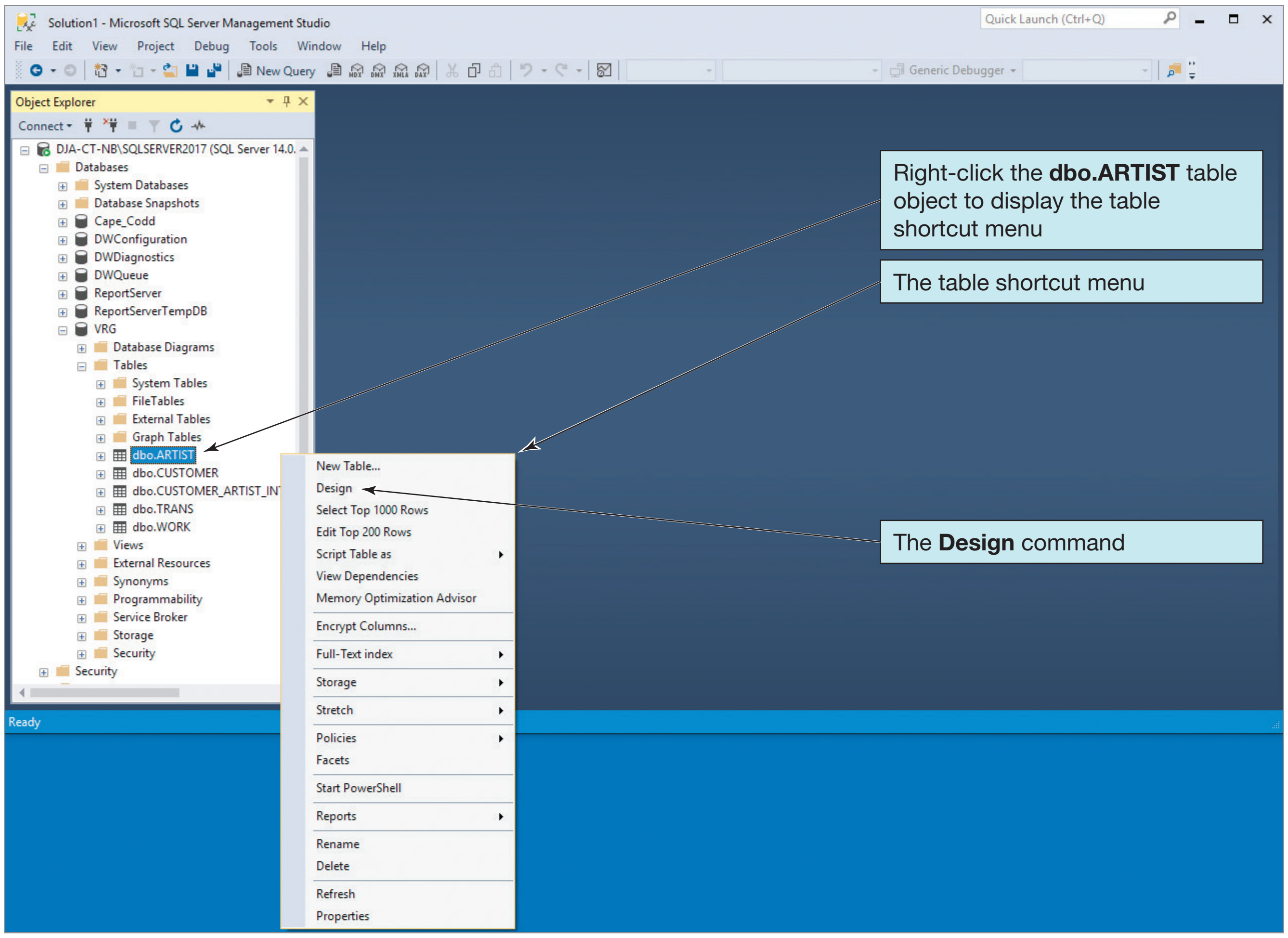


FIGURE 10A-30

The Table Shortcut Menu

7. Click the document window **Close** button to close the ARTIST table's columns and column properties tabbed window, which was automatically opened before the Check Constraints dialog box was displayed.
8. Contract (minimize) the **dbo.ARTIST** table object display.

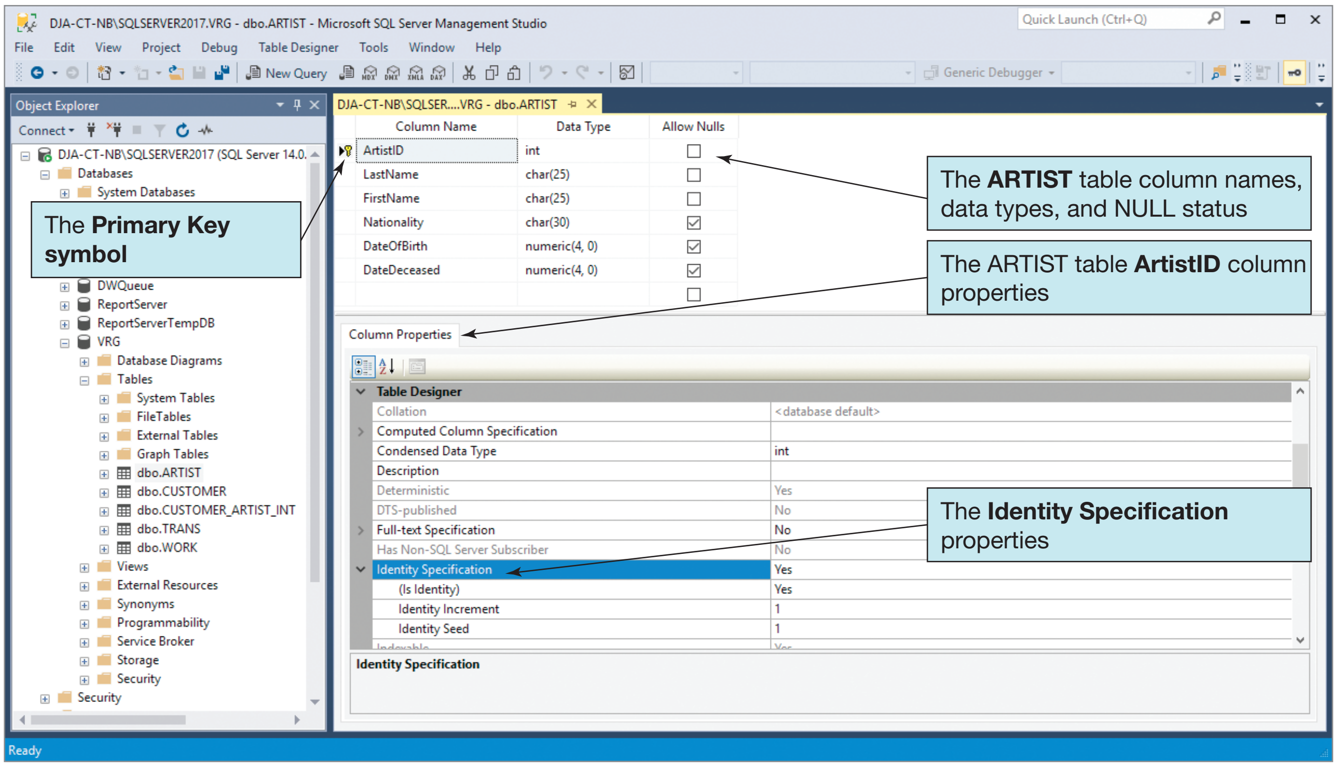
Clearly, it is easier to key the constraint into SQL statements than it would be to type it as shown in this window!

Throughout this chapter, you may see dialog boxes similar to the one shown in Figure 10A-32 that have properties that reference something about replication (for example: Enforce for Replication). All such references refer to distributed SQL Server databases in which data are placed in two or more databases and updates to them are coordinated in some fashion. We will not consider that topic here, so ignore any references to replication. You can learn more by searching for the replication topic in Chapter 12 or the SQL Server 2017 documentation.

To ensure that the VRG database relationships were created correctly, let's create a **database diagram** for the VRG database. We can then use that diagram as the basis for checking the relationships among the various tables.

Creating an SQL Server Database Diagram

1. In the SQL Server Management Studio Object Browser, right-click the **VRG Database Diagrams** folder object to display the shortcut menu.
2. Click the **Install Diagram Support** command on the shortcut menu.
3. A Microsoft SQL Server Management Studio dialog box appears asking if you want to create some objects that support database diagrams. Of course you do! Isn't that the command you just gave? Click the **Yes** button!
4. Right-click the **VRG Database Diagrams** folder object to display the shortcut menu.


The ARTIST table column names, data types, and NULL status
The Primary Key symbol
The ARTIST table ArtistID column properties
The Identity Specification properties
DJA-CT-NB\SQLSERVER2017.VRG - dbo.ARTIST - Microsoft SQL Server Management Studio
Object Explorer
Column Properties
Ready

Column Name	Data Type	Allow Nulls
ArtistID	int	☐
LastName	char(25)	☐
FirstName	char(25)	☐
Nationality	char(30)	☑
DateOfBirth	numeric(4, 0)	☑
DateDeceased	numeric(4, 0)	☑

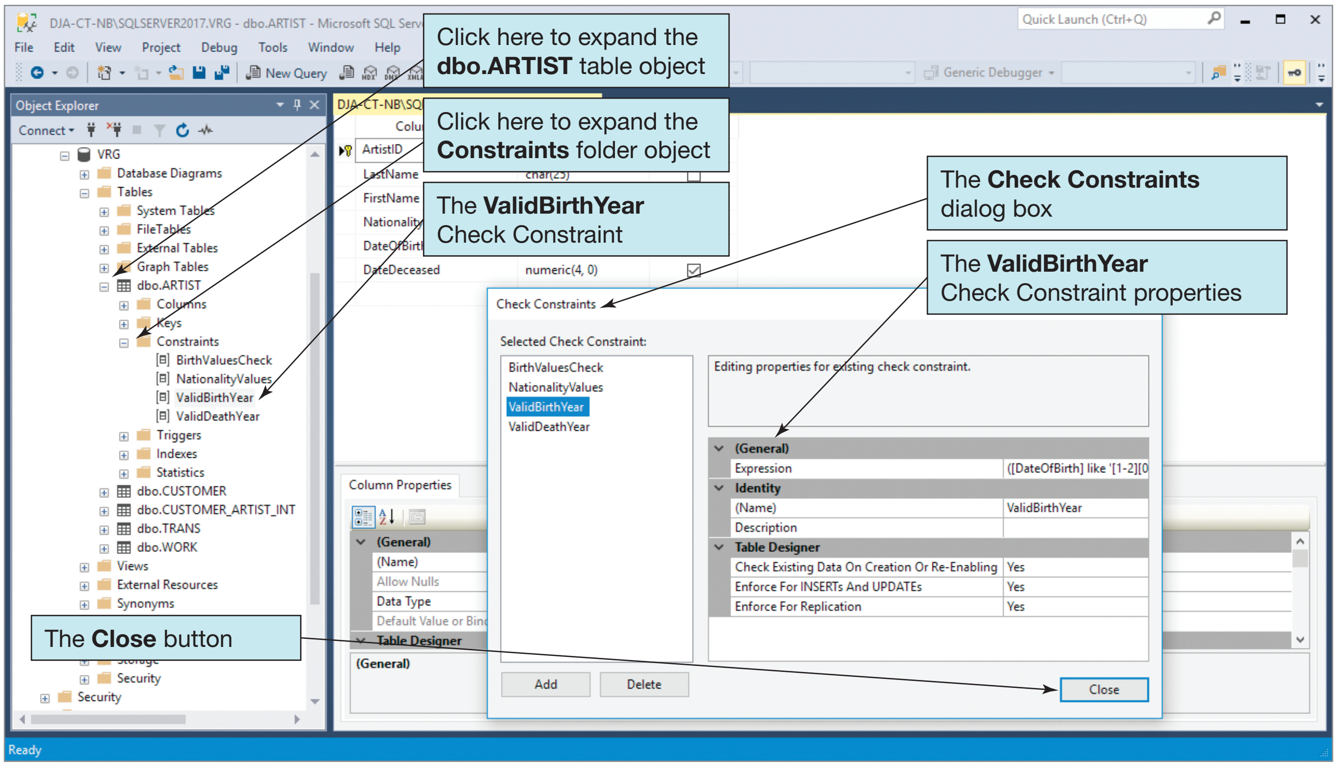
Table Designer	
Collation	<database default>
Computed Column Specification	
Condensed Data Type	int
Description	
Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

Identity Specification

FIGURE 10A-31

The ARTIST Table Columns and Column Properties

5. Click the **New Database Diagram** command on the shortcut menu. The Add Table dialog box appears, as shown in Figure 10A-33.
6. Click the **Add** button to add the highlighted ARTIST table to the database diagram.


Click here to expand the dbo.ARTIST table object
Click here to expand the Constraints folder object
The ValidBirthYear Check Constraint
The Check Constraints dialog box
The ValidBirthYear Check Constraint properties
The Close button
Check Constraints
Add
Delete
Close
Ready

Selected Check Constraint:

BirthValuesCheck
NationalityValues
ValidBirthYear
ValidDeathYear

Editing properties for existing check constraint.

(General)	
Expression	([DateOfBirth] like '[1-2][0
Identity	
(Name)	ValidBirthYear
Description	
Table Designer	
Check Existing Data On Creation Or Re-Enabling	Yes
Enforce For INSERTs And UPDATEs	Yes
Enforce For Replication	Yes

FIGURE 10A-32

The ARTIST Table Check Constraints

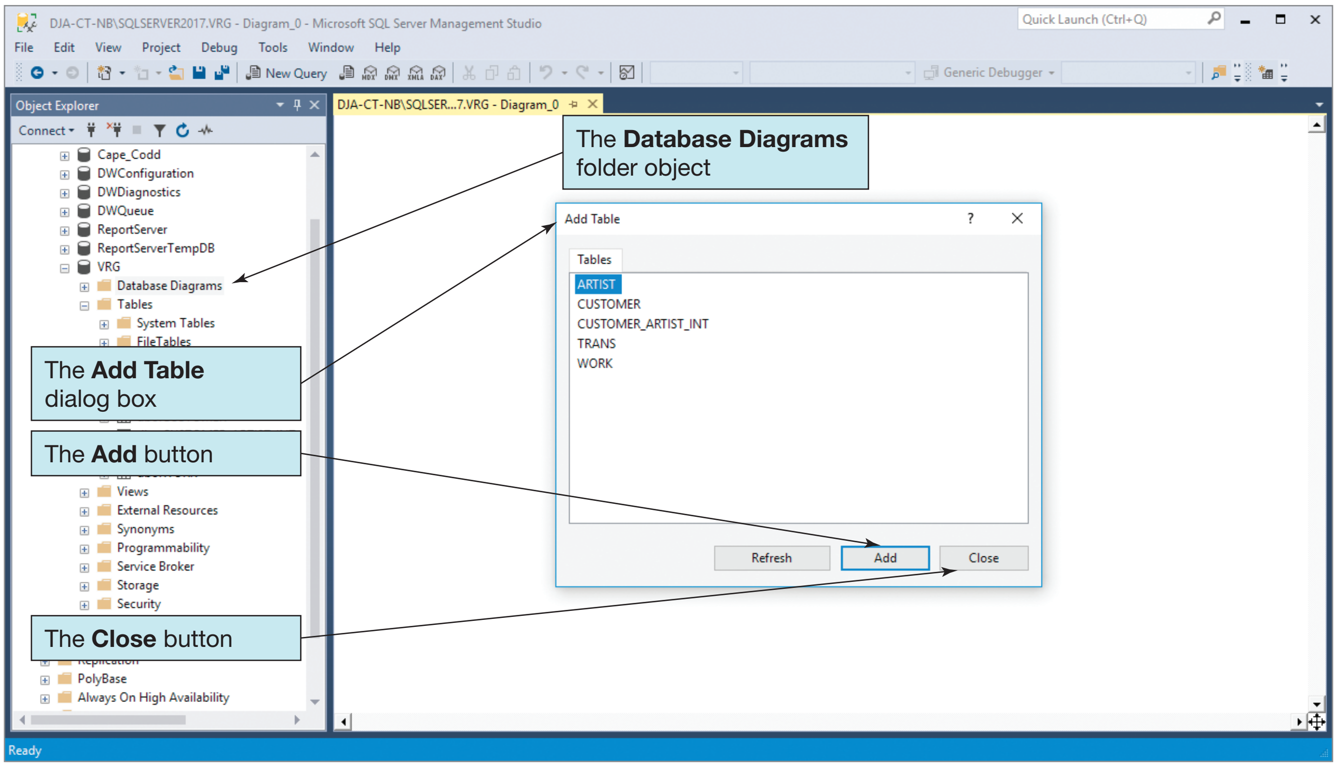


FIGURE 10A-33

The Add Table Dialog Box

7. In the Add Table dialog box, the next table in the list is now highlighted. Click the **Add** button to add the highlighted table to the database diagram. Repeat this process until all of the VRG tables have been added to the diagram.
8. Click the **Close** button to close the Add Table dialog box.
9. Rearrange the VRG tables in the database diagram until it appears as shown in Figure 10A-34, and then click the **Save** button and name the diagram **VRG-Database-Diagram**.
10. Expand the **VRG Database Diagrams** folder object to display the VRG-Database-Diagram object, which is shown in Figure 10A-35.

Now we can use the VRG database diagram to view the properties of a relationship. We will take a look at the relationship between ARTIST and WORK, checking the referential integrity constraint and the cascading update and deletion behavior between ARTIST and WORK.

BY THE WAY

What exactly is a *database diagram*? Let us be very clear not only about *what it is* but also about *what it isn't*:

- It is *not* a *data model* as described in Chapter 5, it is *not* in data model E-R notation, and it is *not* used to determine user requirements.
- It is *not* a database design as described in Chapter 5, it is *not* in database design E-R notation, and it is *not* used as a component design.
- It *is* a visual representation of the database as it has been actually created in SQL server, and it *is* used to help visualize and inspect the database.
  - It *does* indicate primary keys.
  - It *does not* indicate foreign keys.
  - It *does* show relationships and which table contains the foreign key and the corresponding primary key, but it *does not* visually connect these fields (we manipulated the relationship lines in Figure 10A-34 to visually show the linked fields, but this is *not* automatically done and often *isn't* done by the user.

Although the database diagram is useful, do not confuse it with either a *data model* or a *database design*.

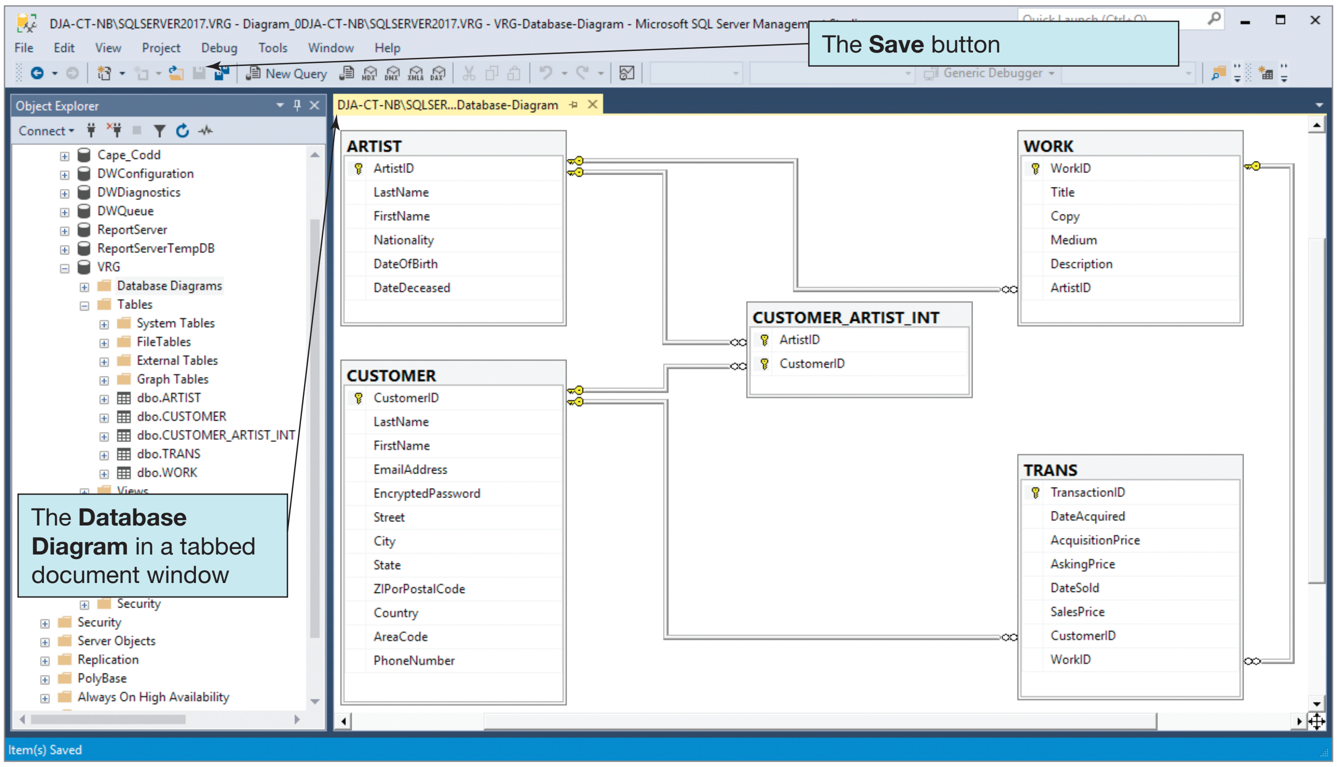

The Save button
The Database Diagram in a tabbed document window
DJA-CT-NB\SQLSERVER2017.VRG - Diagram_0DJA-CT-NB\SQLSERVER2017.VRG - VRG-Database-Diagram - Microsoft SQL Server Management Studio
File Edit View Project Debug Tools Window Help
Object Explorer
DJA-CT-NB\SQLSER...Database-Diagram
Cape_Codd
DWConfiguration
DWDiagnostics
DWQueue
ReportServer
ReportServerTempDB
VRG
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.ARTIST
dbo.CUSTOMER
dbo.CUSTOMER_ARTIST_INT
dbo.TRANS
dbo.WORK
Security
Server Objects
Replication
PolyBase
Always On High Availability
ARTIST: ArtistID, LastName, FirstName, Nationality, DateOfBirth, DateDeceased
CUSTOMER: CustomerID, LastName, FirstName, EmailAddress, EncryptedPassword, Street, City, State, ZIPorPostalCode, Country, AreaCode, PhoneNumber
WORK: WorkID, Title, Copy, Medium, Description, ArtistID
CUSTOMER_ARTIST_INT: ArtistID, CustomerID
TRANS: TransactionID, DateAcquired, AcquisitionPrice, AskingPrice, DateSold, SalesPrice, CustomerID, WorkID
Item(s) Saved

FIGURE 10A-34

The VRG Database Diagram

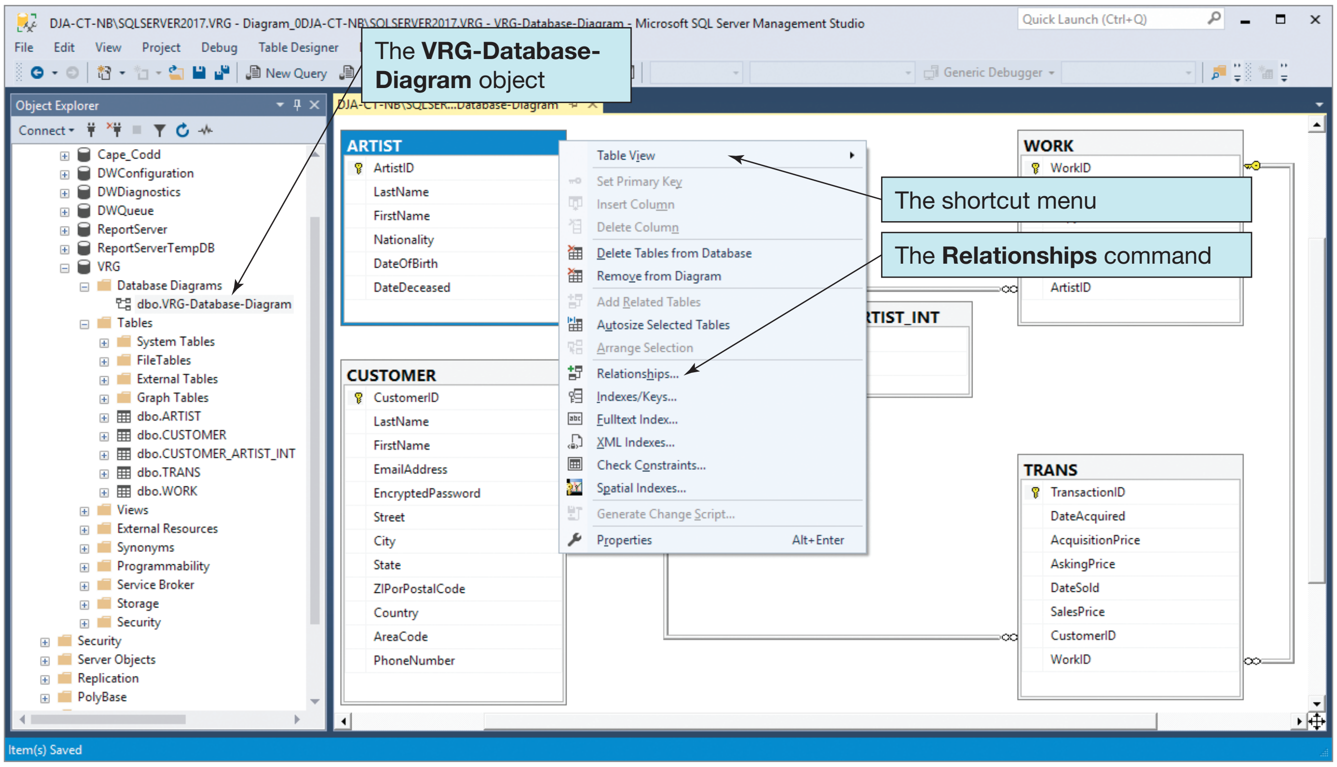
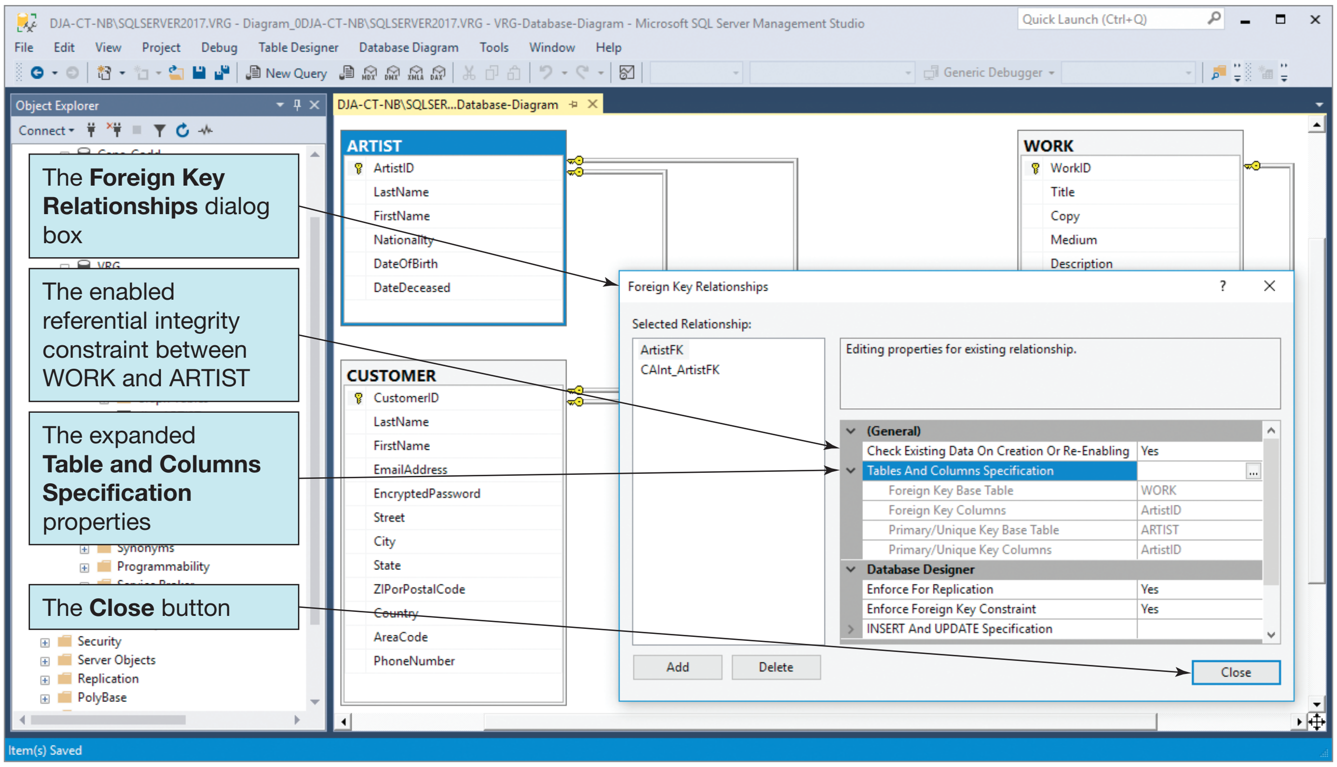


FIGURE 10A-35

The Database Diagram Table Shortcut Menu



The **Foreign Key Relationships** dialog box
The enabled referential integrity constraint between WORK and ARTIST
The expanded **Table and Columns Specification** properties
The **Close** button
DJA-CT-NB\SQLSERVER2017.VRG - Diagram_0DJA-CT-NB\SQLSERVER2017.VRG - VRG-Database-Diagram - Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File Edit View Project Debug Table Designer Database Diagram Tools Window Help
New Query
Generic Debugger
Object Explorer
Connect
Synonyms
Programmability
Security
Server Objects
Replication
PolyBase
DJA-CT-NB\SQLSER...Database-Diagram
ARTIST
ArtistID
LastName
FirstName
Nationality
DateOfBirth
DateDeceased
CUSTOMER
CustomerID
LastName
FirstName
EmailAddress
EncryptedPassword
Street
City
State
ZIPorPostalCode
Country
AreaCode
PhoneNumber
WORK
WorkID
Title
Copy
Medium
Description
Item(s) Saved

Foreign Key Relationships

Selected Relationship:

ArtistFK
CAInt_ArtistFK

Editing properties for existing relationship.

(General)	
Check Existing Data On Creation Or Re-Enabling	Yes
Tables And Columns Specification	
Foreign Key Base Table	WORK
Foreign Key Columns	ArtistID
Primary/Unique Key Base Table	ARTIST
Primary/Unique Key Columns	ArtistID
Database Designer	
Enforce For Replication	Yes
Enforce Foreign Key Constraint	Yes
INSERT And UPDATE Specification	

Add Delete Close

FIGURE 10A-36

The Foreign Key Relationships Dialog Box

Viewing Relationship Properties

1. Right-click the **ARTIST** table in the VRG Database Diagram to display the shortcut menu, as shown in Figure 10A-35. Note how many actions are available from this shortcut menu.
2. Click the **Relationships** command to display the Foreign Key Relationships dialog box, as shown in Figure 10A-36.
3. Expand the **Tables and Columns Specifications** property section, as shown in Figure 10A-36 by clicking the expand arrow next to it.
4. In Figure 10A-36, note that the Tables and Columns Specifications property section details the *primary and foreign keys* in the relationship between the two tables.
5. In Figure 10A-36, note that the Check Existing Data On Creation Or Re-Enabling property is enabled. This is the *referential integrity constraint* between the ArtistID column in WORK and the ArtistID column in ARTIST. You might need to drag the column separator to the right to see all of the text.
6. In the Foreign Key Relationships dialog box, expand the **INSERT and UPDATE Specification**, and then scroll down until it is visible. The property settings indicate that neither updates to the primary key of ARTIST nor deletions of rows in ARTIST will cause any cascading actions in the WORK table.
7. Click the **Close** button to close the Foreign Key Relationships dialog box.
8. Click the **Close** button on the Diagram tab to close the VRG Database Diagram.

Indexes

As discussed in Appendix G, an **index** is a special data structure that is created to improve database performance. SQL Server automatically creates an index on all primary and foreign keys. A developer can also direct SQL Server to create an index on other columns that are frequently used in WHERE clauses or on columns that are used for sorting data when sequentially processing a table for queries and reports.

SQL Server supports two kinds of indexes for tables: clustered and nonclustered. With a **clustered index**, the data are stored in the bottom level of the index and in the same order as that index. With a **nonclustered index**, the bottom level of an index does not contain data; it contains pointers to the data. Because rows can be sorted only in one physical order at a time, only one clustered index is allowed per table. Retrieval is faster with clustered indexes than with nonclustered indexes. Updating is normally faster as well with clustered indexes, but not if there are many updates in the same spot in the middle of the relation. For more information on clustered and nonclustered indexes, see the SQL Server 2017 Books Online article "Tables and Index Data Structures Architecture," which is available at *http://msdn.microsoft.com/en-us/library/ms180978.aspx*. Also see Appendix G, "Physical database Design and Data Structures for Database Processing."

SQL Server 2017 also supports indexes on XML data (see the discussion of XML in Chapter 11 and Appendix I) and geometric or geographic spatial data types. For each column of XML data, four types of indexes can be created: a primary XML index, a PATH secondary XML index, a VALUE secondary XML index, and a PROPERTY secondary XML index. For each column of spatial data, we can create a spatial index. For more information on XML indexes, see the SQL Server 2017 Books Online article "CREATE XML INDEX (Transact-SQL)" (*http://msdn.microsoft.com/en-us/library/bb934097.aspx*). For more information on spatial indexes, see the SQL Server 2017 Books Online article "Spatial Indexing Overview" (*http://msdn.microsoft.com/en-us/library/bb964712.aspx*).

To illustrate how to create indexes, we will create a new index on the ZIPorPostalCode column in the CUSTOMER table.

Creating a New Index

1. In the SQL Server Management Studio Object Browser, expand the **dbo.CUSTOMER** table object.
2. Expand the dbo.CUSTOMER table **Indexes** folder object to display the existing indexes.
3. Right-click the **Indexes** folder object to display the shortcut menu.
4. Click the **New Index** command on the shortcut menu and then the **Non-clustered index** command in the next shortcut menu. The New Index dialog box appears, as shown in Figure 10A-37.
5. Type the name **ZIPorPostalCodeIndex** in the Index Name text box of the New Index dialog box.

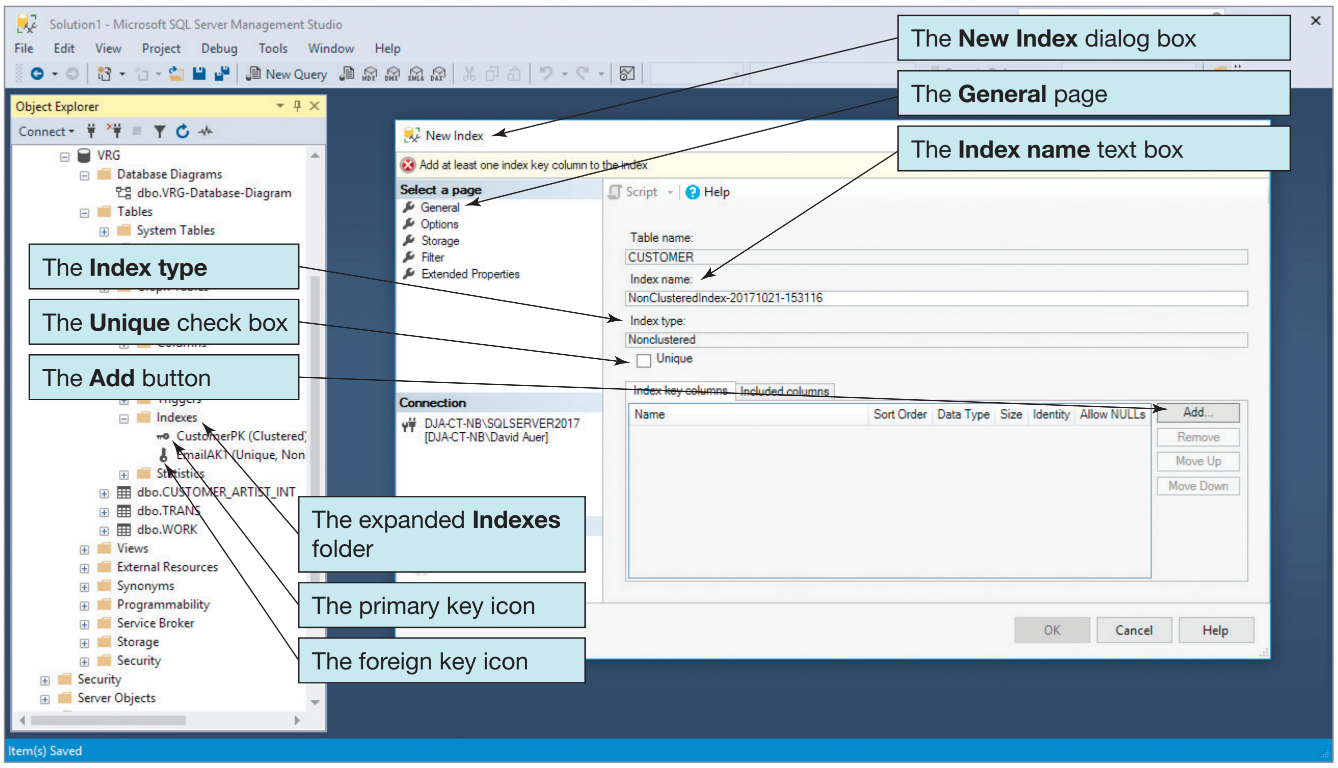


FIGURE 10A-37

The New Index Dialog Box

New Index

Add at least one index key column to the index

Table name: CUSTOMER

Index name: NonClusteredIndex-20171021-153116

Index type: Nonclustered

☐ Unique

Connection: DJA-CT-NB\SQLSERVER2017 [DJA-CT-NB\David Auer]

Item(s) Saved

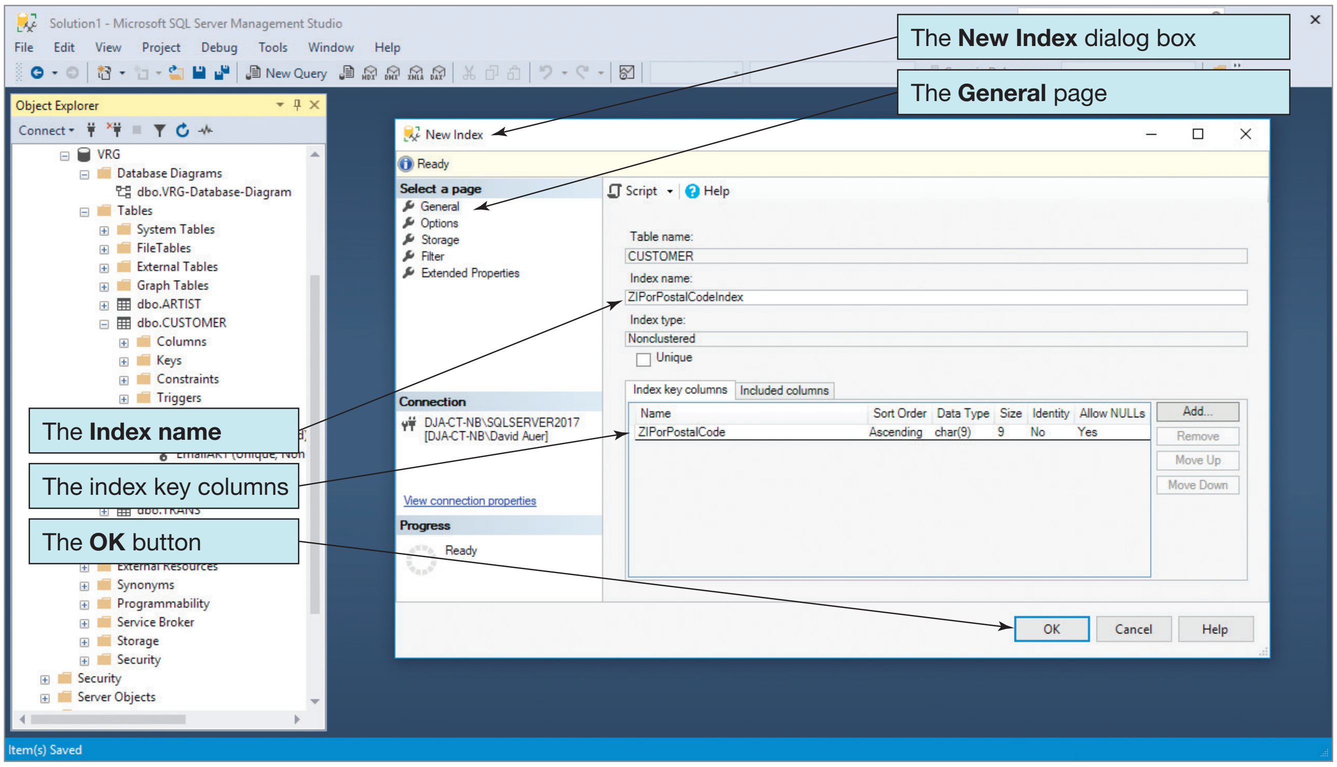


Table name: CUSTOMER

Index name: ZIPorPostalCodeIndex

Index type: Nonclustered

☐ Unique

Index key columns | Included columns

Name	Sort Order	Data Type	Size	Identity	Allow NULLs
ZIPorPostalCode	Ascending	char(9)	9	No	Yes

DJA-CT-NB\SQLSERVER2017
[DJA-CT-NB\David Auer]

FIGURE 10A-38

The Completed New Index Dialog Box

6. Click the **Add** button on the New Index dialog box to display the Select Columns from 'dbo.CUSTOMER' dialog box.
7. Click the **check box** for the ZIPorPostalCode column in the Select Columns from 'dbo.CUSTOMER' dialog box.
8. Click the **OK** button on the Select Columns from 'dbo.CUSTOMER' dialog box. The New Index dialog box now appears, as shown in Figure 10A-38.
9. Click the **OK** button in the New Index dialog box to create the ZIPorPostalCodeIndex. The ZIPorPostalCodeIndex nonclustered index object appears in the Indexes folder of the dbo.CUSTOMER table in the Object Browser.
10. Collapse the **dbo.CUSTOMER** table structure in the Object Browser.

Populating the VRG Database Tables with Data

You can enter data into SQL Server either by entering data into a table grid in the Microsoft SQL Server Management Studio GUI display or by using SQL INSERT statements. The Microsoft SQL Server Management Studio GUI display is more useful for occasional data edits than for populating all the tables of a new database. You can open a table grid for data entry by right-clicking the table name to display a shortcut menu and then clicking the Edit Top 200 Rows command.

However, we will use the same method for populating the VRG database tables that we used to create the table structure: an SQL script. But before we do that, we need to address the surrogate key values issue raised in Chapter 7. The data shown in Figure 7-15 is sample data, and the primary key values of CustomerID, ArtistID, WorkID, and TransactionID shown in that figure are nonsequential. Yet the IDENTITY ({StartValue}, {Increment}) property that we use to populate SQL Server surrogate primary keys creates sequential numbering.

This means that if we write and execute SQL INSERT statements to put the artist data shown in Figure 7-15(b) into the ARTIST table, the values of ArtistID that will be added to the table will be (1, 2, 3, 4, 5, 6, 7, 8, 9) instead of the values of (1, 2, 3, 4, 5, 11, 17, 18, 19) listed in the figure. How can we enter the needed nonsequential values?

The answer is the **Transact-SQL IDENTITY_INSERT property**. When IDENTITY_INSERT is set to OFF (the default setting), only the SQL Server DBMS can enter data into the controlled ID column in the table. When IDENTITY_INSERT is set to ON, values can be input into the controlled column in the table. However, IDENTITY_INSERT can only be set to ON for only one table at a time. Further, IDENTITY_INSERT requires the use of a column list containing the name of the surrogate key in each INSERT command.

Thus, instead of using an SQL INSERT statement that automatically enters the surrogate value, such as:

```
INSERT INTO ARTIST VALUES(
   'Miro', 'Joan', 'Spanish', 1893, 1983);
```

we have to use a set of SQL statements similar to the following:

```
SET IDENTITY_INSERT dbo.ARTIST ON
INSERT INTO ARTIST
       (ArtistID, LastName, FirstName, Nationality,
       DateOfBirth, DateDeceased)
       VALUES (1, 'Miro', 'Joan', 'Spanish', 1893, 1983);
SET IDENTITY_INSERT dbo.ARTIST OFF
```

Note how we set IDENTITY_INSERT to ON, insert the date, and then set IDENTITY_INSERT to OFF. Of course, this is a lot of work if we are inserting one row of data at a time, but when used in an SQL script that inserts a lot of data into a table, it makes sense. So, we will use an SQL script.

The set of SQL INSERT statements needed to populate the VRG database with the View Ridge Gallery data shown in Figure 7-15 is shown in Figure 10A-39. Create and save a new SQL script named **DBP-e15-VRG-Table-Data.sql** based on Figure 10A-39, testing it with the **Parse** command (use the Parse button) until you have corrected any errors. Save the corrected script, and then run the script (use the **Execute** button) to populate the tables. Close the script window after the script has been successfully run.

Creating SQL Views

SQL views were discussed in Chapter 7. One view we created there was CustomerInterestsView. In SQL Server 2017, views can be created in the Microsoft SQL Server Management Studio by using either an SQL statement (as we have done to create and populate the VRG tables) or by using the GUI Display (by right-clicking the Views folder to display a shortcut menu and then clicking the New View command). CustomerInterestsView can be created with the following SQL statement:

```
/* SQL View SQL-CREATE-VIEW-CH07-05 - CustomerInterestsView */
CREATE VIEW CustomerInterestsView AS
      SELECT     C.LastName AS CustomerLastName,
                 C.FirstName AS CustomerFirstName,
                 A.LastName AS ArtistName
      FROM       CUSTOMER AS C JOIN CUSTOMER_ARTIST_INT AS CAI
            ON   C.CustomerID = CAI.CustomerID
                 JOIN       ARTIST AS A
                      ON    CAI.ArtistID = A.ArtistID;
```

Note that the comment labeling this SQL CREATE VIEW statement refers to the view as SQL-CREATE-VIEW-CH07-05:

```
/* SQL View SQL-CREATE-VIEW-CH07-05 – CustomerInterestsView */
```

```
/*******************************************************************************/
/*                                                                             */
/*      Kroenke, Auer, Vandenberg, and Yoder                                   */
/*      Database Processing (15th Edition) Chapters 07/10A                     */
/*                                                                             */
/*      The View Ridge Gallery (VRG) Database - Insert Data                    */
/*                                                                             */
/*      These are the Microsoft SQL Server 2016/2017 SQL code solutions        */
/*                                                                             */
/*******************************************************************************/
/*                                                                             */
/*      This file contains the initial data for each table.                    */
/*                                                                             */
/*      NOTE:  We will have a problem entering the surrogate key values shown in */
/*      the text.                                                              */
/*                                                                             */
/*      The database table structure was set up using the SQL Server T-SQL   */
/*      keyword IDENTITY to create surrogate keys. This means that by default */
/*      we cannot enter values into any column defined with INDENTITY.         */
/*                                                                             */
/*      But that means we cannot enter the primary key values as shown.       */
/*      To work around this, we will use the T-SQL command IDENTITY_INSERT.    */
/*                                                                             */
/*      When IDENTITY_INSERT is set to OFF (the default), only the DBMS        */
/*      can enter data into the controlled column in the table.                */
/*                                                                             */
/*      When IDENTITY_INSERT is set to ON, values can be input into the        */
/*      controlled column in the table. However, IDENTITY_INSERT can only be set */
/*      to ON for only one table at a time. Further, IDENTITY_INSERT requires  */
/*      the use of a column list containing the name of the surrogate key in each */
/*      INSERT command.                                                        */
/*                                                                             */
/*******************************************************************************/

USE VRG
GO

/*      Be sure IDENTITY_INSERT is OFF for all tables.                         */

SET IDENTITY_INSERT dbo.CUSTOMER OFF
SET IDENTITY_INSERT dbo.ARTIST OFF
SET IDENTITY_INSERT dbo.WORK OFF
SET IDENTITY_INSERT dbo.TRANS OFF


/*******************************************************************************/

/*      INSERT data for CUSTOMER                                               */

/*      Set INDENTITY_INSERT to ON for CUSTOMER;                               */
/*      reset it to OFF after CUSTOMER data is inserted.                       */

SET IDENTITY_INSERT dbo.CUSTOMER ON

INSERT INTO CUSTOMER
       (CustomerID, LastName, FirstName, EmailAddress, EncryptedPassword,
        Street, City, [State], ZIPorPostalCode, Country, AreaCode, PhoneNumber)
       VALUES (
       1000, 'Janes', 'Jeffrey' , 'Jeffrey.Janes@somewhere.com' , 'nh98tr3m',
       '123 W. Elm St', 'Renton', 'WA', '98055', 'USA', '425', '543-2345');
```

FIGURE 10A-39

The SQL Statements to Populate the VRG Database Tables

```
INSERT INTO CUSTOMER
       (CustomerID, LastName, FirstName, EmailAddress, EncryptedPassword,
        Street, City, [State], ZIPorPostalCode, Country, AreaCode, PhoneNumber)
       VALUES (
       1001, 'Smith', 'David', 'David.Smith@somewhere.com', 'ty7r932x',
       '813 Tumbleweed Lane', 'Loveland', 'CO', '81201', 'USA', '970', '654-9876');

INSERT INTO CUSTOMER
       (CustomerID, LastName, FirstName, EmailAddress,
        Street, City, [State], ZIPorPostalCode, Country, AreaCode, PhoneNumber)
       VALUES (
       1015, 'Twilight', 'Tiffany' , 'Tiffany.Twilight@somewhere.com' ,
       '88 1st Avenue', 'Langley', 'WA', '98260', 'USA', '360', '765-5566');

INSERT INTO CUSTOMER
       (CustomerID, LastName, FirstName, EmailAddress,
        Street, City, [State], ZIPorPostalCode, Country, AreaCode, PhoneNumber)
       VALUES (
       1033, 'Smathers', 'Fred', 'Fred.Smathers@somewhere.com',
       '10899 88th Ave', 'Bainbridge Island', 'WA', '98110', 'USA', '206', '876-9911');

INSERT INTO CUSTOMER
       (CustomerID, LastName, FirstName, EmailAddress, EncryptedPassword,
        Street, City, [State], ZIPorPostalCode, Country, AreaCode, PhoneNumber)
       VALUES (
       1034, 'Frederickson', 'Mary Beth', 'MaryBeth.Frederickson@somewhere.com', 'xc4vgh87',
       '25 South Lafayette', 'Denver', 'CO', '80201', 'USA', '303', '513-8822');

INSERT INTO CUSTOMER
       (CustomerID, LastName, FirstName, EmailAddress, EncryptedPassword,
        Street, City, [State], ZIPorPostalCode, Country, AreaCode, PhoneNumber)
       VALUES (
       1036, 'Warning', 'Selma', 'Selma.Warning@somewhere.com', 'ca45b32c',
       '205 Burnaby', 'Vancouver', 'BC', 'V6Z 1W2', 'Canada', '604', '988-0512');

INSERT INTO CUSTOMER
       (CustomerID, LastName, FirstName, EmailAddress,
        Street, City, [State], ZIPorPostalCode, Country, AreaCode, PhoneNumber)
       VALUES (
       1037, 'Wu', 'Susan', 'Susan.Wu@somewhere.com',
       '105 Locust Ave', 'Atlanta', 'GA', '30322', 'USA', '404', '653-3465');

INSERT INTO CUSTOMER
       (CustomerID, LastName, FirstName, EmailAddress, EncryptedPassword,
        Street, City, [State], ZIPorPostalCode, Country, AreaCode, PhoneNumber)
       VALUES (
       1040, 'Gray', 'Donald','Donald.Gray@somewhere.com', '98zx3y6',
       '55 Bodega Ave', 'Bodega Bay', 'CA', '94923', 'USA', '707', '568-4839');

INSERT INTO CUSTOMER
       (CustomerID, LastName, FirstName,
        Street, City, [State], ZIPorPostalCode, Country, AreaCode, PhoneNumber)
       VALUES (
       1041, 'Johnson', 'Lynda',
       '117 C Street', 'Washington', 'DC', '20003', 'USA', '202', '438-5498');

INSERT INTO CUSTOMER
       (CustomerID, LastName, FirstName, EmailAddress, EncryptedPassword,
        Street, City, [State], ZIPorPostalCode, Country, AreaCode, PhoneNumber)
       VALUES (
       1051, 'Wilkens', 'Chris', 'Chris.Wilkens@somewhere.com', 'wqpb3yyu',
       '87 Highland Drive', 'Olympia', 'WA', '98508', 'USA', '360', '876-8822');
```

FIGURE 10A-39

Continued

```
SET IDENTITY_INSERT dbo.CUSTOMER OFF

/*****************************************************************************/

/*     INSERT data for ARTIST                                               */

/*     Set INDENTITY_INSERT to ON for ARTIST;                               */
/*     reset it to OFF after ARTIST data is inserted.                       */

SET IDENTITY_INSERT dbo.ARTIST ON

INSERT INTO ARTIST
       (ArtistID, LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
       VALUES (1, 'Miro', 'Joan', 'Spanish', 1893, 1983);
INSERT INTO ARTIST
       (ArtistID, LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
       VALUES (2, 'Kandinsky', 'Wassily', 'Russian', 1866, 1944);
INSERT INTO ARTIST
       (ArtistID, LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
       VALUES (3, 'Klee', 'Paul', 'German', 1879, 1940);
INSERT INTO ARTIST
       (ArtistID, LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
       VALUES (4, 'Matisse', 'Henri', 'French', 1869, 1954);
INSERT INTO ARTIST
       (ArtistID, LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
       VALUES (5, 'Chagall', 'Marc', 'French', 1887, 1985);
INSERT INTO ARTIST
       (ArtistID, LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
       VALUES (11, 'Sargent', 'John Singer', 'United States', 1856, 1925);
INSERT INTO ARTIST
       (ArtistID, LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
       VALUES (17, 'Tobey', 'Mark', 'United States', 1890, 1976);
INSERT INTO ARTIST
       (ArtistID, LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
       VALUES (18, 'Horiuchi', 'Paul', 'United States', 1906, 1999);
INSERT INTO ARTIST
       (ArtistID, LastName, FirstName, Nationality, DateOfBirth, DateDeceased)
       VALUES (19, 'Graves', 'Morris', 'United States', 1920, 2001);

SET IDENTITY_INSERT dbo.ARTIST OFF

/*****************************************************************************/

/*     INSERT data for CUSTOMER_ARTIST_INT                                  */

/*     INDENTITY_INSERT OFF is NOT needed for CUSTOMER_ARTIST_INT -         */
/*     There are NO surrogate keys in CUSTOMER_ARTIST_INT.                  */


INSERT INTO CUSTOMER_ARTIST_INT VALUES (1, 1001);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (1, 1034);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (2, 1001);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (2, 1034);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (4, 1001);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (4, 1034);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (5, 1001);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (5, 1034);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (5, 1036);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (11, 1001);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (11, 1015);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (11, 1036);
```

FIGURE 10A-39

Continued

```
INSERT INTO CUSTOMER_ARTIST_INT VALUES (17, 1000);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (17, 1015);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (17, 1033);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (17, 1040);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (17, 1051);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (18, 1000);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (18, 1015);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (18, 1033);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (18, 1040);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (18, 1051);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (19, 1000);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (19, 1015);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (19, 1033);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (19, 1036);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (19, 1040);
INSERT INTO CUSTOMER_ARTIST_INT VALUES (19, 1051);

/*****************************************************************************/

/*      INSERT data for WORK                                                  */

/*      Set INDENTITY_INSERT to ON for WORK;                                  */
/*      reset it to OFF after WORK data is inserted.                          */

SET IDENTITY_INSERT dbo.WORK ON

INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       500, 'Memories IV', 'Unique', 'Casein rice paper collage',
       '31 x 24.8 in.', 18);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       511, 'Surf and Bird', '142/500', 'High Quality Limited Print',
       'Northwest School Expressionist style', 19);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       521, 'The Tilled Field', '788/1000', 'High Quality Limited Print',
       'Early Surrealist style', 1);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       522, 'La Lecon de Ski', '353/500', 'High Quality Limited Print',
       'Surrealist style', 1);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       523, 'On White II', '435/500', 'High Quality Limited Print',
       'Bauhaus style of Kandinsky', 2);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       524, 'Woman with a Hat', '596/750', 'High Quality Limited Print',
       'A very colorful Impressionist piece', 4);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       537, 'The Woven World', '17/750', 'Color lithograph', 'Signed', 17);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       548, 'Night Bird', 'Unique', 'Watercolor on Paper',
       '50 x 72.5 cm. - Signed', 19);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       551, 'Der Blaue Reiter', '236/1000', 'High Quality Limited Print',
       'The Blue Rider-Early Pointilism influence' , 2);
```

FIGURE 10A-39

Continued

```
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       552, 'Angelus Novus', '659/750', 'High Quality Limited Print',
       'Bauhaus style of Klee', 3);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       553, 'The Dance', '734/1000', 'High Quality Limited Print',
       'An Impressionist masterpiece', 4);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       554, 'I and the Village', '834/1000', 'High Quality Limited Print',
       'Shows Belarusian folk-life themes and symbology', 5);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       555, 'Claude Monet Painting', '684/1000', 'High Quality Limited Print',
       'Shows French Impressionist influence of Monet' , 11);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       561, 'Sunflower' , 'Unique', 'Watercolor and ink','33.3 x 16.1 cm. - Signed', 19);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       562, 'The Fiddler', '251/1000', 'High Quality Limited Print',
       'Shows Belarusian folk-life themes and symbology', 5);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       563, 'Spanish Dancer', '583/750', 'High Quality Limited Print',
       'American realist style - From work in Spain', 11);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       564, 'Farmer''s Market #2',   '267/500', 'High Quality Limited Print',
       'Northwest School Abstract Expressionist style', 17);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       565, 'Farmer''s Market #2',   '268/500', 'High Quality Limited Print',
       'Northwest School Abstract Expressionist style', 17);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       566, 'Into Time', '323/500', 'High Quality Limited Print',
       'Northwest School Abstract Expressionist style', 18);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       570, 'Untitled Number 1', 'Unique', 'Monotype with tempera',
       '4.3 x 6.1 in. Signed', 17);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       571, 'Yellow Covers Blue', 'Unique', 'Oil and collage','71 x 78 in. - Signed', 18);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       578, 'Mid-Century Hibernation', '362/500', 'High Quality Limited Print',
       'Northwest School Expressionist style', 19);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       580, 'Forms in Progress I', 'Unique', 'Color aquatint','19.3 x 24.4 in. - Signed', 17);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       581, 'Forms in Progress II', 'Unique', 'Color aquatint',
       '19.3 x 24.4 in. - Signed', 17);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       585, 'The Fiddler', '252/1000', 'High Quality Limited Print',
       'Shows Belarusian folk-life themes and symbology', 5);
```

FIGURE 10A-39

Continued

```
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       586, 'Spanish Dancer', '588/750', 'High Quality Limited Print',
       'American Realist style - From work in Spain', 11);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       587, 'Broadway Boggie', '433/500', 'High Quality Limited Print',
       'Northwest School Abstract Expressionist style', 17);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       588, 'Universal Field', '114/500', 'High Quality Limited Print',
       'Northwest School Abstract Expressionist style', 17);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       589, 'Color Floating in Time', '487/500', 'High Quality Limited Print',
       'Northwest School Abstract Expressionist style', 18);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       590, 'Blue Interior', 'Unique', 'Tempera on card', '43.9 x 28 in.', 17);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       593, 'Surf and Bird', 'Unique', 'Gouache', '26.5 x 29.75 in. - Signed', 19);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       594, 'Surf and Bird', '362/500', 'High Quality Limited Print',
       'Northwest School Expressionist style', 19);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       595, 'Surf and Bird', '365/500', 'High Quality Limited Print',
       'Northwest School Expressionist style', 19);
INSERT INTO WORK (WorkID, Title, [Copy], Medium, [Description], ArtistID)
       VALUES (
       596, 'Surf and Bird', '366/500', 'High Quality Limited Print',
       'Northwest School Expressionist style', 19);

SET IDENTITY_INSERT dbo.WORK OFF

/*****************************************************************************/

/*     INSERT data for TRANS                                                 */

/*     Set INDENTITY_INSERT to ON for TRANS;                                 */
/*     reset it to OFF after TRANS data is inserted.                         */

SET IDENTITY_INSERT dbo.TRANS ON

INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       100, '11/4/2014', 30000.00, 45000.00, '12/14/2014', 42500.00, 1000, 500);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       101, '11/7/2014', 250.00, 500.00, '12/19/2014', 500.00,  1015, 511);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       102, '11/17/2014', 125.00, 250.00, '01/18/2015', 200.00, 1001, 521);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       103, '11/17/2014', 250.00, 500.00, '12/12/2015', 400.00, 1034, 522);
```

FIGURE 10A-39

Continued

```
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        104, '11/17/2014', 250.00, 250.00, '01/18/2015', 200.00, 1001, 523);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        105, '11/17/2014', 200.00, 500.00, '12/12/2015', 400.00, 1034, 524);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        115, '03/03/2015', 1500.00, 3000.00, '06/07/2015', 2750.00, 1033, 537);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        121, '09/21/2015', 15000.00, 30000.00, '11/28/2015', 27500.00,  1015, 548);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        125, '11/21/2015', 125.00, 250.00, '12/18/2015', 200.00, 1001, 551);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, WorkID)
        VALUES (
        126, '11/21/2015', 200.00, 400.00, 552);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        127, '11/21/2015', 125.00, 500.00, '12/22/2015', 400.00, 1034, 553);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        128, '11/21/2015', 125.00, 250.00, '03/16/2016', 225.00, 1036, 554);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        129, '11/21/2015', 125.00, 250.00, '03/16/2016', 225.00, 1036, 555);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        151, '05/7/2016', 10000.00, 20000.00, '06/28/2016', 17500.00, 1036, 561);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        152, '05/18/2016', 125.00, 250.00, '08/15/2016', 225.00, 1001, 562);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        153, '05/18/2016', 200.00, 400.00, '08/15/2016', 350.00, 1001, 563);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        154, '05/18/2016', 250.00, 500.00, '09/28/2016', 400.00, 1040, 564);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, WorkID)
        VALUES (
        155, '05/18/2016', 250.00, 500.00, 565);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        156, '05/18/2016', 250.00, 500.00, '09/27/2016', 400.00, 1040, 566);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
        AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
        VALUES (
        161, '06/28/2016', 7500.00, 15000.00, '09/29/2016', 13750.00, 1033, 570);
```

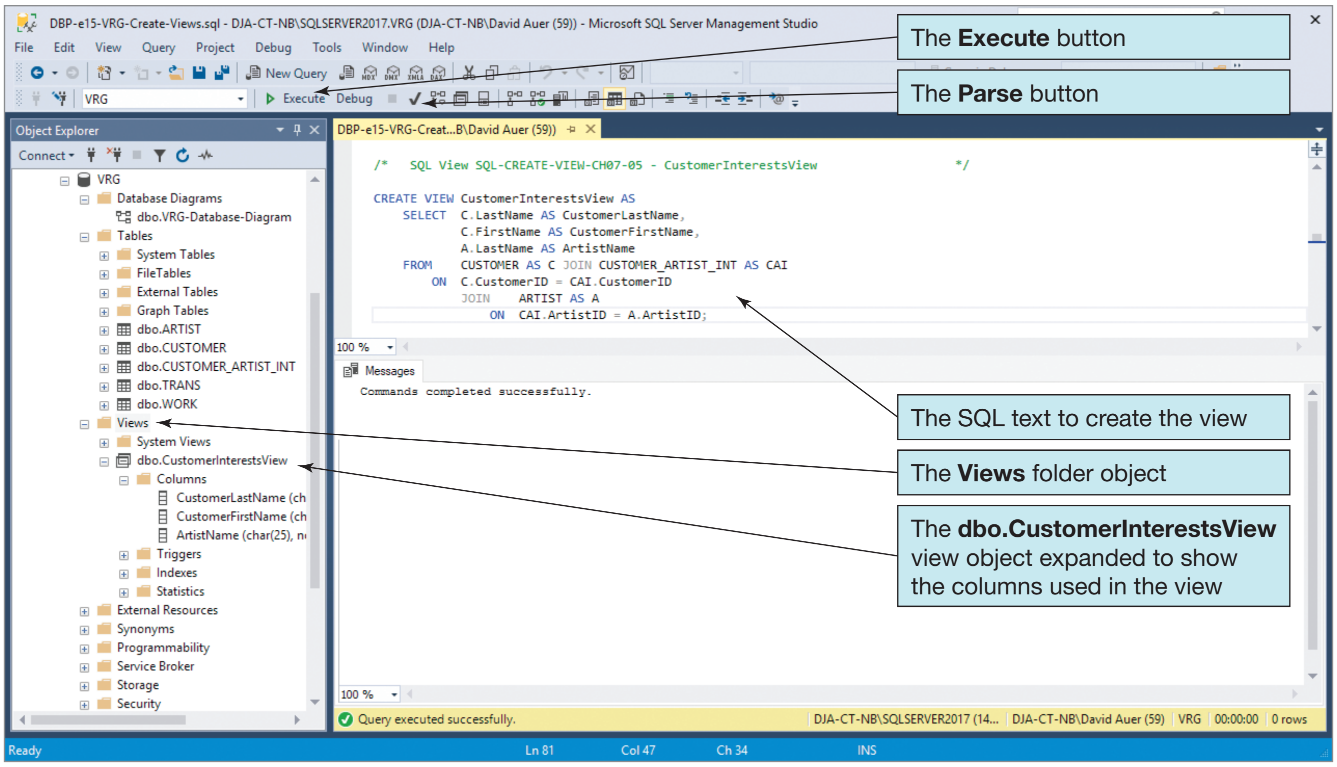
FIGURE 10A-39

Continued

```
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       171, '08/23/2016', 35000.00, 60000.00, '09/29/2016', 55000.00, 1000, 571);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       175, '09/29/2016', 40000.00, 75000.00, '12/18/2016', 72500.00, 1036, 500);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, WorkID)
       VALUES (
       181, '10/11/2016', 250.00, 500.00, 578);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       201, '02/28/2017', 2000.00, 3500.00, '04/26/2017', 3250.00, 1040, 580);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       202, '02/28/2017', 2000.00, 3500.00, '04/26/2017', 3250.00, 1040, 581);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       225, '06/8/2017', 125.00, 250.00, '09/27/2017', 225.00, 1051, 585);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, WorkID)
       VALUES (
       226, '06/8/2017', 200.00, 400.00, 586);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       227, '06/8/2017', 250.00, 500.00, '09/27/2017', 475.00, 1051, 587);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, WorkID)
       VALUES (
       228, '06/8/2017', 250.00, 500.00, 588);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, WorkID)
       VALUES (
       229, '06/8/2017', 250.00, 500.00, 589);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, DateSold, SalesPrice, CustomerID, WorkID)
       VALUES (
       241, '08/29/2017', 2500.00, 5000.00, '09/27/2017', 4750.00, 1015, 590);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, WorkID)
       VALUES (
       251, '10/25/2017', 25000.00, 50000.00, 593);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, WorkID)
       VALUES (
       252, '10/27/2017', 250.00, 500.00, 594);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, WorkID)
       VALUES (
       253, '10/27/2017', 250.00, 500.00, 595);
INSERT INTO TRANS (TransactionID, DateAcquired, AcquisitionPrice,
       AskingPrice, WorkID)
       VALUES (
       254, '10/27/2017', 250.00, 500.00, 596);

SET IDENTITY_INSERT dbo.TRANS OFF

/*********************************************************************************/
```

FIGURE 10A-39

Continued


The Execute button
The Parse button
The SQL text to create the view
The Views folder object
The dbo.CustomerInterestsView view object expanded to show the columns used in the view

```
/*   SQL View SQL-CREATE-VIEW-CH07-05 - CustomerInterestsView              */

CREATE VIEW CustomerInterestsView AS
    SELECT  C.LastName AS CustomerLastName,
            C.FirstName AS CustomerFirstName,
            A.LastName AS ArtistName
    FROM    CUSTOMER AS C JOIN CUSTOMER_ARTIST_INT AS CAI
        ON  C.CustomerID = CAI.CustomerID
            JOIN    ARTIST AS A
                ON  CAI.ArtistID = A.ArtistID;
```

```
Commands completed successfully.
```

FIGURE 10A-40

Creating an SQL View

This numbering corresponds to the numbering we used in Chapter 7 and is used here for continuity and easy reference between this chapter and Chapter 7. Figure 10A-40 shows this SQL CREATE VIEW statement in an SQL script named *DBP-e15-VRG-Create-Views.sql* in the Microsoft SQL Server Management Studio. We will step through creating and testing this view.

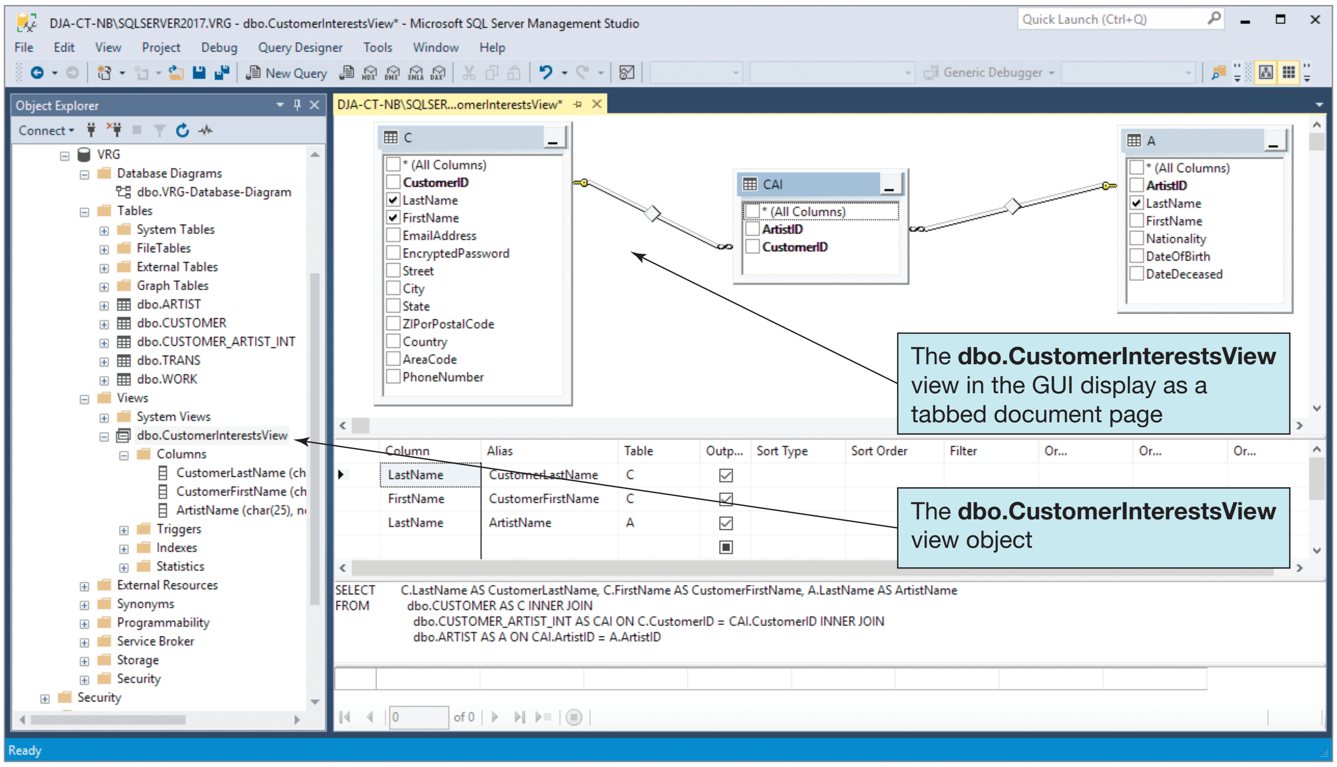
Creating a New View

1. In the Microsoft SQL Server Management Studio Object Explorer, click the **New Query** button to open a new tabbed SQL document window.
2. Click the **Intellisense Enabled** button to *disable* the Intellisense feature.
3. Type the SQL statements for **SQL-CREATE-VIEW-CH07-05—Customer InterestsView** on page 10A-73.
4. Click the **Parse** button. If any SQL coding errors are detected, fix them.
5. Click the **Execute** button.
6. Expand the **Views** folder in the VRG database object. Note that the dbo .CustomerInterestsView object has been added to the VRG database, as shown in Figure 10A-40.
7. To save this CREATE VIEW statement as part of an SQL script, add the comments shown in Figure 10A-40 and then save the script as **DBP-e15-VRG-Create-Views.sql**. You may need to click in the SQL View text to enable the Save command.
8. Click the document window **Close** button to close the window.

We can now look at the view in a GUI display.

Viewing an Existing View in the SQL Server GUI Display

1. Right-click the **dbo.CustomerInterestsView** object in the Object Browser to display a shortcut menu.
2. Click the **Design** command on the shortcut menu. The dbo.CustomerInterests-View is displayed in a new tabbed document window in the GUI display format.
3. Rearrange the components of the view in the GUI display so it appears as shown in Figure 10A-41. You may need to adjust the size of the panes.


DJA-CT-NB\SQLSERVER2017.VRG - dbo.CustomerInterestsView* - Microsoft SQL Server Management Studio
File Edit View Project Debug Query Designer Tools Window Help
Object Explorer
VRG
Database Diagrams
Tables
dbo.ARTIST
dbo.CUSTOMER
dbo.CUSTOMER_ARTIST_INT
dbo.TRANS
dbo.WORK
Views
dbo.CustomerInterestsView
C
CAI
A
Ready
The **dbo.CustomerInterestsView** view in the GUI display as a tabbed document page

The **dbo.CustomerInterestsView** view object

Column	Alias	Table	Outp...	Sort Type	Sort Order	Filter	Or...	Or...	Or...
LastName	CustomerLastName	C	✓						
FirstName	CustomerFirstName	C	✓						
LastName	ArtistName	A	✓						

```
SELECT     C.LastName AS CustomerLastName, C.FirstName AS CustomerFirstName, A.LastName AS ArtistName
FROM        dbo.CUSTOMER AS C INNER JOIN
              dbo.CUSTOMER_ARTIST_INT AS CAI ON C.CustomerID = CAI.CustomerID INNER JOIN
              dbo.ARTIST AS A ON CAI.ArtistID = A.ArtistID
```

FIGURE 10A-41

Viewing an SQL View in the GUI Display

4. Click the **Save** button to save the reconfigured view GUI display.
5. Click the document window tab **Close** button to close the GUI display window.
6. Collapse the **Views** folder in the Object Browser.

As explained in Chapter 7, SQL views are used like tables in other SQL statements. For example, to see all the data in the view, we use the following SQL SELECT statement:

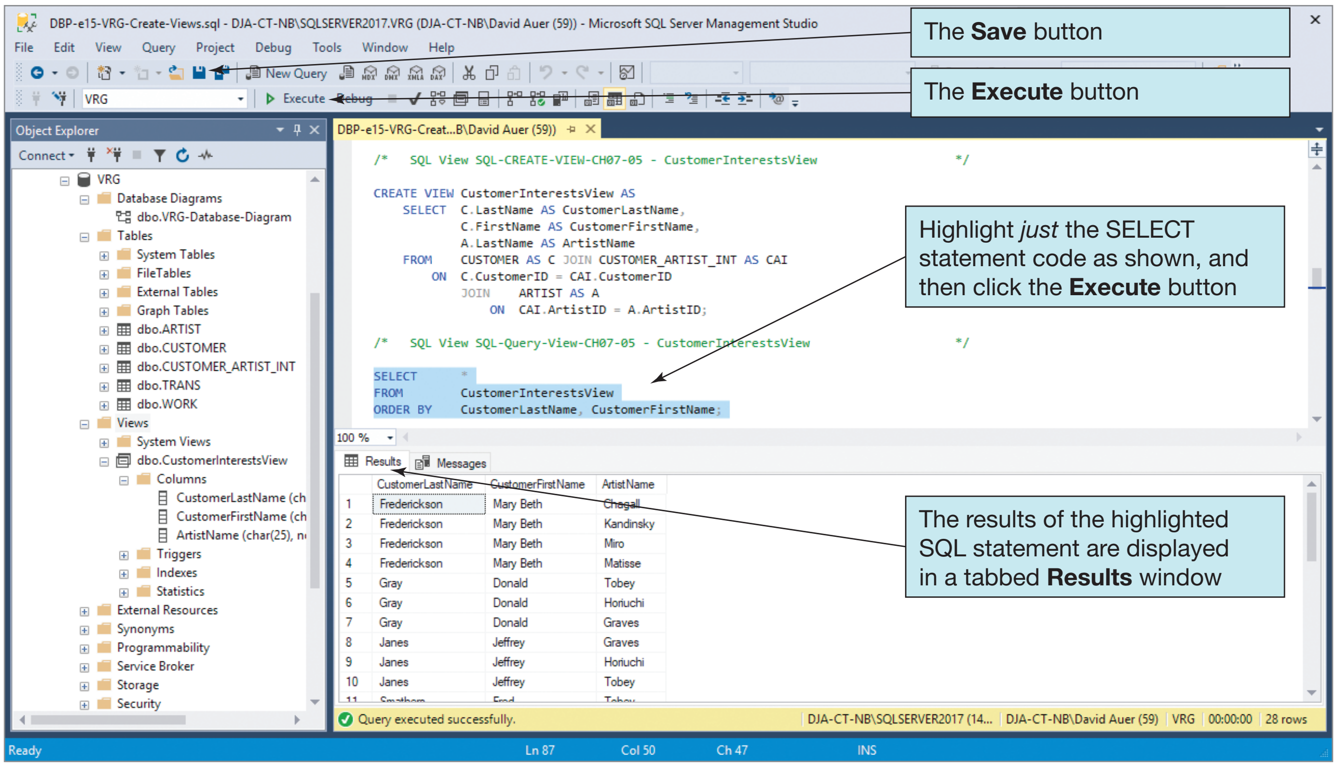
```
/* SQL View SQL-Query-View-CH07-05 - CustomerInterestsView */
SELECT       *
FROM         CustomerInterestsView
ORDER BY     CustomerLastName, CustomerFirstName;
```

Running a Single SQL Command in an SQL Script

1. Open your *DBP-e15-VRG-Create-Views.sql* SQL script file, disable Intellisense, and add the SQL code for SQL-Query-View-07-05 into.
2. Click the **Save** button to save the edited script file.
3. As shown in Figure 10A-42, highlight *just* the SQL-Query-View-07-05 SQL query statement.
4. Click the **Execute** button.
5. Note that *only* the SQL-Query-Views-CH07-05 SQL query statement is executed, not the entire script, and the results are displayed in the tabbed Results window as shown in Figure 10A-42.
6. Close the *DBP-e15-VRG-Create-Views.sql* SQL script file.

This is a handy technique to know—it allows you to store multiple SQL commands in a single SQL script and then run only the command or commands that you want to run. The result is shown in Figure 10A-43.

At this point, you should create and add all the SQL views and SQL view queries discussed in Chapter 7 to the *DBP-e15-VRG-Create-Views.sql* SQL script file. With these views created in the VRG database, they will be available for use in later sections of this chapter.

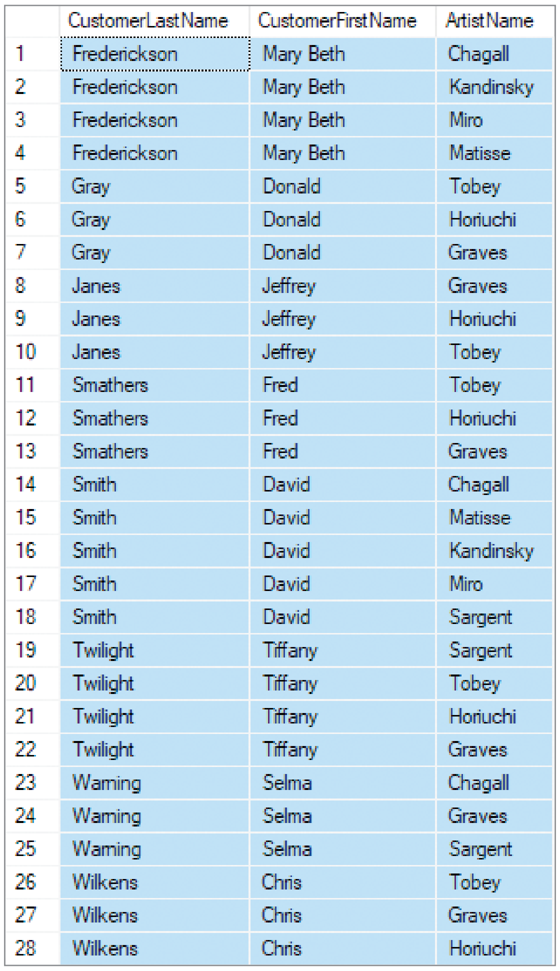

DBP-e15-VRG-Create-Views.sql - DJA-CT-NB\SQLSERVER2017.VRG (DJA-CT-NB\David Auer (59)) - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
Execute
VRG
The **Save** button
The **Execute** button
Object Explorer
Connect
VRG
Database Diagrams
dbo.VRG-Database-Diagram
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.ARTIST
dbo.CUSTOMER
dbo.CUSTOMER_ARTIST_INT
dbo.TRANS
dbo.WORK
Views
System Views
dbo.CustomerInterestsView
Columns
CustomerLastName (ch
CustomerFirstName (ch
ArtistName (char(25), n
Triggers
Indexes
Statistics
External Resources
Synonyms
Programmability
Service Broker
Storage
Security
DBP-e15-VRG-Creat...B\David Auer (59))
Highlight *just* the SELECT statement code as shown, and then click the **Execute** button
100 %
Results
Messages
The results of the highlighted SQL statement are displayed in a tabbed **Results** window
Query executed successfully.
DJA-CT-NB\SQLSERVER2017 (14... DJA-CT-NB\David Auer (59) VRG 00:00:00 28 rows
Ready Ln 87 Col 50 Ch 47 INS

```
/*   SQL View SQL-CREATE-VIEW-CH07-05 - CustomerInterestsView            */

CREATE VIEW CustomerInterestsView AS
    SELECT  C.LastName AS CustomerLastName,
            C.FirstName AS CustomerFirstName,
            A.LastName AS ArtistName
    FROM    CUSTOMER AS C JOIN CUSTOMER_ARTIST_INT AS CAI
        ON  C.CustomerID = CAI.CustomerID
            JOIN    ARTIST AS A
                ON  CAI.ArtistID = A.ArtistID;

/*   SQL View SQL-Query-View-CH07-05 - CustomerInterestsView             */

SELECT      *
FROM        CustomerInterestsView
ORDER BY    CustomerLastName, CustomerFirstName;
```

	CustomerLastName	CustomerFirstName	ArtistName
1	Frederickson	Mary Beth	Chagall
2	Frederickson	Mary Beth	Kandinsky
3	Frederickson	Mary Beth	Miro
4	Frederickson	Mary Beth	Matisse
5	Gray	Donald	Tobey
6	Gray	Donald	Horiuchi
7	Gray	Donald	Graves
8	Janes	Jeffrey	Graves
9	Janes	Jeffrey	Horiuchi
10	Janes	Jeffrey	Tobey

FIGURE 10A-42

Running a Selected SQL Command in an SQL Script



	CustomerLastName	CustomerFirstName	ArtistName
1	Frederickson	Mary Beth	Chagall
2	Frederickson	Mary Beth	Kandinsky
3	Frederickson	Mary Beth	Miro
4	Frederickson	Mary Beth	Matisse
5	Gray	Donald	Tobey
6	Gray	Donald	Horiuchi
7	Gray	Donald	Graves
8	Janes	Jeffrey	Graves
9	Janes	Jeffrey	Horiuchi
10	Janes	Jeffrey	Tobey
11	Smathers	Fred	Tobey
12	Smathers	Fred	Horiuchi
13	Smathers	Fred	Graves
14	Smith	David	Chagall
15	Smith	David	Matisse
16	Smith	David	Kandinsky
17	Smith	David	Miro
18	Smith	David	Sargent
19	Twilight	Tiffany	Sargent
20	Twilight	Tiffany	Tobey
21	Twilight	Tiffany	Horiuchi
22	Twilight	Tiffany	Graves
23	Warning	Selma	Chagall
24	Warning	Selma	Graves
25	Warning	Selma	Sargent
26	Wilkens	Chris	Tobey
27	Wilkens	Chris	Graves
28	Wilkens	Chris	Horiuchi

FIGURE 10A-43

Result of Using the View CustomerInterestsView

Importing Microsoft Excel Data into a Microsoft SQL Server Database Table

When developing a database to support an application, it is common to find that some (if not all) of the data needed in the database exists as data in user **worksheets** (also called **spreadsheets**). A typical example of this is a Microsoft Excel 2016 worksheet that a user has been maintaining and which must now be converted to data stored in the database.

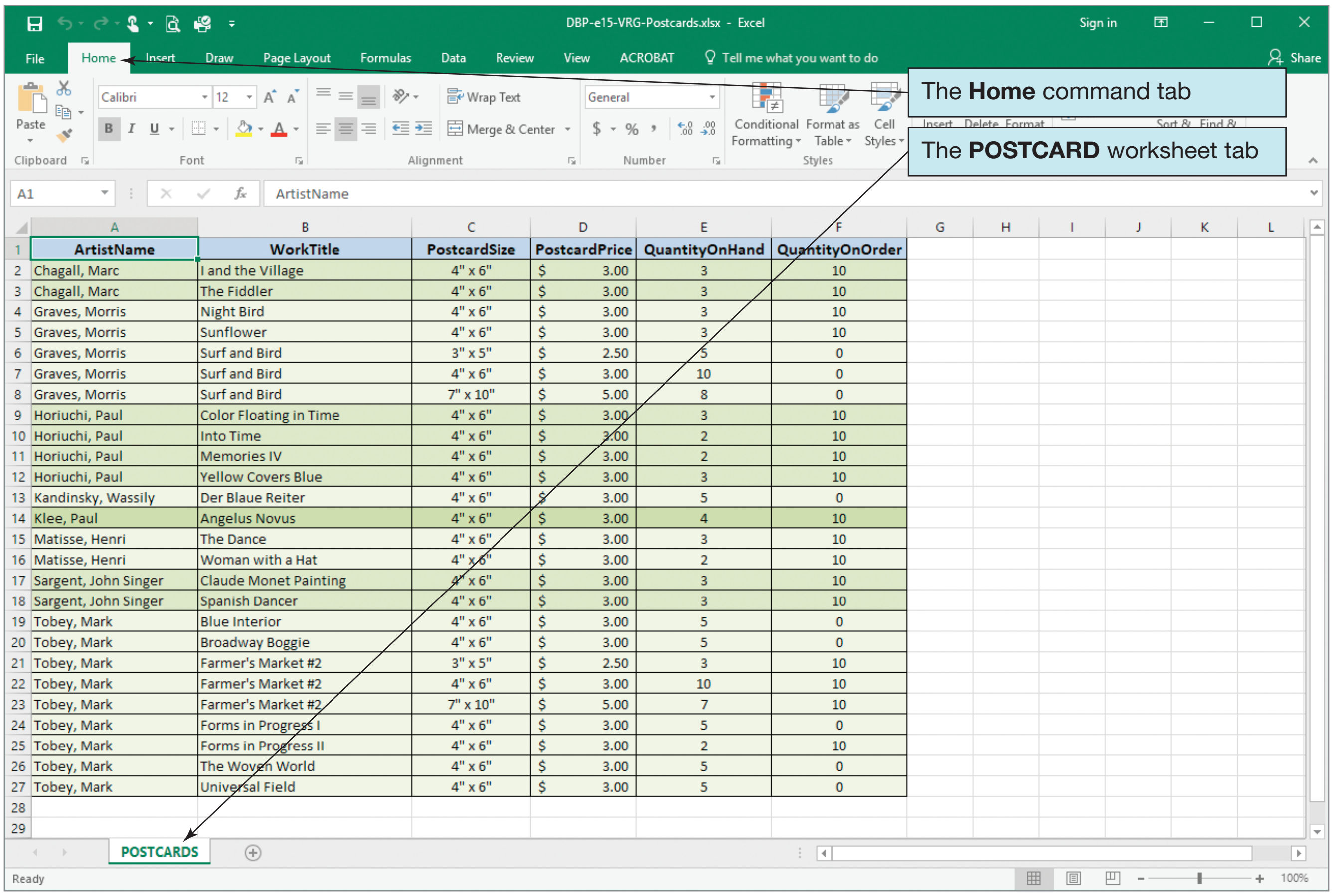
If we are really lucky, the worksheet will already be organized like a database table, with appropriate column labels and unique data in each row. And if we are *really, really lucky*, there will be one or more columns that can be used as the primary key in the new database table. In that case, we can easily import the data into the database. More likely, we will have to modify the worksheet and organize and clean up the data in it before we can import the data. In essence, we are following a procedure that we will encounter again in Chapter 12 in our discussion of data warehouses known as **extract, transform, and load (ETL)**.

As an example, let's consider the problem of postcards sold by the View Ridge Gallery. These postcards are pictures of the art work sold by the View Ridge Gallery (postcards of other popular works of art by well-known artists such as Claude Monet's *Water Lilies* are also stocked and sold, but to simplify the problem we will consider only postcards of art work at View Ridge Gallery). The inventory of postcards is currently kept in a Microsoft Excel 2016 worksheet named POSTCARDS and shown in Figure 10A-44(a).

We will need a primary key in our SQL Server table, and although we could possibly add this column in the SQL Server Import and Export Wizard, it will be easier to add that column to the Microsoft Excel 2016 worksheet. Therefore, we will make a copy of the POSTCARDS worksheet named POSTCARDSwithID and manually add a *PostcardID* identifier column, as shown in Figure 10A-44(b).

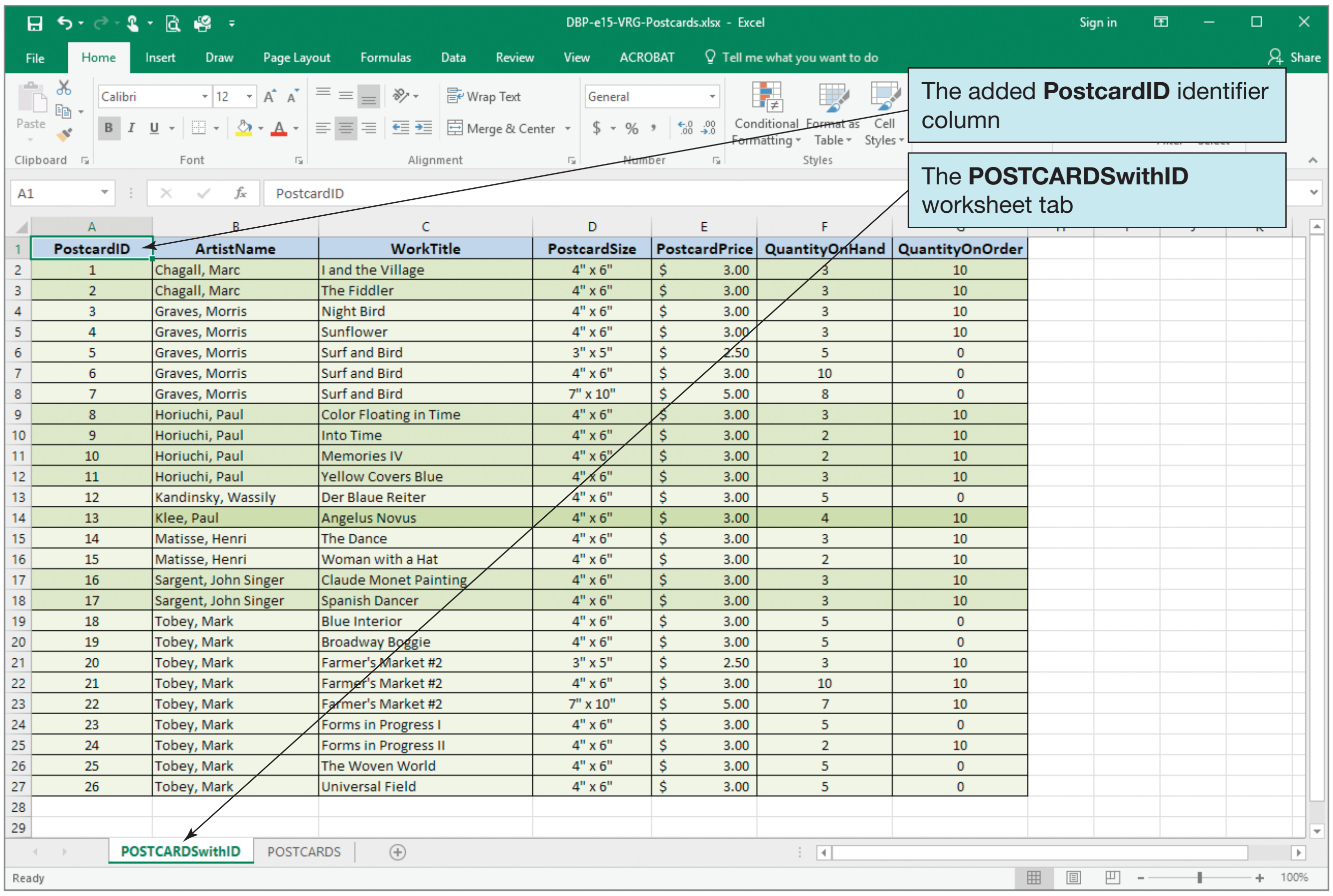
FIGURE 10A-44

The View Ridge Gallery POSTCARD Worksheet



	A	B	C	D	E	F
1	ArtistName	WorkTitle	PostcardSize	PostcardPrice	QuantityOnHand	QuantityOnOrder
2	Chagall, Marc	I and the Village	4" x 6"	$ 3.00	3	10
3	Chagall, Marc	The Fiddler	4" x 6"	$ 3.00	3	10
4	Graves, Morris	Night Bird	4" x 6"	$ 3.00	3	10
5	Graves, Morris	Sunflower	4" x 6"	$ 3.00	3	10
6	Graves, Morris	Surf and Bird	3" x 5"	$ 2.50	5	0
7	Graves, Morris	Surf and Bird	4" x 6"	$ 3.00	10	0
8	Graves, Morris	Surf and Bird	7" x 10"	$ 5.00	8	0
9	Horiuchi, Paul	Color Floating in Time	4" x 6"	$ 3.00	3	10
10	Horiuchi, Paul	Into Time	4" x 6"	$ 3.00	2	10
11	Horiuchi, Paul	Memories IV	4" x 6"	$ 3.00	2	10
12	Horiuchi, Paul	Yellow Covers Blue	4" x 6"	$ 3.00	3	10
13	Kandinsky, Wassily	Der Blaue Reiter	4" x 6"	$ 3.00	5	0
14	Klee, Paul	Angelus Novus	4" x 6"	$ 3.00	4	10
15	Matisse, Henri	The Dance	4" x 6"	$ 3.00	3	10
16	Matisse, Henri	Woman with a Hat	4" x 6"	$ 3.00	2	10
17	Sargent, John Singer	Claude Monet Painting	4" x 6"	$ 3.00	3	10
18	Sargent, John Singer	Spanish Dancer	4" x 6"	$ 3.00	3	10
19	Tobey, Mark	Blue Interior	4" x 6"	$ 3.00	5	0
20	Tobey, Mark	Broadway Boggie	4" x 6"	$ 3.00	5	0
21	Tobey, Mark	Farmer's Market #2	3" x 5"	$ 2.50	3	10
22	Tobey, Mark	Farmer's Market #2	4" x 6"	$ 3.00	10	10
23	Tobey, Mark	Farmer's Market #2	7" x 10"	$ 5.00	7	10
24	Tobey, Mark	Forms in Progress I	4" x 6"	$ 3.00	5	0
25	Tobey, Mark	Forms in Progress II	4" x 6"	$ 3.00	2	10
26	Tobey, Mark	The Woven World	4" x 6"	$ 3.00	5	0
27	Tobey, Mark	Universal Field	4" x 6"	$ 3.00	5	0
28						
29						

(a) The Original Worksheet



	A	B	C	D	E	F	G
1	PostcardID	ArtistName	WorkTitle	PostcardSize	PostcardPrice	QuantityOnHand	QuantityOnOrder
2	1	Chagall, Marc	I and the Village	4" x 6"	$ 3.00	3	10
3	2	Chagall, Marc	The Fiddler	4" x 6"	$ 3.00	3	10
4	3	Graves, Morris	Night Bird	4" x 6"	$ 3.00	3	10
5	4	Graves, Morris	Sunflower	4" x 6"	$ 3.00	3	10
6	5	Graves, Morris	Surf and Bird	3" x 5"	$ 2.50	5	0
7	6	Graves, Morris	Surf and Bird	4" x 6"	$ 3.00	10	0
8	7	Graves, Morris	Surf and Bird	7" x 10"	$ 5.00	8	0
9	8	Horiuchi, Paul	Color Floating in Time	4" x 6"	$ 3.00	3	10
10	9	Horiuchi, Paul	Into Time	4" x 6"	$ 3.00	2	10
11	10	Horiuchi, Paul	Memories IV	4" x 6"	$ 3.00	2	10
12	11	Horiuchi, Paul	Yellow Covers Blue	4" x 6"	$ 3.00	3	10
13	12	Kandinsky, Wassily	Der Blaue Reiter	4" x 6"	$ 3.00	5	0
14	13	Klee, Paul	Angelus Novus	4" x 6"	$ 3.00	4	10
15	14	Matisse, Henri	The Dance	4" x 6"	$ 3.00	3	10
16	15	Matisse, Henri	Woman with a Hat	4" x 6"	$ 3.00	2	10
17	16	Sargent, John Singer	Claude Monet Painting	4" x 6"	$ 3.00	3	10
18	17	Sargent, John Singer	Spanish Dancer	4" x 6"	$ 3.00	3	10
19	18	Tobey, Mark	Blue Interior	4" x 6"	$ 3.00	5	0
20	19	Tobey, Mark	Broadway Boggie	4" x 6"	$ 3.00	5	0
21	20	Tobey, Mark	Farmer's Market #2	3" x 5"	$ 2.50	3	10
22	21	Tobey, Mark	Farmer's Market #2	4" x 6"	$ 3.00	10	10
23	22	Tobey, Mark	Farmer's Market #2	7" x 10"	$ 5.00	7	10
24	23	Tobey, Mark	Forms in Progress I	4" x 6"	$ 3.00	5	0
25	24	Tobey, Mark	Forms in Progress II	4" x 6"	$ 3.00	2	10
26	25	Tobey, Mark	The Woven World	4" x 6"	$ 3.00	5	0
27	26	Tobey, Mark	Universal Field	4" x 6"	$ 3.00	5	0
28							
29							

(b) The Copied Worksheet with PostcardID Column

FIGURE 10A-44
Continued

Even with the added PostcardID column, this worksheet breaks our basic rule of one theme per table and combines artist, artwork, and inventory into the same worksheet. Fortunately, there is no multivalue, multicolumn problem (as discussed in Chapter 4) in this worksheet. Even as it stands, this worksheet would need normalizing into BCNF to create the proper set of tables for the VRG database.

However, we will deal with the normalization of this data in the VRG database itself in the Exercises at the end of this chapter. For now, we will simply import the worksheet into the VRG database and use it as a temporary table and source of data for populating the properly normalized tables.

Because Microsoft creates both Microsoft Excel 2016 and Microsoft SQL Server 2017, we would expect that importing data from Microsoft Excel into SQL Server would be simple and without problems. Unfortunately, in our experience, the **SQL Server Import and Export Wizard**, which is the tool we use for data import, has some glitches.[6]

First, while SQL Server 2017 is a 64-bit program, by default it launches a 32-bit version of the SQL Server Import and Export Wizard if you start the Wizard from within the SQL Server Management Studio. This is fine if you are running the 32-bit version of Microsoft Excel 2016, but it will not work if you are running the 64-bit version of Microsoft Excel. In that, case you need to manually launch the 64-bit version of the Wizard from the Windows menu, where the program can be found in the Microsoft SQL Server 2017 folder (You can pin a shortcut onto the toolbar for easier access).[7]

[6] For a good discussion of this problem, see the SQL Blog.com article SQL Server 2016 Import and Export Wizard and Excel by John Paul Cook at *http://sqlblog.com/blogs/john_paul_cook/archive/2017/06/11/sql-server-2016-import-and-export-wizard-and-excel.aspx*. Although this article discusses SQL Server 2016, the same problem and solution apply to SQL Server 2017.
[7] Discrepancies between the 32-bit and 64-bit versions of Microsoft Office 2016 are enough that Microsoft recommends that you install the 32-bit version. We have seen enough problems that we agree.

Second, apparently the Microsoft Access Database Engine (ADE) drivers needed to support later versions of Microsoft Excel in SQL Server Import and Export Wizard are not automatically installed. Even though we are using Microsoft Excel 2016, we have to download and install the Microsoft Access Database Engine 2016 Redistributable.[8] Install the 32-bit version if you are running the 32-bit version of Microsoft Excel 2016, and the 64-bit version if you are running the 64-bit version of Microsoft Excel—you will only be able to install the version compatible with the version of Microsoft Excel installed on you computer. If you don't install this software, you will get an error message during the wizard, and it will not complete its tasks.[9]

Third, the wizard does not handle data types or NULL/NOT NULL constraints smoothly. We cannot modify the wizard-detected data types or NULL/NOT NULL settings into the data types we want in the database—if we try, the wizard generates an error message and will not complete its tasks.

Fourth, the wizard does not allow a primary key to be set on the imported table and also imports a set of blank rows (all NULL values) in addition to the actual data (this is only possible because no primary key has been set).

Our solution is to:

- Use the SQL Server Import and Export Wizard to import the data into a temporary table as created by the wizard, then
- Use an SQL CREATE TABLE statement to create the actual table we want in the database, then
- Use an SQL INSERT statement to copy the data from the temporary table to the actual table, then
- Delete the temporary table from the database.

Note that in these steps we will use a new variant of the SQL INSERT statement, a **bulk INSERT statement**. We use this form of the SQL INSERT statement when we want to copy a lot of data from one table to another, and copying from a temporary table to a final table is a great place to use this statement.

Here are the actual steps:

Importing a Microsoft Excel 2016 Worksheet into an SQL Server Table

1. In the Microsoft SQL Server Management Studio, expand the **VRG** database.
2. **If you want to use the default 32-bit version of the SQL Server Import and Export Wizard**, right-click on the VRG database object to display a shortcut menu, and in the shortcut menu click on the **Tasks** command to display the Tasks menu. In the Tasks menu, click the **Import Data** command shown in Figure 10A-45 to launch the SQL Server Import and Export Wizard.
3. **If you want to use the 64-bit version of the SQL Server Import and Export Wizard,** browse C:\Program Files\Microsoft SQL Server\140\DTS\Binn\DTSWizard.exe. Right-click on the DTSWizard.exe file to display a shortcut menu, and in the shortcut menu click on the **Run as administrator** command to launch the SQL Server Import and Export Wizard. Or, as indicated above, you can launch it from the Microsoft SQL Server section of the Windows 10 App menu.
   - **NOTE:** You can also pin DTSWizard.exe to the Task bar, and launch it from there.
4. The SQL Server Import and Export Wizard is displayed with the *Welcome to SQL Server Import and Export Wizard* page appearing shown in Figure 10A-46. Click the **Next** button to display the Choose a Data Source page as shown in Figure 10A-47.

[8] Downloadable from *https://www.microsoft.com/en-us/download/details.aspx?id=54920*.

[9] This statement is true on Windows 10 Version 1703, Microsoft Office 2016, and Microsoft SQL Server 2017 Developer Edition with all updates and patches installed as of October 24, 2017. For support for Microsoft Excel 2013, download the Microsoft Access 2013 Runtime from *https://www.microsoft.com/en-us/download/details.aspx?id=39358*. For support for Microsoft Excel 2010, download the Microsoft Access Engine 2010 Redistributable from *https://www.microsoft.com/en-us/download/details.aspx?id=13255*.

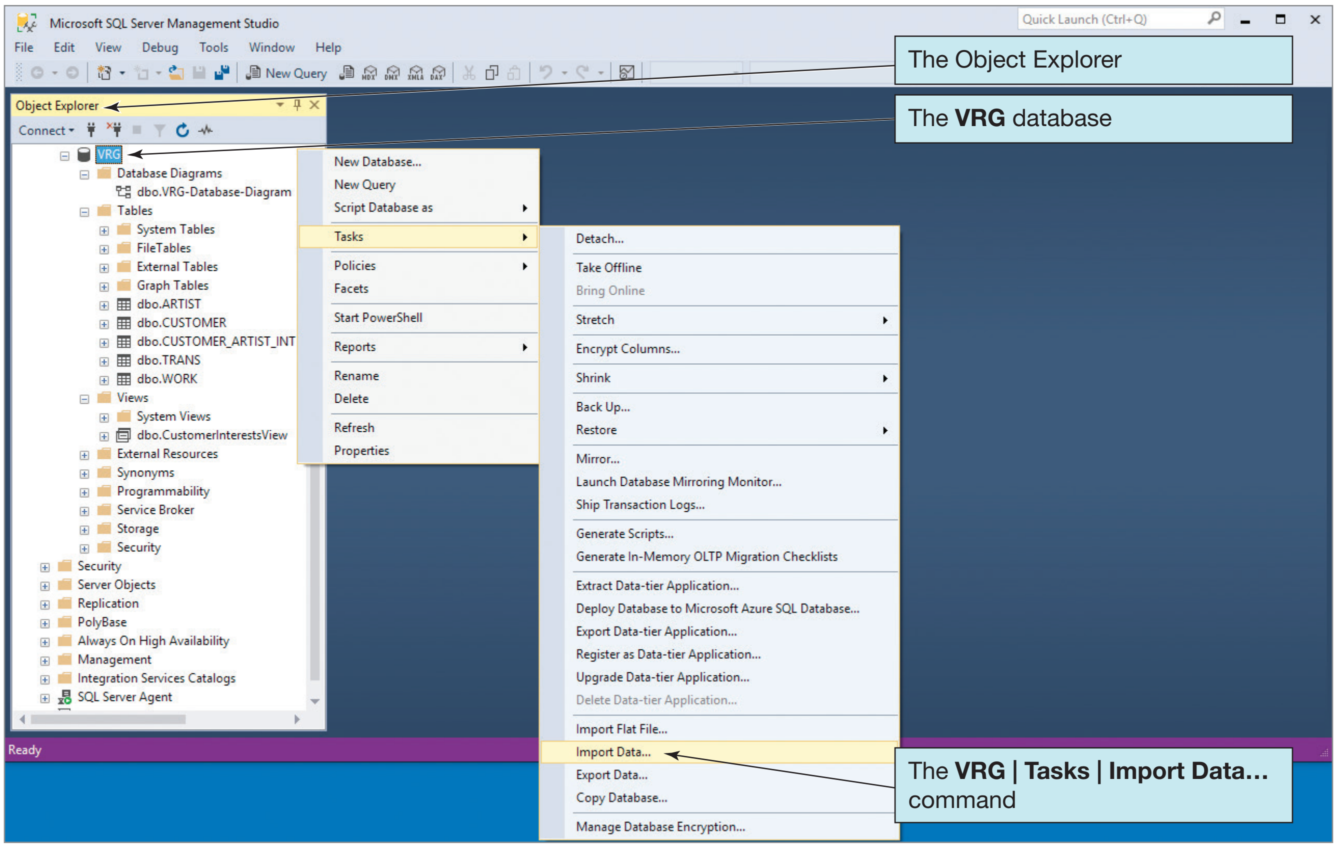


FIGURE 10A-45

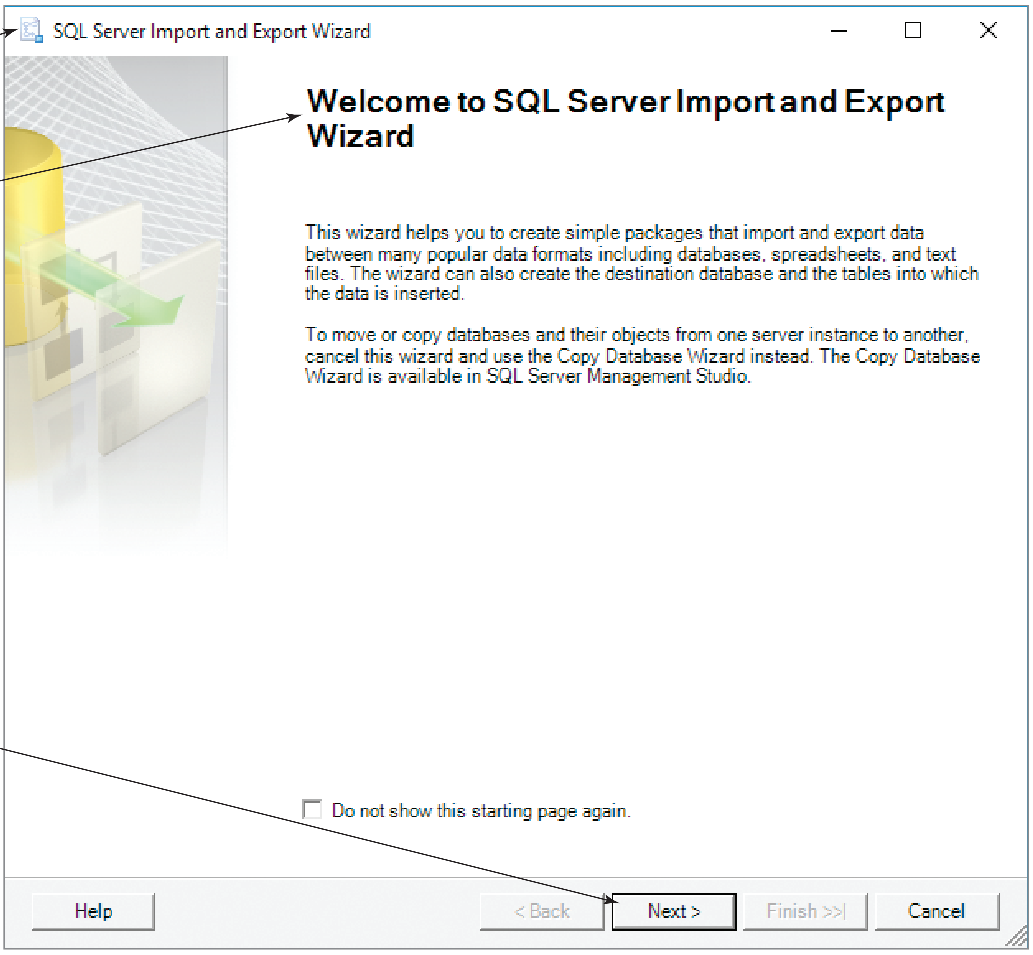
Launching the Microsoft SQL Server Import and Export Wizard

5. On the *Choose a Data Source* page shown in Figure 10A-47, select **Microsoft Excel** as the data source.
6. On the *Choose a Data Source* page shown in Figure 10A-47, select browse to go to the location of the Microsoft Excel file, select **Microsoft Excel 2016** in Excel version drop-down list, and make sure the check box for **First row has column names** is checked, as shown in Figure 10A-47.
7. Click the **Next** button to display the *Choose a Destination* page as shown in Figure 10A-48, and select **SQL Server Native Client 11.0** as the destination. Select the *{ComputerName\{SQLServerInstanceName}* from the Server name drop-down list, and the database name from the Database drop-down list.
8. Click the **Next** button to display the *Specify Table Copy or Query* page as shown in Figure 10A-49. Make sure the **Copy data from one or more tables or views** radio button is selected.
9. Click the **Next** button to display the *Select Source Tables and Views* page as shown in Figure 10A-50, and check the **'POSTCARDSwithID$'** check box in the *Source* column. The table name *[dbo].[POSTCARDSwithID$]* is generated and displayed in the *Destination* column. This is the name we will use for the temporary table in the VRG database.
10. Click the **Edit Mappings** button to display the *Column Mappings* dialog box shown in Figure 10A-51. This dialog box shows the column names, data types, and NULL/NOT NULL settings that will be used to create the POSTCARDS$ table during the import.
    - **Note:** We should be able to edit these values, but if we do, we are likely to generate errors during the import process. Therefore, we leave them alone and leave the temporary POSTCARDSwithID$ table as created by the wizard.
    - **Note:** You may want to try some other imports where you do edit these values in order to understand what you can and cannot successfully edit. When in doubt, leave it alone!

The **SQL Server Import and Export** Wizard dialog box

The **Welcome to SQL Server Import and Export Wizard** page

The **Next** button


SQL Server Import and Export Wizard

Welcome to SQL Server Import and Export Wizard

This wizard helps you to create simple packages that import and export data between many popular data formats including databases, spreadsheets, and text files. The wizard can also create the destination database and the tables into which the data is inserted.

To move or copy databases and their objects from one server instance to another, cancel this wizard and use the Copy Database Wizard instead. The Copy Database Wizard is available in SQL Server Management Studio.

☐ Do not show this starting page again.

FIGURE 10A-46

The Microsoft SQL Server Import and Export Wizard

The **Choose a Data Source** page

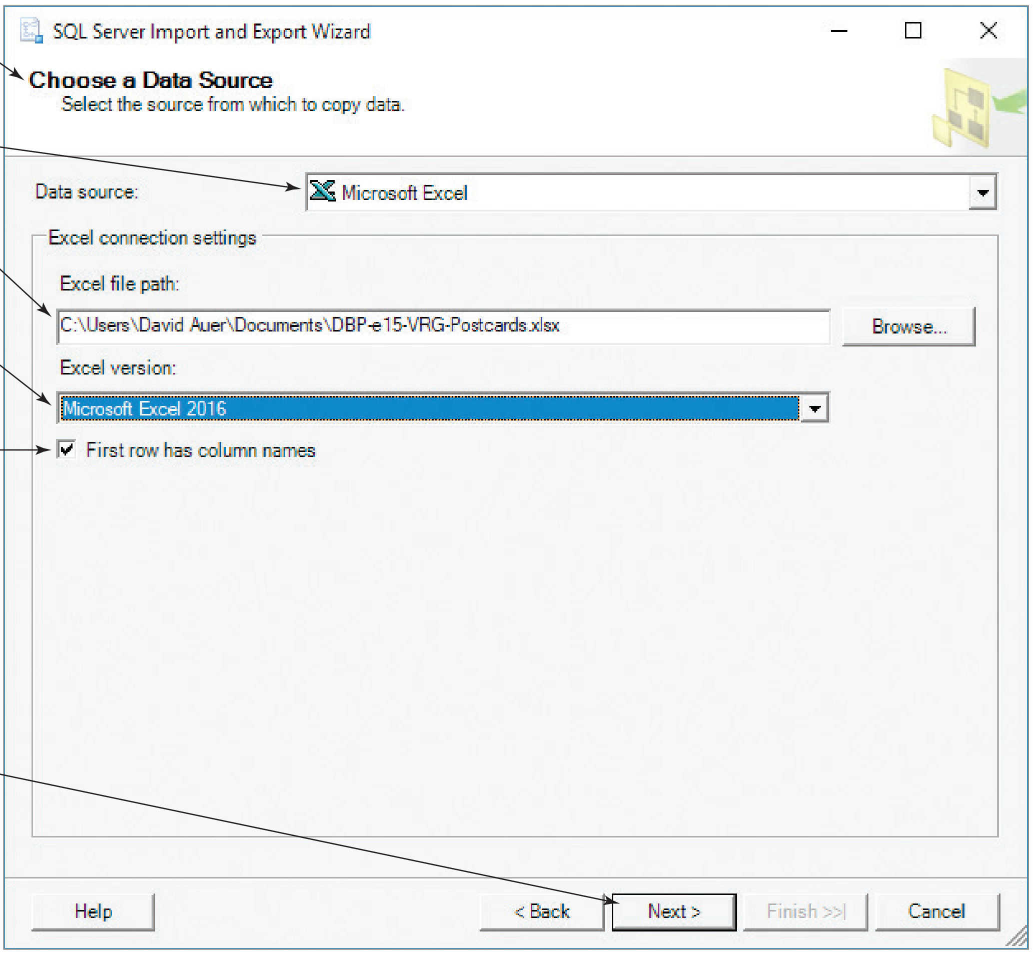
Select **Microsoft Excel** from the drop-down list

Browse to the Excel file

Select **Microsoft Excel 2016** from the drop-down list

Check the **First row has column names** check box

The **Next** button


SQL Server Import and Export Wizard

Choose a Data Source

Select the source from which to copy data.

Data source: Microsoft Excel

Excel connection settings

Excel file path: C:\Users\David Auer\Documents\DBP-e15-VRG-Postcards.xlsx

Excel version: Microsoft Excel 2016

☑ First row has column names

FIGURE 10A-47

The Choose a Data Source Page

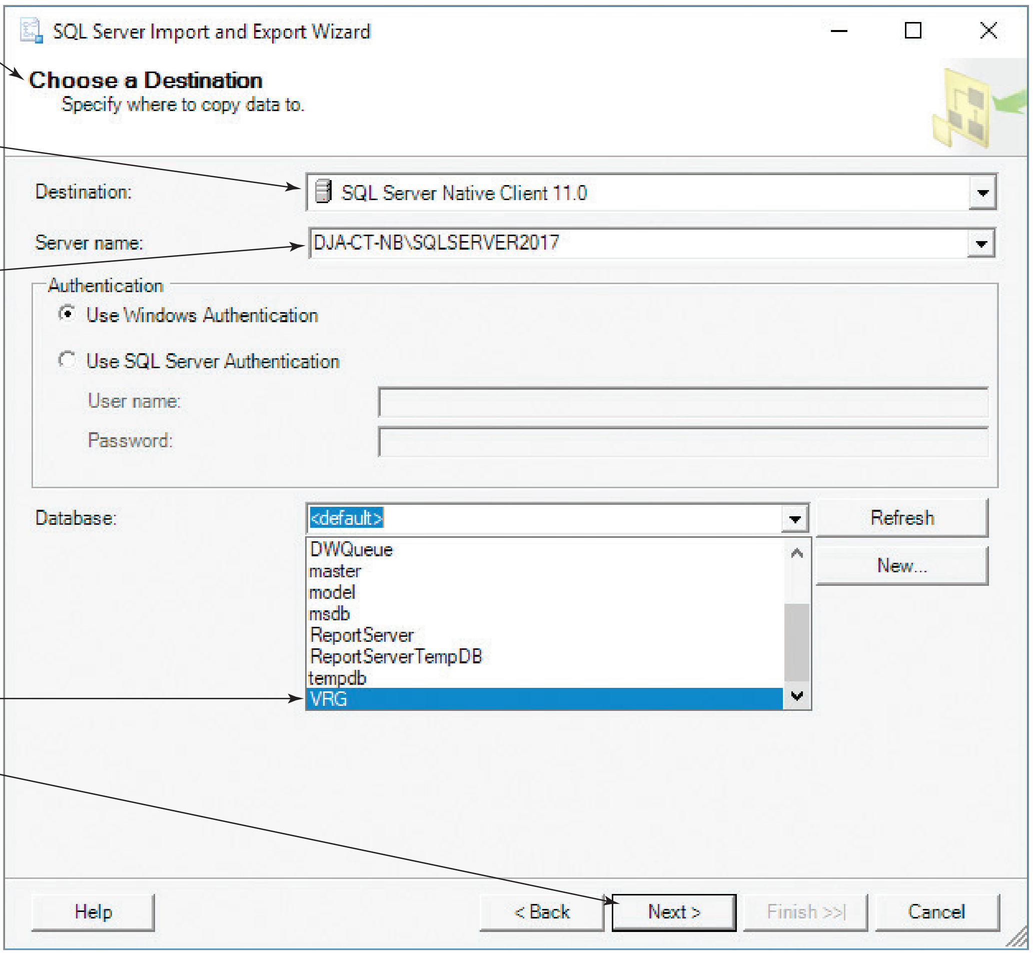
The **Choose a Destination** page

Select **SQL Server Native Client 11.0** from the drop-down list

Select the correct SQL Server instance from the drop-down list

Select the **VRG** database from the drop-down list

The **Next** button



Choose a Destination
Specify where to copy data to.

Destination: SQL Server Native Client 11.0

Server name: DJA-CT-NB\SQLSERVER2017

Authentication
- Use Windows Authentication (selected)
- Use SQL Server Authentication

User name:
Password:

Database: <default>

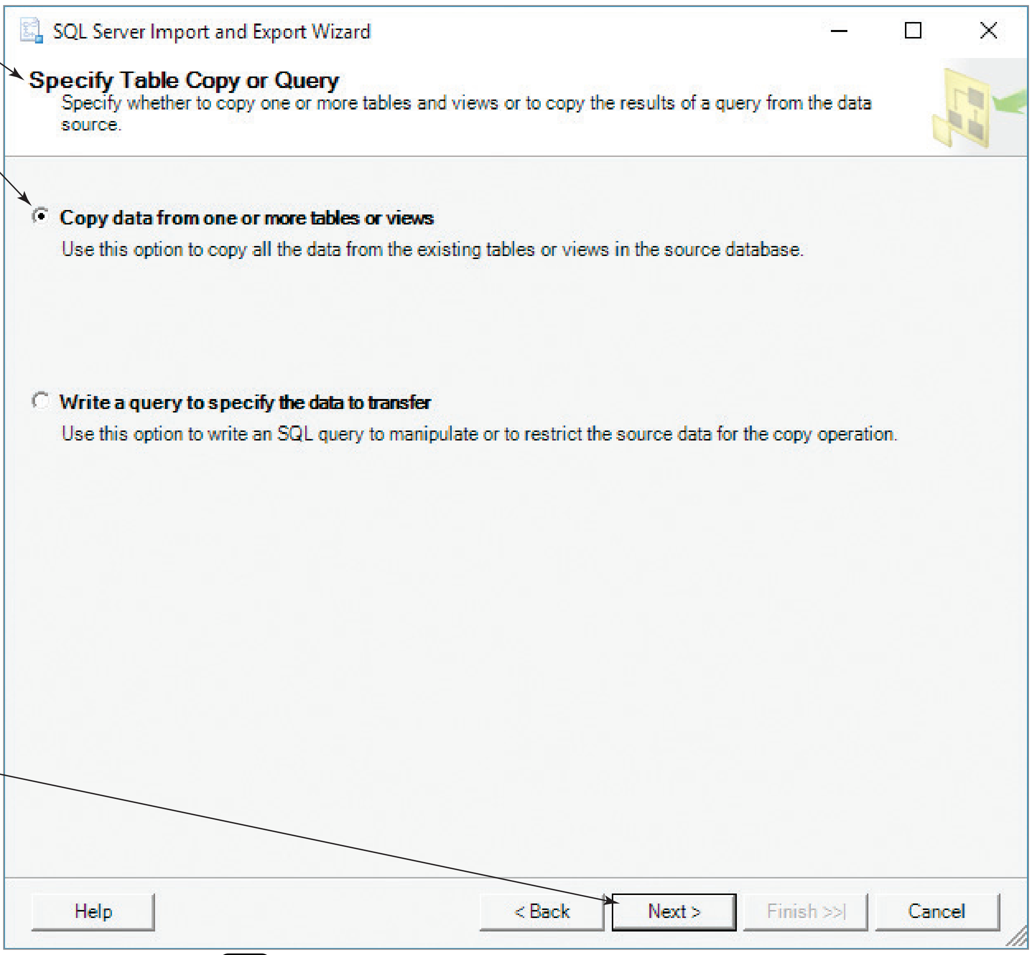
DWQueue
master
model
msdb
ReportServer
ReportServerTempDB
tempdb
VRG

FIGURE 10A-48
The Choose a Destination Page

The **Specify Table Copy or Query** page

Select **Copy data from one or more tables or views**

The **Next** button



Specify Table Copy or Query
Specify whether to copy one or more tables and views or to copy the results of a query from the data source.

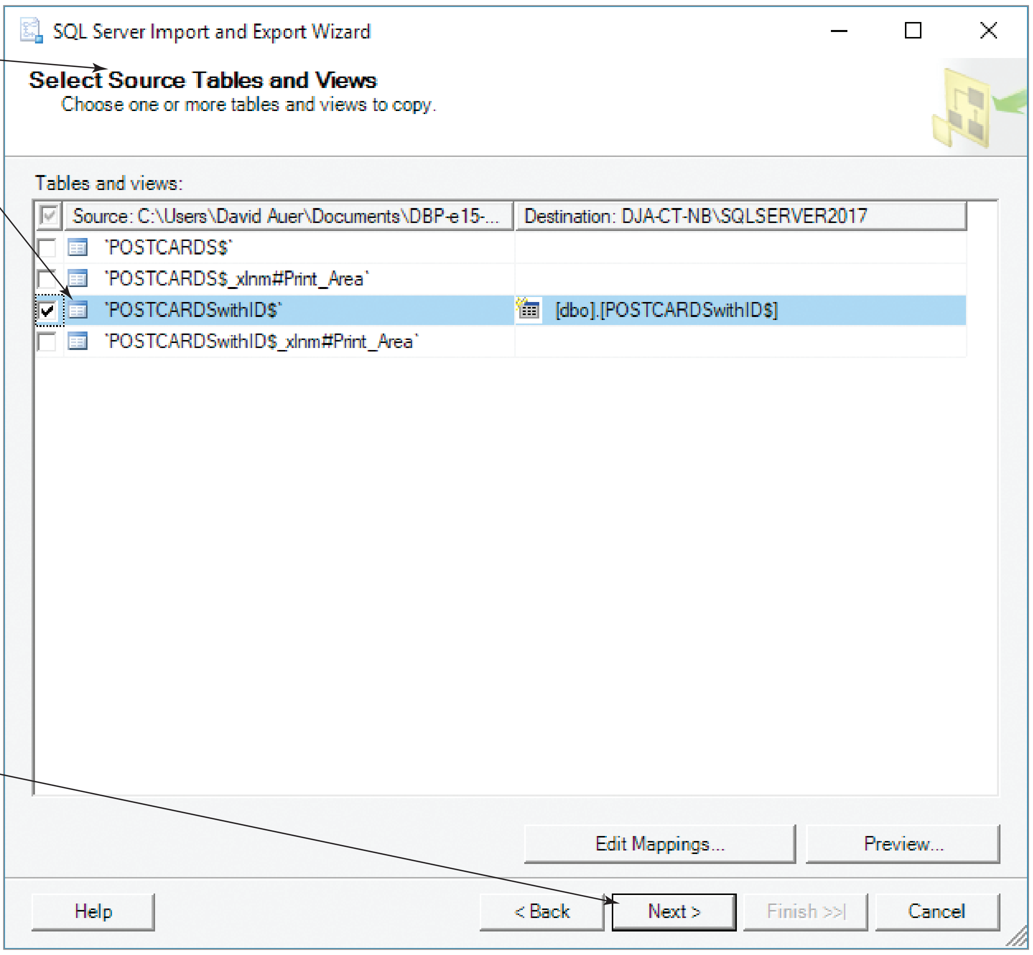
- **Copy data from one or more tables or views** (selected)
  Use this option to copy all the data from the existing tables or views in the source database.
- **Write a query to specify the data to transfer**
  Use this option to write an SQL query to manipulate or to restrict the source data for the copy operation.

FIGURE 10A-49
The Specify Table Copy or Query Page

The **Select Source Tables and Views** page

Check the **'POSTCARDwithID$'** check box

The **Next** button


SQL Server Import and Export Wizard
Select Source Tables and Views
Choose one or more tables and views to copy.
Tables and views:
Edit Mappings...
Preview...
Help
< Back
Next >
Finish >>|
Cancel

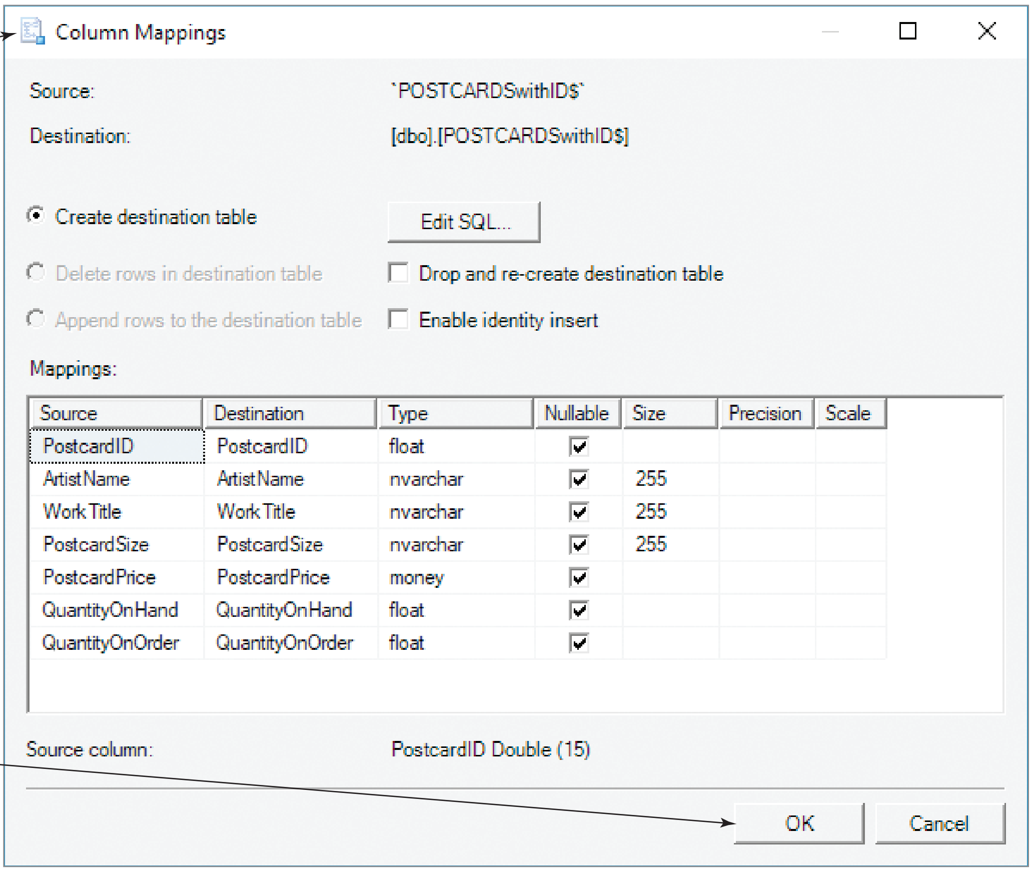
	Source: C:\Users\David Auer\Documents\DBP-e15-...	Destination: DJA-CT-NB\SQLSERVER2017
☐	\`POSTCARDS$\`	
☐	\`POSTCARDS$_xlnm#Print_Area\`	
✓	\`POSTCARDSwithID$\`	[dbo].[POSTCARDSwithID$]
☐	\`POSTCARDSwithID$_xlnm#Print_Area\`	

FIGURE 10A-50

The Select Source Tables and Views Page

The **Column Mappings** dialog box

The **OK** button



Source: \`POSTCARDSwithID$\`

Destination: [dbo].[POSTCARDSwithID$]

[● Create destination table]

○ Delete rows in destination table

○ Append rows to the destination table

☐ Drop and re-create destination table

☐ Enable identity insert

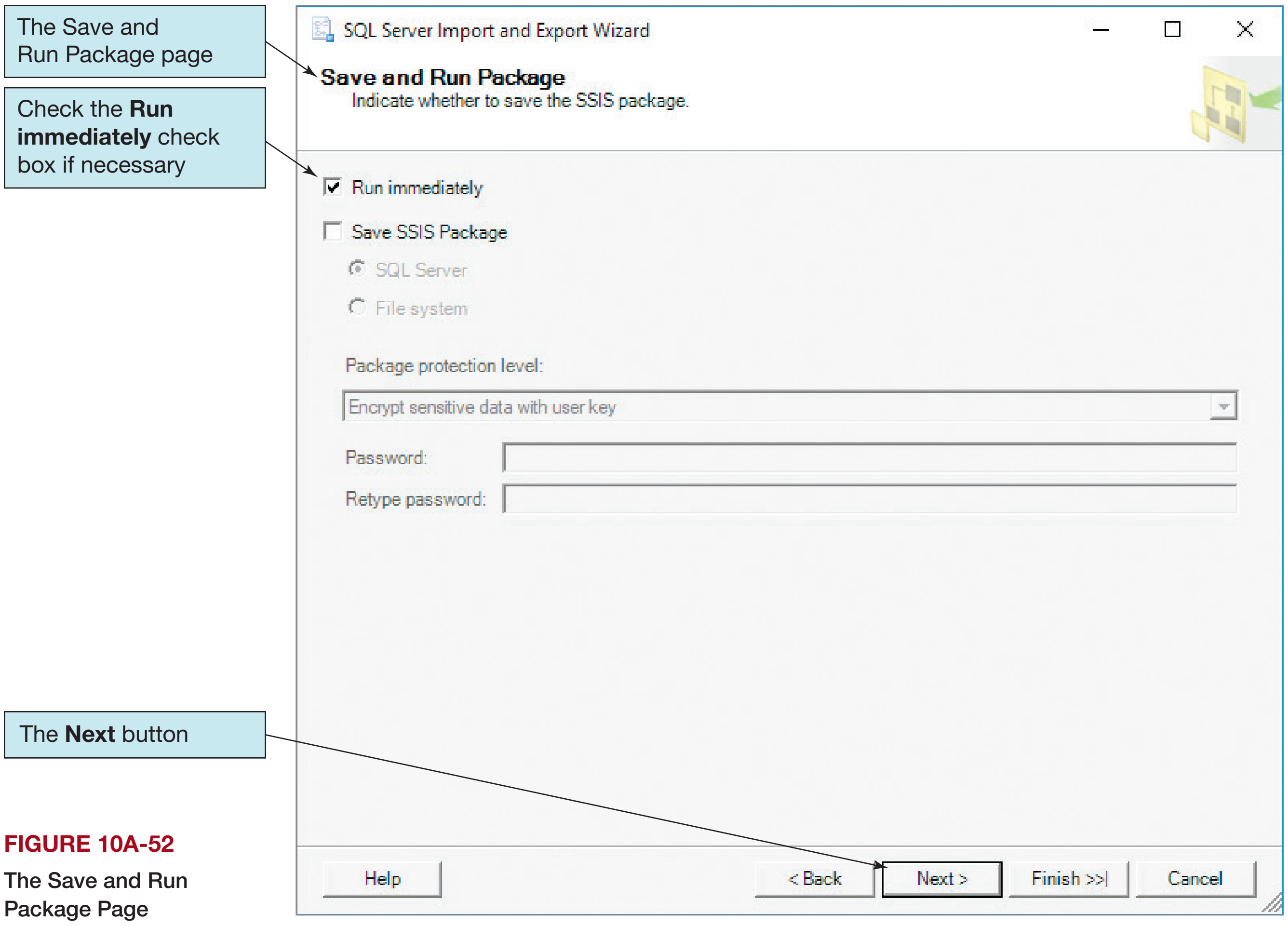
Mappings:

Source	Destination	Type	Nullable	Size	Precision	Scale
PostcardID	PostcardID	float	✓			
ArtistName	ArtistName	nvarchar	✓	255		
WorkTitle	WorkTitle	nvarchar	✓	255		
PostcardSize	PostcardSize	nvarchar	✓	255		
PostcardPrice	PostcardPrice	money	✓			
QuantityOnHand	QuantityOnHand	float	✓			
QuantityOnOrder	QuantityOnOrder	float	✓			

Source column: PostcardID Double (15)

FIGURE 10A-51

The Column Mappings Dialog Box


The Save and Run Package page
Check the **Run immediately** check box if necessary
The **Next** button
SQL Server Import and Export Wizard
Save and Run Package
Indicate whether to save the SSIS package.
Help
< Back
Next >
Finish >>|
Cancel

☑ Run immediately
☐ Save SSIS Package
● SQL Server
○ File system
Package protection level: Encrypt sensitive data with user key
Password:
Retype password:

FIGURE 10A-52

The Save and Run Package Page

11. Click the **OK** button to return to the *Select Source Tables and Views* page, and then click the **Next** button.
12. The *Save and Run Package* page is displayed as shown in Figure 10A-52. Click the **Next** button to display the *Complete the Wizard* page as shown in Figure 10A-53, and then click the **Finish** button.
13. The SQL Server Import and Export Wizard runs the actual import and then displays *The execution was successful page* as shown in Figure 10A-54. Note that there are no errors in the process. Click the **Close** button to close the wizard.
14. In SQL Server Management Studio, refresh the **VRG** database. In Object Explorer, expand the **VRG** database, then expand the **Tables** object, then expand the **dbo.POSTCARDSwithID$** object, and finally expand the **Columns** object.
15. Open a New Query window, and run SQL-Query-CH10A-01:

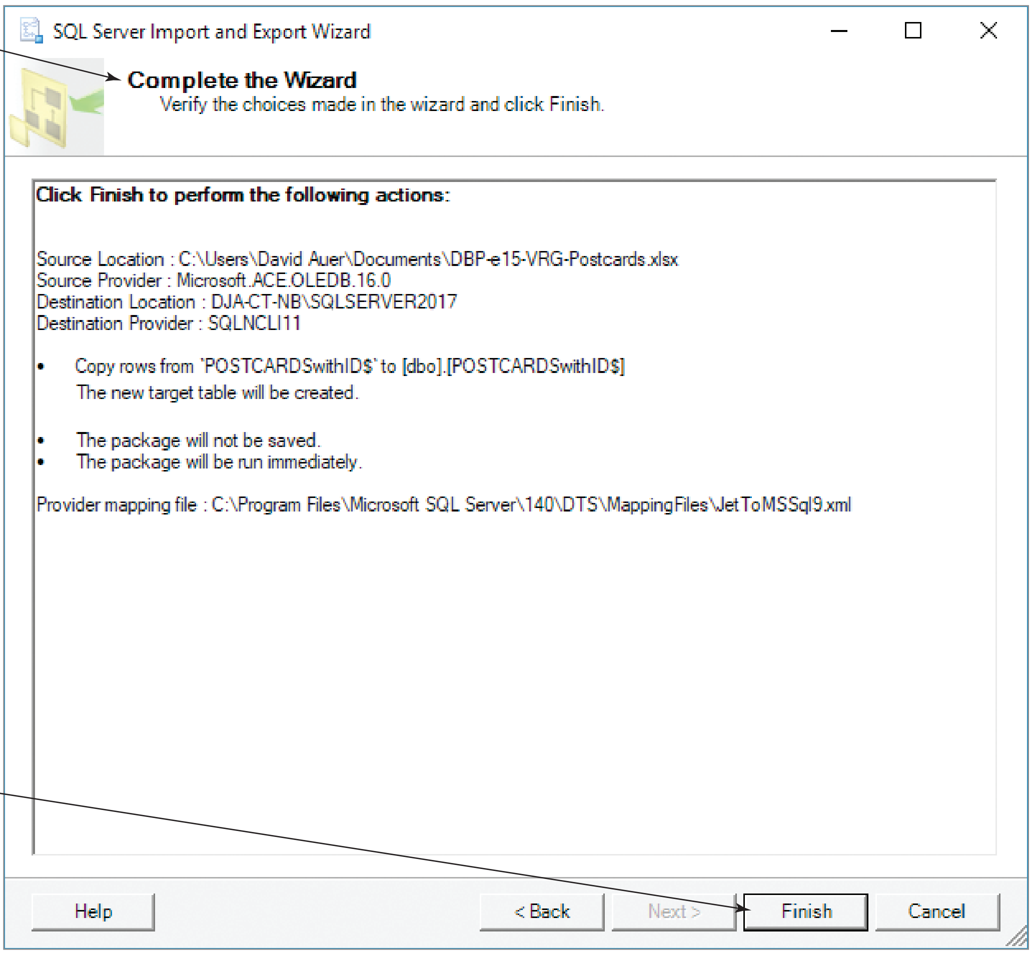
```
/* *** SQL-Query-CH10A-01 *** */
SELECT     *
FROM       POSTCARDSwithID$;
```

16. The results of SQL-Query-CH10A-01 are shown in Figure 10A-55. Note that in this case the data was imported correctly, with no extraneous rows of data.

Now that we have successfully imported the temporary POSTCARDSwithID$ table, we will need to design and implement the actual final table or tables in the VRG database to store this data in the form we want and then populate those tables. We will deal with these steps in Exercise 10A.41.

The **Complete the Wizard** page

The **Finish** button



Complete the Wizard

Verify the choices made in the wizard and click Finish.

Click Finish to perform the following actions:

Source Location : C:\Users\David Auer\Documents\DBP-e15-VRG-Postcards.xlsx
Source Provider : Microsoft.ACE.OLEDB.16.0
Destination Location : DJA-CT-NB\SQLSERVER2017
Destination Provider : SQLNCLI11

- Copy rows from `POSTCARDSwithID$` to [dbo].[POSTCARDSwithID$]
  The new target table will be created.

- The package will not be saved.
- The package will be run immediately.

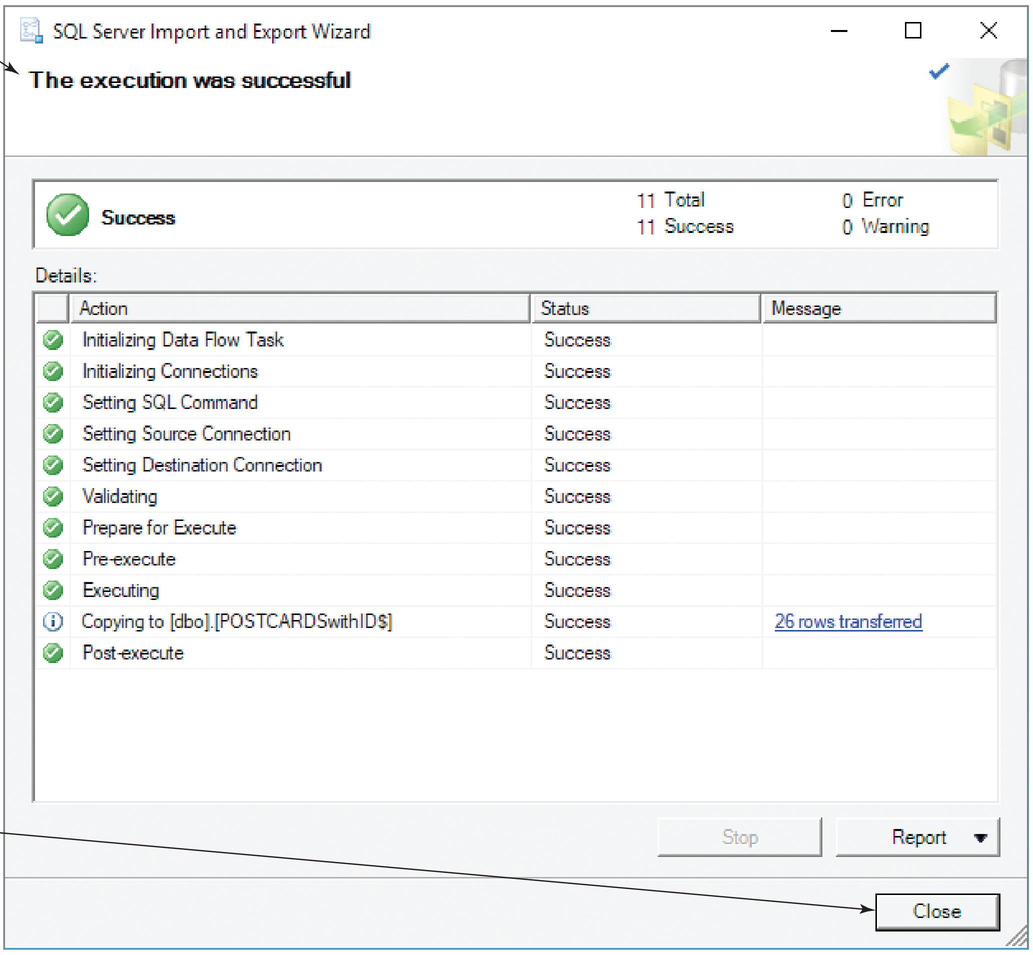
Provider mapping file : C:\Program Files\Microsoft SQL Server\140\DTS\MappingFiles\JetToMSSql9.xml

FIGURE 10A-53

The Complete the Wizard Page

The **The execution was successful** page

The **Close** button



The execution was successful

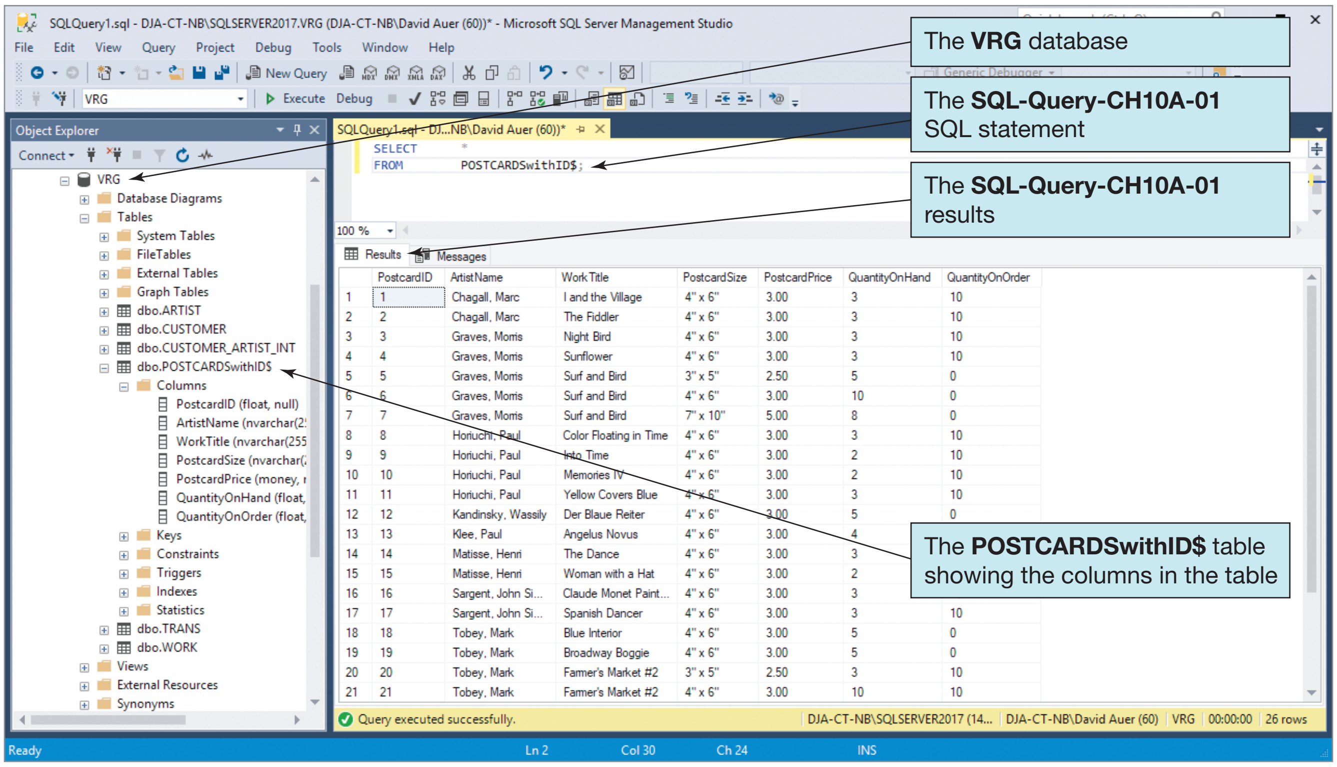
Success 11 Total 11 Success 0 Error 0 Warning

Details:

Action	Status	Message
Initializing Data Flow Task	Success	
Initializing Connections	Success	
Setting SQL Command	Success	
Setting Source Connection	Success	
Setting Destination Connection	Success	
Validating	Success	
Prepare for Execute	Success	
Pre-execute	Success	
Executing	Success	
Copying to [dbo].[POSTCARDSwithID$]	Success	26 rows transferred
Post-execute	Success	

FIGURE 10A-54

The Execution Was Successful Page



```
SELECT      *
FROM        POSTCARDSwithID$;
```

	PostcardID	ArtistName	WorkTitle	PostcardSize	PostcardPrice	QuantityOnHand	QuantityOnOrder
1	1	Chagall, Marc	I and the Village	4" x 6"	3.00	3	10
2	2	Chagall, Marc	The Fiddler	4" x 6"	3.00	3	10
3	3	Graves, Morris	Night Bird	4" x 6"	3.00	3	10
4	4	Graves, Morris	Sunflower	4" x 6"	3.00	3	10
5	5	Graves, Morris	Surf and Bird	3" x 5"	2.50	5	0
6	6	Graves, Morris	Surf and Bird	4" x 6"	3.00	10	0
7	7	Graves, Morris	Surf and Bird	7" x 10"	5.00	8	0
8	8	Horiuchi, Paul	Color Floating in Time	4" x 6"	3.00	3	10
9	9	Horiuchi, Paul	Into Time	4" x 6"	3.00	2	10
10	10	Horiuchi, Paul	Memories IV	4" x 6"	3.00	2	10
11	11	Horiuchi, Paul	Yellow Covers Blue	4" x 6"	3.00	3	10
12	12	Kandinsky, Wassily	Der Blaue Reiter	4" x 6"	3.00	5	0
13	13	Klee, Paul	Angelus Novus	4" x 6"	3.00	4	
14	14	Matisse, Henri	The Dance	4" x 6"	3.00	3	
15	15	Matisse, Henri	Woman with a Hat	4" x 6"	3.00	2	
16	16	Sargent, John Si...	Claude Monet Paint...	4" x 6"	3.00	3	
17	17	Sargent, John Si...	Spanish Dancer	4" x 6"	3.00	3	10
18	18	Tobey, Mark	Blue Interior	4" x 6"	3.00	5	0
19	19	Tobey, Mark	Broadway Boggie	4" x 6"	3.00	5	0
20	20	Tobey, Mark	Farmer's Market #2	3" x 5"	2.50	3	10
21	21	Tobey, Mark	Farmer's Market #2	4" x 6"	3.00	10	10

Query executed successfully. | DJA-CT-NB\SQLSERVER2017 (14... | DJA-CT-NB\David Auer (60) | VRG | 00:00:00 | 26 rows

FIGURE 10A-55

The SQL-Query-CH10A-01 Query and Results

Microsoft SQL Server 2017 Application Logic

An SQL Server database can be processed from an application in a number of different ways. The first is one that we are already familiar with—SQL scripts. For security, however, such files should only be used during application development and testing and never on an operational database.

Another way is to create application code using a Microsoft .NET language such as C#.NET, C++.NET, VB.NET, or some other programming language and then invoke SQL Server DBMS commands from those programs. The modern way to do this is to use a library of object classes, create objects that accomplish database work, and then process those objects by setting object properties and invoking object methods. We will look at another alternative—embedding SQL statements in Web page code using the PHP scripting language—in Chapter 11.

Finally, based on the SQL standard, application logic can be embedded in **SQL/Persistent Stored Modules (SQL/PSM)** functions, triggers, and stored procedures.

Functions, or more specifically user-defined functions (to distinguish them from SQL Server provided functions), are short blocks of reusable code that are accessed from SQL queries, stored procedures, or other functions. User-defined functions take input values, manipulate those values, and return a value as the output of the functions.

As you learned in Chapter 7, triggers can be used to set default values, to enforce data constraints, to update views, and to enforce referential integrity constraints. In this chapter, we will describe four triggers, one for each of the four trigger uses. These triggers will be invoked by SQL Server when the specified actions occur.

Stored procedures, as described in Chapter 7, can be invoked from application programs or from Web pages using languages such as VBScript or PHP. Stored procedures can also be executed from the SQL Server Management Studio, but this should be done only when the procedures are being developed and tested. As described in Chapter 9, for security reasons, no one other than authorized members of the DBA staff should be allowed to interactively process an operational database.

In this chapter, we will describe and illustrate two stored procedures. Here we will test those procedures by invoking them from the SQL Server Management Studio, and some of our output will be designed specifically for this environment. Again, this should be done only during development and testing. You will learn how to invoke these stored procedures from application code in Chapter 11.

Transact-SQL

Transact-SQL (T-SQL) is Microsoft's name for the SQL Server variant of SQL and includes SQL/PSM capabilities for use in functions, stored procedures, and triggers. We will use certain SQL/PSM elements of Transact-SQL in such code, and therefore we need to discuss them at this point. Information on these and other Transact-SQL language components can be found at the Microsoft SQL Server 2017 Transact-SQL Reference at *https://docs.microsoft.com/en-us/sql/t-sql/language-reference*.

Basic SQL/PSM Structure

Transact-SQL uses the same basic structure for user-defined functions, triggers, and stored procedures, although each varies according to what it needs to accomplish. This basic structure is:

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
/* *** SQL-Code-Example-CH10A-01 *** */
CREATE     {FUNCTION, PROCEDURE, TRIGGER} {}
           {Additional TRIGGER declarations here for TRIGGER only}
(
-- Define input parameters if any here
)
AS
BEGIN
-- DECLARE local variables here
-- PSM module statements here
END
```

Note the use of the **BEGIN . . . END keywords**, which define a block of Transact-SQL statements so more than one statement can be executed to contain all procedural statements in the user-defined function, stored procedure, or trigger.

Transact-SQL Variables

Transact-SQL identifies variables and parameters with an @ symbol. Thus, WorkID is a column name, but @WorkID is a Transact-SQL variable or parameter. A **parameter** is a value that is passed to a stored procedure when it is called. A **variable** is a value used within a stored procedure or trigger itself. Comments in Transact-SQL are either enclosed in /* (slash asterisk) and */ (asterisk slash) signs or follow -- (two dashes) if they are restricted to one line.

BY THE WAY Transact-SQL statements in procedures and triggers can be written with or without the ending semicolon (;) required in nonprocedural SQL statements. For consistency, we will continue to use semicolons for SQL statements, but we will not use them for some of the other procedural elements discussed in the following sections.

Transact-SQL Control-of-Flow Statements

The **Transact-SQL control-of-flow language** contains procedural language components that let you control exactly which parts of your code are used and the conditions required for their use. These components include BEGIN . . . END, IF . . . ELSE, WHILE, RETURN, and other keywords that can be used to direct the operations of a block of code.

The *BEGIN . . . END* keywords define a block of Transact-SQL statements so more than one statement can be executed. The **IF . . . ELSE keywords** are used to test for a condition and then direct which blocks of code are used based on the result of that test. Note that the END keyword is *not* used as part of the IF . . . ELSE construct in Transact-SQL, although it is commonly used here in many other programming languages. Therefore, we have to use BEGIN . . . END blocks within the IF . . . ELSE statement. Without this grouping, only one SQL statement can be used in either the IF or ELSE section of an IF . . . ELSE conditional branching. The **WHILE keyword** is used to create loops in Transact-SQL, where a section of code is repeated as long as ("while") some condition is true. The **RETURN keyword** is used to exit a block of code and terminate whatever code structure (stored procedure or trigger) is running. In this case, control is "returned" to the DBMS.

As an example, let's consider a new customer at the View Ridge Gallery who needs to have customer data entered into the CUSTOMER table and artist interest data entered into the CUSTOMER_ARTIST_INT table. The new customer is Michael Bench, with phone number 206-876-8822, email address Michael.Bench@somewhere.com, and an interest in French artists.

Before we enter Michael's data, we need to check to see whether he is already in the database. To do this, we can use the following Transact-SQL code in a trigger or stored procedure:

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
/* *** SQL-Code-Example-CH10A-02 *** */
DECLARE    @RowCount    AS    Int
-- Check to see if Customer already exists in database
SELECT    @RowCount = COUNT(*)
FROM      CUSTOMER
WHERE     LastName = 'Bench'
     AND  FirstName = 'Michael'
     AND  EmailAddress = 'Michael.Bench@somewhere.com'
     AND  AreaCode = '206'
     AND  PhoneNumber = '876-8822';
-- IF @RowCount > 0 THEN Customer already exists.
IF (@RowCount > 0)
     BEGIN
          PRINT ' The Customer is already in the database. '
          RETURN
     END
-- IF (@RowCount = 0) THEN Customer does not exist in database.
ELSE
    BEGIN
    -- Insert new Customer data.
         INSERT INTO dbo.CUSTOMER
              (LastName, FirstName, EmailAddress, AreaCode, PhoneNumber)
              VALUES('Bench', 'Michael', 'Michael.Bench@somewhere.com',
              '206', '876-8822');
         PRINT ' The new Customer is now in the database. '
     END
```

This block of SQL code illustrates the use of all of the control-of-flow keywords we have discussed except WHILE. The WHILE keyword is used in loops, and one use of a loop is in an SQL cursor.

Transact-SQL Cursor Statements

As we discussed in Chapter 7, a cursor is used so SQL results stored in a table can be processed one row at a time. The **Transact-SQL cursor** is Microsoft's implementation of a cursor. Related cursor keywords include DECLARE, OPEN, FETCH, CLOSE, and DEALLOCATE. The **DECLARE CURSOR keywords** are used to create a cursor, whereas the **OPEN CURSOR keywords** start the use of the cursor. The **FETCH keyword** is used to retrieve row data. The **CLOSE CURSOR keywords** are used to exit the use of a cursor, and the **DEALLOCATE CURSOR keywords** remove the cursor from the DBMS. When using a cursor, the WHILE keyword is used to control how long the cursor is active.

Let's consider Michael Bench's interest in French artists. The ARTIST table has two French artists—Henri Matisse and Marc Chagall. Therefore, we need to add new rows to CUSTOMER_ARTIST_INT, both of which will contain Michael's CustomerID number (now that he has one) and each of which contains the ArtistID for one of these artists. To do this, we can use the following Transact-SQL code in a trigger or stored procedure:

```
/* *** EXAMPLE CODE - DO NOT RUN *** */
/* *** SQL-Code-Example-CH10A-03 *** */
DECLARE     @ArtistID      AS     Int
DECLARE     @CustomerID    AS     Int
-- GET the CustomerID surrogate key value.
SELECT      @CustomerID = CustomerID
FROM        CUSTOMER
WHERE       LastName = 'Bench'
     AND    FirstName = 'Michael'
     AND    EmailAddress = 'Michael.Bench@somewhere.com'
     AND    AreaCode = '206'
     AND    PhoneNumber = '876-8822';
-- Create intersection record for each appropriate Artist.
-- Create cursor ArtistCursor
DECLARE ArtistCursor CURSOR FOR
        SELECT      ArtistID
        FROM        ARTIST
        WHERE       Nationality = 'French';
--Process each appropriate Artist
OPEN ArtistCursor
     FETCH NEXT FROM ArtistCursor INTO @ArtistID
           WHILE @@FETCH_STATUS = 0
           BEGIN
                INSERT INTO CUSTOMER_ARTIST_INT
                (ArtistID, CustomerID)
                VALUES(@ArtistID, @CustomerID)
                PRINT ' New CUSTOMER_ARTIST_INT row added. '
                PRINT ' ArtistID = '+CONVERT(Char(6), @ArtistID)
                PRINT ' CustomerID = '+CONVERT(Char(6), @CustomerID);
```

```
          FETCH NEXT FROM ArtistCursor INTO @ArtistID
          END
CLOSE ArtistCursor
-- Remove cursor ArtistCursor
DEALLOCATE ArtistCursor
```

In this code, the ArtistCursor loops through the set of ArtistID values for French artists as long as ("while") the value of @@FETCH_STATUS is equal to 0. The **Transact-SQL @@FETCH_STATUS function** returns a value based on whether the FETCH NEXT statement actually returned a value from the next row. If there is no next row, then @@FETCH_STATUS returns -1.

Transact-SQL Output Statements

The two previous code segments also illustrate the use of the **Transact-SQL PRINT command**. This command will send a message to the SQL Server Management Console Messages window. We will use this output as a proxy for the real output that would be returned to an application, as it would be in most situations. Note that the actual output to be printed is marked by a single quote at the beginning and the end. Also note the use of the CONVERT function. The **Transact-SQL CONVERT function** is used to change one data type to another. In this case, we can only print character strings, so numbers such as ArtistID must be converted to character strings using the CONVERT function. Character strings can be combined by using the plus sign (+), and this is also shown in the code segments earlier.

User-Defined Functions

As we discussed in Chapter 7, a **user-defined function** (also known as a **stored function**) is a stored set of SQL statements that:

- is *called by name* from another SQL statement,
- may have *input parameters* passed to it by the calling SQL statement, and
- *returns an output value* to the SQL statement that called the function.

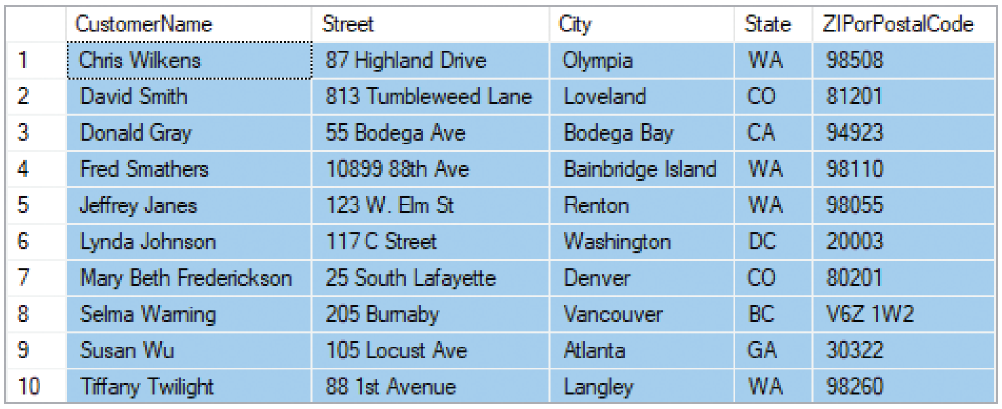
The logical process flow of a user-defined function is illustrated in Figure 7-20. SQL/PSM user-defined functions are similar to the SQL built-in functions (COUNT, SUM, AVG, MAX, and MIN) that we discussed and used in Chapter 2, except that, as the name implies, we create them ourselves to perform specific tasks that we need to do.

In Chapter 7, we used as our example the common problem of needing a name in the format *LastName, FirstName* (including the comma) in a report when the database stores the basic data in two fields named *FirstName* and *LastName*. Here we will build a user-defined function for a similar task: concatenating the *FirstName* and *LastName* fields into a name in the format *First Name LastName* (including the space!). This construct is commonly used, for example, for mailing labels.

Using the data in the VRG database, we could, of course, simply include the code to do this in an SQL statement (similar to SQL-Query-CH02-45 in Chapter 2) such as:

```
/* *** SQL-Query-CH10A-02 *** */
SELECT      RTRIM(FirstName)+' '+RTRIM(LastName) AS CustomerName,
            Street, City, State, ZIPorPostalCode
FROM        CUSTOMER
ORDER BY    CustomerName;
```

This produces the desired results but at the expense of working out some cumbersome coding:



	CustomerName	Street	City	State	ZIPorPostalCode
1	Chris Wilkens	87 Highland Drive	Olympia	WA	98508
2	David Smith	813 Tumbleweed Lane	Loveland	CO	81201
3	Donald Gray	55 Bodega Ave	Bodega Bay	CA	94923
4	Fred Smathers	10899 88th Ave	Bainbridge Island	WA	98110
5	Jeffrey Janes	123 W. Elm St	Renton	WA	98055
6	Lynda Johnson	117 C Street	Washington	DC	20003
7	Mary Beth Frederickson	25 South Lafayette	Denver	CO	80201
8	Selma Warning	205 Burnaby	Vancouver	BC	V6Z 1W2
9	Susan Wu	105 Locust Ave	Atlanta	GA	30322
10	Tiffany Twilight	88 1st Avenue	Langley	WA	98260

The alternative is to create a user-defined function to store this code. Not only does this make it easier to use, but it also makes it available for use in other SQL statements. Figure 10A-56 shows a user-defined function named *FirstNameFirst* written in T-SQL for use with Microsoft SQL Server 2017, and the SQL code for the function uses, as we would expect, specific syntax requirements for Microsoft SQL Server's T-SQL 2017:

- The function is created and stored in the database by using the **Transact-SQL CREATE FUNCTION statement**.
- The function name starts with *dbo*, which is a Microsoft SQL Server *schema* name (SQL Server schemas are discussed later in this chapter). This use of a schema name appended to a database object name is common in Microsoft SQL Server.
- The variable names of both the input parameters and the returned output value start with @.
- The concatenation syntax is T-SQL syntax.

Now that we have created and stored the user-defined function, we can use it in SQL-Query-CH10A-03:

```
/* *** SQL-Query-CH10A-03 *** */
SELECT      dbo.FirstNameFirst(FirstName, LastName) AS CustomerName,
            Street, City, State, ZIPorPostalCode
FROM        CUSTOMER
ORDER BY    CustomerName;
```

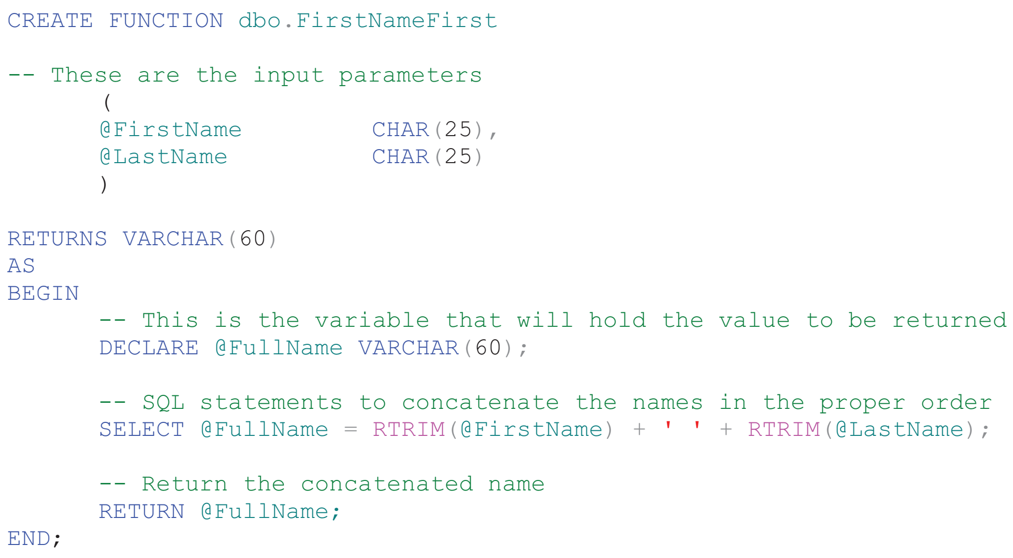
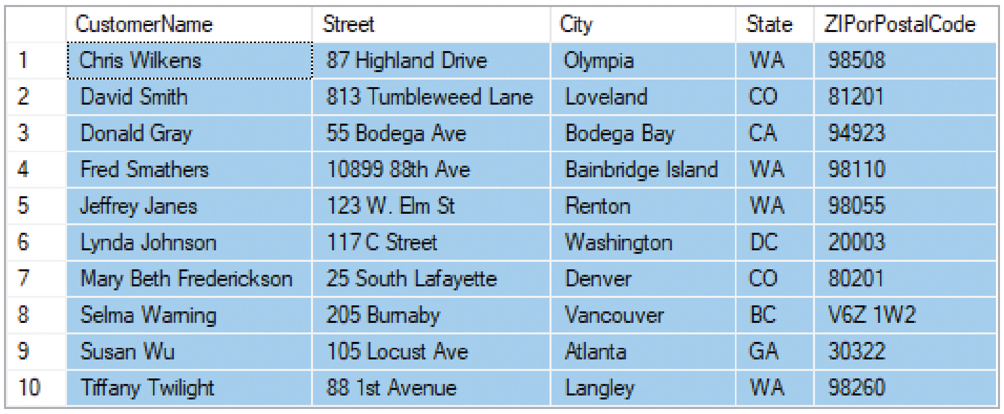


FIGURE 10A-56

The SQL Statements for the FirstNameFirst User-Defined Function

```
CREATE FUNCTION dbo.FirstNameFirst

-- These are the input parameters
      (
      @FirstName          CHAR(25),
      @LastName           CHAR(25)
      )

RETURNS VARCHAR(60)
AS
BEGIN
      -- This is the variable that will hold the value to be returned
      DECLARE @FullName VARCHAR(60);

      -- SQL statements to concatenate the names in the proper order
      SELECT @FullName = RTRIM(@FirstName) + ' ' + RTRIM(@LastName);

      -- Return the concatenated name
      RETURN @FullName;
END;
```

Now we have a function that produces the results we want, which, of course, are identical to the results for SQL-Query-CH10A-02 earlier:

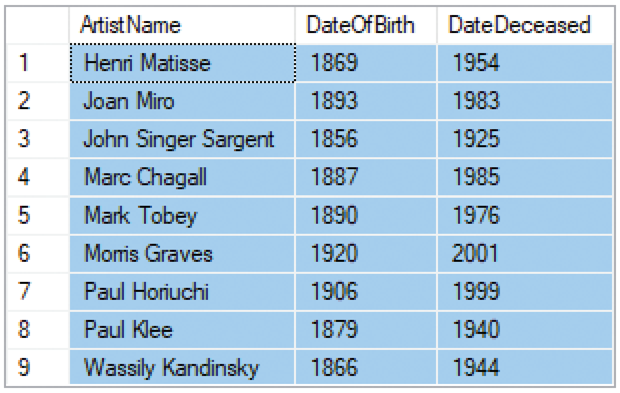


	CustomerName	Street	City	State	ZIPorPostalCode
1	Chris Wilkens	87 Highland Drive	Olympia	WA	98508
2	David Smith	813 Tumbleweed Lane	Loveland	CO	81201
3	Donald Gray	55 Bodega Ave	Bodega Bay	CA	94923
4	Fred Smathers	10899 88th Ave	Bainbridge Island	WA	98110
5	Jeffrey Janes	123 W. Elm St	Renton	WA	98055
6	Lynda Johnson	117 C Street	Washington	DC	20003
7	Mary Beth Frederickson	25 South Lafayette	Denver	CO	80201
8	Selma Warning	205 Burnaby	Vancouver	BC	V6Z 1W2
9	Susan Wu	105 Locust Ave	Atlanta	GA	30322
10	Tiffany Twilight	88 1st Avenue	Langley	WA	98260

The advantage of having a user-defined function is that we can now use it whenever we need to without having to re-create the code. For example, our previous query used data in the View Ridge Gallery CUSTOMER table, but we could just as easily use the function with the data in the ARTIST table:

```
/* *** SQL-Query-CH10A-04 *** */
SELECT      dbo.FirstNameFirst(FirstName, LastName) AS ArtistName,
            DateOfBirth, DateDeceased
FROM        ARTIST
ORDER BY    ArtistName;
```

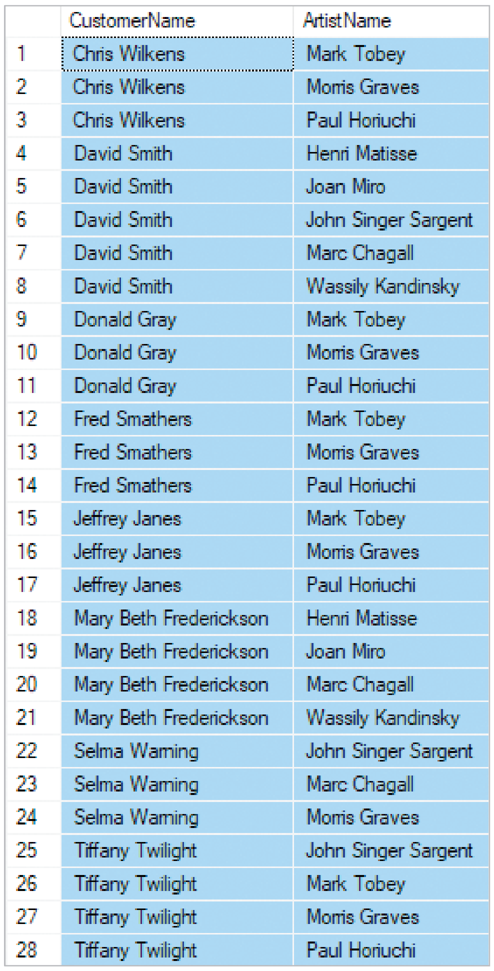
This query produces the expected result:



	ArtistName	DateOfBirth	DateDeceased
1	Henri Matisse	1869	1954
2	Joan Miro	1893	1983
3	John Singer Sargent	1856	1925
4	Marc Chagall	1887	1985
5	Mark Tobey	1890	1976
6	Morris Graves	1920	2001
7	Paul Horiuchi	1906	1999
8	Paul Klee	1879	1940
9	Wassily Kandinsky	1866	1944

We can even use the function multiple times in the same SQL statement, as shown in SQL-Query-CH10A-05, which is a variant on the SQL query we used to create the SQL view CustomerInterestsView in our discussion of SQL views in this chapter:

```
/* *** SQL-Query-CH10A-05 *** */
SELECT      dbo.FirstNameFirst(C.FirstName, C.LastName) AS CustomerName,
            dbo.FirstNameFirst(A.FirstName, A.LastName) AS ArtistName
FROM        CUSTOMER AS C JOIN CUSTOMER_ARTIST_INT AS CAI
   ON       C.CustomerID = CAI.CustomerID
               JOIN    ARTIST AS A
                  ON    CAI.ArtistID = A.ArtistID
ORDER BY    CustomerName, ArtistName;
```



	CustomerName	ArtistName
1	Chris Wilkens	Mark Tobey
2	Chris Wilkens	Morris Graves
3	Chris Wilkens	Paul Horiuchi
4	David Smith	Henri Matisse
5	David Smith	Joan Miro
6	David Smith	John Singer Sargent
7	David Smith	Marc Chagall
8	David Smith	Wassily Kandinsky
9	Donald Gray	Mark Tobey
10	Donald Gray	Morris Graves
11	Donald Gray	Paul Horiuchi
12	Fred Smathers	Mark Tobey
13	Fred Smathers	Morris Graves
14	Fred Smathers	Paul Horiuchi
15	Jeffrey Janes	Mark Tobey
16	Jeffrey Janes	Morris Graves
17	Jeffrey Janes	Paul Horiuchi
18	Mary Beth Frederickson	Henri Matisse
19	Mary Beth Frederickson	Joan Miro
20	Mary Beth Frederickson	Marc Chagall
21	Mary Beth Frederickson	Wassily Kandinsky
22	Selma Warning	John Singer Sargent
23	Selma Warning	Marc Chagall
24	Selma Warning	Morris Graves
25	Tiffany Twilight	John Singer Sargent
26	Tiffany Twilight	Mark Tobey
27	Tiffany Twilight	Morris Graves
28	Tiffany Twilight	Paul Horiuchi

FIGURE 10A-57

Result of SQL Query Using the FirstNameFirst User-Defined Function

This query produces the expected large result that is shown in Figure 10A-57, except now we see that both CustomerName and ArtistName display the names in the FirstName LastName syntax produced by the *FirstNameFirst* user-defined function. Compare the results in this figure with those in Figure 10A-43, which presents somewhat similar results but without the formatting provided by the *FirstNameFirst* function.

Having written the user-defined function, if you subsequently need to change the code in the function, you would use the **SQL ALTER FUNCTION statement**. Otherwise, you will get an error message saying that the function already exists when you execute the modified user-defined function code. Note that as of SQL Server 2016 SP1, these statements can be combined into the **SQL CREATE OR ALTER FUNCTION statement**. If you use this statement, you can create or alter your code without having to change the CREATE keyword to the ALTER keyword.

BY THE WAY

For user-defined functions, the final semicolon (;) after the END statement in Figures 10A-56 and 10A-58 is optional, but we prefer to include it to match the syntax of other SQL statements. For more information, see the T-SQL CREATE FUNCTION documentation at *https://docs.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql*.

Having dealt with the problem of concatenating two separate name values into one, let's consider the opposite problem: separating a combined name into separate elements. This is a problem that commonly occurs when we use data provided to us in a Microsoft Excel worksheet, where the user simply put an entire name into one cell. As a practical example of this, consider the VRG POSTCARDSwithID$ table we imported in our discussion of how to import

FIGURE 10A-58

The SQL Statements for the GetLastName CommaSeparated User-Defined Function

```
CREATE FUNCTION dbo.GetLastNameCommaSeparated

-- These are the input parameters
(
      @Name        VARCHAR(25)
)
RETURNS VARCHAR(25)
AS
BEGIN
      -- This is the variable that will hold the value to be returned
      DECLARE @LastName           VARCHAR(25);

      -- This is the variable that will hold the position of the comma
      DECLARE @IndexValue         INT;

      -- SQL statement to find the comma separator

      SELECT @IndexValue = CHARINDEX(',', @Name);

      -- SQL statement to determine last name
      SELECT @LastName = SUBSTRING(@Name, 1, (@IndexValue - 1));

      -- Return the last name
      RETURN @LastName;
END;
```

Microsoft Excel data into a Microsoft SQL Server table. Looking at Figure 10A-44, we can see that the ArtistName column contains the combined artist name in *LastName, FirstName* format.

Because a best practice of database design is to separate data like this into its separate elements, we have the problem of breaking this data into *LastName* and *FirstName*. We can write a user-defined function to do this.

Note that the delineator or separator between LastName and FirstName is a comma. We can search for the comma using the SQL Server built-in character string function **CHARINDEX**, which will return the numeric position of the comma. The full syntax of the function is:

```
CHARINDEX (ExpressionToFind, ExpressionToSearch [, StartLocation])
```

In the POSTCARDSwithID$ table, we want to find the comma (",") in ArtistName starting at the default location of 1 (character strings are indexed starting at 1, not 0).

Once we have found the comma, we can retrieve the last name by using the SQL Server built-in character string function **SUBSTRING**, which will return a contiguous subset of the characters from a character string. The full syntax of the function is:

```
SUBSTRING (ExpressionToSearch, StartLocation, Length)
```

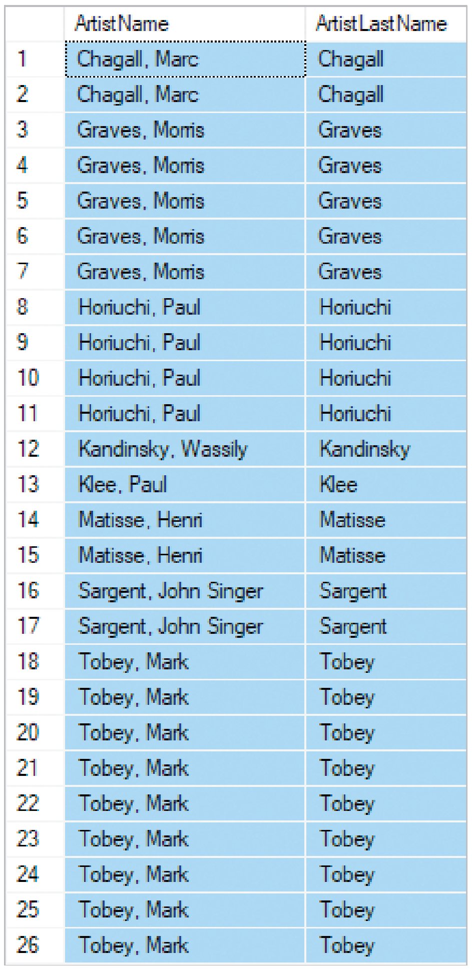
In the POSTCARDSwithID$ table, we want to return the subset of ArtistName starting at 1 and ending one character before the comma ([Value returned by CHARINDEX] – 1). We put these together into a user-defined function named *GetLastNameCommaSeparated* as shown in Figure 10A-58.

Now that we have created and stored the user-defined function, we can use it in SQL-Query-CH10A-06:

```
/* *** SQL-Query-CH10A-06 *** */
SELECT      ArtistName,
            dbo.GetLastNameCommaSeparated(ArtistName) AS ArtistLastName
FROM        POSTCARDSwithID$
ORDER BY    ArtistName;
```

FIGURE 10A-59

The Results for SQL-Query-CH10A-06



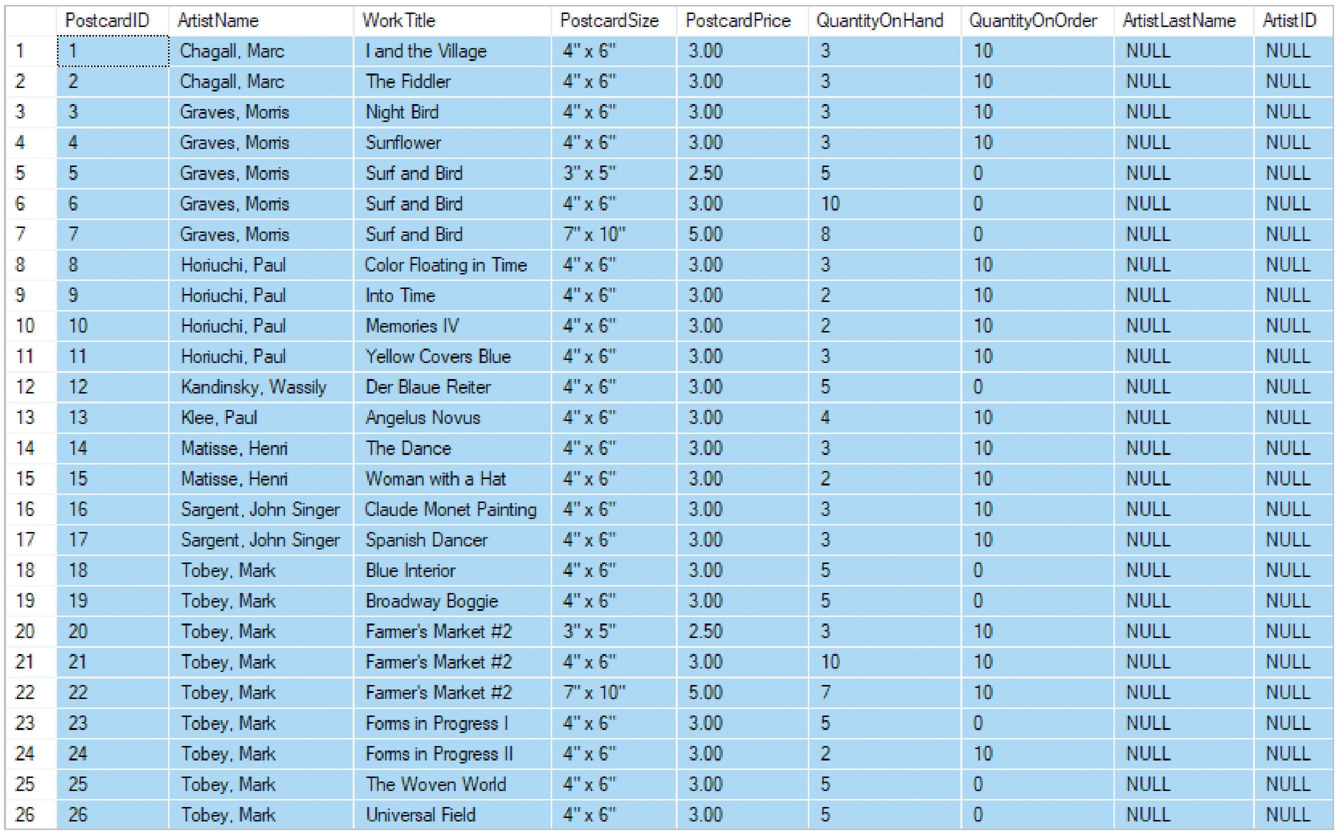
	ArtistName	ArtistLastName
1	Chagall, Marc	Chagall
2	Chagall, Marc	Chagall
3	Graves, Morris	Graves
4	Graves, Morris	Graves
5	Graves, Morris	Graves
6	Graves, Morris	Graves
7	Graves, Morris	Graves
8	Horiuchi, Paul	Horiuchi
9	Horiuchi, Paul	Horiuchi
10	Horiuchi, Paul	Horiuchi
11	Horiuchi, Paul	Horiuchi
12	Kandinsky, Wassily	Kandinsky
13	Klee, Paul	Klee
14	Matisse, Henri	Matisse
15	Matisse, Henri	Matisse
16	Sargent, John Singer	Sargent
17	Sargent, John Singer	Sargent
18	Tobey, Mark	Tobey
19	Tobey, Mark	Tobey
20	Tobey, Mark	Tobey
21	Tobey, Mark	Tobey
22	Tobey, Mark	Tobey
23	Tobey, Mark	Tobey
24	Tobey, Mark	Tobey
25	Tobey, Mark	Tobey
26	Tobey, Mark	Tobey

The results are shown in Figure 10A-59 and are exactly what we want, with the ArtistLastName column containing only the last name.

Now that we can determine the last name of the artists in the POSTCARDSwithID$ table, let's return to our discussion of that table and how we will integrate the data in it into the VRG database. Because the ARTIST table uses ArtistID as the primary key and WORK uses WorkID for the primary key, we have to find some way of associating these primary key values with the data in POSTCARDSwithID$. We can use the *GetLastNameCommaSeparated* user-defined function to help us do this.

First, we need to alter the POSTCARDSwithID$ table by adding a column named ArtistLastName to hold the last name values. The full discussion of how to do this is in Chapter 8 on database redesign, where we discuss how to use the **SQL ALTER TABLE statement**. Here, we will use it to add an ArtistLastName column and an ArtistID column.

```
/* *** SQL-ALTER-TABLE-CH10A-01 *** */
ALTER TABLE POSTCARDSwithID$
   ADD      ArtistLastName   Char(25)       NULL;
/* *** SQL-ALTER-TABLE-CH10A-02 *** */
ALTER TABLE POSTCARDSwithID$
   ADD      ArtistID         Int            NULL;
/* *** SQL-Query-CH10A-03 *** */
SELECT      *
FROM        POSTCARDSwithID$;
```



	PostcardID	ArtistName	WorkTitle	PostcardSize	PostcardPrice	QuantityOnHand	QuantityOnOrder	ArtistLastName	ArtistID
1	1	Chagall, Marc	I and the Village	4" x 6"	3.00	3	10	NULL	NULL
2	2	Chagall, Marc	The Fiddler	4" x 6"	3.00	3	10	NULL	NULL
3	3	Graves, Morris	Night Bird	4" x 6"	3.00	3	10	NULL	NULL
4	4	Graves, Morris	Sunflower	4" x 6"	3.00	3	10	NULL	NULL
5	5	Graves, Morris	Surf and Bird	3" x 5"	2.50	5	0	NULL	NULL
6	6	Graves, Morris	Surf and Bird	4" x 6"	3.00	10	0	NULL	NULL
7	7	Graves, Morris	Surf and Bird	7" x 10"	5.00	8	0	NULL	NULL
8	8	Horiuchi, Paul	Color Floating in Time	4" x 6"	3.00	3	10	NULL	NULL
9	9	Horiuchi, Paul	Into Time	4" x 6"	3.00	2	10	NULL	NULL
10	10	Horiuchi, Paul	Memories IV	4" x 6"	3.00	2	10	NULL	NULL
11	11	Horiuchi, Paul	Yellow Covers Blue	4" x 6"	3.00	3	10	NULL	NULL
12	12	Kandinsky, Wassily	Der Blaue Reiter	4" x 6"	3.00	5	0	NULL	NULL
13	13	Klee, Paul	Angelus Novus	4" x 6"	3.00	4	10	NULL	NULL
14	14	Matisse, Henri	The Dance	4" x 6"	3.00	3	10	NULL	NULL
15	15	Matisse, Henri	Woman with a Hat	4" x 6"	3.00	2	10	NULL	NULL
16	16	Sargent, John Singer	Claude Monet Painting	4" x 6"	3.00	3	10	NULL	NULL
17	17	Sargent, John Singer	Spanish Dancer	4" x 6"	3.00	3	10	NULL	NULL
18	18	Tobey, Mark	Blue Interior	4" x 6"	3.00	5	0	NULL	NULL
19	19	Tobey, Mark	Broadway Boggie	4" x 6"	3.00	5	0	NULL	NULL
20	20	Tobey, Mark	Farmer's Market #2	3" x 5"	2.50	3	10	NULL	NULL
21	21	Tobey, Mark	Farmer's Market #2	4" x 6"	3.00	10	10	NULL	NULL
22	22	Tobey, Mark	Farmer's Market #2	7" x 10"	5.00	7	10	NULL	NULL
23	23	Tobey, Mark	Forms in Progress I	4" x 6"	3.00	5	0	NULL	NULL
24	24	Tobey, Mark	Forms in Progress II	4" x 6"	3.00	2	10	NULL	NULL
25	25	Tobey, Mark	The Woven World	4" x 6"	3.00	5	0	NULL	NULL
26	26	Tobey, Mark	Universal Field	4" x 6"	3.00	5	0	NULL	NULL

FIGURE 10A-60

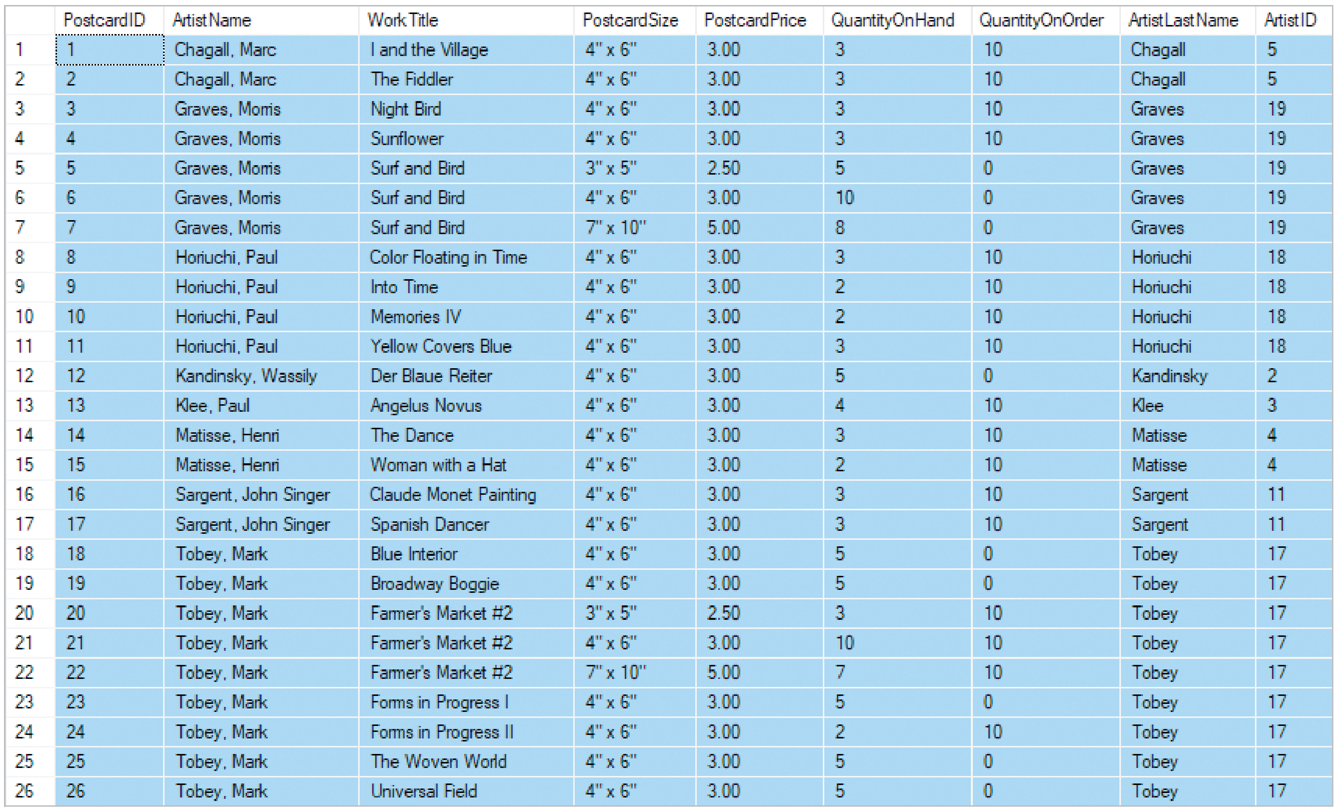
The POSTCARDSwithID$ Table with the Added Columns

Note that we are allowing the values of both of these columns to be NULL, because we have not inserted any data and therefore we cannot create them as NOT NULL even if that is what we ultimately want them to be (see Chapter 8 for a discussion of the steps to add a NOT NULL column to a table). The POSTCARDSwithID$ table now appears as shown in Figure 10A-60, with the two new columns displayed at the right side of the table.

To populate these columns, we can simply use two SQL UPDATE statements:

```
/* *** SQL-UPDATE-CH10A-01 *** */
UPDATE       POSTCARDSwithID$
   SET       ArtistLastName = dbo.GetLastNameCommaSeparated(ArtistName);
/* *** SQL-UPDATE-CH10A-02 *** */
UPDATE       POSTCARDSwithID$
   SET       ArtistID =
             (SELECT  ArtistID
              FROM    ARTIST
              WHERE   ARTIST.LastName = POSTCARDSwithID$.
                      ArtistLastName);
/* *** SQL-Query-CH10A-04 *** */
SELECT       *
FROM         POSTCARDSwithID$;
```

Note the use of the *GetLastNameCommaSeparated* user-defined function in SQL-UPDATE-CH10A-01. Also note the use of an SQL subquery in SQL-UPDATE-CH10A-02, which illustrates that SQL subqueries can be used in SQL statements beyond just the SQL SELECT statement. In fact, we can use an SQL subquery in INSERT, UPDATE, DELETE, and MERGE statements, and it is often exactly what we need! Finally, note that



	PostcardID	ArtistName	WorkTitle	PostcardSize	PostcardPrice	QuantityOnHand	QuantityOnOrder	ArtistLastName	ArtistID
1	1	Chagall, Marc	I and the Village	4" x 6"	3.00	3	10	Chagall	5
2	2	Chagall, Marc	The Fiddler	4" x 6"	3.00	3	10	Chagall	5
3	3	Graves, Morris	Night Bird	4" x 6"	3.00	3	10	Graves	19
4	4	Graves, Morris	Sunflower	4" x 6"	3.00	3	10	Graves	19
5	5	Graves, Morris	Surf and Bird	3" x 5"	2.50	5	0	Graves	19
6	6	Graves, Morris	Surf and Bird	4" x 6"	3.00	10	0	Graves	19
7	7	Graves, Morris	Surf and Bird	7" x 10"	5.00	8	0	Graves	19
8	8	Horiuchi, Paul	Color Floating in Time	4" x 6"	3.00	3	10	Horiuchi	18
9	9	Horiuchi, Paul	Into Time	4" x 6"	3.00	2	10	Horiuchi	18
10	10	Horiuchi, Paul	Memories IV	4" x 6"	3.00	2	10	Horiuchi	18
11	11	Horiuchi, Paul	Yellow Covers Blue	4" x 6"	3.00	3	10	Horiuchi	18
12	12	Kandinsky, Wassily	Der Blaue Reiter	4" x 6"	3.00	5	0	Kandinsky	2
13	13	Klee, Paul	Angelus Novus	4" x 6"	3.00	4	10	Klee	3
14	14	Matisse, Henri	The Dance	4" x 6"	3.00	3	10	Matisse	4
15	15	Matisse, Henri	Woman with a Hat	4" x 6"	3.00	2	10	Matisse	4
16	16	Sargent, John Singer	Claude Monet Painting	4" x 6"	3.00	3	10	Sargent	11
17	17	Sargent, John Singer	Spanish Dancer	4" x 6"	3.00	3	10	Sargent	11
18	18	Tobey, Mark	Blue Interior	4" x 6"	3.00	5	0	Tobey	17
19	19	Tobey, Mark	Broadway Boggie	4" x 6"	3.00	5	0	Tobey	17
20	20	Tobey, Mark	Farmer's Market #2	3" x 5"	2.50	3	10	Tobey	17
21	21	Tobey, Mark	Farmer's Market #2	4" x 6"	3.00	10	10	Tobey	17
22	22	Tobey, Mark	Farmer's Market #2	7" x 10"	5.00	7	10	Tobey	17
23	23	Tobey, Mark	Forms in Progress I	4" x 6"	3.00	5	0	Tobey	17
24	24	Tobey, Mark	Forms in Progress II	4" x 6"	3.00	2	10	Tobey	17
25	25	Tobey, Mark	The Woven World	4" x 6"	3.00	5	0	Tobey	17
26	26	Tobey, Mark	Universal Field	4" x 6"	3.00	5	0	Tobey	17

FIGURE 10A-61

The POSTCARDSwithID$ Table with the Populated Columns

SQL-UPDATE-CH10A-02 only works if artists have unique last names—if they do not, we can simply include more columns in the comparison.

The updated POSTCARDSwithID$ table is shown in Figure 10A-61, with the new column data displayed in the new columns. We still have work to do to integrate the POSTCARDSwithID$ table data into the VRG database, and we will continue that work in Exercise 10A.41.

Stored Procedures

When using SQL Server 2000, stored procedures must be written in Transact-SQL. Starting with the release of SQL Server 2005, stored procedures and triggers can be written in any of the .NET or CLR languages, such as Visual Basic.NET, Visual C#.NET, and Visual C++.NET .Transact-SQL continues to be supported and will have improved features and functions for error handling, as well as other language upgrades, and the Transact-SQL shown here will operate in SQL Server 2005, SQL Server 2008 R2, and SQL Server 2012, as well as in SQL Server 2017. In the long run, however, Visual Basic.NET, Visual C#.NET, or Visual C++.NET may become better choices for stored procedures and triggers.

As with other database structures, you can write a stored procedure in an SQL script text file and process the commands using the SQL Server Management Studio. However, there is, as with user-defined functions, one little gotcha. The first time you create a stored procedure in an SQL script, you use the **SQL CREATE PROCEDURE statement**. Subsequently, if you change the procedure, use the **SQL ALTER PROCEDURE statement**. Otherwise, you will get an error message saying that the procedure already exists when you execute the modified procedure code. Note that as of SQL Server 2016 SP1, these statements can be combined into the **SQL CREATE OR ALTER PROCEDURE statement**. If you use this statement you can create or alter your code without having to change the CREATE keyword to the ALTER keyword.

You can also create a stored procedure within the SQL Server Management Studio by expanding a database's Programmability folder object, right-clicking Stored Procedures, and selecting New Stored Procedure. If you do this, however, SQL Server simply opens an editable draft template in a tabbed document window.

BY THE WAY For stored procedures, the first BEGIN statement, the final END statement, and the semicolon (;) after the final END statement in Figure 10A-62 and other figures showing stored procedure code are optional, but we prefer to include them to use an enclosing BEGIN . . . END block and to match the syntax of other SQL statements. For more information, see the T-SQL CREATE PROCEDURE documentation at *https://docs.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql*.

The Stored Procedure InsertCustomerAndInterests

In our preceding discussion of Transact-SQL, we used as our example the need to enter data for a new customer and the artists of interest to that customer. The code segments we wrote were specifically tied to the data we used and thus of limited use. Is there a way to write a general block of code that could be used for more than one customer? Yes, and that block of code is a stored procedure.

Figure 10A-62 shows the SQL code for the InsertCustomerAndInterests stored procedure. This stored procedure generalizes our previous code and can be used to insert data for any new customer into CUSTOMER and then store data for that

FIGURE 10A-62

The SQL Statements for the InsertCustomerAndInterests Stored Procedure

```
CREATE PROCEDURE InsertCustomerAndInterests
              @NewLastName     Char(25),
              @NewFirstName    Char(25),
              @NewAreaCode     Char(3),
              @NewPhoneNumber  Char(8),
              @NewEmailAddress Varchar(100),
              @Nationality     Char(30)
AS
BEGIN
       DECLARE    @RowCount     AS  Int
       DECLARE    @ArtistID     AS  Int
       DECLARE    @CustomerID   AS  Int

       -- Check to see if Customer already exists in database

       SELECT @RowCount = COUNT(*)
       FROM   dbo.CUSTOMER
       WHERE  LastName = @NewLastName
          AND FirstName = @NewFirstName
          AND AreaCode = @NewAreaCode
          AND PhoneNumber = @NewPhoneNumber
          AND EmailAddress = @NewEmailAddress;

       -- IF @RowCount > 0 THEN Customer already exists.
       IF (@RowCount > 0)
          BEGIN
             PRINT '*******************************************************'
             PRINT ''
             PRINT '    The Customer is already in the database. '
             PRINT ''
             PRINT '    Customer Last Name          =   '+@NewLastName
             PRINT '    Customer First Name         =   '+@NewFirstName
             PRINT ''
             PRINT '*******************************************************'
             RETURN
          END
```

```
        -- IF @RowCount = 0 THEN Customer does not exist in database.
        ELSE
            BEGIN
            -- Insert new Customer data.
                INSERT INTO dbo.CUSTOMER
                    (LastName, FirstName, AreaCode, PhoneNumber, EmailAddress)
                    VALUES(
                    @NewLastName, @NewFirstName, @NewAreaCode,
                    @NewPhoneNumber, @NewEmailAddress);
                -- Get new CustomerID surrogate key value.
                SELECT @CustomerID = CustomerID
                FROM   dbo.CUSTOMER
                WHERE  LastName = @NewLastName
                    AND FirstName = @NewFirstName
                    AND AreaCode = @NewAreaCode
                    AND PhoneNumber = @NewPhoneNumber
                    AND EmailAddress = @NewEmailAddress;
                PRINT '*******************************************************'
                PRINT ''
                PRINT '   The new Customer is now in the database. '
                PRINT ''
                PRINT '   Customer Last Name         =  '+@NewLastName
                PRINT '   Customer First Name        =  '+@NewFirstName
                PRINT ''
                PRINT '*******************************************************'
                -- Create intersection record for each appropriate Artist.
                DECLARE ArtistCursor CURSOR FOR
                        SELECT ArtistID
                        FROM   dbo.ARTIST
                        WHERE  Nationality=@Nationality
                --Process each appropriate Artist
                OPEN   ArtistCursor
                    FETCH NEXT FROM ArtistCursor INTO @ArtistID
                    WHILE @@FETCH_STATUS = 0
                    BEGIN
                         INSERT INTO dbo.CUSTOMER_ARTIST_INT
                             (ArtistID, CustomerID)
                             VALUES(@ArtistID, @CustomerID);
                             PRINT '**************************************************'
                             PRINT ''
                             PRINT '   New CUSTOMER_ARTIST_INT row added. '
                             PRINT ''
                             PRINT '   ArtistID     =  '+CONVERT(Char(6), @ArtistID)
                             PRINT '   CustomerID   =  '+CONVERT(Char(6), @CustomerID)
                             PRINT ''
                             PRINT '**************************************************'
                         FETCH NEXT FROM ArtistCursor INTO @ArtistID
                    END
                CLOSE  ArtistCursor
                DEALLOCATE   ArtistCursor
            END
END;
```

FIGURE 10A-62
Continued

customer in CUSTOMER_ARTIST_INT, linking the customer to all artists with a particular nationality.

Six parameters are input to the procedure: @NewLastName, @NewFirstName, @NewEmailAddress, @NewAreaCode, @NewPhoneNumber, and @Nationality. The first five parameters are the new customer data, and the sixth one is the nationality of the artists for which the new customer has an interest. The stored procedure also uses three variables: @RowCount, @ArtistID, and @CustomerID. These variables are used to store values of the number of rows, the value of the ArtistID primary key, and the value of the CustomerID primary key, respectively.

The first task performed by this stored procedure is to determine whether the customer already exists. If the value of @RowCount in the first SELECT statement is greater than zero, then a row for that customer already exists. In this case, nothing is done, and the stored procedure prints an error message and exits (using the RETURN command). As stated earlier, the error message is visible in the Microsoft SQL Server Management Studio, but it generally would not be visible to application programs that invoked this procedure. Instead, a parameter or other facility needs to be used to return the error message back to the user via the application program. That topic is beyond the scope of the present discussion, but we will send a message back to the Microsoft SQL Server Management Studio to mimic such actions and provide a means to make sure our stored procedures are working correctly.

If the customer does not already exist, the procedure inserts the new data into the table dbo.CUSTOMER, and then a new value of CustomerID is read into the variable @CustomerID. Internally, SQL Server adds a prefix to the table name that shows the name of the user who created it. Here the prefix *dbo* is used to ensure that the CUSTOMER table created by the database owner (dbo) is processed. Without the dbo prefix, if the user invoking the stored procedure had created a table named CUSTOMER, then the user's table and not the dbo's table would be used.

The purpose of the second SELECT in Figure 10A-62 is to obtain the value of the surrogate key CustomerID that was created by the INSERT statement. Another option is to use the **SQL Server function @@Identity**, which provides the value of the most recently created surrogate key value. Using this function, you could replace the second SELECT statement with the following expression:

```
SET @CustomerID = @@Identity
```

To create the appropriate intersection table rows, an SQL cursor named ArtistCursor is created on an SQL statement that obtains all ARTIST rows where Nationality equals the parameter @Nationality. The cursor is opened and positioned on the first row by calling FETCH NEXT, and then the cursor is processed in a WHILE loop. In this loop, statements between BEGIN and END are iterated until SQL Server signals the end of the data by setting the value of the SQL Server function @@FETCH_STATUS to -1. Upon each iteration of the WHILE loop, a new row is inserted into the intersection table CUSTOMER_ARTIST_INT. The FETCH NEXT statement at the end of the block moves the cursor to the next row.

To create the InsertCustomerAndInterests stored procedure in the VRG database, create a new SQL script named *DBP-e15-VRG-Create-Stored-Procedures.sql* containing the SQL code in Figure 10A-62. Include a beginning comments section similar to the one shown in Figure 10A-27. Use the Parse button to check your SQL code, and after it parses correctly, save the completed code before running it. Finally, check to make sure that the VRG database is selected in the Available Databases list, and then use the Execute button to create the stored procedure.

To invoke the InsertCustomerAndInterests stored procedure for Michael Bench, we use the following SQL statement:

```
/* *** SQL-EXEC-CH10A-01 *** */
EXEC InsertCustomerAndInterests
     @NewLastName = 'Bench', @NewFirstName = 'Michael',
     @NewEmailAddress = 'Michael.Bench@somewhere.com',
     @NewAreaCode = '206', @NewPhoneNumber = '876-8822',
     @Nationality = 'French';
```

BY THE WAY Before we test any new functionality of a database, such as a stored procedure or a trigger, it is always a good idea to refresh the content of the database in Microsoft SQL Server Management Studio. This can be done by using the Refresh command in the object shortcut menu. For example, after creating the InsertCustomerAndInterests stored procedure, right-click the VRG Stored Procedures folder in the Programmability folder and then click the Refresh command. Then expand the Stored Procedures folder and make sure the stored procedure object (shown as dbo.InsertCustomerAndInterests) is visible before running any test data.

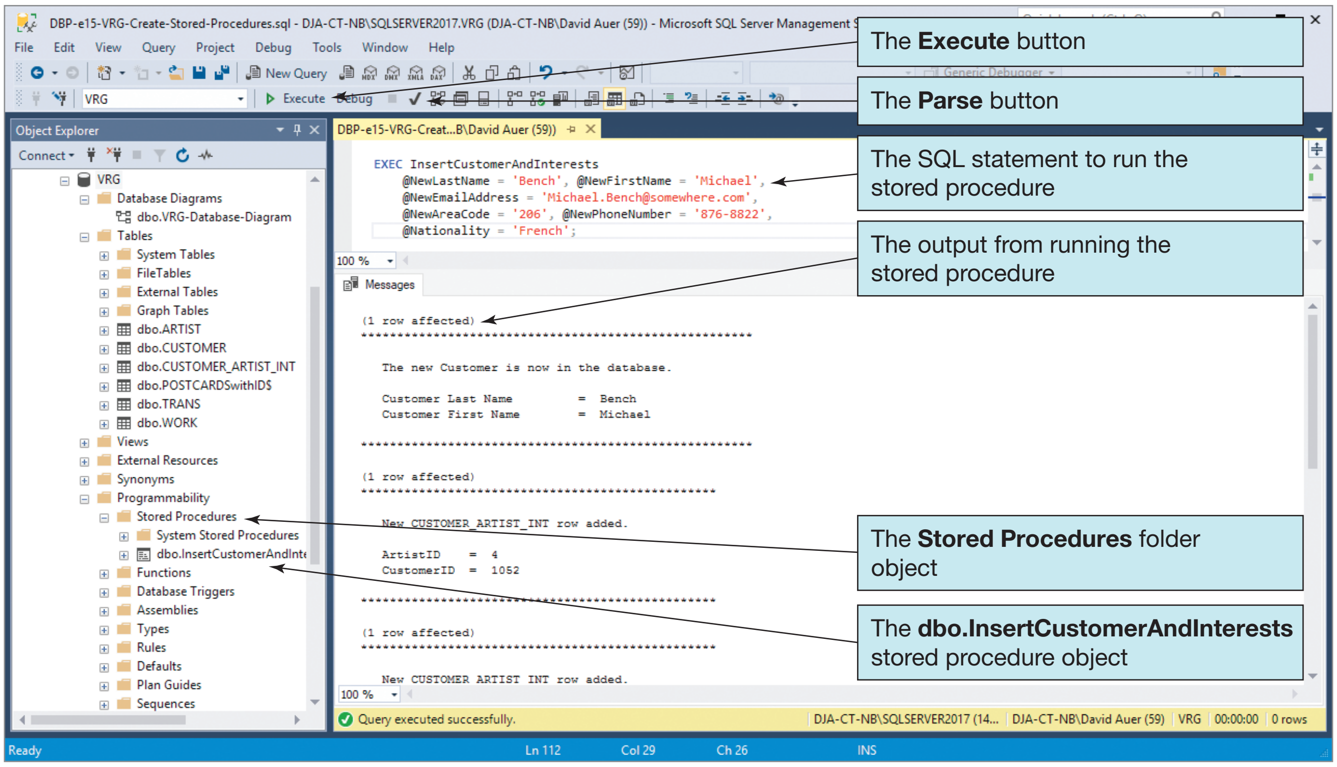
Figure 10A-63 shows the execution of the stored procedure in the SQL Server Management Studio. Notice how our sections of PRINT commands have produced the necessary output so we can see what actions were taken. If we now wanted to check the tables themselves, we could do so, but that is not necessary at this point. In the output in Figure 10A-63, we see that customer Michael Bench has been added to the CUSTOMER table and that new rows (a total of two, though we would have to scroll through the output to see that there are no more than two) have been inserted into the CUSTOMER_ARTIST_INT table.

BY THE WAY You can include the preceding EXEC InsertCustomerAndInterests statement in your *DBP-e15-VRG-Create-Stored-Procedures.sql* script and run it from there using the highlighting technique we illustrated in the section on SQL views. There we noted that when there are multiple SQL statements in an open SQL script file, you can still run them one at a time by highlighting *just* the SQL statement or statements that you want to use. The Microsoft SQL Server Management Studio will then apply actions such as parsing (using the Parse button) and executing (using the Execute button) to only the highlighted SQL statement or statements.

This is a good trick to know because it allows you to store multiple SQL statements (e.g., a set of queries) in one SQL script for convenience but still control which statements are actually executed. Even better, if you highlight more than one SQL statement, the highlighted *set* of commands can be controlled the same way.

FIGURE 10A-63

Running the InsertCustomerAndInterests Stored Procedure


DBP-e15-VRG-Create-Stored-Procedures.sql - DJA-CT-NB\SQLSERVER2017.VRG (DJA-CT-NB\David Auer (59)) - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
Execute
Debug
Object Explorer
VRG
Database Diagrams
dbo.VRG-Database-Diagram
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.ARTIST
dbo.CUSTOMER
dbo.CUSTOMER_ARTIST_INT
dbo.POSTCARDSwithIDS
dbo.TRANS
dbo.WORK
Views
External Resources
Synonyms
Programmability
Stored Procedures
System Stored Procedures
dbo.InsertCustomerAndInte
Functions
Database Triggers
Assemblies
Types
Rules
Defaults
Plan Guides
Sequences
DBP-e15-VRG-Creat...B\David Auer (59))
Messages
The **Execute** button
The **Parse** button
The SQL statement to run the stored procedure
The output from running the stored procedure
The **Stored Procedures** folder object
The **dbo.InsertCustomerAndInterests** stored procedure object

```
EXEC InsertCustomerAndInterests
    @NewLastName = 'Bench', @NewFirstName = 'Michael',
    @NewEmailAddress = 'Michael.Bench@somewhere.com',
    @NewAreaCode = '206', @NewPhoneNumber = '876-8822',
    @Nationality = 'French';
```

```
(1 row affected)
*********************************************************

   The new Customer is now in the database.

   Customer Last Name         =  Bench
   Customer First Name        =  Michael

*********************************************************

(1 row affected)
****************************************************

   New CUSTOMER_ARTIST_INT row added.

   ArtistID    =  4
   CustomerID  =  1052

****************************************************

(1 row affected)
****************************************************

   New CUSTOMER ARTIST INT row added.
```

Query executed successfully. | DJA-CT-NB\SQLSERVER2017 (14... | DJA-CT-NB\David Auer (59) | VRG | 00:00:00 | 0 rows

Ready Ln 112 Col 29 Ch 26 INS

The Stored Procedure InsertCustomerWithTransaction

Now we will write a stored procedure that inserts data for a new customer, records a purchase, and creates an entry in the CUSTOMER_ARTIST_INT table. We will name this stored procedure *InsertCustomerWithTransaction*, and the necessary SQL code is shown in Figure 10A-64. This procedure receives nine parameters with data about the new customer and about the customer's purchase. We will use this procedure to discuss transaction processing in SQL Server 2017.

The first action is to see whether the customer already exists. If so, the procedure exits with an error message. If the customer does not exist, this procedure then starts a

FIGURE 10A-64

The SQL Statements for the InsertCustomerWith Transaction Stored Procedure

```
CREATE PROCEDURE InsertCustomerWithTransaction
             @NewCustomerLastName          Char(25),
             @NewCustomerFirstName         Char(25),
             @NewCustomerAreaCode          Char(3),
             @NewCustomerPhoneNumber       Char(8),
             @NewCustomerEmailAddress      Varchar(100),
             @ArtistLastName               Char(25),
             @WorkTitle                    Char(35),
             @WorkCopy                     Char(12),
             @TransSalesPrice              Numeric(8,2)

AS
BEGIN
      DECLARE   @RowCount        AS  Int,
                @ArtistID        AS  Int,
                @CustomerID      AS  Int,
                @WorkID          AS  Int,
                @TransactionID   AS  Int

      -- Check to see if Customer already exists in database

      SELECT    @RowCount = COUNT(*)
      FROM      dbo.CUSTOMER
      WHERE     LastName = @NewCustomerLastName
         AND    FirstName = @NewCustomerFirstName
         AND    AreaCode = @NewCustomerAreaCode
         AND    PhoneNumber = @NewCustomerPhoneNumber
         AND    EmailAddress = @NewCustomerEmailAddress;

      -- IF @RowCount > 0 THEN Customer already exists.
      IF (@RowCount > 0)
         BEGIN
            PRINT '******************************************************'
            PRINT ''
            PRINT '   The Customer is already in the database. '
            PRINT ''
            PRINT '   Customer Last Name     =   '+@NewCustomerLastName
            PRINT '   Customer First Name    =   '+@NewCustomerFirstName
            PRINT ''
            PRINT '******************************************************'
            RETURN
         END
      -- IF @RowCount = 0 THEN Customer does not exist in database.
      ELSE
         -- Start transaction - Rollback everything if unable to complete it.
         BEGIN TRANSACTION
            -- Insert new Customer data.
            INSERT INTO dbo.CUSTOMER
```

```
       (LastName, FirstName, AreaCode, PhoneNumber, EmailAddress)
       VALUES(
       @NewCustomerLastName, @NewCustomerFirstName,
       @NewCustomerAreaCode, @NewCustomerPhoneNumber, @NewCustomerEmailAddress);
    -- Get new CustomerID surrogate key value.
    SELECT    @CustomerID = CustomerID
    FROM      dbo.CUSTOMER
    WHERE     LastName = @NewCustomerLastName
       AND    FirstName = @NewCustomerFirstName
       AND    AreaCode = @NewCustomerAreaCode
       AND    PhoneNumber = @NewCustomerPhoneNumber
       AND    EmailAddress =  @NewCustomerEmailAddress;

    -- Get ArtistID surrogate key value, check for validity.
    SELECT    @ArtistID = ArtistID
    FROM      dbo.ARTIST
    WHERE     LastName = @ArtistLastName;

    IF @ArtistID IS NULL
       BEGIN
          PRINT '*******************************************************'
          PRINT ''
          PRINT '   Invalid ArtistID '
          PRINT ''
          PRINT '*******************************************************'
          ROLLBACK TRANSACTION
          RETURN
       END

    -- Get WorkID surrogate key value, check for validity.
    SELECT    @WorkID = WorkID
    FROM      dbo.WORK
    WHERE     ArtistID = @ArtistID
       AND    Title = @WorkTitle
       AND    Copy = @WorkCopy;

    IF @WorkID IS NULL
       BEGIN
          PRINT '*******************************************************'
          PRINT ''
          PRINT '   Invalid WorkID '
          PRINT ''
          PRINT '*******************************************************'
          ROLLBACK TRANSACTION
          RETURN
       END

    -- Get TransID surrogate key value, check for validity.
    SELECT    @TransactionID = TransactionID
    FROM      dbo.TRANS
    WHERE     WorkID = @WorkID
       AND    SalesPrice IS NULL;

    IF @TransactionID IS NULL
       BEGIN
```

FIGURE 10A-64
Continued

```
                PRINT '*******************************************************'
                PRINT ''
                PRINT '   Invalid TransactionID '
                PRINT ''
                PRINT '*******************************************************'
                ROLLBACK TRANSACTION
                RETURN
            END

        -- All surrogate key values are OK, complete the transaction
        BEGIN
            -- Update TRANS row
            UPDATE     dbo.TRANS
            SET        DateSold = GETDATE(),
                       SalesPrice = @TransSalesPrice,
                       CustomerID = @CustomerID
            WHERE      TransactionID = @TransactionID;
            -- Create CUSTOMER_ARTIST_INT row
            INSERT INTO dbo.CUSTOMER_ARTIST_INT (CustomerID, ArtistID)
                   VALUES(@CustomerID, @ArtistID);
        END
    -- Commit the Transaction
    COMMIT TRANSACTION
    -- The transaction is completed. Print output
    BEGIN
        -- Print Customer results.
        PRINT '*******************************************************'
        PRINT ''
        PRINT '   The new Customer is now in the database. '
        PRINT ''
        PRINT '   Customer Last Name  =  '+@NewCustomerLastName
        PRINT '   Customer First Name =  '+@NewCustomerFirstName
        PRINT ''
        PRINT '*******************************************************'
        -- Print Transaction result
        PRINT ''
        PRINT '   Transaction complete. '
        PRINT ''
        PRINT '   TransactionID    =  '+CONVERT(Char(6), @TransactionID)
        PRINT '   ArtistID         =  '+CONVERT(Char(6), @ArtistID)
        PRINT '   WorkID           =  '+CONVERT(Char(6), @WorkID)
        PRINT '   Sales Price      =  '+CONVERT(Char(12), @TransSalesPrice)
        PRINT ''
        PRINT '*******************************************************'
        -- Print CUSTOMER_ARTIST_INT update
        PRINT ''
        PRINT '   New CUSTOMER_ARTIST_INT row added. '
        PRINT ''
        PRINT '   ArtistID         =  '+CONVERT(Char(6), @ArtistID)
        PRINT '   CustomerID       =  '+CONVERT(Char(6), @CustomerID)
        PRINT ''
        PRINT '*******************************************************'
    END
END;
```

FIGURE 10A-64

Continued

transaction with the **Transact-SQL BEGIN TRANSACTION command**. Recall from Chapter 9 that transactions ensure that all of the database activity is committed atomically; either all of the updates occur or none of them does. The transaction begins, and the new customer row is inserted. The new value of CustomerID is obtained, as shown in the

InsertCustomerAndInterests stored procedure. Next, the procedure checks to determine whether ArtistID, WorkID, and TransactionID are valid. If any are invalid, the transaction is rolled back using the **Transact-SQL ROLLBACK TRANSACTION command**.

If all the surrogate key values are valid, two actions in the transaction are completed. First, an UPDATE statement updates DateSold, SalesPrice, and CustomerID in the appropriate TRANS row. DateSold is set to system date via the **Transact-SQL GETDATE() function**, SalesPrice is set to the value of @TransSalesPrice, and CustomerID is set to the value of @CustomerID. Second, a row is added to CUSTOMER_ARTIST_INT to record the customer's interest in this artist.

If everything proceeds normally to this point, the transaction is committed using the **Transact-SQL COMMIT TRANSACTION command**. After, and only after, the transaction is committed, we print the results messages. Do not be confused by the presence of two transactions here. One of them, which already exists in the TRANS table, records the purchase (and eventual sale) of a work by VRG. The transaction being committed (or possibly rolled back) in the stored procedure encompasses the changes being made to the database tables as a result of adding a new customer with a purchase of an existing work.

To create the InsertCustomerWithTransaction stored procedure in the VRG database, add the SQL code in Figure 10A-60 to your *DBP-e15-VRG-Create-Stored-Procedures.sql* script. Include a comment to separate the code sections in this script file. Use the highlighting technique described in the preceding By the Way feature to parse and execute the SQL code.

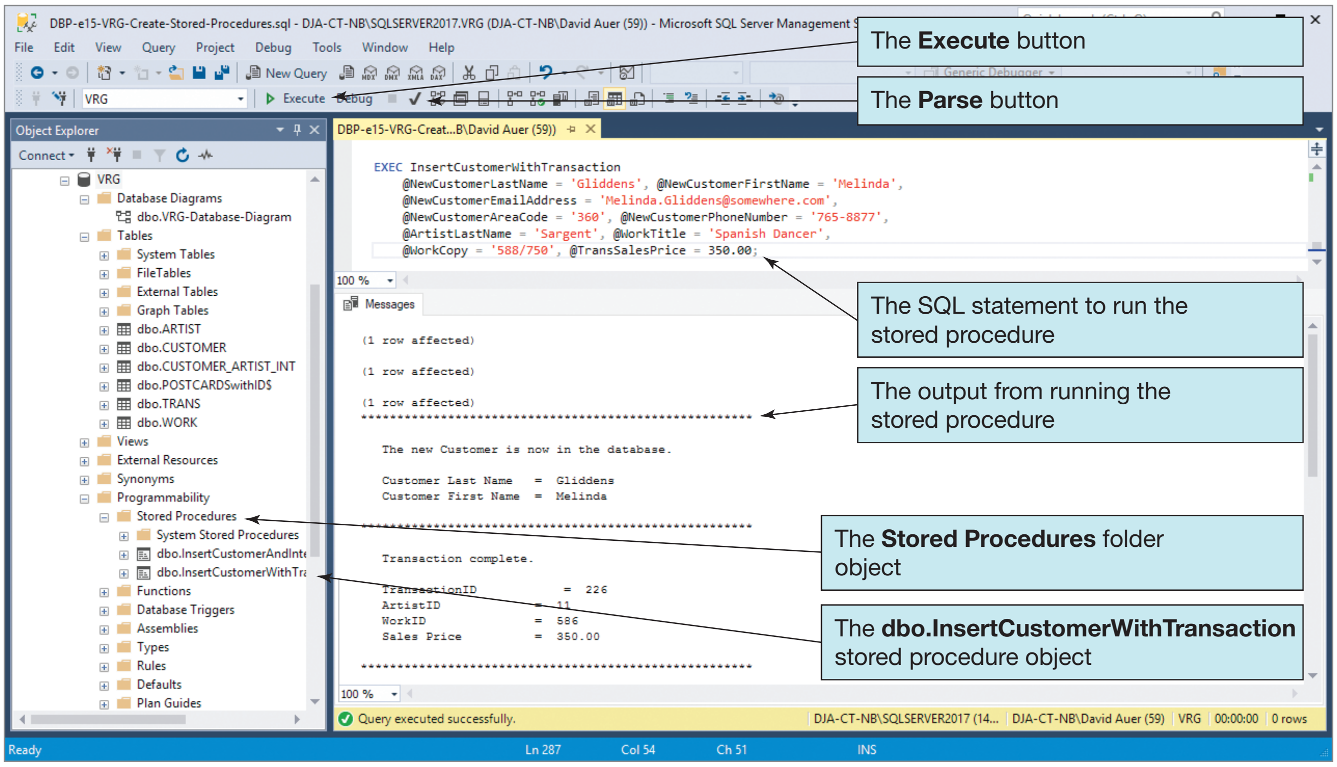
To test the InsertCustomerWithTransaction stored procedure, we will record the following purchase by our next new customer, Melinda Gliddens, who just bought a print of John Singer Sargent's *Spanish Dancer* for $350.00. The SQL statement is:

```
/* *** SQL-EXEC-CH10A-02 *** */
EXEC InsertCustomerWithTransaction
     @NewCustomerLastName = 'Gliddens',
     @NewCustomerFirstName = 'Melinda',
     @NewCustomerEmailAddress = 'Melinda.Gliddens@somewhere.com',
     @NewCustomerAreaCode = '360',
     @NewCustomerPhoneNumber = '765-8877',
     @ArtistLastName = 'Sargent', @WorkTitle = 'Spanish Dancer',
     @WorkCopy = '588/750', @TransSalesPrice = 350.00;
```

To execute this EXEC InsertCustomerWithTransaction statement, add the SQL statement to your *VRG-Create-Stored-Procedures.sql* script. Include a comment to separate the new SQL statement. Use the highlighting technique described in the preceding By the Way feature to parse and execute the SQL statement. Figure 10A-65 shows the invocation of this procedure using sample data.

BY THE WAY If we now look at the SQL code that has actually been stored for these two stored procedures (by right-clicking the stored procedure object and then choosing Modify), we will find that SQL Server has added the following lines before the code we wrote:

```
USE [VRG]
GO
/****** Object: StoredProcedure [ {StoredProcedureName}]
     Script Date: {Date and Time created or altered} ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```



```
EXEC InsertCustomerWithTransaction
    @NewCustomerLastName = 'Gliddens', @NewCustomerFirstName = 'Melinda',
    @NewCustomerEmailAddress = 'Melinda.Gliddens@somewhere.com',
    @NewCustomerAreaCode = '360', @NewCustomerPhoneNumber = '765-8877',
    @ArtistLastName = 'Sargent', @WorkTitle = 'Spanish Dancer',
    @WorkCopy = '588/750', @TransSalesPrice = 350.00;
```

```
(1 row affected)

(1 row affected)

(1 row affected)
******************************************************

  The new Customer is now in the database.

  Customer Last Name    =  Gliddens
  Customer First Name   =  Melinda

******************************************************

  Transaction complete.

  TransactionID             =  226
  ArtistID              =  11
  WorkID                =  586
  Sales Price           =  350.00

******************************************************
```

Query executed successfully.

DBP-e15-VRG-Create-Stored-Procedures.sql - DJA-CT-NB\SQLSERVER2017.VRG (DJA-CT-NB\David Auer (59)) - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
VRG
Execute Debug
Object Explorer
Connect
VRG
Database Diagrams
dbo.VRG-Database-Diagram
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.ARTIST
dbo.CUSTOMER
dbo.CUSTOMER_ARTIST_INT
dbo.POSTCARDSwithIDS
dbo.TRANS
dbo.WORK
Views
External Resources
Synonyms
Programmability
Stored Procedures
System Stored Procedures
dbo.InsertCustomerAndInte
dbo.InsertCustomerWithTra
Functions
Database Triggers
Assemblies
Types
Rules
Defaults
Plan Guides
DBP-e15-VRG-Creat...B\David Auer (59))
100 %
Messages
DJA-CT-NB\SQLSERVER2017 (14... DJA-CT-NB\David Auer (59) VRG 00:00:00 0 rows
Ready Ln 287 Col 54 Ch 51 INS
The **Execute** button
The **Parse** button
The SQL statement to run the stored procedure
The output from running the stored procedure
The **Stored Procedures** folder object
The **dbo.InsertCustomerWithTransaction** stored procedure object

FIGURE 10A-65

Running the InsertCustomerWithTransaction Stored Procedure

The **Transact-SQL USE [{DatabaseName}] command** tells the stored procedure to use the VRG database when it is called. This is the SQL command equivalent of selecting the database name in the Available Databases drop-down list in the Microsoft SQL Server Management Studio.

The **Transact-SQL SET ANSI_NULLS ON command** specifies how SQL server handles comparisons of NULL values using equal (=) and not equal (<>).[10] The **Transact-SQL SET QUOTED_IDENTIFIER ON command** specifies that object identifiers (table names, column names, etc.) can be enclosed in double quotes ("), which allows the use of SQL reserved words as object names. For example, we could run the following SQL statement:

```
SELECT    "Select"
FROM      "FROM"
WHERE     "Where" = 'San Francisco';
```

Note that literals (San Francisco in this example) are still enclosed in single quotes.[11] Finally, the **GO command** is not a Transact-SQL statement but rather a command used by the Microsoft SQL Server Management Studio (and other SQL utilities) to mark the end of a batch of commands so the utility can process sections of the code (as marked by the GO commands) separately instead of all at once. Declared variables (@RowCount) only exist within a section of code delineated by the GO statement and are cleared after that group of statements is run.[12]

Triggers

SQL Server 2017 supports the INSTEAD OF and AFTER triggers but not a BEFORE trigger. A table may have one or more AFTER triggers for insert, update, and delete actions. AFTER triggers may not be assigned to views. A view or table may have, at most, one INSTEAD OF trigger for each triggering by an insert, an update, or a delete action.

[10] For more information, see *https://docs.microsoft.com/en-us/sql/t-sql/statements/set-ansi-nulls-transact-sql*.
[11] For more information, see *https://docs.microsoft.com/en-us/sql/t-sql/statements/set-quoted-identifier-transact-sql*.
[12] For more information, see *https://docs.microsoft.com/en-us/sql/t-sql/ language-elements/sql-server-utilities-statements-go*.

In SQL Server, triggers can roll back the transactions that caused them to be fired. When a trigger executes a ROLLBACK or ROLLBACK TRANSACTION command (two versions of the same command), all work done by the transaction that caused the trigger to be fired will be rolled back. If the trigger contains instructions after the ROLLBACK command, those instructions will be executed. However, any instructions in the transaction after the statement that caused the trigger to be fired will not be executed.

For insert and update triggers, the new values for every column of the table being processed will be stored in a pseudotable named *inserted*. If, for example, a new row is being added to the ARTIST table, the pseudotable *inserted* will have five columns: LastName, FirstName, Nationality, DateOfBirth, and DateDeceased. Similarly, for update and delete commands, the old values for every column of the table being updated or deleted will be stored in the pseudotable named *deleted*. You will see how to use these pseudotables in the examples that follow.

The next four sections illustrate four triggers for each of the trigger functions described in Chapter 7. The best way to create these triggers is by keying them into an SQL script in the Microsoft SQL Server Management Studio SQL editor. The first time you create a stored procedure in an SQL script, you use the **SQL CREATE TRIGGER statement**. Subsequently, if you change the procedure, use the **SQL ALTER TRIGGER statement**. Otherwise, you will get an error message saying that the procedure already exists when you execute the modified procedure code. However, just as with the SQL CREATE FUNCTION and CREATE PROCEDURE statements, as of SQL Server 2016 SP1, these statements can be combined into the **SQL CREATE OR ALTER TRIGGER statement**. If you use this statement you can create or alter your code without having to change the CREATE keyword to the ALTER keyword. You can create a trigger by right-clicking the Triggers folder object for each table, but this only creates a skeleton trigger template for you to edit. It is better to create and test your own SQL script. For the VRG database, use an SQL script named *DBP-e15-VRG-Create-Triggers.sql*, with introductory comments and comment lines to separate the triggers and the SQL statements used to test them. Use the previously discussed highlighting technique to control which SQL statement is actually being run.

BY THE WAY For triggers, the first BEGIN statement, the final END statement, and the semicolon (;) after the final END statement in Figure 10A-66 and other figures showing trigger code are optional, but we prefer to include them to use an enclosing BEGIN . . . END block and to match the syntax of other SQL statements. For more information, see the T-SQL CREATE TRIGGER documentation at *https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql*.

A Trigger for Setting Default Values

Triggers can be used to set default values that are more complex than those that can be set with the Default constraint on a column definition. For example, the View Ridge Gallery has a pricing policy that says that the default AskingPrice of an artwork depends on whether the piece has been in the gallery before. If not, the default AskingPrice is twice the AcquisitionPrice. If the artwork has been in the gallery before, the default price is the larger of twice the AcquisitionPrice or the AcquisitionPrice plus the average net gain of the work in the past. The TRANS_AfterInsertSetAskingPrice trigger shown in Figure 10A-66 implements this pricing policy.

Note the use of the AFTER keyword in this trigger code. We can use the keywords FOR or AFTER here, but both of these indicate an AFTER trigger, and SQL Server does not support BEFORE triggers.

In the trigger, after the variables are declared, the new values of WorkID and AcquisitionPrice are obtained from the *inserted* pseudotable. Then a SELECT FROM dbo.TRANS is executed to count the number of rows with the given WorkID. This trigger uses the system function @@rowCount, which contains the number of rows processed in the preceding

```
CREATE TRIGGER TRANS_AfterInsertSetAskingPrice
   ON TRANS
   AFTER INSERT

AS
BEGIN
   SET NOCOUNT ON;

   DECLARE   @PriorRowCount      AS Int,
             @WorkID             AS Int,
             @TransactionID      AS Int,
             @AcquisitionPrice   AS Numeric(8,2),
             @NewAskingPrice     AS Numeric(8,2),
             @SumNetProfit       AS Numeric(8,2),
             @AvgNetProfit       AS Numeric(8,2)

   SELECT @TransactionID = TransactionID,
          @AcquisitionPrice = AcquisitionPrice,
          @WorkID = WorkID
   FROM   inserted

   -- First find if work has been here before.

   SELECT    *
   FROM   dbo.TRANS AS T
   WHERE  T.WorkID = @WorkID;

   -- Since this is an AFTER trigger, @@Rowcount includes the new row.

   SET @PriorRowCount = (@@Rowcount - 1)

   IF (@PriorRowCount = 0)
      -- This is first time work has been in the gallery.
      -- Set @NewAskingPrice to twice the acquisition cost.
      SET @NewAskingPrice = (2 * @AcquisitionPrice)

   ELSE
      -- The work has been here before
      -- We have to determine the value of @NewAskingPrice
      BEGIN
         SELECT    @SumNetProfit = SUM(NetProfit)
         FROM      dbo.ArtistWorkNetView AWNV
         WHERE     AWNV.WorkID = @WorkID
         GROUP BY  AWNV.WorkID;

         SET @AvgNetProfit = (@SumNetProfit / @PriorRowCount);

            -- Now choose larger value for the new AskingPrice.

         IF ((@AcquisitionPrice + @AvgNetProfit)
               > (2 * @AcquisitionPrice))
            SET @NewAskingPrice = (@AcquisitionPrice + @AvgNetProfit)
         ELSE
            SET @NewAskingPrice = (2 * @AcquisitionPrice)
      END
   -- Update TRANS with the value of AskingPrice

   UPDATE dbo.TRANS
   SET    AskingPrice = @NewAskingPrice
   WHERE  TransactionID = @TransactionID
```

FIGURE 10A-66

The SQL Statements for the TRANS_AfterInsert SetAskingPrice Trigger

```
        -- The INSERT is completed. Print output
        BEGIN
               PRINT '*******************************************************'
               PRINT ''
               PRINT '   INSERT complete. '
               PRINT ''
               PRINT '   TransactionID      =  '+CONVERT(Char(6), @TransactionID)
               PRINT '   WorkID             =  '+CONVERT(Char(6), @WorkID)
               PRINT '   Acquisition Price  =  '+CONVERT(Char(12), @AcquisitionPrice)
               PRINT '   Asking Price       =  '+CONVERT(Char(12), @NewAskingPrice)
               PRINT ''
               PRINT '*******************************************************'
        END
     END;
```

FIGURE 10A-66

Continued

Transact-SQL statement. Its value must be checked immediately after the statement is executed or saved in a variable to be checked later. Here the trigger immediately uses the value of @@rowCount.

The variable @PriorRowCount is set to @@rowCount minus one because this is an AFTER trigger, and the new row will already be in the database. The expression (@@rowCount-1) is the correct number of qualifying TRANS rows that were in the database prior to the insert.

We test to see if the work is new. If so, @NewAskingPrice is then set to twice the AcquisitionPrice. If the work has been in the gallery before, we must calculate which value of @NewAskingPrice to use. Thus, if @PriorRowCount is greater than zero, there were TRANS rows for this work in the database, and the SQL view ArtistWorkNetView (see pages 358–360) is used to obtain the sum of the WorkNetProfit for the work. The AVG built-in function cannot be used because it will compute the average using @@rowCount rather than @PriorRowCount. Next, the two possible values of @NewAskingPrice are compared, and @NewAskingPrice is set to the larger value. Finally, an update is made to TRANS to set the computed value of @NewAskingPrice for AskingPrice.

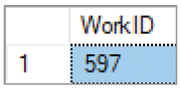
BY THE WAY

In the two stored procedures we discussed in the previous section, the @@rowCount function could have been used instead of counting rows using COUNT(*) to set the value of the variable @RowCount. Just remember to check it or store it immediately after the SQL statement is executed.

To test the trigger, we will begin by obtaining a new work for the View Ridge Gallery. Because Melinda Gliddens just bought the only copy of the print of John Singer Sargent's *Spanish Dancer*, we will replace it:

```
-- INSERT new work into WORK
/* *** SQL-INSERT-CH10A-01 *** */
INSERT INTO WORK VALUES(
     'Spanish Dancer', '635/750', 'High Quality Limited Print',
     'American Realist style - From work in Spain', 11);
-- Obtain the new WorkID form WORK
/* *** SQL-Query-CH10A-07 *** */
SELECT     WorkID
FROM       dbo.WORK
WHERE      ArtistID = 11
     AND   Title = 'Spanish Dancer'
     AND   Copy = '635/750';
```

The result of SQL-Query-CH10A-07 gives us the WorkID of the new art work, which in this case is 597:



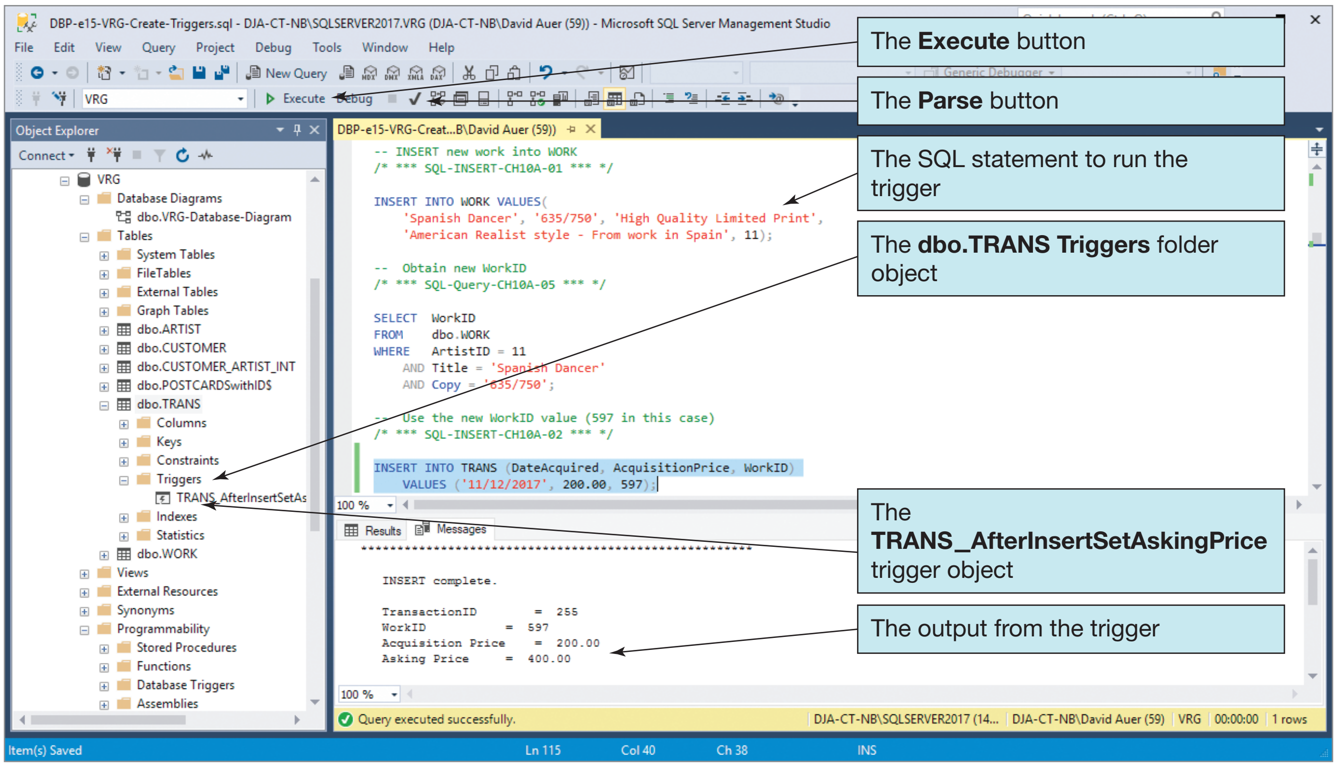
	WorkID
1	597

We use this value in the SQL INSERT statement to record the new transaction:

```
-- Use the new WorkID value (597 in this case)
/* *** SQL-INSERT-CH10A-02 *** */
INSERT INTO TRANS (DateAcquired, AcquisitionPrice, WorkID)
      VALUES ('11/12/2017', 200.00, 597);
```

Figure 10A-67 shows the results of the events triggered by the INSERT statement on TRANS. Note that the asking price for the new work (400.00) has been set to twice the acquisition cost (200.00), which is the correct value for a work that has not previously been in the gallery. This trigger provides useful functionality for the gallery. It saves the gallery personnel considerable manual work in implementing their pricing policy and likely improves the accuracy of the results as well.

BY THE WAY At the beginning of the trigger code, you will see the **Transact-SQL SET NOCOUNT ON command**, which controls some of the output generated by SQL Server 2017. You may have noticed that when, for example, you execute INSERT statements, SQL Server generates the message *(1 row(s) affected)* after each INSERT statement so this message appears in the Microsoft SQL Server Management Studio Messages window. This is useful if you are actually watching these actions (it tells you that the intended action successfully completed), but it is meaningless when a trigger is firing based on an application action. Further, the time it takes to generate these messages can slow performance down in some cases. Therefore, we set NOCOUNT to ON so these messages are not generated by trigger actions.



```
-- INSERT new work into WORK
/* *** SQL-INSERT-CH10A-01 *** */

INSERT INTO WORK VALUES(
    'Spanish Dancer', '635/750', 'High Quality Limited Print',
    'American Realist style - From work in Spain', 11);

--  Obtain new WorkID
/* *** SQL-Query-CH10A-05 *** */

SELECT  WorkID
FROM    dbo.WORK
WHERE   ArtistID = 11
    AND Title = 'Spanish Dancer'
    AND Copy = '635/750';

--  Use the new WorkID value (597 in this case)
/* *** SQL-INSERT-CH10A-02 *** */

INSERT INTO TRANS (DateAcquired, AcquisitionPrice, WorkID)
    VALUES ('11/12/2017', 200.00, 597);
```

```
**************************************************************

 INSERT complete.

 TransactionID        =  255
 WorkID            =  597
 Acquisition Price    =  200.00
 Asking Price      =  400.00
```

Query executed successfully. | DJA-CT-NB\SQLSERVER2017 (14... | DJA-CT-NB\David Auer (59) | VRG | 00:00:00 | 1 rows

Ln 115 Col 40 Ch 38 INS

FIGURE 10A-67

Results of the TRANS_AfterInsertSetAsking Price Trigger

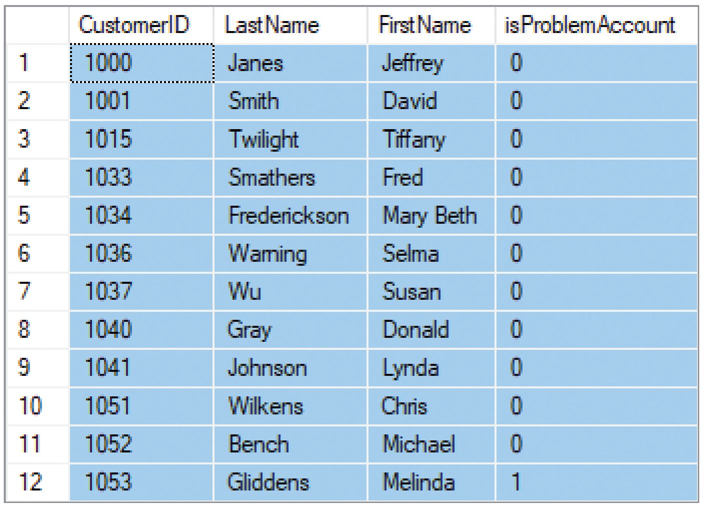
A Trigger for Enforcing a Data Constraint

The View Ridge Gallery needs to track problem-customer accounts; these are customers who have either not paid promptly or who have presented other problems to the gallery. When a customer who is on the problem list attempts to make a purchase at the gallery, the gallery wants the transaction to be rolled back and a message displayed. Note that this feature requires an intertable CHECK constraint between the TRANS table and the CUSTOMER table, which, as we discussed in Chapter 7, requires a trigger to implement.

To enforce this policy and the corresponding constraint, we need to add a column to the CUSTOMER table named isProblemAccount. This column will use the **SQL Server bit data type**, which can have the values NULL, 0, or 1. Zero indicates a good account, whereas a 1 indicates a problem account. And it looks like our new customer Melinda Gliddens had trouble with her previous payment, so we set her isProblemAccount value to 1:

```
-- Add column isProblemAccount to CUSTOMER
/* *** SQL-ALTER-TABLE-CH10A-03 *** */
ALTER TABLE dbo.CUSTOMER
     ADD isProblemAccount    Bit       NULL DEFAULT '0';
-- Set initial column values for CUSTOMER.isProblemAccount
/* *** SQL-UPDATE-CH10A-03 *** */
UPDATE        dbo.CUSTOMER
      SET     isProblemAccount = 0;
-- Set column value for Melinda Gliddens
/* *** SQL-UPDATE-CH10A-04 *** */
UPDATE          dbo.CUSTOMER
      SET       isProblemAccount = 1
      WHERE     LastName = 'Gliddens'
         AND    FirstName = 'Melinda';
-- Check CUSTOMER.isProblemAccount column values
/* *** SQL-Query-CH10A-08 *** */
SELECT       CustomerID, LastName, FirstName, isProblemAccount
FROM         dbo.CUSTOMER;
```

The results of the SELECT statement are:



	CustomerID	LastName	FirstName	isProblemAccount
1	1000	Janes	Jeffrey	0
2	1001	Smith	David	0
3	1015	Twilight	Tiffany	0
4	1033	Smathers	Fred	0
5	1034	Frederickson	Mary Beth	0
6	1036	Warning	Selma	0
7	1037	Wu	Susan	0
8	1040	Gray	Donald	0
9	1041	Johnson	Lynda	0
10	1051	Wilkens	Chris	0
11	1052	Bench	Michael	0
12	1053	Gliddens	Melinda	1

Now we will create a trigger on TRANS named TRANS_CheckIsProblemAccount. With this trigger, when a customer makes a purchase the trigger determines whether the customer is flagged by the value of the isProblemAccount data in the CUSTOMER table. If so, the transaction is rolled back and a message is displayed. The trigger code in Figure 10A-68 enforces this policy.

Note one interesting feature of the trigger code in Figure 10A-68. As noted there, this trigger will fire for every update on TRANS, including updates fired by another trigger, such as TRANS_AfterInsertSetAskingPrice. But in that trigger, no customer is involved. Therefore, before completing the rest of this trigger, we have to be sure that there is actually

FIGURE 10A-68

The SQL Statements for the TRANS_CheckIs ProblemAccount Trigger

```
CREATE TRIGGER TRANS_CheckIsProblemAccount
    ON dbo.TRANS
    FOR UPDATE

AS
BEGIN
    SET NOCOUNT ON;

    DECLARE   @TransactionID      AS Int,
              @CustomerID         AS Int,
              @isProblemAccount   AS Bit

    SELECT @TransactionID = TransactionID,
           @CustomerID   = CustomerID
    FROM   inserted;

    /* This trigger will fire for every update of TRANS.
     * This includes updates without a Customer participating,
     * such as an update of AskingPrice using the
     * TRANS_AfterInsertSetAskingPrice trigger.
     * Therefore, make sure there is a Customer particpating
     * in the Update of TRANS/
     */

    -- Check if Customer ID is NULL and if so RETURN.
    -- Do not ROLLBACK the transaction, just don't complete this trigger.

    IF (@CustomerID IS NULL) RETURN

    -- Valid CustomerID.
    -- Obtain value of @isProblemAcocunt.

    SELECT    @isProblemAccount = isProblemAccount
    FROM   dbo.CUSTOMER AS C
    WHERE  C.CustomerID = @CustomerID;

    IF (@isProblemAccount = 1)
        -- This is a problem account.
        -- Rollback the transaction and send message.
        BEGIN
            ROLLBACK TRANSACTION
            PRINT '*******************************************************'
            PRINT ''
            PRINT '    Transaction canceled.'
            PRINT ''
            PRINT '    CustomerID       = '+CONVERT(Char(6), @CustomerID)
            PRINT ''
            PRINT '    Refer customer to the manager immediately.'
            PRINT ''
            PRINT '*******************************************************'
            RETURN
        END
```

```
ELSE
        -- This is a good account
        -- Let the transaction stand.
        BEGIN
            PRINT '*******************************************************'
            PRINT ''
            PRINT '    Transaction complete.'
            PRINT '    TransactionID    = '+CONVERT(Char(6), @TransactionID)
            PRINT '    CustomerID       = '+CONVERT(Char(6), @CustomerID)
            PRINT ''
            PRINT '    Thank the customer for their business.'
            PRINT ''
            PRINT '*******************************************************'
        END
END;
```

FIGURE 10A-68
Continued

a customer participating in a transaction whose account status needs to be checked. This is done by the line:

```
IF (@CustomerID IS NULL) RETURN
```

Note that we only want to exit the TRANS_CheckIsProblemAccount trigger if there is no customer, not roll back the transaction that fired the trigger. When writing multiple triggers, remember that they may be run from other actions besides the one that you originally created them to handle.

OK, here comes Melinda to make another purchase—let's see what happens.

```
/* *** SQL-UPDATE-CH10A-05 *** */
UPDATE        TRANS
     SET      DateSold = '11/18/2017',
              SalesPrice = 475.00,
              CustomerID = 1053
     WHERE    TransactionID = 229;
```

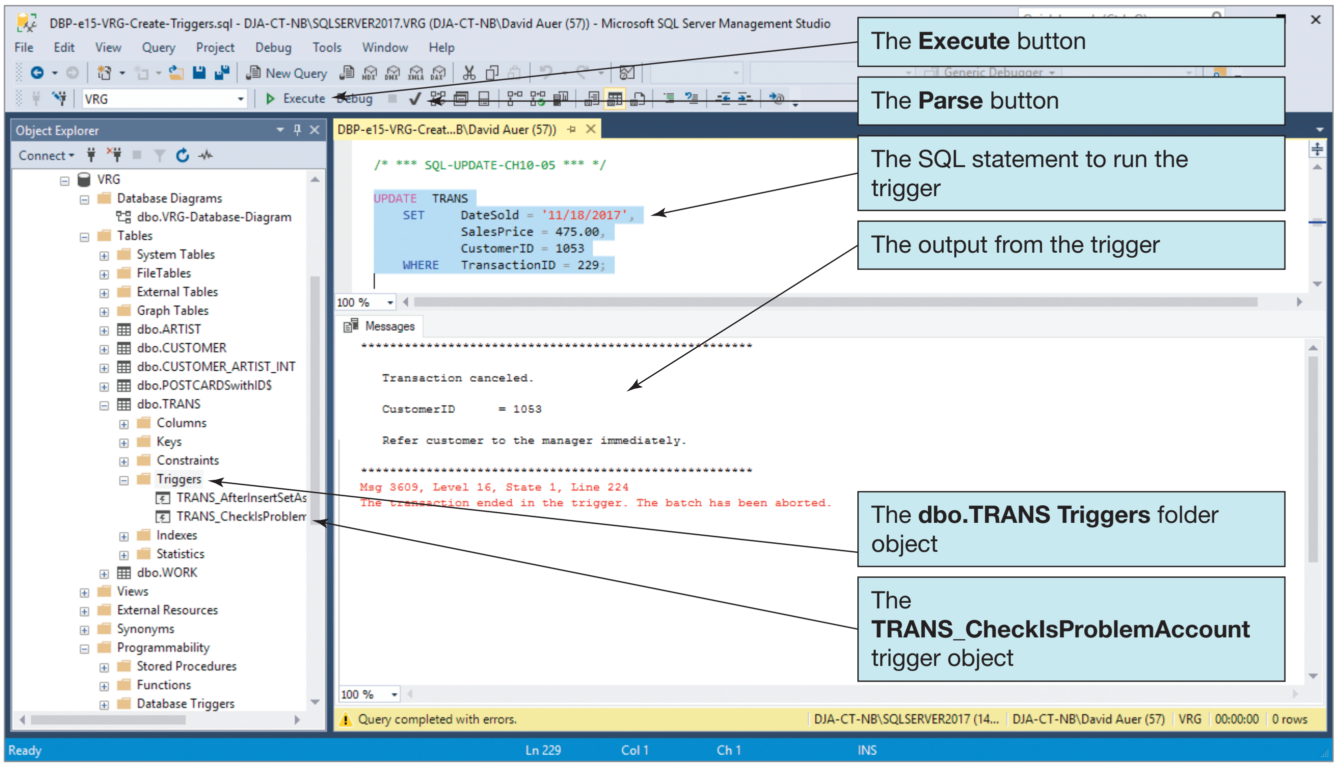
The resulting output is shown in Figure 10A-69. Looks like Melinda is off to talk to the manager about her account!

BY THE WAY Using a table of valid or invalid values is more flexible and dynamic than placing such values in a CHECK constraint. For example, consider the CHECK constraint on Nationality values in the ARTIST table. If the gallery manager wants to expand the list of allowed nationalities for artists, the manager will have to change the CHECK constraint using the ALTER TABLE statement. In reality, the gallery manager will have to hire a consultant to change this constraint.

A better approach is to place the allowed values of Nationality in a table, say, ALLOWED_NATIONALITY. Then write a trigger like that shown in Figure 10A-68 to enforce the constraint that new values of Nationality exist in ALLOWED_NATIONALITY. When the gallery owner wants to change the allowed nationalities, the manager would simply add or remove values in the ALLOWED_NATIONALITY table.

A Trigger for Updating a View

In Chapter 7, we discussed the problem of updating views. One such problem concerns updating views created via joins; it is normally not possible for the DBMS to know how to update tables that underlie the join. However, sometimes application-specific knowledge can be used to determine how to interpret a request to update a joined view.


The **Execute** button
The **Parse** button
The SQL statement to run the trigger
The output from the trigger
The **dbo.TRANS Triggers** folder object
The **TRANS_CheckIsProblemAccount** trigger object
Object Explorer
VRG
Tables
dbo.TRANS
Triggers
TRANS_AfterInsertSetAs
TRANS_CheckIsProblem
Messages
Query completed with errors.

```
/* *** SQL-UPDATE-CH10-05 *** */

UPDATE  TRANS
   SET     DateSold = '11/18/2017',
           SalesPrice = 475.00,
           CustomerID = 1053
   WHERE   TransactionID = 229;
```

```
******************************************************

  Transaction canceled.

  CustomerID      = 1053

  Refer customer to the manager immediately.

******************************************************
Msg 3609, Level 16, State 1, Line 224
The transaction ended in the trigger. The batch has been aborted.
```

FIGURE 10A-69

Running the TRANS_CheckIsProblemAccount Trigger

Consider the view CustomerInterestsView shown in Figures 10A-40, 10A-41, and 10A-42. It contains rows of CUSTOMER and ARTIST joined over their intersection table. CUSTOMER.LastName is given the alias CustomerLastName, CUSTOMER.FirstName is given the alias CustomerFirstName, and ARTIST.LastName is given the alias ArtistName. A request to change the last name of a customer in CustomerInterests can be interpreted as a request to change the last name of the underlying CUSTOMER table. Such a request, however, can be processed only if the values of CUSTOMER.LastName and CUSTOMER .FirstName are unique. If not, the request cannot be processed.

The INSTEAD OF trigger shown in Figure 10A-70 implements this logic. First, the new and old values of the CustomerLastName and CustomerFirstName columns in CustomerInterestsView are obtained. Then a correlated subquery is used to determine whether the old values of CUSTOMER.LastName and CUSTOMER.FirstName are unique. If so, the name can be changed, but otherwise no update can be made.

This trigger needs to be tested against cases in which the name is unique and cases in which the name is not unique. Figure 10A-71 shows the case in which the customer name was unique: View Ridge Gallery's two newest customers, Michael Bench and Melinda Gliddens, just got married after meeting at a View Ridge Gallery opening, and Melinda wants to change her last name. To do this, we use the following SQL statement:

```
/* *** SQL-UPDATE-CH10A-06 *** */
UPDATE        dbo.CustomerInterestsView
     SET      CustomerLastName = 'Bench',
              CustomerFirstName = 'Melinda'
     WHERE    CustomerLastName = 'Gliddens'
        AND   CustomerFirstName = 'Melinda';
```

Note that the UPDATE command was issued against the view. As indicated in the Messages pane, Melinda Gliddens is now Melinda Bench.

A Trigger for Enforcing a Required Child Constraint

The VRG database design includes an M-M relationship between WORK and TRANS. Every WORK must have a TRANS to store the price of the work and the date the work was acquired, and every TRANS must relate to a WORK parent. Figure 10A-72 shows the tasks that must be accomplished to enforce this constraint; it is based on the boilerplate shown in Figure 6-29(b).

FIGURE 10A-70

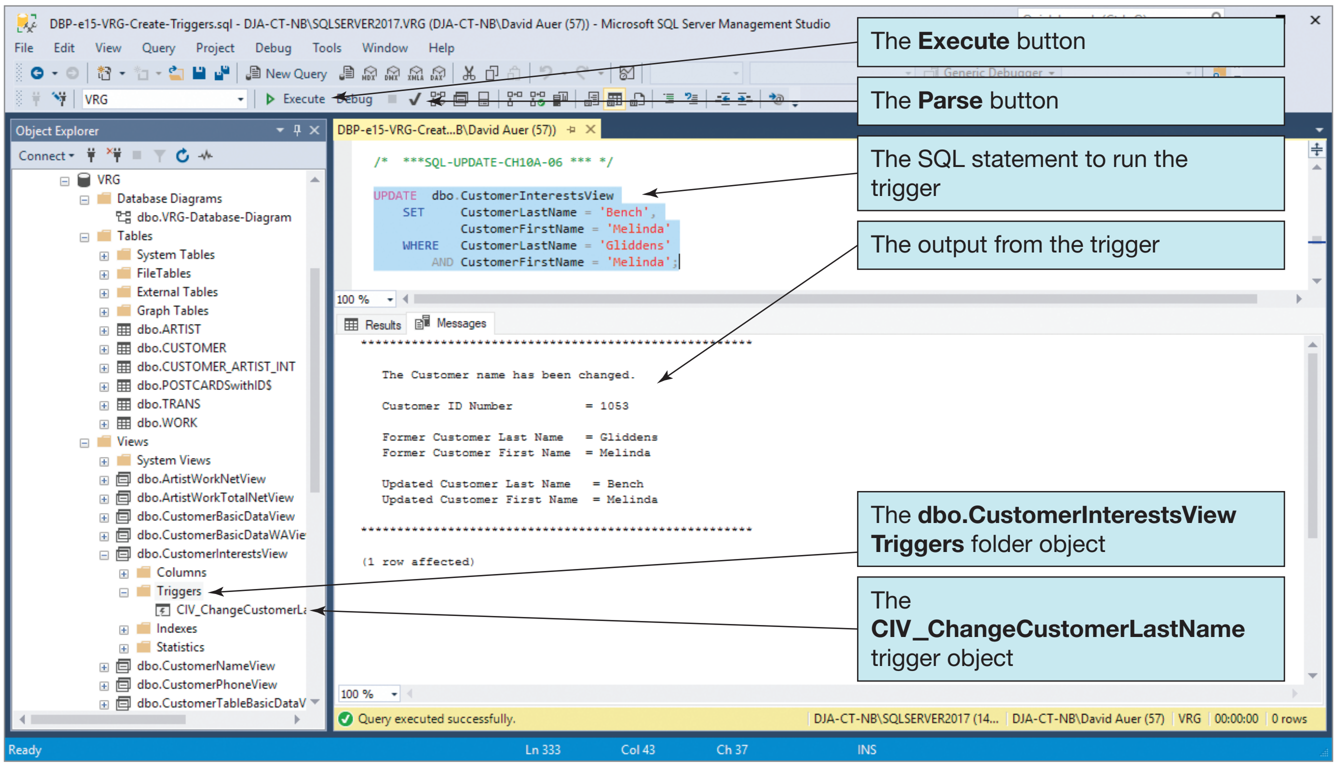
The SQL Statements for the CIV_ChangeCustomerName Trigger

```
CREATE TRIGGER CIV_ChangeCustomerLastName
       ON dbo.CustomerInterestsView
       INSTEAD OF UPDATE

AS
BEGIN
       SET NOCOUNT ON;

       DECLARE   @OldCustomerLastName    AS Char(25),
                 @OldCustomerFirstName   AS Char(25),
                 @NewCustomerLastName    AS Char(25),
                 @NewCustomerFirstName   AS Char(25),
                 @CustomerID             AS Int

       -- Get values of new and old names.
       SELECT @NewCustomerLastName = CustomerLastName
       FROM   inserted;
       SELECT @NewCustomerFirstName = CustomerFirstName
       FROM   inserted;
       SELECT @OldCustomerLastName = CustomerLastName
       FROM   deleted;
       SELECT @OldCustomerFirstName = CustomerFirstName
       FROM   deleted;

       -- Count number of synonyms in CUSTOMER.
       SELECT    *
       FROM      dbo.CUSTOMER AS C1
       WHERE     C1.LastName = @OldCustomerLastName
          AND    C1.FirstName = @OldCustomerFirstName
          AND    EXISTS
                 (SELECT       *
                  FROM         dbo.CUSTOMER AS C2
                  WHERE        C1.LastName = C2.LastName
                        AND    C1.FirstName = C2.FirstName
                        AND    C1.CustomerID <> C2.CustomerID);

       IF (@@rowCount = 0)
          -- The Customer last name is unique.
          -- Update the Customer record.
          BEGIN

             -- Get CustomerID vlaue
             SELECT    @CustomerID = CustomerID
             FROM      dbo.CUSTOMER AS C
             WHERE     C.LastName = @OldCustomerLastName
                AND    C.FirstName = @OldCustomerFirstName;

             UPDATE    dbo.CUSTOMER
             SET       LastName = @NewCustomerLastName
             WHERE     CustomerID = @CustomerID;

             UPDATE    dbo.CUSTOMER
             SET       FirstName = @NewCustomerFirstName
             WHERE     CustomerID = @CustomerID;
```

```
                -- Print update message.
                   PRINT '******************************************************'
                   PRINT ''
                   PRINT '   The Customer name has been changed.'
                   PRINT ''
                   PRINT '   Customer ID Number  = '+CONVERT(Char(6), @CustomerID)
                   PRINT ''
                   PRINT '   Former Customer Last Name  = '+@OldCustomerLastName
                   PRINT '   Former Customer First Name = '+@OldCustomerFirstName
                   PRINT ''
                   PRINT '   Updated Customer Last Name  = '+@NewCustomerLastName
                   PRINT '   Updated Customer First Name = '+@NewCustomerFirstName
                   PRINT ''
                   PRINT '******************************************************'
                END
             ELSE
                -- The Customer name is not unique.
                -- Rollback the transaction and send message.
                BEGIN
                   PRINT '******************************************************'
                   PRINT ''
                   PRINT '   Transaction canceled.'
                   PRINT ''
                   PRINT '   Customer Last Name    = '+@NewCustomerLastName
                   PRINT '   Customer First Name   = '+@NewCustomerFirstName
                   PRINT ''
                   PRINT '   The customer name is not unique.'
                   PRINT ''
                   PRINT '******************************************************'
                END
      END;
```

FIGURE 10A-70

Continued


DBP-e15-VRG-Create-Triggers.sql - DJA-CT-NB\SQLSERVER2017.VRG (DJA-CT-NB\David Auer (57)) - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
Execute Debug
VRG
Object Explorer
Connect
DBP-e15-VRG-Creat...B\David Auer (57))
The **Execute** button
The **Parse** button
The SQL statement to run the trigger
The output from the trigger
The **dbo.CustomerInterestsView Triggers** folder object
The **CIV_ChangeCustomerLastName** trigger object
VRG
Database Diagrams
dbo.VRG-Database-Diagram
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.ARTIST
dbo.CUSTOMER
dbo.CUSTOMER_ARTIST_INT
dbo.POSTCARDSwithIDS
dbo.TRANS
dbo.WORK
Views
System Views
dbo.ArtistWorkNetView
dbo.ArtistWorkTotalNetView
dbo.CustomerBasicDataView
dbo.CustomerBasicDataWAVie
dbo.CustomerInterestsView
Columns
Triggers
CIV_ChangeCustomerLa
Indexes
Statistics
dbo.CustomerNameView
dbo.CustomerPhoneView
dbo.CustomerTableBasicDataV
100 %
Results
Messages
Query executed successfully.
DJA-CT-NB\SQLSERVER2017 (14... DJA-CT-NB\David Auer (57) VRG 00:00:00 0 rows
Ready Ln 333 Col 43 Ch 37 INS

```
/*  ***SQL-UPDATE-CH10A-06 *** */

UPDATE  dbo.CustomerInterestsView
   SET     CustomerLastName = 'Bench',
           CustomerFirstName = 'Melinda'
   WHERE   CustomerLastName = 'Gliddens'
      AND  CustomerFirstName = 'Melinda';
```

```
******************************************************

  The Customer name has been changed.

  Customer ID Number          = 1053

  Former Customer Last Name   = Gliddens
  Former Customer First Name  = Melinda

  Updated Customer Last Name   = Bench
  Updated Customer First Name  = Melinda

******************************************************

(1 row affected)
```

FIGURE 10A-71

Results of the CIV_Change CustomerName Trigger

FIGURE 10A-72

Actions to Enforce Minimum Cardinality for the WORK-to-TRANS Relationship

WORK Is Required Parent TRANS Is Required Child	Action on WORK (Parent)	Action on TRANS (Child)
Insert	Create a TRANS row	New TRANS must have a valid WorkID (enforced by DBMS)
Modify key or foreign key	Prohibit—WORK uses a surrogate key	Prohibit—TRANS uses a surrogate key, and TRANS cannot change to a different WORK
Delete	Prohibit—Cannot delete a WORK with TRANS children (enforced by DBMS by lack of CASCADE DELETE)	Cannot delete the last child [actually, data related to a transaction is never deleted (business rule)]

Because the CREATE TABLE statement for TRANS in Figure 10A-27 defines TRANS.WorkID as NOT NULL and defines the FOREIGN KEY constraint without cascading deletions, the DBMS will ensure that every TRANS has a WORK parent. So we need not be concerned with enforcing the insert on TRANS or the deletion on WORK. As stated in Figure 10A-72, the DBMS will do that for us. Also, we need not be concerned with updates to WORK.WorkID because it is a surrogate key.

Three constraints remain that must be enforced by triggers: (1) ensuring that a TRANS row is created when a new WORK is created; (2) ensuring that TRANS.WorkID never changes; and (3) ensuring that the last TRANS child for a WORK is never deleted.

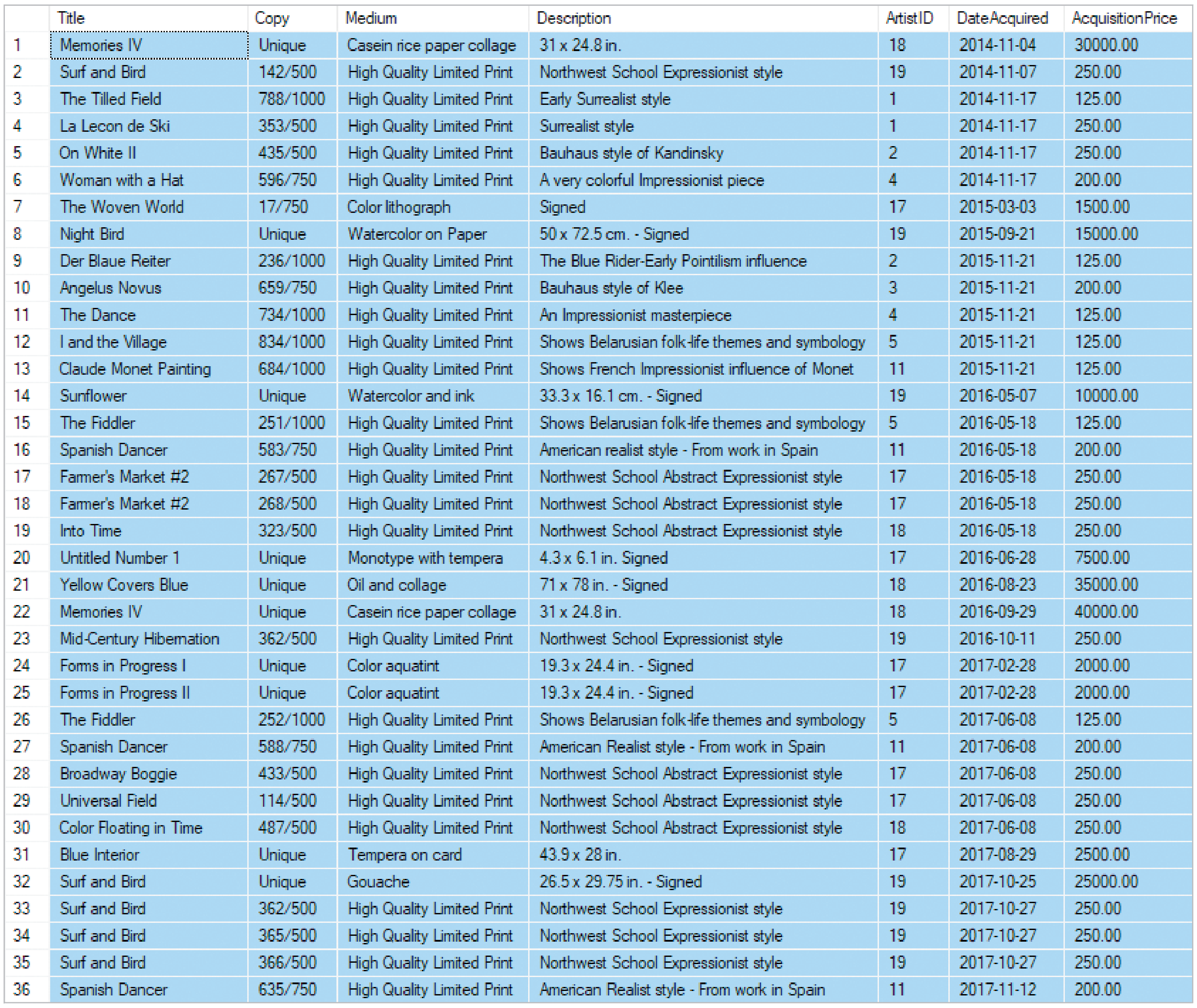
We can enforce the second constraint by writing a trigger on the update of TRANS that checks for a change in WorkID. If there is such a change, the trigger can roll back the change.

With regard to the third constraint, View Ridge Gallery has a business policy that no TRANS data ever be deleted. Thus, we not only need to disallow the deletion of the last child, we also need to disallow the deletion of any child. We can do this by writing a trigger on the deletion of TRANS that rolls back any attempted deletion. (If the gallery allowed TRANS deletions, we could enforce the deletion constraint using views, as shown in Chapter 7, Figures 7-29 and 7-30.) The triggers for enforcing the second and third constraints are simple.

However, the first constraint is a problem. We could write a trigger on WORK INSERT to create a default TRANS row, but this trigger will be called before the application has a chance to create the TRANS row itself. The trigger would create a TRANS row, and then the application may create a second one. To guard against the duplicate, we could then write a trigger on TRANS to remove the row the WORK trigger created in those cases when the application creates its own trigger. However, this solution is awkward at best.

A better design is to require the applications to create the WORK and TRANS combination via a view. For example, we can create a view named WorkAndTransView (we will add the code into our *DBP-e15-VRG-Create-Views.sql* script and execute it from there):

```
/* *** SQL-CREATE-VIEW-CH10A-01 – WorkAndTransView *** */
CREATE VIEW WorkAndTransView AS
     SELECT     Title, Copy, Medium, [Description], ArtistID,
                DateAcquired, AcquisitionPrice
     FROM       WORK AS W JOIN TRANS AS T
          ON    W.WorkID = T.WorkID;
```



	Title	Copy	Medium	Description	ArtistID	DateAcquired	AcquisitionPrice
1	Memories IV	Unique	Casein rice paper collage	31 x 24.8 in.	18	2014-11-04	30000.00
2	Surf and Bird	142/500	High Quality Limited Print	Northwest School Expressionist style	19	2014-11-07	250.00
3	The Tilled Field	788/1000	High Quality Limited Print	Early Surrealist style	1	2014-11-17	125.00
4	La Lecon de Ski	353/500	High Quality Limited Print	Surrealist style	1	2014-11-17	250.00
5	On White II	435/500	High Quality Limited Print	Bauhaus style of Kandinsky	2	2014-11-17	250.00
6	Woman with a Hat	596/750	High Quality Limited Print	A very colorful Impressionist piece	4	2014-11-17	200.00
7	The Woven World	17/750	Color lithograph	Signed	17	2015-03-03	1500.00
8	Night Bird	Unique	Watercolor on Paper	50 x 72.5 cm. - Signed	19	2015-09-21	15000.00
9	Der Blaue Reiter	236/1000	High Quality Limited Print	The Blue Rider-Early Pointilism influence	2	2015-11-21	125.00
10	Angelus Novus	659/750	High Quality Limited Print	Bauhaus style of Klee	3	2015-11-21	200.00
11	The Dance	734/1000	High Quality Limited Print	An Impressionist masterpiece	4	2015-11-21	125.00
12	I and the Village	834/1000	High Quality Limited Print	Shows Belarusian folk-life themes and symbology	5	2015-11-21	125.00
13	Claude Monet Painting	684/1000	High Quality Limited Print	Shows French Impressionist influence of Monet	11	2015-11-21	125.00
14	Sunflower	Unique	Watercolor and ink	33.3 x 16.1 cm. - Signed	19	2016-05-07	10000.00
15	The Fiddler	251/1000	High Quality Limited Print	Shows Belarusian folk-life themes and symbology	5	2016-05-18	125.00
16	Spanish Dancer	583/750	High Quality Limited Print	American realist style - From work in Spain	11	2016-05-18	200.00
17	Farmer's Market #2	267/500	High Quality Limited Print	Northwest School Abstract Expressionist style	17	2016-05-18	250.00
18	Farmer's Market #2	268/500	High Quality Limited Print	Northwest School Abstract Expressionist style	17	2016-05-18	250.00
19	Into Time	323/500	High Quality Limited Print	Northwest School Abstract Expressionist style	18	2016-05-18	250.00
20	Untitled Number 1	Unique	Monotype with tempera	4.3 x 6.1 in. Signed	17	2016-06-28	7500.00
21	Yellow Covers Blue	Unique	Oil and collage	71 x 78 in. - Signed	18	2016-08-23	35000.00
22	Memories IV	Unique	Casein rice paper collage	31 x 24.8 in.	18	2016-09-29	40000.00
23	Mid-Century Hibernation	362/500	High Quality Limited Print	Northwest School Expressionist style	19	2016-10-11	250.00
24	Forms in Progress I	Unique	Color aquatint	19.3 x 24.4 in. - Signed	17	2017-02-28	2000.00
25	Forms in Progress II	Unique	Color aquatint	19.3 x 24.4 in. - Signed	17	2017-02-28	2000.00
26	The Fiddler	252/1000	High Quality Limited Print	Shows Belarusian folk-life themes and symbology	5	2017-06-08	125.00
27	Spanish Dancer	588/750	High Quality Limited Print	American Realist style - From work in Spain	11	2017-06-08	200.00
28	Broadway Boggie	433/500	High Quality Limited Print	Northwest School Abstract Expressionist style	17	2017-06-08	250.00
29	Universal Field	114/500	High Quality Limited Print	Northwest School Abstract Expressionist style	17	2017-06-08	250.00
30	Color Floating in Time	487/500	High Quality Limited Print	Northwest School Abstract Expressionist style	18	2017-06-08	250.00
31	Blue Interior	Unique	Tempera on card	43.9 x 28 in.	17	2017-08-29	2500.00
32	Surf and Bird	Unique	Gouache	26.5 x 29.75 in. - Signed	19	2017-10-25	25000.00
33	Surf and Bird	362/500	High Quality Limited Print	Northwest School Expressionist style	19	2017-10-27	250.00
34	Surf and Bird	365/500	High Quality Limited Print	Northwest School Expressionist style	19	2017-10-27	250.00
35	Surf and Bird	366/500	High Quality Limited Print	Northwest School Expressionist style	19	2017-10-27	250.00
36	Spanish Dancer	635/750	High Quality Limited Print	American Realist style - From work in Spain	11	2017-11-12	200.00

FIGURE 10A-73

Result of Using the View WorkAndTransView

We can display the view results by using the SQL SELECT command:

```
/* *** SQL-Query-View-CH10A-01 *** */
SELECT      *
FROM        WorkAndTransView
ORDER BY    DateAcquired;
```

The results of this query are shown in Figure 10A-73.

The DBMS will *not* be able to process an INSERT on this view. We can, however, define an INSTEAD OF trigger to process the insert. Our trigger, named *WATV_InsertTransactionWithWork*, will create both a new row in WORK and the new required child in TRANS. The code for this trigger is shown in Figure 10A-74. Note that applications that use this solution must *not* be allowed to insert WORK rows directly. They must always insert them via the view WorkAndTransView.

To test our trigger, we will add a new work to the *VRG* database. Melinda, now Mrs. Michael Bench, has worked out her account problems with the View Ridge Gallery and completed her purchase of the print of Horiuchi's *Color Floating in Time*.

FIGURE 10A-74

The SQL Statements for the WATV_Insert-TransactionWithWork Trigger

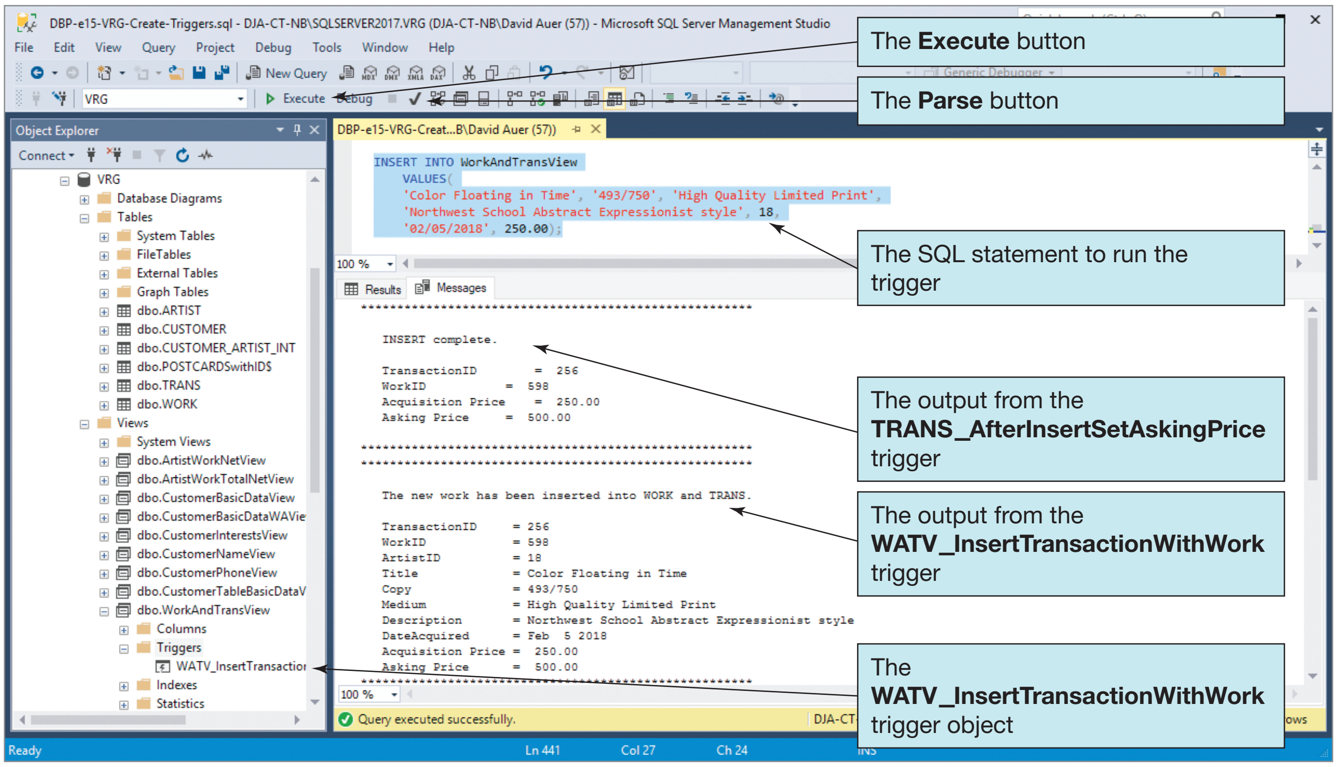
```
ALTER TRIGGER WATV_InsertTransactionWithWork
    ON dbo.WorkAndTransView
    INSTEAD OF INSERT

AS
BEGIN
    SET NOCOUNT ON;

    DECLARE    @TransactionID       AS Int,
               @WorkID              AS Int,
               @Title               AS Char(35),
               @Copy                AS Char(12),
               @Medium              AS Char(35),
               @Description         AS Varchar(1000),
               @ArtistID            AS Int,
               @DateAcquired        AS DateTime,
               @AcquisitionPrice    AS Numeric(8,2),
               @AskingPrice         AS Numeric(8,2)

    -- Get available values from Insert on the view.
    SELECT     @Title = Title, @Copy = Copy, @Medium = Medium,
               @Description = [Description],
               @ArtistID = ArtistID, @DateAcquired =DateAcquired,
               @AcquisitionPrice = AcquisitionPrice
    FROM       inserted;

    -- Insert new row into WORK.
    INSERT INTO WORK VALUES(
               @Title, @Copy, @Medium, @Description, @ArtistID);

    -- Get new WorkID surrogate key value using @@Identity funcion.
    SET @WorkID = @@Identity;

    -- Insert new row into TRANS.
    -- Note that INSERT will trigger TRANS_AfterInsertSetAskingPrice.
    INSERT INTO TRANS (DateAcquired, AcquisitionPrice, WorkID)
               VALUES(
               @DateAcquired, @AcquisitionPrice, @WorkID);

    -- Get new TranasctionID surrogate key value.
    SET @TransactionID = @@Identity;

    -- Get new AskingPrice set by TRANS_AfterInsertSetAskingPrice.
    SELECT     @AskingPrice = AskingPrice
    FROM       TRANS
    WHERE      TransactionID = @TransactionID;

    -- Print results message.
    PRINT '*******************************************************'
    PRINT ''
    PRINT '    The new work has been inserted into WORK and TRANS.'
    PRINT ''
    PRINT '    TransactionID      = '+CONVERT(Char(6), @TransactionID)
    PRINT '    WorkID             = '+CONVERT(Char(6), @WorkID)
    PRINT '    ArtistID           = '+CONVERT(Char(6), @ArtistID)
    PRINT '    Title              = '+@Title
    PRINT '    Copy               = '+@Copy
    PRINT '    Medium             = '+@Medium
    PRINT '    Description        = '+@Description
    PRINT '    DateAcquired       = '+CONVERT(Char(12), @DateAcquired)
    PRINT '    Acquisition Price =  '+CONVERT(Char(12), @AcquisitionPrice)
    PRINT '    Asking Price       =  '+CONVERT(Char(12), @AskingPrice)
    PRINT '*******************************************************'
END;
```

```
-- Reset Melinda Bench's Problem Account status.
/* *** SQL-UPDATE-CH10A-07 *** */
UPDATE        dbo.CUSTOMER
     SET      isProblemAccount = 0
     WHERE    LastName = 'Bench'
        AND   FirstName = 'Melinda';
-- Record the completed purchase of "Color Floating in Time".
/* *** SQL-UPDATE-CH10A-08 *** */
UPDATE        TRANS
     SET      DateSold = '11/18/2017',
              SalesPrice = 475.00,
              CustomerID = 1053
     WHERE    TransactionID = 229;
```

Note that the SQL-UPDATE-CH10-08 statement fired the TRAN_CheckIsProblemAccount trigger, but now that Melinda is a good customer again, the transaction was accepted. Now we will restock a copy of *Color Floating in Time* into the gallery.

```
/* *** SQL-INSERT-CH10A-03 *** */
INSERT INTO WorkAndTransView
       VALUES(
       'Color Floating in Time', '493/750',
       'High Quality Limited Print',
       'Northwest School Abstract Expressionist style', 18,
       '02/05/2018', 250.00);
```

The results of the transaction are shown in Figure 10A-75. Note that the WATV_InsertTransactionWithWork trigger actually sets off two other triggers. First, it fires the INSERT


DBP-e15-VRG-Create-Triggers.sql - DJA-CT-NB\SQLSERVER2017.VRG (DJA-CT-NB\David Auer (57)) - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
Execute Debug
Object Explorer
Connect
VRG
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
dbo.ARTIST
dbo.CUSTOMER
dbo.CUSTOMER_ARTIST_INT
dbo.POSTCARDSwithIDS
dbo.TRANS
dbo.WORK
Views
System Views
dbo.ArtistWorkNetView
dbo.ArtistWorkTotalNetView
dbo.CustomerBasicDataView
dbo.CustomerBasicDataWAVie
dbo.CustomerInterestsView
dbo.CustomerNameView
dbo.CustomerPhoneView
dbo.CustomerTableBasicDataV
dbo.WorkAndTransView
Columns
Triggers
WATV_InsertTransactio
Indexes
Statistics
DBP-e15-VRG-Creat...B\David Auer (57))
100 %
Results
Messages
Query executed successfully.
Ready
Ln 441
Col 27
Ch 24
INS
The **Execute** button
The **Parse** button
The SQL statement to run the trigger
The output from the **TRANS_AfterInsertSetAskingPrice** trigger
The output from the **WATV_InsertTransactionWithWork** trigger
The **WATV_InsertTransactionWithWork** trigger object

```
INSERT INTO WorkAndTransView
    VALUES(
    'Color Floating in Time', '493/750', 'High Quality Limited Print',
    'Northwest School Abstract Expressionist style', 18,
    '02/05/2018', 250.00);
```

```
******************************************************

   INSERT complete.

   TransactionID        =  256
   WorkID            =  598
   Acquisition Price     =  250.00
   Asking Price      =  500.00

******************************************************
******************************************************

   The new work has been inserted into WORK and TRANS.

   TransactionID      = 256
   WorkID             = 598
   ArtistID           = 18
   Title              = Color Floating in Time
   Copy               = 493/750
   Medium             = High Quality Limited Print
   Description        = Northwest School Abstract Expressionist style
   DateAcquired       = Feb  5 2018
   Acquisition Price =  250.00
   Asking Price       =  500.00
******************************************************
```

FIGURE 10A-75

Results of the WATV_InsertTranactionWithWork Trigger

trigger TRANS_AfterInsertSetAskingPrice, which then fires the UPDATE trigger TRANS_CheckIsProblemAccount. Because no customer is involved in this transaction, the TRANS_CheckIsProblemAccount trigger returns control without finishing (see the previous discussion of the code for this trigger). However, the TRANS_AfterInsertSetAskingPrice trigger does run and sets the new asking price for the new work. Therefore, the results of two triggers show up in the output in Figure 10A-75, and the new work now has an asking price.

BY THE WAY If we look at the SQL code that has actually been stored for these triggers, we will again find that SQL Server has added some lines before the code we wrote:

```
USE [VRG]
GO
/****** Object: Trigger [ {StoredProcedureName}]
     Script Date: {Date and Time created or altered} ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

This is essentially the same code used for stored procedures. However, we should also note that there is an important use of the GO command when running scripts. Triggers are run only *once* for a set of operations, such as typically found in a script, not for each individual command.

For example, suppose that in our last trigger example we had decided to use an SQL script to input the data for three copies of *Color Floating in Time* at once as follows (note the change in the value of Copy):

```
/* *** EXAMPLE CODE- DO NOT RUN *** */
/* *** SQL-INSERT-CH10A-04 *** */
INSERT INTO WorkAndTransView
     VALUES(
     'Color Floating in Time', '494/750',
     'High Quality Limited Print',
     'Northwest School Abstract Expressionist style', 18,
     '02/05/2018', 250.00);
/* *** SQL-INSERT-CH10A-05 *** */
INSERT INTO WorkAndTransView
     VALUES(
     'Color Floating in Time', '495/750',
     'High Quality Limited Print',
     'Northwest School Abstract Expressionist style', 18,
     '02/05/2018', 250.00);
/* *** SQL-INSERT-CH10A-06 *** */
INSERT INTO WorkAndTransView
     VALUES(
     'Color Floating in Time', '496/750',
     'High Quality Limited Print',
     'Northwest School Abstract Expressionist style', 18,
     '02/05/2018', 250.00);
```

(*continued*)

In this case, the WATV_InsertTransactionWithWork trigger only fires *once*! But this is *not* what we want—we want it to fire once per INSERT. This is where the GO command comes in handy, and we will use the following code:

```
/* *** SQL-INSERT-CH10A-04 *** */
INSERT INTO WorkAndTransView
     VALUES(
     'Color Floating in Time', '494/750',
     'High Quality Limited Print',
     'Northwest School Abstract Expressionist style', 18,
     '02/05/2018', 250.00);
GO
/* *** SQL-INSERT-CH10A-05 *** */
INSERT INTO WorkAndTransView
     VALUES(
     'Color Floating in Time', '495/750',
     'High Quality Limited Print',
     'Northwest School Abstract Expressionist style', 18,
     '02/05/2018', 250.00);
GO
/* *** SQL-INSERT-CH10A-06 *** */
INSERT INTO WorkAndTransView
     VALUES(
     'Color Floating in Time', '496/750',
     'High Quality Limited Print',
     'Northwest School Abstract Expressionist style', 18,
     '02/05/2018', 250.00);
GO
```

Now each statement is considered its own “batch of commands,” and the trigger fires three times, as we intended. You should run the second set of INSERT statements (with the GO commands) on the VRG database and note the output.

Microsoft SQL Server 2017 Concurrency Control

SQL Server 2017 provides a comprehensive set of capabilities to control concurrent processing. Many choices and options are available, and the resulting behavior is determined by the interaction of three factors: the transaction isolation level, the cursor concurrency setting, and locking hints provided in the SELECT clause. Locking behavior also depends on whether the cursor is processed as part of a transaction; whether the SELECT statement is part of a cursor; and whether INSERT, UPDATE, or DELETE commands occur inside of transactions or independently.

Figure 10A-76 summarizes the concurrency control options. In this section, we will describe just the basics.[13] With SQL Server, developers do not place explicit locks. Instead, developers declare the concurrency control behavior they want, and SQL Server determines where to place the locks. Locks are applied on rows, pages, keys, indexes, tables, and even

[13] For more information, see the article “Set Transaction Isolation Level” at *http://msdn.microsoft.com/en-us/library/ms173763.aspx*.

FIGURE 10A-76

Concurrency Options for SQL Server 2017

Type	Scope	Options
Transaction isolation level	Connection—all transactions	READ UNCOMMITTED READ COMMITTED REPEATABLE READ SNAPSHOT SERIALIZABLE
Cursor concurrency	CURSOR statements	READ_ONLY SCROLL_LOCKS OPTIMSTIC
Locking hints	SELECT statements	READCOMMITTED READUNCOMMITTED REPEATABLEREAD SERIALIZABLE NOLOCK HOLDLOCK And many more...

the entire database. SQL Server determines what level of lock to use and may promote or demote a lock level while processing. It also determines when to place the lock and when to release it, depending on the declarations made by the developer.

Transaction Isolation Level

The broadest level of concurrency settings is the **transaction isolation level**. As shown in Figure 10A-76, there are five transaction isolation level options, listed in ascending level of restriction. You studied four of these options in Chapter 9, and they are the SQL-92 standard levels. A new level, SNAPSHOT, is unique to SQL Server. With the **SNAPSHOT transaction isolation level**, the data read by an SQL statement will be the same as the data that existed and were committed at the start of the transaction. SNAPSHOT does not request locks during reads (except during database recovery) and does not prevent other transactions from writes. With SQL Server, READ COMMITTED is the default isolation level, though it is possible to make dirty reads by setting the isolation level to READ UNCOMMITTED. However, the exact behavior of READ UNCOMMITTED now depends on the setting of the READ_COMMITTED_SNAPSHOT option.

An SQL command to set the isolation level can be issued anyplace Transact-SQL is allowed prior to any other database activity. An example Transact-SQL statement to set the isolation level of, say, REPEATABLE READ is:

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Cursor Concurrency

The second way in which the developer can declare locking characteristics is with cursor concurrency. Possibilities are read only, optimistic, and pessimistic, here called **SCROLL_LOCK**. As described in Chapter 9, with optimistic locking, no lock is obtained until the user updates the data. At that point, if the data have been changed since they were read, the update is refused. Of course, the application program must specify what to do when such a refusal occurs.[14]

[14] For more information, see the SQL Server 2017 documentation on "Declare Cursor" at *https://docs.microsoft.com/en-us/sql/t-sql/language-elements/declare-cursor-transact-sql*.

SCROLL_LOCK is a version of pessimistic locking. With it, an update lock is placed on a row when the row is read. If the cursor is opened within a transaction, the lock is held until the transaction commits or rolls back. If the cursor is outside of a transaction, the lock is dropped when the next row is read. Recall from Chapter 9 that an update lock blocks another update lock, but it does not block a shared lock. Thus, other connections can read the row with shared locks.

As described in Chapter 9, the default cursor concurrency setting depends on the cursor type. It is read only for static and forward only cursors, and it is optimistic for dynamic and keyset cursors.

Cursor concurrency is set with the DECLARE CURSOR statement. An example to declare a dynamic SCROLL_LOCK cursor on all rows of the TRANS table is as follows:

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
DECLARE MyCursor CURSOR DYNAMIC SCROLL_LOCKS
FOR
      SELECT    *
      FROM      dbo.TRANS;
```

Locking Hints

Locking behavior can be further modified by providing locking hints in the WITH parameter of the FROM clause in SELECT statements. Figure 10A-76 lists several of the locking hints available with SQL Server. The first four hints override the transaction isolation level; the next two influence the type of lock issued.[15]

Consider the following statements:

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
DECLARE MyCursor CURSOR DYNAMIC SCROLL_LOCKS
FOR
      SELECT    *
      FROM      dbo.TRANS WITH READUNCOMMITTED NOLOCK;
```

Without the locking hints, the cursor MyCursor would have REPEATABLE READ isolation and would issue update locks on all rows read. The locks would be held until the transaction committed. With the locking hints, the isolation level for this cursor becomes READ UNCOMMITTED. Furthermore, the specification of NOLOCK changes this cursor from DYNAMIC to READ_ONLY.

Consider another example:

```
/* *** EXAMPLE CODE – DO NOT RUN *** */
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
DECLARE MyCursor CURSOR DYNAMIC SCROLL_LOCKS
FOR
      SELECT    *
      FROM      dbo.TRANS WITH HOLDLOCK;
```

Here the locking hint will cause SQL Server to hold update locks on all rows read until the transaction commits. The effect is to change the transaction isolation level for this cursor from REPEATABLE READ to SERIALIZABLE.

[15] For a full list of locking hints, see the documentation at *https://docs.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-table*.

In general, the beginner is advised not to provide locking hints. Rather, until you have become an expert, set the isolation level and cursor concurrency to appropriate values for your transactions and cursors and leave it at that. In fact, the SQL Server 2017 documentation specifically suggests relying on the SQL Server query optimizer and using locking hints only when absolutely necessary.

Microsoft SQL Server 2017 Security

We discussed security in general terms in Chapter 9. Here we will summarize how those general ideas pertain to SQL Server security. Starting with SQL Server 2005 and continuing with SQL Server 2017, Microsoft introduced a more complex version of the model of DBMS security shown in Figure 9-14. The SQL Server 2017 model is shown in Figure 10A-77.

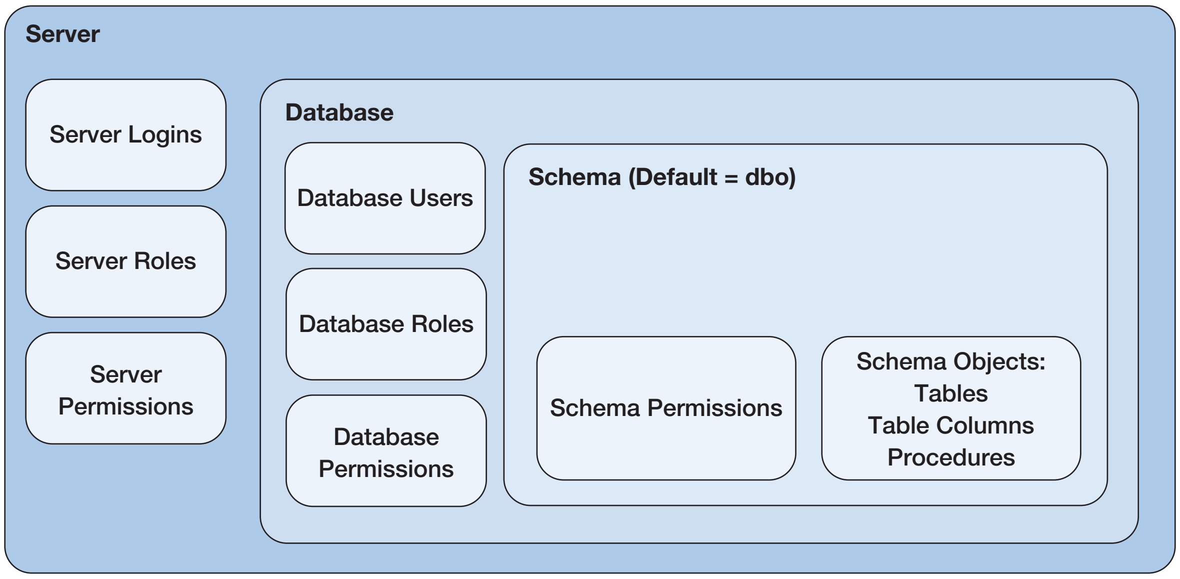
As usual, security consists of authentication and authorization. As shown in Figure 10A-6, we have to log into SQL Server itself first, and we log in using one of the server logins illustrated in Figure 10A-77. This login will be a member of one or more server roles, and each of these roles has specific permissions at the server level.

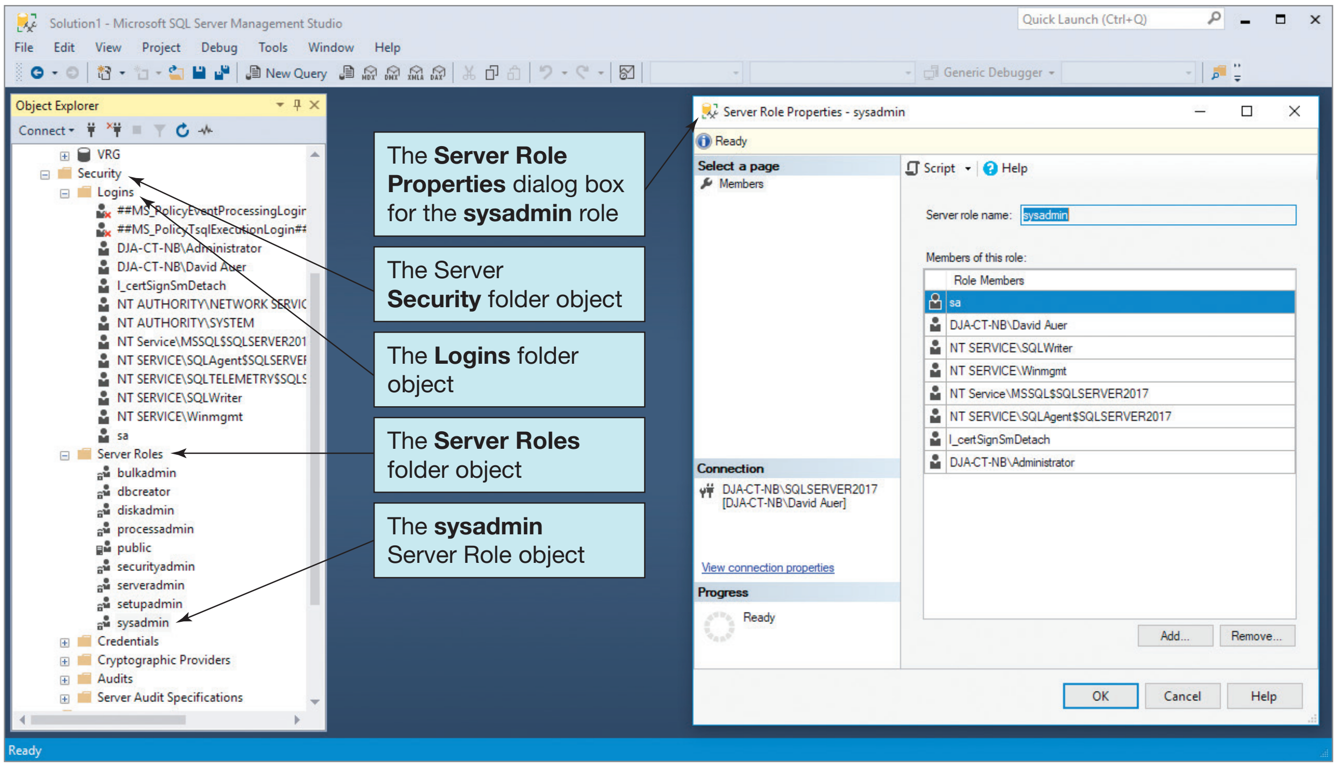
We are currently logged into SQL Server 2017 (on a computer running Windows 10 and named DJA-CT-NT) as the login DJA-CT-NT\David Auer (see the username in Figure 10A-6, which is grayed out but is still the username being used), which means that we logged in based on the local computer user account *David Auer*, and as shown in Figure 10A-78, that login is assigned to the *sysadmin* role. This means that the user *Auer* has all the privileges of an SQL Server 2017 systems administrator (which is everything!). SQL Server 2017 logins can be based on Windows operating system security (recommended by Microsoft), SQL Server–specific logins (see the *sa* login in Figure 10A-78—*sa* is the SQL Server system administrator account, and if you use it, be sure to set a secure password for it), or both (which is how our SQL Server 2017 instance is set up).

To see the server permissions that the login *David Auer* has, we look at the Permissions page of the Server Properties dialog box (right-click the SQL Server object at the top of the Object Explorer, and then click the Properties command and finally, select the Permissions page). Figure 10A-79 shows the effective permissions for the login *David Auer*.

FIGURE 10A-77

SQL Server 2017 Security Model




The Server Role Properties dialog box for the sysadmin role
The Server Security folder object
The Logins folder object
The Server Roles folder object
The sysadmin Server Role object
Solution1 - Microsoft SQL Server Management Studio
Object Explorer
Server Role Properties - sysadmin
Select a page
Members
Script
Help
OK
Cancel
Add...
Remove...

Server role name: sysadmin

Members of this role:

Role Members
sa
DJA-CT-NB\David Auer
NT SERVICE\SQLWriter
NT SERVICE\Winmgmt
NT Service\MSSQL$SQLSERVER2017
NT SERVICE\SQLAgent$SQLSERVER2017
l_certSignSmDetach
DJA-CT-NB\Administrator

Connection: DJA-CT-NB\SQLSERVER2017 [DJA-CT-NB\David Auer]

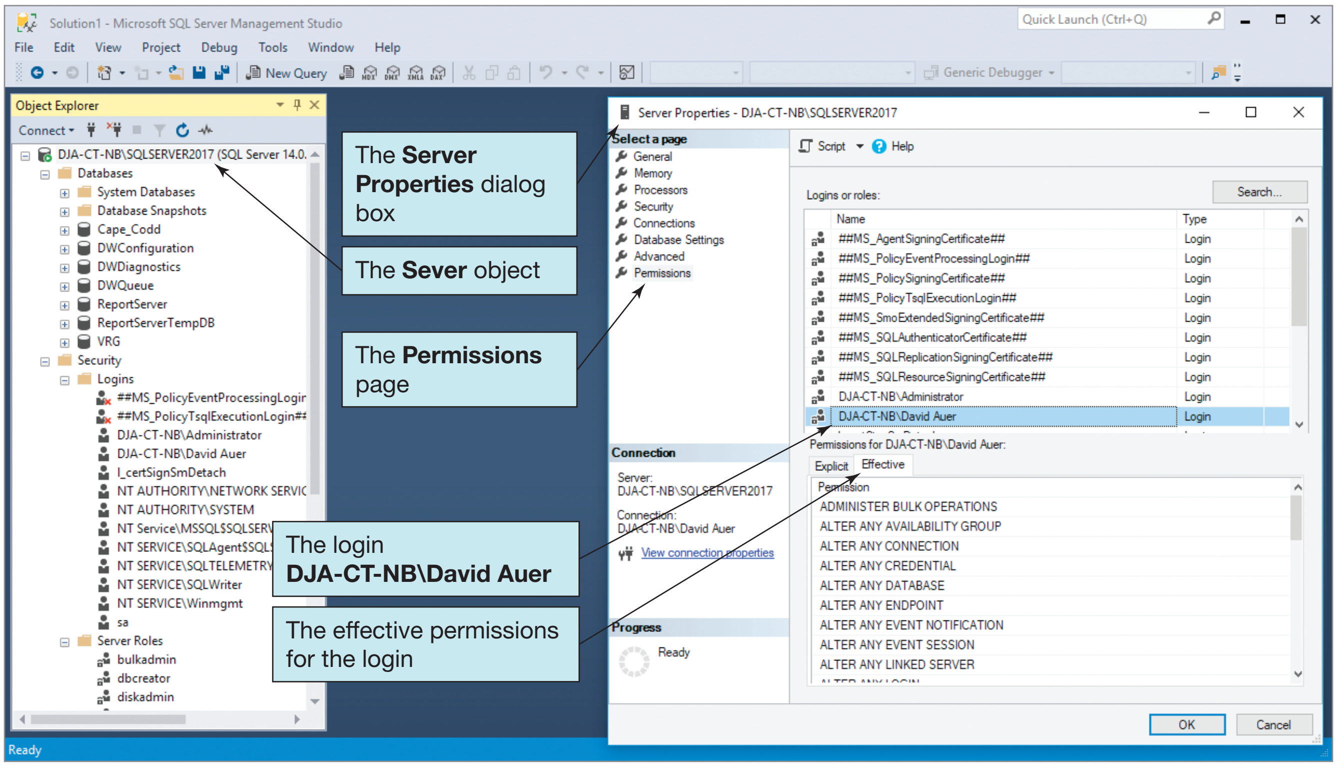
FIGURE 10A-78

Server Logins and Server Roles

This gives us an overview of the SQL Server authorization model. **SQL Server principals** (either **server principals**, such as logins and server roles, or **database principals**, such as users and database roles) are granted permissions to **SQL Server securable objects** (databases, schemas, tables, procedures, etc.). The new concept here is the schema. An SQL Server schema is a named container for SQL Server. In other words,

FIGURE 10A-79

Effective Server Permissions for a Login



Logins or roles:

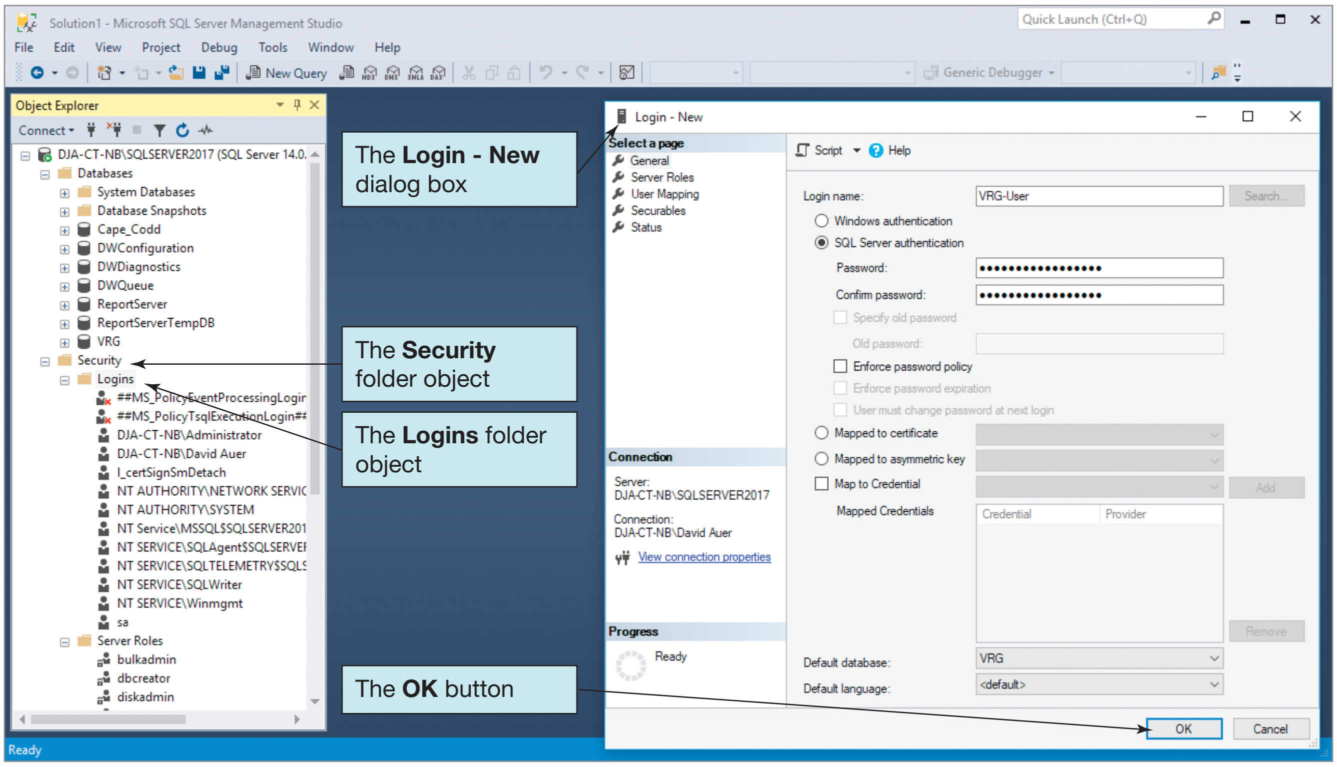
Name	Type
##MS_AgentSigningCertificate##	Login
##MS_PolicyEventProcessingLogin##	Login
##MS_PolicySigningCertificate##	Login
##MS_PolicyTsqlExecutionLogin##	Login
##MS_SmoExtendedSigningCertificate##	Login
##MS_SQLAuthenticatorCertificate##	Login
##MS_SQLReplicationSigningCertificate##	Login
##MS_SQLResourceSigningCertificate##	Login
DJA-CT-NB\Administrator	Login
DJA-CT-NB\David Auer	Login

Permissions for DJA-CT-NB\David Auer:

Permission
ADMINISTER BULK OPERATIONS
ALTER ANY AVAILABILITY GROUP
ALTER ANY CONNECTION
ALTER ANY CREDENTIAL
ALTER ANY DATABASE
ALTER ANY ENDPOINT
ALTER ANY EVENT NOTIFICATION
ALTER ANY EVENT SESSION
ALTER ANY LINKED SERVER

Server: DJA-CT-NB\SQLSERVER2017

Connection: DJA-CT-NB\David Auer



DJA-CT-NB\SQLSERVER2017 (SQL Server 14.0...
Databases: System Databases, Database Snapshots, Cape_Codd, DWConfiguration, DWDiagnostics, DWQueue, ReportServer, ReportServerTempDB, VRG
Security
Logins: ##MS_PolicyEventProcessingLogin##, ##MS_PolicyTsqlExecutionLogin##, DJA-CT-NB\Administrator, DJA-CT-NB\David Auer, l_certSignSmDetach, NT AUTHORITY\NETWORK SERVICE, NT AUTHORITY\SYSTEM, NT Service\MSSQL$SQLSERVER2017, NT SERVICE\SQLAgent$SQLSERVER..., NT SERVICE\SQLTELEMETRY$SQLS..., NT SERVICE\SQLWriter, NT SERVICE\Winmgmt, sa
Server Roles: bulkadmin, dbcreator, diskadmin

Login name: VRG-User
Windows authentication ( ) / SQL Server authentication (●)
Password: ●●●●●●●●●●●●●●●●●
Confirm password: ●●●●●●●●●●●●●●●●●
Enforce password policy ☐
Server: DJA-CT-NB\SQLSERVER2017
Connection: DJA-CT-NB\David Auer
Default database: VRG
Default language: <default>

FIGURE 10A-80

Creating the VRG-User Login

SQL objects such as tables and stored procedures are now held in a container called a *schema*. A schema can be owned by any SQL server principal. The main advantage is that schema ownership can be transferred, and thus a database principal can be deleted without the loss of the objects owned by that principal.[16] The default schema, the one an object will automatically be assigned to, is the **dbo schema**. In earlier versions of SQL Server, dbo (database owner) was only a *user*; now dbo is also a *schema name*.

As an example, let's create a login for use by the View Ridge Gallery.

Creating a New Login

1. Expand the **Security** folder object in the Object Explorer so the Logins folder object is displayed.
2. Right-click the **Logins** folder object to display the shortcut menu.
3. In the shortcut menu, click the **New Login** command. The Login–New dialog box is displayed.
4. The completed new login data is shown in Figure 10A-80. Use that figure for reference in the following steps.
5. In the Login name text box, type in the login name **VRG-User**.
6. Click the **SQL Server authentication** radio button to enable the use of SQL Server authentication mode.
7. In the Password text box, type the password **VRG-User+password**.
8. In the Confirm password text box, type the password **VRG-User+password**.
9. Uncheck the **Enforce password policy** check box to disable enforcing password policy, disable enforcing password expiration, and disable forcing a password change.
   - **NOTE:** This step is to simplify the use of this login in our test environment only. Do *not* do this in a production environment.
10. Select **VRG** as the Default database.
11. Click the **OK** button. The new login is created.

[16] For more information, see "User-Schema Separation" at *http://msdn.microsoft.com/en-us/library/ms190387.aspx*.

SQL Server 2017 Database Security Settings

Now we need to move to the database level of security. Here we create specific database users and assign database roles (and associated permissions) to those users. SQL Server 2017 database roles and permissions are shown in Figure 10A-81. To illustrate the use of a database-specific user, we will create a database user based on the VRG-User login we have created for use by the View Ridge Gallery.

Creating a New Database User

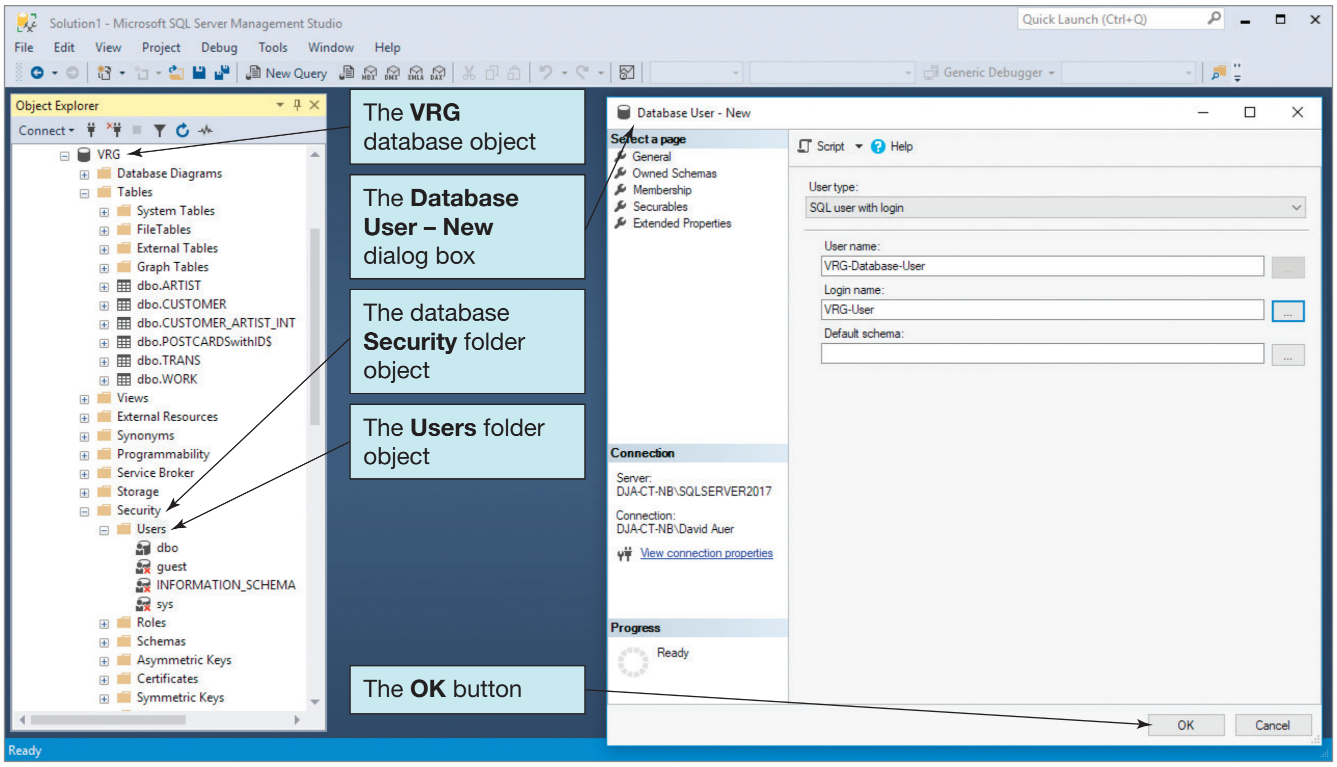
1. Expand the **VRG** database object in the Object Explorer so the database Security folder object is displayed.
2. Expand the **VRG Security** folder object in the Object Explorer so the Users folder object is displayed.
3. Right-click the **Users** folder object to display a shortcut menu.
4. In the shortcut menu, click the **New User** command. The Database User–New dialog box is displayed.
5. The completed new login data is shown in Figure 10A-82. Use that figure for reference in the following steps.
6. In the User name text box, type in the username **VRG-Database-User**.
7. In the Login name text box, type in the login name **VRG-User**.
   - **NOTE:** You can also browse to the correct login name using the Browse button shown in Figure 10A-82.
8. Click the **OK** button to close the dialog box.

FIGURE 10A-81

SQL Server Fixed Database Roles

Fixed Database Role	Database-Specific Permissions	DBMS Server Permissions
db_accessadmin	Permissions granted: ALTER ANY USER, CREATE SCHEMA Permissions granted with GRANT option: CONNECT	Permissions granted: VIEW ANY DATABASE
db_backupoperator	Permissions granted: BACKUP DATABASE, BACKUP LOG, CHECKPOINT	Permissions granted: VIEW ANY DATABASE
db_datareader	Permissions granted: SELECT	Permissions granted: VIEW ANY DATABASE
db_datawriter	Permissions granted: DELETE, INSERT, UPDATE	Permissions granted: VIEW ANY DATABASE
db_ddladmin	Permissions granted: See SQL Server 2017 documentation	Permissions granted: VIEW ANY DATABASE
db_denydatareader	Permissions denied: SELECT	Permissions granted: VIEW ANY DATABASE
db_denydatawriter	Permissions denied: DELETE, INSERT, UPDATE	Permissions granted: VIEW ANY DATABASE
db_owner	Permissions granted with GRANT option: CONTROL	Permissions granted: VIEW ANY DATABASE
db_securityadmin	Permissions granted: ALTER ANY APPLICATION ROLE, ALTER ANY ROLE, CREATE SCHEMA, VIEW DEFINITION	Permissions granted: VIEW ANY DATABASE

Note: For the definitions of each of the SQL Server permissions shown in the table, consult the SQL Server documentation.



User type: SQL user with login

User name: VRG-Database-User

Login name: VRG-User

Default schema:

Connection

Server: DJA-CT-NB\SQLSERVER2017

Connection: DJA-CT-NB\David Auer

View connection properties

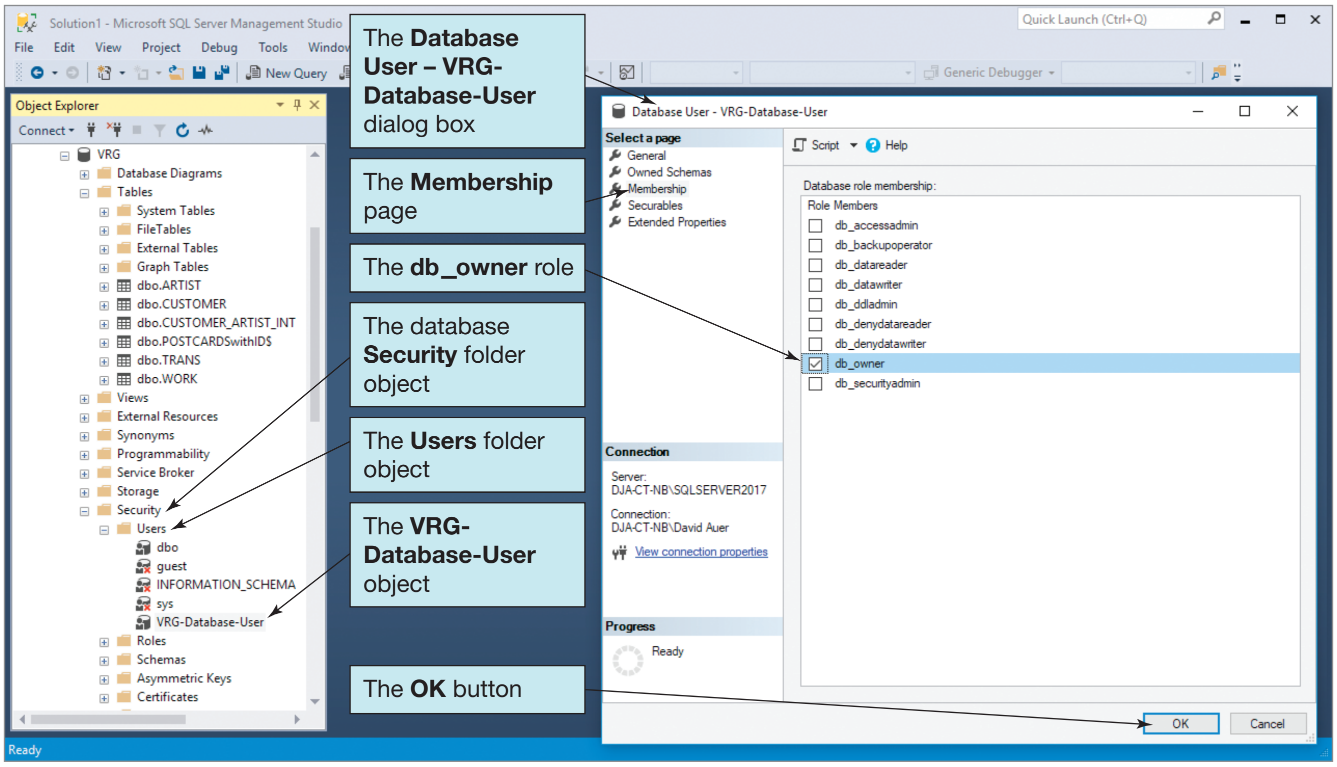
Progress

Ready

FIGURE 10A-82

Creating the VRG-Database-User Database User

9. Right-click the newly created **VRG-Database-User** object to display the shortcut menu, and then click the **Properties** command to display the Database User–VRG-Database-User dialog box, as shown in Figure 10A-83.
10. In the **Select a page** pane, click **Membership** to display the Database role membership page.



Database role membership:

Role Members
☐ db_accessadmin
☐ db_backupoperator
☐ db_datareader
☐ db_datawriter
☐ db_ddladmin
☐ db_denydatareader
☐ db_denydatawriter
☑ db_owner
☐ db_securityadmin

Connection

Server: DJA-CT-NB\SQLSERVER2017

Connection: DJA-CT-NB\David Auer

View connection properties

Progress

Ready

FIGURE 10A-83

Assigning the db_owner Role to VRG-Database-User

11. In the Database role membership list, check **db_owner**.
12. Click the **OK** button. The new login is created.

Now we have completed creating the needed logins and database users for the VRG database. We will use these in other chapters when we develop applications based on this database.

BY THE WAY Note that in SQL Server 2017 we can create, and have in fact created, both a server login and a database user. We then assigned database *role memberships* (and thus permissions) to the database user. However, as an alternative, we could have assigned the database permissions to the user login using *user mappings!*

This makes SQL Server database permissions administration rather complex, and somewhat a matter of DBA preferences. We did it this way to illustrate the creation of both server logins and database users.

Microsoft SQL Server 2017 Backup and Recovery

When you create an SQL Server 2017 database, both data and log files are created. As explained in Chapter 9, these files should be backed up periodically. When backups are available, it is possible to recover a failed database by restoring it from a prior database save and applying changes from the log.

To recover a database with SQL Server, the database is restored from a prior database backup and then log after-images are applied to the restored database. When the end of the log is reached, changes from any transaction that failed to commit are then rolled back.

It is also possible to process the log to a particular point in time or to a transaction mark. For example, the following statement causes a mark labeled *NewCustomer* to be placed into the log every time this transaction is run:

```
BEGIN TRANSACTION NewCustomer WITH MARK;
```

If this is done, the log can be restored to a point either just before or just after the first *NewCustomer* mark or the first *NewCustomer* mark after a particular point in time. The restored log can then be used to restore the database. Such marks consume log space, however, so they should not be used without good reason.

Backing Up a Database

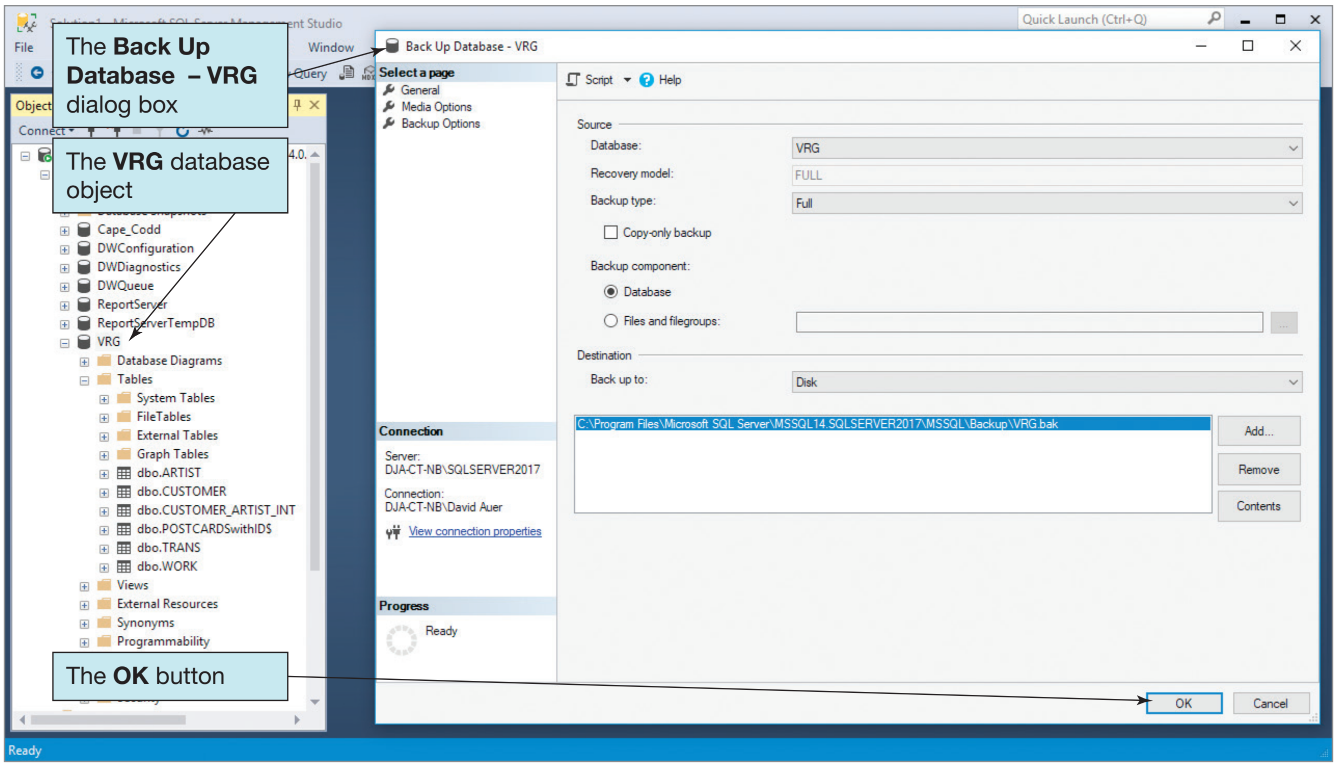
SQL Server 2017 supports several types of backup. We can, for example, choose to do a full backup, a differential backup, or a transaction log backup. A **full backup** backs up the complete database and the transaction logs. A **differential backup** backs up only the changes since the last full backup, which is useful if a full backup takes a long time. This allows us to make periodic full backups (e.g., once a week) and still capture daily changes to the database using differential backups. A **transaction log backup** backs up only the current transaction log. This is useful in keeping a series of transaction log files backed up more currently than even our differential backups. For example, we may do a transaction log backup every six hours.

Here we will make a complete backup of the VRG database. Other backup options may require the use of backup device objects and management policies. A full discussion of these topics is beyond the scope of this book.[17]

Backing Up the VRG Database

1. Right-click the **VRG** database object in the Object Explorer to display a shortcut menu.
2. In the shortcut menu, click the **Tasks** command, then click the **Backup** command. The Back Up Database–VRG dialog box is displayed, as shown in Figure 10A-84.
3. SQL Server 2017 has already set up all the backup options correctly for a simple full backup of the database.
4. Click the **OK** button. The database backup is created.

[17] For more information, see the Microsoft SQL Server 2017 Books Online at *https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation.*



Back Up Database - VRG

Source
Database: VRG
Recovery model: FULL
Backup type: Full
Copy-only backup
Backup component:
(●) Database
( ) Files and filegroups:

Destination
Back up to: Disk
C:\Program Files\Microsoft SQL Server\MSSQL14.SQLSERVER2017\MSSQL\Backup\VRG.bak

Connection
Server: DJA-CT-NB\SQLSERVER2017
Connection: DJA-CT-NB\David Auer
View connection properties

Progress
Ready

FIGURE 10A-84

Backing Up the VRG Database

We could use the same backup task we just ran to make differential backups if we wanted—we would simply choose Differential instead of Full as the backup type.

In a more production-oriented backup system, the transaction log needs to be periodically backed up to ensure that its contents are preserved. Further, the transaction log must be backed up before it can be used to recover a database. Backups can be made either to disk or to tape—our backup was just made to disk. When possible, the backups should be made to devices other than those that store the operational database and log. Backing up to removable devices allows the backups to be stored in a location physically removed from the database data center. This is important for recovery in the event of disasters caused by floods, hurricanes, earthquakes, and the like.

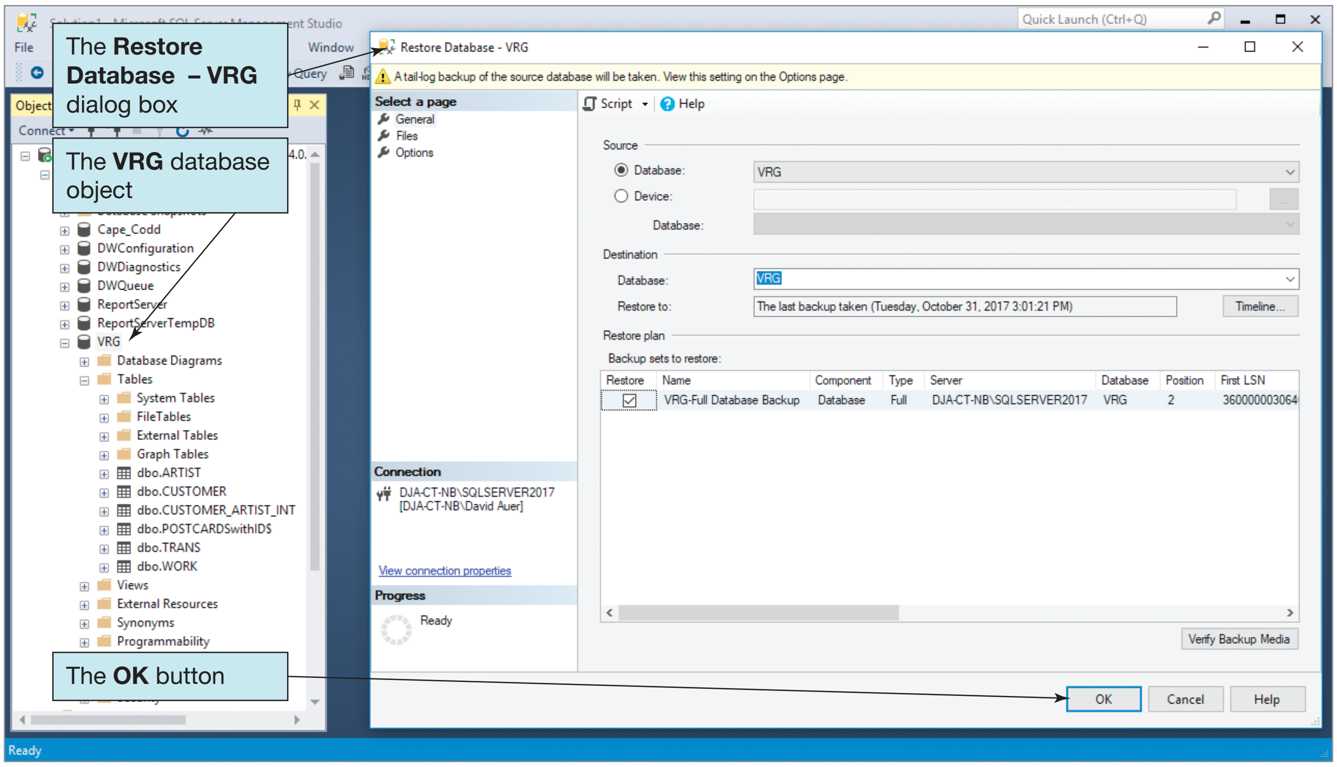
SQL Server Recovery Models

SQL Server supports three recovery models: simple, full, and bulk logged. With the simple recovery model, no logging is done. The only way to recover a database is to restore it to the last backup. Changes made since that last backup are lost. The simple recovery model can be used for a database that is never changed—one with the names and locations of the occupants of a full graveyard, for example—or for one that is used for read-only analysis of data that are copied from some transactional database.

With full recovery, all database changes are logged. With bulk-logged database recovery, all changes are logged except those that cause large log entries. With bulk-logged recovery, changes to large text and graphic data items are not recorded to the log, actions such as CREATE INDEX are not logged, and some other bulk-oriented actions are not logged. An organization uses bulk-logged recovery if conserving log space is important and if the data used in the bulk operations are saved in some other way.

Restoring a Database

If the database and log files have been properly backed up, restoring the database is straightforward.



Restore Database - VRG

A tail-log backup of the source database will be taken. View this setting on the Options page.

Source
(● Database:) VRG
Device:

Destination
Database: VRG
Restore to: The last backup taken (Tuesday, October 31, 2017 3:01:21 PM)

Restore plan
Backup sets to restore:

Restore	Name	Component	Type	Server	Database	Position	First LSN
✓	VRG-Full Database Backup	Database	Full	DJA-CT-NB\SQLSERVER2017	VRG	2	360000003064

Connection
DJA-CT-NB\SQLSERVER2017
[DJA-CT-NB\David Auer]
View connection properties

Progress
Ready

Object Explorer: Cape_Codd, DWConfiguration, DWDiagnostics, DWQueue, ReportServer, ReportServerTempDB, VRG, Database Diagrams, Tables, System Tables, FileTables, External Tables, Graph Tables, dbo.ARTIST, dbo.CUSTOMER, dbo.CUSTOMER_ARTIST_INT, dbo.POSTCARDSwithIDS, dbo.TRANS, dbo.WORK, Views, External Resources, Synonyms, Programmability

FIGURE 10A-85

Restoring the VRG Database

Restoring the VRG Database

1. Right-click the **VRG** database object in the Object Explorer to display a shortcut menu.
2. In the shortcut menu, click the **Tasks** command, then click the **Restore** command. The Restore Database–VRG dialog box is displayed, as shown in Figure 10A-85.
3. SQL Server 2017 has already set up all the restore options correctly for a simple full restore of the database.
4. If we were going to actually restore the VRG database, we would click the OK button at this point, and the database backup would be restored. However, there is no need to do so at this point, so click the **Cancel** button.

You can use backup and restore to transfer a database to another computer or user (such as your professor!). Just do a full backup of the database you want to share to a file, say, the file *MyBackup*. Then create a new database on another computer, and name it whatever you want. Then restore the database using the file *MyBackup*.

Database Maintenance Plans

You can create a database maintenance plan to facilitate the making of database and log backups, among other tasks, using SQL Server policy management. This topic, however, is beyond the scope of this book.[18]

Topics Not Discussed in This Chapter

Several important SQL Server topics are beyond the scope of this discussion. For one, SQL Server provides utilities to measure database activity and performance. The DBA can use these utilities when tuning the database. Another facility not described here is connecting Access to SQL Server. You can check the Access documentation for more information about this topic.

[18] For more information, see the Microsoft SQL Server 2017 Books Online at *https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation*.

SQL Server 2017 provides facilities to support distributed database processing (called **replication** in SQL Server). Although very important in its own right, distributed database processing is discussed briefly in Chapter 12.[19]

Finally, SQL Server has facilities for processing database views in the form of XML documents. We will discuss those facilities in Chapter 11 and Appendix I, "XML."

Summary

SQL Server 2017 can be installed on computers running various versions of the Microsoft Windows OS, Microsoft Windows Server, several distributions of Linux (Red Hat, Ubuntus, and SUSE Linux Enterprise Server), and is available for use in the Microsoft Azure cloud service. Tables, views, indexes, and other database structures can be created in two ways. One is to use the graphical design tools, which are similar to those in Microsoft Access. The other is to write SQL statements to create the structures and then submit them to SQL Server via the SQL Server Management Studio utility. SQL Server supports all of the SQL DDL that you have learned in this text, including the IDENTITY keyword for defining surrogate keys. The only change required for the View Ridge schema was to change the name of the TRANSACTION table to TRANS.

Indexes are special data structures used to improve performance. SQL Server automatically creates an index on all primary and foreign keys. Additional indexes can be created using CREATE INDEX or the Manage Index graphical tool. SQL Server supports clustered and nonclustered indexes.

Data can be imported directly from Excel spreadsheets into SQL Server tables using the SQL Server Import and Export Wizard.

SQL Server supports a language called Transact-SQL, which surrounds basic SQL statements with programming constructs such as parameters, variables, and logic structures, such as IF, WHILE, and so forth. User-defined functions, stored procedures, and triggers can all be defined in Transact-SQL.

SQL Server databases can be processed from application programs coded in standard programming languages, such as Visual Basic .NET, Visual C# .NET, or Visual C++ .NET, or application logic can be placed in user-defined functions, stored procedures, and triggers. Stored procedures can be invoked from standard languages or from VBScript and JScript in Web pages. In this chapter, stored procedures were invoked from the SQL Server Query Manager. This technique should be used only during development and testing.

For security reasons, no one should process an SQL Server operational database directly through utilities such as SQL Server Management Studio. This chapter demonstrated SQL Server triggers for computing default values, for enforcing a data constraint, for updating a view, and for enforcing a mandatory child referential integrity constraint.

Three factors determine the concurrency control behavior of SQL Server: the transaction isolation level, the cursor concurrency setting, and locking hints provided in the SELECT clause. These factors are summarized in Figure 10A-76. Behavior also changes depending on whether actions occur in the context of transactions or cursors or independently. Given these behavior declarations, SQL Server places locks on behalf of the developer. Locks may be placed at many levels of granularity and may be promoted or demoted as work progresses.

SQL Server supports log backups and both complete and differential database backups. Three recovery models are available: simple, full, and bulk logged. With simple recovery, no logging is done, nor are log records applied. Full recovery logs all database operations and applies them for restoration. Bulk-logged recovery omits certain transactions that would otherwise consume large amounts of space in the log.

[19]For more information, see the Microsoft SQL Server 2017 Books Online at *https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation*.

Key Terms

/* (slash asterisk) and */ (asterisk slash)
- - (two dashes)
.NET Framework 3.5 Service Pack 1
BEGIN ... END keywords
bulk INSERT statement
CHARINDEX
CLOSE CURSOR keywords
clustered index
cmdlets
command-line utility
Connect to Server dialog box
database diagram
database owner
database principals
dbo schema
DEALLOCATE CURSOR keywords
DECLARE CURSOR keywords
default instance
delimited identifier
differential backup
extract, transform, and load (ETL)
FETCH keyword
full backup
GO command
graphical user interface (GUI) utility
IF ... ELSE keywords
import
index
integrated development environment (IDE)
Intellisense
instance
Meltdown
Microsoft Command Line Utilities 13.1 for SQL Server
Microsoft SQL Server
Microsoft SQL Server Management Studio
Microsoft SQL Server Reporting Services
Microsoft Windows PowerShell
named instance
nonclustered index
OPEN CURSOR keywords
Oracle Java Runtime Environment (JRE)
parameter
procedural language extensions
replication
reporting system
reserved word
RETURN keyword
SCROLL_LOCK
server principals
SNAPSHOT transaction isolation level
Spectre
spreadsheet
SQL/Persistent Stored Module (SQL/PSM)
SQL ALTER FUNCTION statement
SQL ALTER TABLE statement
SQL ALTER PROCEDURE statement
SQL CMD utility
SQL Common Language Runtime (CLR)
SQL CREATE OR ALTER FUNCTION statement
SQL CREATE PROCEDURE statement
SQL CREATE OR ALTER PROCEDURE statement
SQL script
SQL script comment
SQL Server 2017 Express
SQL Server 2017 for Microsoft Windows Latest Cumulative Update
SQL Server bit data type
SQL Server function @@Identity
SQL Server Import and Export Wizard
SQL Server principals
SQL Server schema
SQL Server securable objects
sqlps cmdlet
stored function
SUBSTRING
transaction isolation level
transaction log backup
Transact-SQL (T-SQL)
Transact-SQL @@FETCH_STATUS function
Transact-SQL BEGIN TRANSACTION command
Transact-SQL COMMIT TRANSACTION command
Transact-SQL control-of-flow language
Transact-SQL CONVERT function
Transact-SQL CREATE FUNCTION statement
Transact-SQL cursor
Transact-SQL GETDATE() function
Transact-SQL IDENTITY ({StartValue}, {Increment}) property
Transact-SQL IDENTITY_INSERT property
Transact-SQL PRINT command
Transact-SQL ROLLBACK TRANSACTION command
Transact-SQL SET ANSI_NULLS ON command
Transact-SQL SET NOCOUNT ON command
Transact-SQL SET QUOTED_IDENTIFIER ON command
Transact-SQL USE [{DatabaseName}] command
user-defined function
variable
WHILE keyword
worksheet
xSQLServer cmdlet

Review Questions

If you have not already installed SQL Server 2017 (or do not otherwise have it available to you), you need to install a version of it at this point.

Review Questions 10A.1–10A.15 are based on a database named MEDIA that is used to record data about photographs that are stored in the database.

10A.1 Create a database named MEDIA in SQL Server 2017. Use the default settings for file sizes, names, and locations.

FIGURE 10A-86

Column Characteristics for the MEDIA Database PICTURE Table

Column Name	Type	Key	Required	Remarks
PictureID	Integer	Primary Key	Yes	Surrogate Key (1, 1)
PictureName	Character (35)	No	Yes	
PictureDescription	Character (255)	No	No	Use Varchar, default “None”
DateTaken	Date	No	Yes	
PictureFileName	Character (45)	No	Yes	Use Varchar

10A.2 In the SQL Server Management Studio folder structure in your *Documents* folder, create a folder named *DBP-e15-Media-Database* in the *Projects* folder. Use this folder to save and store *.sql scripts containing the SQL statements that you are asked to create in the remaining Review Questions in this section.

10A.3 Using the MEDIA database, open a new tabbed SQL Query window, and save it as *MEDIA-CH10A-RQ-Solutions.sql* in the *DBP-e15-Media-Database* folder. Use this script to record and save the SQL statements that you are asked to create in the remaining Review Questions in this section.

10A.4 Write an SQL CREATE TABLE statement to create a table named PICTURE using the column characteristics as shown in Figure 10A-86. Run the SQL statement to create the PICTURE table in the MEDIA database.

10A.5 Write an SQL CREATE TABLE statement to create the table SLIDE_SHOW using the column characteristics as shown in Figure 10A-87. Run the SQL statement to create the SLIDE_SHOW table in the MEDIA database.

10A.6 Write an SQL CREATE TABLE statement to create the table SLIDE_SHOW_PICTURE_INT using the column characteristics as shown in Figure 10A-88. SLIDE_SHOW_PICTURE_INT is an intersection table between PICTURE and SLIDE_SHOW, so create appropriate relationships between PICTURE and SLIDE_SHOW_PICTURE_INT and between SLIDE_SHOW and SLIDE_SHOW_PICTURE_INT. Set the referential integrity properties to disallow any deletion of a SLIDE_SHOW row that has any SLIDE_SHOW_PICTURE_INT rows related to

FIGURE 10A-87

Column Characteristics for the MEDIA Database SLIDE_SHOW Table

Column Name	Type	Key	Required	Remarks
ShowID	Integer	Primary Key	Yes	Surrogate Key (1000, 1)
ShowName	Character (35)	No	Yes	
ShowDescription	Character (255)	No	No	Use Varchar, default “None”
Purpose	Character (15)	No	Yes	Data value must be one of the following: Home Office Family Recreation Sports Pets

FIGURE 10A-88

Column Characteristics for the MEDIA Database SLIDE_SHOW_PICTURE_INT Table

Column Name	Type	Key	Required	Remarks
ShowID	Integer	Primary Key, Foreign Key	Yes	REF: SLIDE_SHOW
PictureID	Integer	Primary Key, Foreign Key	Yes	REF: PICTURE

it. Set the referential integrity properties to cascade deletions when a PICTURE is deleted. Do not cascade updates to PICTURE.PictureID.

10A.7 Write SQL INSERT statements to populate the PICTURE table using the data shown in Figure 10A-89. Run the SQL statements to populate the PICTURE table.

10A.8 Write SQL INSERT statements to populate the SLIDE_SHOW table using the data shown in Figure 10A-90. Run the SQL statements to populate the SLIDE_SHOW table.

10A.9 Write SQL INSERT statements to populate the SLIDE_SHOW_PICTURE_INT table using the data shown in Figure 10A-91. Run the SQL statements to populate the SLIDE_SHOW_PICTURE_INT table.

10A.10 Write an SQL statement to create an SQL view named *PopularShowsView* that has SLIDE_SHOW.ShowName and PICTURE.PictureName for all slide shows that have a Purpose of either 'Home' or 'Pets'. Execute this statement to create the view in the MEDIA database.

10A.11 Run an SQL SELECT query to demonstrate that the view *PopularShowsView* was constructed correctly.

10A.12 Use an SQL ALTER VIEW statement to modify the *PopularShowsView* view to include PICTURE.PictureDescription and PICTURE.PictureFileName.

10A.13 Run an SQL SELECT query to demonstrate that the modified *PopularShowsView* view was constructed correctly.

FIGURE 10A-89

Sample Data for the MEDIA Database PICTURE Table

PictureID	PictureName	PictureDescription	DateTaken	PictureFileName
1	SpotAndBall	My dog Spot chasing a ball	2018-09-07	spot00001.jpg
2	SpotAndCat	My dog Spot chasing a cat	2018-09-08	spot00002.jpg
3	SpotAndCar	My dog Spot chasing a car	2018-10-11	spot00003.jpg
4	SpotAndMailman	My dog Spot chasing a mailman - BAD DOG!	2018-11-22	spot00004.jpg
5	TheJudgeAndI	I explain that Spot is really a good dog and did not mean to chase the mailman	2018-12-13	me00001.jpg

FIGURE 10A-90

Sample Data for the MEDIA Database SLIDE_SHOW Table

ShowID	ShowName	ShowDescription	Purpose
1000	My Dog Spot	My dog Spot likes to chase things	Pets
1001	My Day In Court	I explain that Spot is really a good dog	Home

FIGURE 10A-91

Sample Data for the MEDIA Database SLIDE_SHOW_PICTURE_INT Table

ShowID	PictureID
1000	1
1000	2
1000	3
1000	4
1001	4
1001	5

10A.14 Can the SQL DELETE statement be used with the *PopularShowsView* view? Why or why not?

10A.15 Under what circumstances can the *PopularShowsView* view be used for inserts and modifications?

Review Questions 10A.16–10A.31 are about terms and concepts discussed in this chapter.

10A.16 In Figure 10A-62, what is the purpose of the @RowCount variable?

10A.17 In Figure 10A-62, why is the SELECT statement that begins SELECT @CustomerID necessary?

10A.18 Explain how you would change the stored procedure in Figure 10A-62 to connect the customer to all artists who either (a) were born before 1900 or (b) had a null value for DateOfBirth.

10A.19 Explain the purpose of the transaction shown in Figure 10A-64.

10A.20 What happens if an incorrect value of Copy is input to the stored procedure in Figure 10A-64?

10A.21 In Figure 10A-64, what happens if the ROLLBACK statement is executed?

10A.22 In Figure 10A-66, why is SUM used instead of AVG?

10A.23 What are the three primary factors that influence SQL Server locking behavior?

10A.24 Explain why the strategy for storing CHECK constraint values in a separate table is better than implementing them in a table-based constraint. How can this strategy be used to implement the constraint on ARTIST.Nationality?

10A.25 Explain why the CustomerInterestsView view in Figure 10A-40 is not updatable. Describe the logic of the INSTEAD OF UPDATE trigger in Figure 10A-70.

10A.26 Explain what limitation must be enforced for the trigger in Figure 10A-70 to be effective.

10A.27 Explain the meaning of each of the transaction isolation levels under Options shown in Figure 10A-76.

10A.28 Explain the meaning of each of the cursor concurrency settings listed in Figure 10A-76.

10A.29 What is the purpose of locking hints?

10A.30 What is the difference between full and differential backups? Under what conditions are full backups preferred? Under what conditions are differential backups preferred?

10A.31 Explain how the simple, full, and bulk-logged recovery models differ. Under what conditions would you choose each one?

Exercises

Wedgewood Pacific Exercises

In the Chapter 7 Review Questions, we introduced the Wedgewood Pacific (WP) company and developed the WP database. Two of the tables that are used in the WP database are:

> **DEPARTMENT (<u>DepartmentName</u>, BudgetCode, OfficeNumber, DepartmentPhone)**
>
> **EMPLOYEE (<u>EmployeeNumber</u>, FirstName, LastName, *Department*, Position, *Supervisor*, OfficePhone, EmailAddress)**

Assume that the relationship between these tables is M-M and use them as the basis for your answers to Exercises 10A.32–10A.39.

10A.32 Create a database named *WP-CH10A-PQ* in SQL Server 2017. Use the default settings for file sizes, names, and locations.

10A.33 In the SQL Server Management Studio folder structure in your *My Documents* folder, create a folder named *DBP-e15-WP-CH10A-PQ-Database* in the *Projects* folder. Use this folder to save and store *.sql scripts containing the SQL statements that you are asked to create in the remaining questions in this section.

10A.34 Code an SQL script to create *only* the WP DEPARTMENT and EMPLOYEE tables in the WP-CH10A-PQ database. Run your script to create the tables.

10A.35 Code an SQL script to populate the WP DEPARTMENT and EMPLOYEE tables in the WP-CH10A-PQ database with data from Figures 1-29 and 1-31. Run your script to populate the tables.

10A.36 Code an SQL Server trigger named *Deny_EMPLOYEE_Change_Of_DEPARTMENT* to enforce the constraint that an employee can never change his or her department. Create test data, and demonstrate that your trigger works.

10A.37 Code an SQL Server trigger named *Allow_Deletion_Of_DEPARTMENT* to allow the deletion of a department if it has only one employee. Assign the last employee to the Human Resources department. Create test data, and demonstrate that your trigger works.

10A.38 Design and code a system of triggers to enforce the M-M relationship. Use Figure 10A-72 as an example, but assume that departments with only one employee can be deleted by assigning the last employee in a department to Human Resources. Create test data, and demonstrate that your triggers work.

10A.39 Create an SQL Server 2017 login named WP-CH10A-User, with a password of WP-CH10A-User+password. Create a WP-CH10A-PQ database user named WP-CH10A-Database-User, which is linked to the WP-CH10A-User login. Assign WP-CH10A-Database-User *db_owner* permissions to the WP-CH10A-PQ database.

View Ridge Gallery Exercises

Exercises 10A.40 and 10A.41 are based on the View Ridge Gallery *VRG* database discussed in this chapter.

10A.40 Write SQL statements to accomplish the following tasks and submit them to SQL Server 2017 via the Microsoft SQL Server Management Studio. For steps C-K, save your work in an SQL script named VRG-CH10A-PQ-10A-40.sql.

A. Create an SQL Server 2017 database named *VRG-CH10A-PQ*.

B. In the SQL Server Management Studio folder structure in your *My Documents* folder, create a folder named *DBP-e15-VRG-CH10A-PQ* in the *Projects* folder.

Use this folder to save and store *.sql scripts containing the SQL statements that you are asked to create in the remaining questions in this section.

C. In the VRG-CH10A-PQ database, create the VRG database tables shown in Figure 10A-27, except do *not* create the NationalityValues constraint. *NOTE:* An SQL script to create the VRG database tables is available at *www.pearsonhighered.com/kroenke*. You can modify this SQL script for use in your solution to this question.

D. Populate your database with the sample data from Figure 10A-39. *NOTE:* An SQL script to populate the VRG database tables is available at *www.pearsonhighered.com/kroenke*. You can modify this SQL script for use in your solution to this question.

E. Create all the VRG views discussed in the Chapter 7 section on SQL views.

F. Write a stored procedure named *PrintArtistData* to read the ARTIST table, and display the artist data using the Transact-SQL PRINT command. Using the database data, demonstrate that your stored procedure works.

G. Write a stored procedure named *PrintArtistAndWorkData* that reads the ARTIST and WORK tables and that accepts the name of the artist to display as an input parameter. Your procedure should then display the data for that artist, followed by a display of all the works for that artist stored in WORK. Using the database data, demonstrate that your stored procedure works.

H. Write a stored procedure named *UpdateCustomerPhoneData* to update customer phone data. Assume that your stored procedure receives LastName, FirstName, PriorAreaCode, NewAreaCode, PriorPhoneNumber, and NewPhoneNumber. Your procedure should first ensure that there is only one customer with the values of (LastName, FirstName, PriorAreaCode, PriorPhoneNumber). If not, produce an error message and quit. Otherwise, update the customer data with the new phone number data and print a results message. Create test data, and demonstrate that your stored procedure works.

I. Create a table named ALLOWED_NATIONALITY with one column, called *Nation*. Place the values of all nationalities currently in the VRG database into the table. Write a trigger named *CheckNationality* that will check to determine whether a new or updated value of Nationality resides in this table. If not, write an error message and roll back the insert or change. Create test data, and demonstrate that your trigger works.

J. Create an SQL view named *WorkWithTransactionsView* with all of the data from the WORK and TRANS tables *except* for the surrogate keys. Write an INSTEAD OF trigger on this view that will create new rows in both WORK and TRANS. Create test data, and demonstrate that your trigger works. *Hint:* Recall that you can issue INSERT commands on WORK and TRANS without specifying a value for the surrogate key—SQL Server will provide it.

K. Create an SQL Server login named VRG-CH10A-User, with a password of VRG-CH10A-User+password. Create a VRG-CH10A-PQ database user named VRG-CH10A-Database-User, which is linked to the VRG-CH10A-User login. Assign VRG-CH10-A-Database-User *db_owner* permissions to the VRG-CH10A-PQ database.

10A.41 Write SQL statements to accomplish the following tasks, and submit them to SQL Server 2017 via the Microsoft SQL Server Management Studio. Save your work in an SQL script named VRG-CH10A-PQ-10A-41.sql. This Project Question shows the steps necessary to integrate the POSTCARDS$ table data into the *VRG* database. A database diagram showing how the *VRG* database will

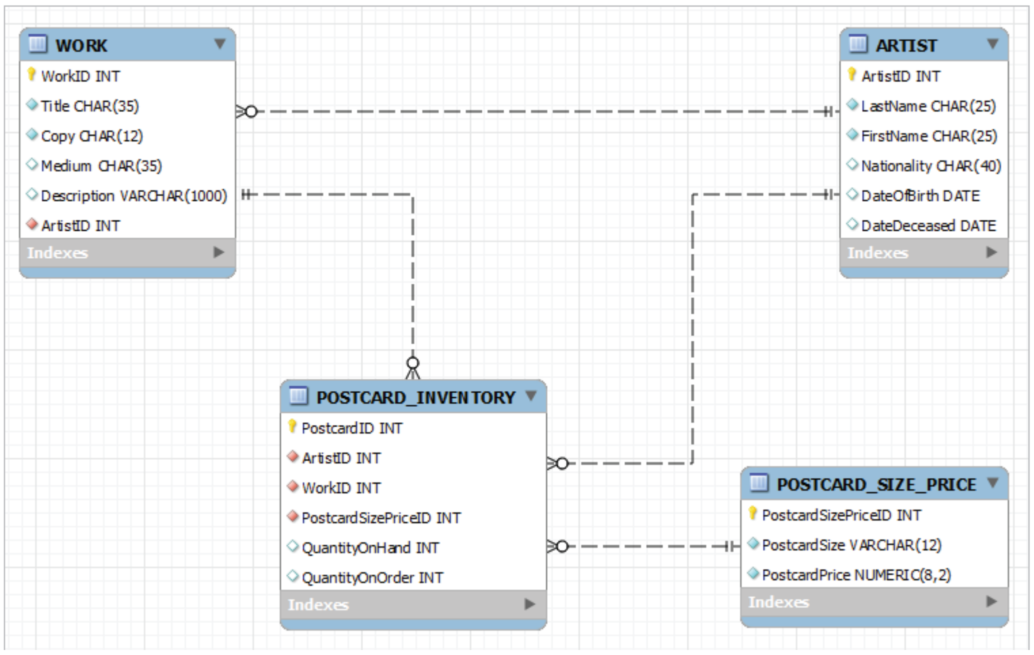


FIGURE 10A-92

Partial Database Design for the Revised VRG Database

appear after these steps are completed (drawn in MySQL Workbench) is shown in Figure 10A-92.

A. If you haven't done so, work through Project Question 10A-40 to create the SQL Server 2017 database named *VRG-CH10A-PQ* as described in that Project Question.

B. Use the steps described in this chapter to:

- Create a Microsoft Excel 2016 workbook containing the POSTCARDS worksheet shown in Figure 10A-44.
- Import the data in the POSTCARDS worksheet into a table in the VRG database named POSTCARDS$.
- Create the *GetLastNameCommaSeparated* user-defined function shown in Figure 10A-58.
- Alter the POSTCARDS$ table to include the ArtistLastName and ArtistID columns as discussed in the text and as shown in Figure 10A-60.
- Populate the POSTCARDS$ table ArtistLastName and ArtistID columns as discussed in the text and as shown in Figure 10A-61.

C. Create a user-defined function named *GetFirstNameCommaSeparated* that will return the first name from a combined name in last-name-first order and with the names separated by a comma and one space. Write an SQL SELECT statement using the POSTCARDS$ table to test your function.

D. Alter the POSTCARDS$ table to include an ArtistFirstName column (Character data, allow NULL values). Use the *GetFirstNameCommaSeparated* function that you created in part C to populate this column.

E. Alter the POSTCARDS$ table to include a WorkID column ((Integer data, allow NULL values). By using and comparing the data in the POSTCARDS$ .WorkTitle and the WORK.Title columns, populate this column. (*Hint:* In the WORK table, the WorkTitle may appear more than once. For these cases, use the lowest numbered WorkID. This will be the WorkID of the first occurrence of the WorkTitle, and can be found using the TOP keyword.)

F. Create a new table named POSTCARD_SIZE_PRICE, as shown in Figure 10A-92. Use the column characteristics shown in Figure 10A-92, where PostcardSizePriceID is a surrogate key starting at 1 and incrementing by 1.

G. Populate the POSTCARD_SIZE_PRICE table using the data stored in the POSTCARDS$ table. *Hint:* You should insert distinct data into the table, and your final table will only have three records.

H. Alter the POSTCARDS$ table to include a PostcardSizePriceID column (Integer data, allow NULL values). By using and comparing the data in the POSTCARDS$.PostCardSize and the POSTCARD_SIZE_PRICE.PostCardSize columns, populate this column.

I. Create a new table named POSTCARD_INVENTORY, as shown in Figure 10A-92. Use the column characteristics shown in Figure 10A-92, where PostcardID is a surrogate key starting at 1 and incrementing by 1.

J. Populate the POSTCARD_INVENTORY table using the data stored in the POSTCARDS$ table. *Hint:* You will have one record in this table for every record in the POSTCARDS$ table, and your final table will only have 26 records.

K. We have completed our modifications of the *VRG* database, and we are done with the temporary POSTCARDS$ table. We *could* delete it if we wanted to, but we will *keep the POSTCARDS$ table* in the database.

Case Questions

Marcia's Dry Cleaning Case Questions

Marcia Wilson owns and operates Marcia's Dry Cleaning, which is an upscale dry cleaner in a well-to-do suburban neighborhood. Marcia makes her business stand out from the competition by providing superior customer service. She wants to keep track of each of her customers and their orders. Ultimately, she wants to notify them that their clothes are ready via email. Suppose that you have designed a database for Marcia's Dry Cleaning that has the following tables:

CUSTOMER (<u>CustomerID</u>, FirstName, LastName, Phone, EmailAddress)

INVOICE (<u>InvoiceNumber</u>, *CustomerID*, DateIn, DateOut, Subtotal, Tax, TotalAmount)

INVOICE_ITEM (<u>*InvoiceNumber*</u>, <u>ItemNumber</u>, *ServiceID*, Quantity, UnitPrice, ExtendedPrice)

SERVICE (<u>ServiceID</u>, ServiceDescription, UnitPrice)

The referential integrity constraints are:

CustomerID in INVOICE must exist in CustomerID in CUSTOMER

InvoiceNumber in INVOICE_ITEM must exist in InvoiceNumber in INVOICE

ServiceID in INVOICE_ITEM must exist in ServiceID in SERVICE

Assume that CustomerID of CUSTOMER and InvoiceNumber of INVOICE are surrogate keys with values as follows:

CustomerID	**Start at 100**	**Increment by 1**
InvoiceNumber	**Start at 2018001**	**Increment by 1**

Further, assume that ServiceID is a surrogate key, but not one that automatically increments—the values of ServiceID are assigned by Marcia's Dry Cleaning management when new services are added at Marcia's Dry Cleaning.

A. Specify NULL/NOT NULL constraints for each table column.

B. Specify alternate keys, if any.

C. State relationships as implied by foreign keys, and specify the maximum and minimum cardinalities of each relationship. Justify your choices.

D. Explain how you will enforce the minimum cardinalities in your answer to part C. Use referential integrity actions for required parents, if any. Use Figure 6-29(b) as a boilerplate for required children, if any.

E. Using SQL Server 2017 and the Microsoft SQL Server Management Studio, create a database named *MDC*.

F. In the SQL Server Management Studio folder structure in your *My Documents* folder, create a folder named *DBP-e15-MDC-Database* in the *Projects* folder. Use this folder to save and store *.sql scripts containing the SQL statements that you are asked to create in the remaining questions in this section.

Using the MDC database, create an SQL script named *MDC-Create-Tables.sql* to answer parts G and H.

G. Write CREATE TABLE statements for each of the tables using your answers to parts A–D, as necessary. Set the first value of CustomerID to 100 and increment it by 1. Set the first value of InvoiceNumber to 2018001 and increment it by 1. Use FOREIGN KEY constraints to create appropriate referential integrity constraints. Set UPDATE and DELETE behavior in accordance with your referential integrity action design. Set the default value of Quantity to 1. Write a constraint that SERVICE.UnitPrice be between 1.50 and 10.00.

H. Explain how you would enforce the data constraint that INVOICE_ITEM.UnitPrice be equal to SERVICE.UnitPrice, where INVOICE_ITEM.ServiceID = SERVICE.ServiceID.

Using the MDC database, create an SQL script named *MDC-Insert-Data.sql* to answer part I.

I. Write INSERT statements to insert the data shown in Figures 10A-93, 10A-94, 10A-95, and 10A-96.

Using the MDC database, create an SQL script named *MDC-DML-CH10A.sql* to answer parts J and K.

J. Write an UPDATE statement to change values of ServiceDescription from Mens Shirt to Mens' Shirt.

K. Write a DELETE statement(s) to delete an INVOICE and all of the items on that INVOICE.

FIGURE 10A-93

Sample Data for the MDC Database CUSTOMER Table

CustomerID	FirstName	LastName	Phone	EmailAddress
100	Nikki	Kaccaton	723-543-1233	Nikki.Kaccaton@somewhere.com
101	Brenda	Catnazaro	723-543-2344	Brenda.Catnazaro@somewhere.com
102	Bruce	LeCat	723-543-3455	Bruce.LeCat@somewhere.com
103	Betsy	Miller	723-654-3211	Betsy.Miller@somewhere.com
104	George	Miller	723-654-4322	George.Miller@somewhere.com
105	Kathy	Miller	723-514-9877	Kathy.Miller@somewhere.com
106	Betsy	Miller	723-514-8766	Betsy.Miller@elsewhere.com

FIGURE 10A-94

Sample Data for the MDC Database SERVICE Table

ServiceID	ServiceDescription	UnitPrice
10	Mens Shirt	$1.50
11	Dress Shirt	$2.50
15	Women's Shirt	$1.50
16	Blouse	$3.50
20	Slacks-Men's	$5.00
25	Slacks-Women's	$6.00
30	Skirt	$5.00
31	Dress Skirt	$6.00
40	Suit-Men's	$9.00
45	Suit-Women's	$8.50
50	Tuxedo	$10.00
60	Formal Gown	$10.00

Using the MDC database, create an SQL script named MDC-Create-Views-and-Functions.sql to answer parts L through T. Your answer to Part P should be in the form of a comment in the SQL script.

L. Create a view called OrderSummaryView that contains INVOICE.InvoiceNumber, INVOICE.DateIn, INVOICE.DateOut, INVOICE_ITEM.ItemNumber, INVOICE_ITEM.ServiceID, and INVOICE_ITEM.ExtendedPrice.

M. Create a view called CustomerOrderSummaryView that contains INVOICE.InvoiceNumber, CUSTOMER.FirstName, CUSTOMER.LastName, CUSTOMER.Phone, INVOICE.DateIn, INVOICE.DateOut, INVOICE.SubTotal, INVOICE_ITEM.ItemNumber, INVOICE_ITEM.ServiceID, and INVOICE_ITEM.ExtendedPrice.

N. Create a view called CustomerOrderHistoryView that (1) includes all columns of CustomerOrderSummaryView except INVOICE_ITEM.ItemNumber, INVOICE_ITEM.ExtendedPrice, and INVOICE_ITEM.ServiceID; (2) groups orders by CUSTOMER.LastName, CUSTOMER.FirstName, and INVOICE.InvoiceNumber, in that order; and (3) sums and averages INVOICE_ITEM.ExtendedPrice for each order for each

FIGURE 10A-95

Sample Data for the MDC Database INVOICE Table

InvoiceNumber	CustomerID	DateIn	DateOut	SubTotal	Tax	TotalAmount
2018001	100	04-Oct-15	06-Oct-18	$158.50	$12.52	$171.02
2018002	101	04-Oct-15	06-Oct-18	$25.00	$1.98	$26.98
2018003	100	06-Oct-15	08-Oct-18	$49.00	$3.87	$52.87
2018004	103	06-Oct-15	08-Oct-18	$17.50	$1.38	$18.88
2018005	105	07-Oct-15	11-Oct-18	$12.00	$0.95	$12.95
2018006	102	11-Oct-15	13-Oct-18	$152.50	$12.05	$164.55
2018007	102	11-Oct-15	13-Oct-18	$7.00	$0.55	$7.55
2018008	106	12-Oct-15	14-Oct-18	$140.50	$11.10	$151.60
2018009	104	12-Oct-15	14-Oct-18	$27.00	$2.13	$29.13

FIGURE 10A-96
Sample Data for the MDC Database INVOICE_ITEM Table

InvoiceNumber	ItemNumber	ServiceID	Quantity	UnitPrice	ExtendedPrice
2018001	1	16	2	$3.50	$7.00
2018001	2	11	5	$2.50	$12.50
2018001	3	50	2	$10.00	$20.00
2018001	4	20	10	$5.00	$50.00
2018001	5	25	10	$6.00	$60.00
2018001	6	40	1	$9.00	$9.00
2018002	1	11	10	$2.50	$25.00
2018003	1	20	5	$5.00	$25.00
2018003	2	25	4	$6.00	$24.00
2018004	1	11	7	$2.50	$17.50
2018005	1	16	2	$3.50	$7.00
2018005	2	11	2	$2.50	$5.00
2018006	1	16	5	$3.50	$17.50
2018006	2	11	10	$2.50	$25.00
2018006	3	20	10	$5.00	$50.00
2018006	4	25	10	$6.00	$60.00
2018007	1	16	2	$3.50	$7.00
2018008	1	16	3	$3.50	$10.50
2018008	2	11	12	$2.50	$30.00
2018008	3	20	8	$5.00	$40.00
2018008	4	25	10	$6.00	$60.00
2018009	1	40	3	$9.00	$27.00

customer. *HINT:* In (2), note that the GROUP BY clause will also have to include Phone, DateIn, etc., because the selected columns must be a subset of the grouping columns.

O. Create a view called CustomerOrderCheckView that uses CustomerOrderHistoryView and that shows any customers for whom the sum of INVOICE_ITEM.ExtendedPrice is not equal to INVOICE.SubTotal. Note that this will display customers who had more than one item in their order.

P. Explain in general terms how you will use triggers to enforce minimum cardinality actions as required by your design. You need not write the triggers, just specify which triggers you need and describe their logic in general terms.

Q. Create and test a user-defined function named *LastNameFirst* that combines two parameters named *FirstName* and *LastName* into a concatenated name field formatted *LastName, FirstName* (including the comma and space).

R. Create and test a view called CustomerInvoiceSummaryView that contains the customer name concatenated and formatted as *LastName, FirstName* in a field named CustomerName, INVOICE.InvoiceNumber, INVOICE.DateIn, INVOICE.DateOut, and INVOICE.TotalAmount.

S. Create and test a user-defined function named *FirstNameFirst* that combines two parameters named *FirstName* and *LastName* into a concatenated name field formatted *FirstName LastName* (including the space).

T. Create and test a view called CustomerDataView that contains the customer name concatenated and formatted as *FirstName LastName* in a field named CustomerName, Phone, and EmailAddress.

Using the MDC database, create an SQL script named *MDC-Create-Triggers.sql* to answer parts U and V.

U. Assume that the relationship between INVOICE and INVOICE_ITEM is M-M. Design triggers to enforce this relationship. Use Figure 10A-72 and the discussion of that figure as an example, but assume that Marcia does allow INVOICEs and their related INVOICE_ITEM rows to be deleted. Use the deletion strategy shown in Figures 7-28 and 7-29 for this case.

V. Write and test the triggers you designed in part U.

W. Create an SQL Server login named MDC-User, with a password of MDC-User+password. Create an MDC database user named MDC-Database-User, which is linked to the MDC-User login. Assign MDC-Database-User *db_owner* permissions to the MDC database.

Marcia's Dry Cleaning tracks which employees have worked on specific dry cleaning jobs. This is somewhat complicated by the fact that more than one employee may have helped a customer with a particular order. So far, the company has kept their records in a Microsoft Excel 2016 worksheet, as shown in Figure 10A-97. They have decided to integrate this data into the MDC database. The modifications to the MDC database needed to accomplish this are shown in Figure 10A-98 (as a MySQL Workbench EER diagram). Using the MDC database, create an SQL script named *MDC-Import-Excel-Data.sql* to answer parts Z through AJ.

X. Duplicate the EMPLOYEE worksheet in Figure 10A-97 in a worksheet (or spreadsheet) in Microsoft Excel 2016 (or another tool such as Apache OpenOffice Calc).

Y. Import the data in the EMPLOYEE worksheet into a new table in the MDC database named EMPLOYEE$.

Z. Create the *GetLastNameCommaSeparated* user-defined function shown in Figure 10A-58.

AA. Create a user-defined function named *GetFirstNameCommaSeparated* that will return the first name from a combined name in last-name-first order, with the names separated by a comma and one space.

AB. Alter the EMPLOYEE$ table to include EmployeeLastName and EmployeeFirstName columns (Char (25), allow NULL values).

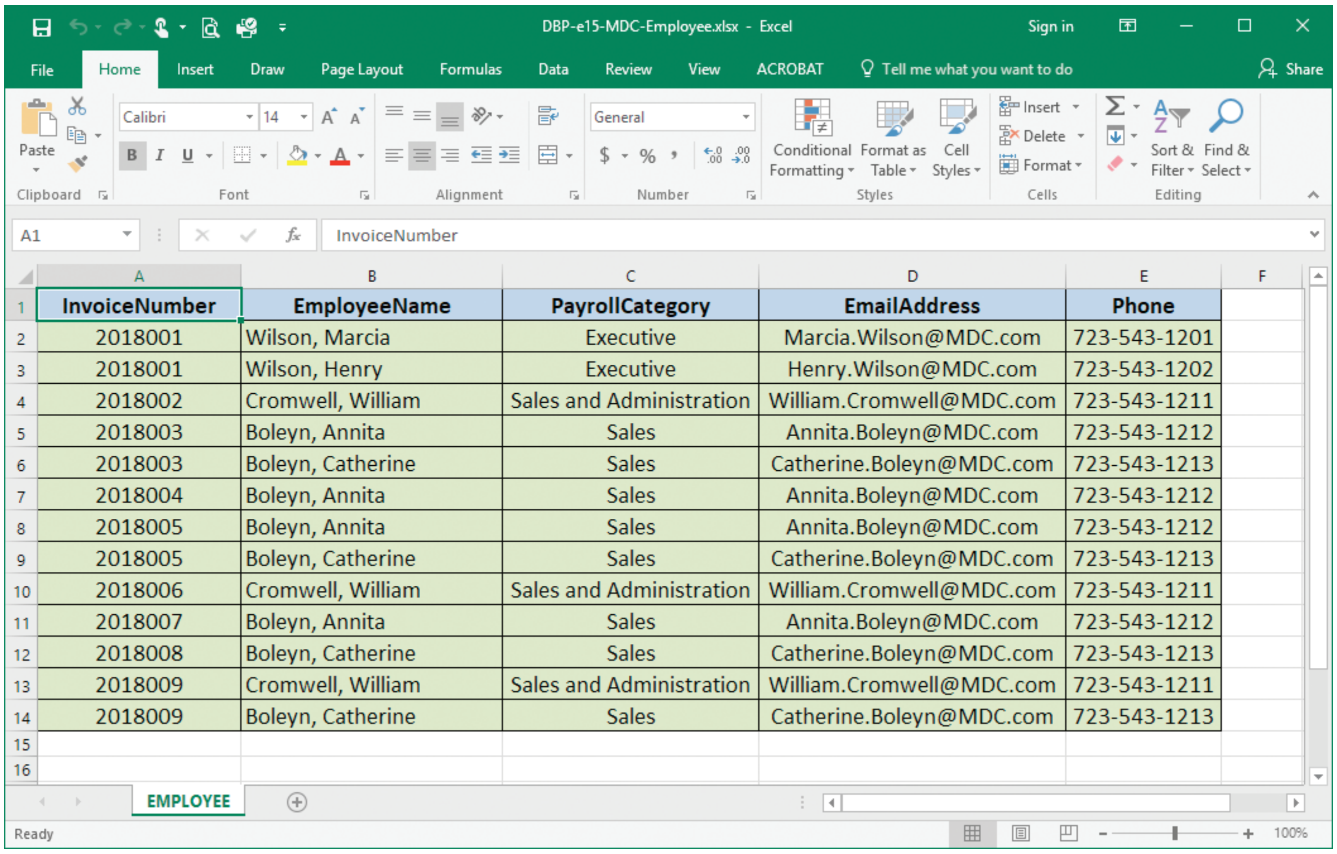
AC. Use the *GetLastNameCommaSeparated* user-defined function you created in step Z to populate the EmployeeLastName column.

AD. Use the *GetFirstNameCommaSeparated* user-defined function you created in step AA to populate the EmployeeFirstName column.

AE. Create a new table named EMPLOYEE, as shown in Figure 10A-98. Use the column characteristics shown in Figure 10A-98, where EmployeeID is a surrogate key starting at 1 and incrementing by 1.

AF. Populate the EMPLOYEE table using the data stored in the EMPLOYEE$ table. *Hint:* You should insert distinct data into the table, and your final table will have only five records.

AG. Alter the EMPLOYEE$ table to include an EmployeeID column (Integer data, allow nulls). By using and comparing the EMPLOYEE$.EmployeeLastName and EMPLOYEE$.EmployeeFirstName columns with the EMPLOYEE.EmployeeLastName



	A	B	C	D	E
1	InvoiceNumber	EmployeeName	PayrollCategory	EmailAddress	Phone
2	2018001	Wilson, Marcia	Executive	Marcia.Wilson@MDC.com	723-543-1201
3	2018001	Wilson, Henry	Executive	Henry.Wilson@MDC.com	723-543-1202
4	2018002	Cromwell, William	Sales and Administration	William.Cromwell@MDC.com	723-543-1211
5	2018003	Boleyn, Annita	Sales	Annita.Boleyn@MDC.com	723-543-1212
6	2018003	Boleyn, Catherine	Sales	Catherine.Boleyn@MDC.com	723-543-1213
7	2018004	Boleyn, Annita	Sales	Annita.Boleyn@MDC.com	723-543-1212
8	2018005	Boleyn, Annita	Sales	Annita.Boleyn@MDC.com	723-543-1212
9	2018005	Boleyn, Catherine	Sales	Catherine.Boleyn@MDC.com	723-543-1213
10	2018006	Cromwell, William	Sales and Administration	William.Cromwell@MDC.com	723-543-1211
11	2018007	Boleyn, Annita	Sales	Annita.Boleyn@MDC.com	723-543-1212
12	2018008	Boleyn, Catherine	Sales	Catherine.Boleyn@MDC.com	723-543-1213
13	2018009	Cromwell, William	Sales and Administration	William.Cromwell@MDC.com	723-543-1211
14	2018009	Boleyn, Catherine	Sales	Catherine.Boleyn@MDC.com	723-543-1213

FIGURE 10A-97

The Marcia's Dry Cleaning Employee Worksheet

and EMPLOYEE.EmployeeFirstName columns, populate this column. *Hint:* Assume for this question that no two employees have the same first and last names.

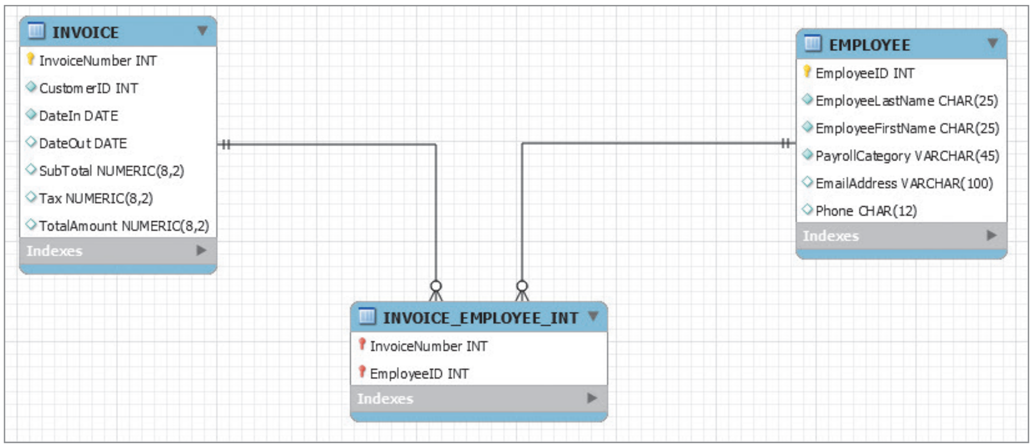
AH. Create a new table named INVOICE_EMPLOYEE_INT, as shown in Figure 10A-98. Use the column characteristics shown in Figure 10A-98.

AI. Populate the INVOICE_EMPLOYEE_INT table using the data stored in the EMPLOYEE$ table. *Hint:* You will have one record in the INVOICE_EMPLOYEE_INT table for every record in the EMPLOYEE$ table, and your final table will have 13 records.

AJ. We have completed the modifications of the MDC database and are done with the temporary EMPLOYEE$ table. We *could* delete it if we wanted to, but we will *keep the EMPLOYEE$ table* in the database.

FIGURE 10A-98

Partial Database Design for the Modified MDC Database



The Queen Anne Curiosity Shop Project Questions

If you have not completed the discussion of the Queen Anne Curiosity Shop database at the end of Chapter 7 on page 409, work through the Chapter 7 QACS Project Questions now. Use the QACS database that you created in the Chapter 7 QACS Project Questions as the basis for your answers to the following questions.

Using the QACS database, create an SQL script named *QACS-Create-Views-and-Functions.sql* to answer parts A through E.

A. Create and test a user-defined function named *LastNameFirst* that combines two parameters named *FirstName* and *LastName* into a concatenated name field formatted *LastName, FirstName* (including the comma and space).

B. Create and test a view called CustomerSaleReviewView that contains the customer name concatenated and formatted as *LastName, FirstName* in a field named CustomerName, SALE.SaleID, SALE.SaleDate, and SALE.Total.

C. Create and test a user-defined function named *FirstNameFirst* that combines two parameters named *FirstName* and *LastName* into a concatenated name field formatted *FirstName LastName* (including the space).

D. Create and test a user-defined function named *CityStateZIP* that combines three parameters named *City, State*, and *ZIP* into a concatenated name field formatted *City, State ZIP* (including the comma and the spaces).

E. Create and test a view called CustomerMailingAddressView that contains the customer name concatenated and formatted as *FirstName LastName* in a field named CustomerName; the customer's street address in a field named CustomerStreetAddress; and the customer's City, State, ZIP concatentated and formatted as *City, State ZIP* in a field named CustomerCityStateZIP.

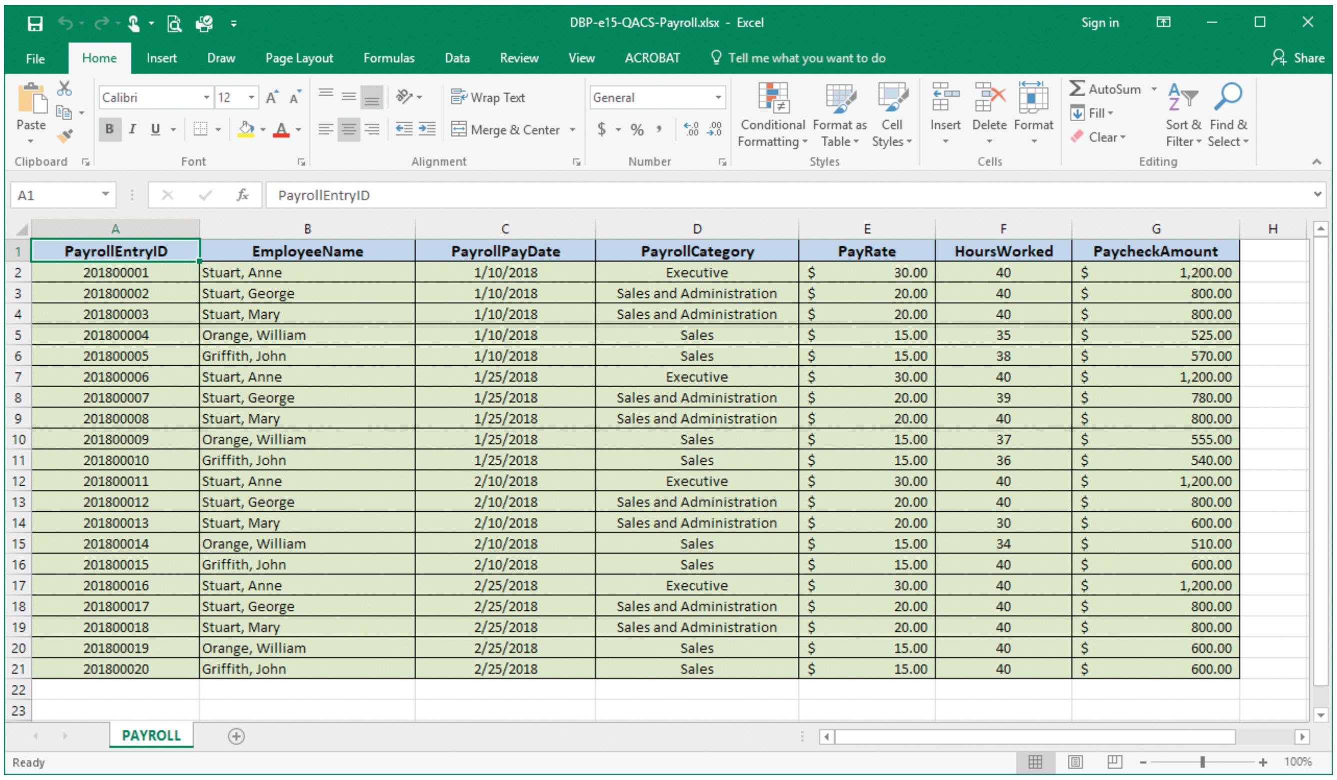
Using the QACS database, create an SQL script named *QACS-Create-Triggers.sql* to answer parts F and G.

F. Assume that the relationship between SALE and SALE_ITEM is M-M. Design triggers to enforce this relationship. Use Figure 10A-72 and the discussion of that figure as an example, but assume that The Queen Anne Curiosity Shop does allow SALESs and their related SALE_ITEM rows to be deleted. Use the deletion strategy shown in Figures 7-28 and 7-29 for this case.

G. Write and test the triggers you designed in part F.

H. Create an SQL Server 2017 login named QACS-User, with a password of QACS-User+password. Create a QACS database user named QACS-Database-User, which is linked to the QACS-User login. Assign QACS-Database-User *db_owner* permissions to the QACS database.

The Queen Anne Curiosity Shop payroll is paid twice monthly, once on the 10th of the month and once on the 25th of the month. Pay is determined by the type of job (the payroll category) and the number of hours worked (rounded to a whole number). Of course, the QACS owners keep detailed payroll records. So far, they have kept their records for these items in a Microsoft Excel 2016 worksheet, as shown in Figure 10A-99. They have decided to integrate this data into the QACS database. The modifications to the QACS database needed to accomplish this are shown in Figure 10A-100 (as a MySQL Workbench EER diagram). Using the QACS database, create an SQL script named *QACS-Import-Excel-Data.sql* to answer parts I through V.



PayrollEntryID	EmployeeName	PayrollPayDate	PayrollCategory	PayRate	HoursWorked	PaycheckAmount
201800001	Stuart, Anne	1/10/2018	Executive	$ 30.00	40	$ 1,200.00
201800002	Stuart, George	1/10/2018	Sales and Administration	$ 20.00	40	$ 800.00
201800003	Stuart, Mary	1/10/2018	Sales and Administration	$ 20.00	40	$ 800.00
201800004	Orange, William	1/10/2018	Sales	$ 15.00	35	$ 525.00
201800005	Griffith, John	1/10/2018	Sales	$ 15.00	38	$ 570.00
201800006	Stuart, Anne	1/25/2018	Executive	$ 30.00	40	$ 1,200.00
201800007	Stuart, George	1/25/2018	Sales and Administration	$ 20.00	39	$ 780.00
201800008	Stuart, Mary	1/25/2018	Sales and Administration	$ 20.00	40	$ 800.00
201800009	Orange, William	1/25/2018	Sales	$ 15.00	37	$ 555.00
201800010	Griffith, John	1/25/2018	Sales	$ 15.00	36	$ 540.00
201800011	Stuart, Anne	2/10/2018	Executive	$ 30.00	40	$ 1,200.00
201800012	Stuart, George	2/10/2018	Sales and Administration	$ 20.00	40	$ 800.00
201800013	Stuart, Mary	2/10/2018	Sales and Administration	$ 20.00	30	$ 600.00
201800014	Orange, William	2/10/2018	Sales	$ 15.00	34	$ 510.00
201800015	Griffith, John	2/10/2018	Sales	$ 15.00	40	$ 600.00
201800016	Stuart, Anne	2/25/2018	Executive	$ 30.00	40	$ 1,200.00
201800017	Stuart, George	2/25/2018	Sales and Administration	$ 20.00	40	$ 800.00
201800018	Stuart, Mary	2/25/2018	Sales and Administration	$ 20.00	40	$ 800.00
201800019	Orange, William	2/25/2018	Sales	$ 15.00	40	$ 600.00
201800020	Griffith, John	2/25/2018	Sales	$ 15.00	40	$ 600.00

FIGURE 10A-99

The Queen Anne Curiosity Shop Payroll Worksheet

I. Duplicate the PAYROLL worksheet in Figure 10A-99 in a worksheet (or spreadsheet) in an appropriate tool (such as Microsoft Excel or Apache OpenOffice Calc).

J. Import the data in the PAYROLL worksheet into a table in the QACS database named PAYROLL$.

K. Create the *GetLastNameCommaSeparated* user-defined function shown in Figure 10A-58.

L. Create a user-defined function named *GetFirstNameCommaSeparated* that will return the first name from a combined name in last-name-first order, with the names separated by a comma and one space.

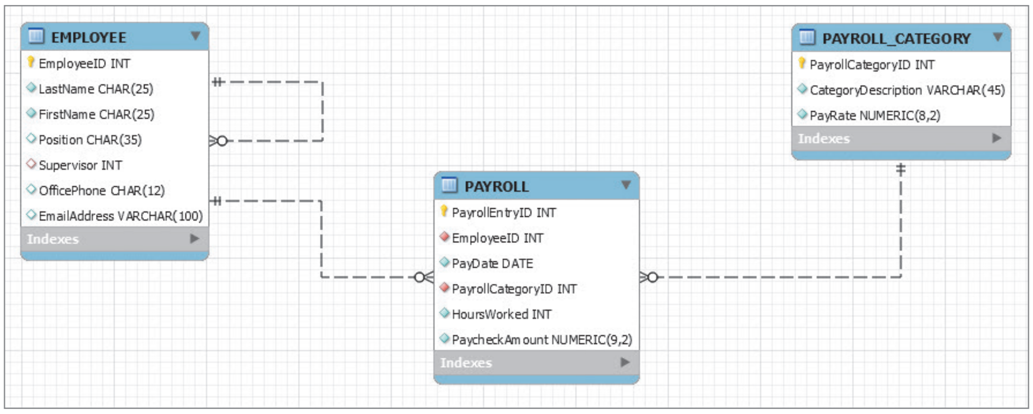
M. Alter the PAYROLL$ table to include EmployeeLastName and EmployeeFirstName columns (Char (25), allow NULL values).

N. Use the *GetLastNameCommaSeparated* user-defined function you created in step K to populate the EmployeeLastName column.

O. Use the *GetFirstNameCommaSeparated* user-defined function you created in step L to populate the EmployeeFirstName column.

FIGURE 10A-100

Partial Database Design for the Modified QACS Database



P. Create a new table named PAYROLL_CATEGORY, as shown in Figure 10A-100. Use the column characteristics shown in Figure 10A-100, where PayrollCategoryID is a surrogate key starting at 1 and incrementing by 1.

Q. Populate the PAYROLL_CATEGORY table using the data stored in the PAYROLL$ table. *Hint:* You should insert distinct data into the table, and your final table will have only three records.

R. Alter the PAYROLL$ table to include an EmployeeID column (Integer data, allow nulls). By using and comparing the PAYROLL$.EmployeeLastName and PAYROLL$ .EmployeeFirstName columns with the EMPLOYEE.LastName and EMPLOYEE .FirstName columns, populate this column. *Hint:* Assume for this question that no two employees have the same first and last names.

S. Alter the PAYROLL$ table to include a PayrollCategoryID column (Integer data, allow nulls). By using and comparing the PAYROLL$.PayrollCategory column with the PAYROLL_CATEGORY.CategoryDescription column, populate this column.

T. Create a new table named PAYROLL, as shown in Figure 10A-100. Use the column characteristics shown in Figure 10A-100. Note that PayrollEntryID is a surrogate key, with initial value 20180001 and incrementing by 1.

U. Populate the PAYROLL table using the data stored in the PAYROLL$ table. *Hint:* You will have one record in the PAYROLL table for every record in the PAYROLL$ table, and your final table will have 20 records.

V. We have completed the modifications of the QACS database and are done with the temporary PAYROLL$ table. We *could* delete it if we wanted to, but we will *keep the PAYROLL$ table* in the database.

Morgan Importing Project Questions

If you have not completed the discussion of Morgan Importing database at the end of Chapter 7 on pages 416–423, work through that chapter's Morgan Importing Project Questions now. Use the MI database that you created in the Chapter 7 MI Project Questions as the basis for your answers to the following questions:

Using the MI database, create an SQL script named MI-Create-Views-and-Functions. sql to answer parts A through D.

A. Create and test a user-defined function named *LastNameFirst* that combines two parameters named *FirstName* and *LastName* into a concatenated name field formatted *LastName, FirstName* (including the comma and space).

B. Create and test a view called *PurchasingAgentSummaryView* that contains the employee name of any MI employees who purchase items for the company, concatenated and formatted as *LastName, FirstName* in a field named PurchasingAgentName, ITEM.ItemDescription, ITEM.PurchaseDate, STORE.StoreName, STORE.City, and STORE.Country.

C. Create and test a user-defined function named *FirstNameFirst* that combines two parameters named *FirstName* and *LastName* into a concatenated name field formatted *FirstName LastName* (including the space).

D. Create and test a view called ReceivingAgentSummaryView that contains the employee name of any MI employees who received items for the company, concatenated and formatted as *FirstName LastName* in a field named ReceivingAgentName, SHIPMENT_RECEIPT.ReceiptNumber, SHIPMENT.ShipmentID, SHIPPER

.ShipperName, SHIPMENT.EstimatedArrivalDate, SHIPMENT_RECEIPT.ReceiptDate, and SHIPMENT_RECEIPT.ReceiptTime.

Using the MI database, create an SQL script named *MI-Create-Triggers.sql* to answer parts E and F.

E. Assume that the relationship between SHIPMENT and SHIPMENT_ITEM is M-M. Design triggers to enforce this relationship. Use Figure 10A-72 and the discussion of that figure as an example, but assume that Morgan does allow SHIPMENTs and their related SHIPMENT_ITEM rows to be deleted. Use the deletion strategy shown in Figures 7-28 and 7-29 for this case.

F. Write and test the triggers you designed in part E.

G. Create an SQL Server 2017 login named MI-User, with a password of MI-User+password. Create an MI-database user named MI-Database-User, which is linked to the MI-User login. Assign MI-Database-User *db_owner* permissions to the MI database.

Morgan Importing purchases marine insurance to protect the company from monetary loss during shipping. So far, the company has kept their insurance records in a Microsoft Excel 2016 worksheet, as shown in Figure 10A-101. They have decided to integrate this data into the *MI* database. The modifications to the MI database needed to accomplish this are shown in Figure 10A-102 (as a MySQL Workbench EER diagram). Using the *MI* database, create an SQL script named *MI-Import-Excel-Data.sql* to answer parts H through T.

H. Duplicate the INSURANCE worksheet Figure 10A-101 in a worksheet (or spreadsheet) in Microsoft Excel 2016 (or another tool such as Apache OpenOffice Calc).

I. Import the data in the INSURANCE worksheet into a table in the MI database named INSURANCE$.

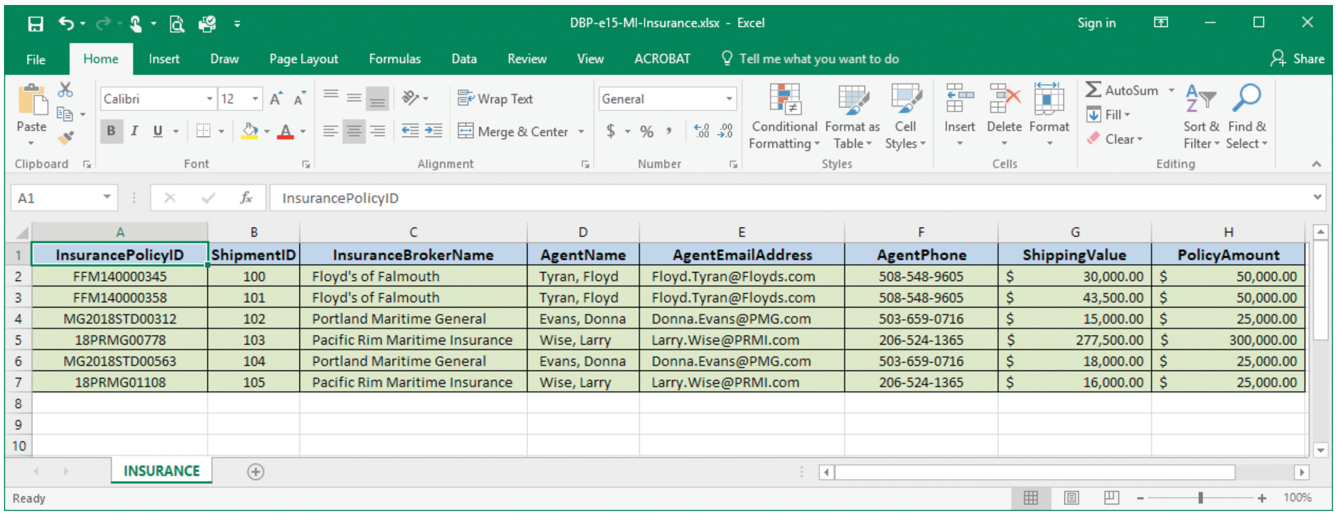
J. Create the *GetLastNameCommaSeparated* user-defined function shown in Figure 10A-58.

K. Create a user-defined function named *GetFirstNameCommaSeparated* that will return the first name from a combined name in last-name-first order, with the names separated by a comma and one space.

L. Alter the INSURANCE$ table to include AgentLastName and AgentFirstName columns (Varchar (45), allow NULL values).

FIGURE 10A-101

The Morgan Importing Maritime Insurance Worksheet



	A	B	C	D	E	F	G	H
1	InsurancePolicyID	ShipmentID	InsuranceBrokerName	AgentName	AgentEmailAddress	AgentPhone	ShippingValue	PolicyAmount
2	FFM140000345	100	Floyd's of Falmouth	Tyran, Floyd	Floyd.Tyran@Floyds.com	508-548-9605	$ 30,000.00	$ 50,000.00
3	FFM140000358	101	Floyd's of Falmouth	Tyran, Floyd	Floyd.Tyran@Floyds.com	508-548-9605	$ 43,500.00	$ 50,000.00
4	MG2018STD00312	102	Portland Maritime General	Evans, Donna	Donna.Evans@PMG.com	503-659-0716	$ 15,000.00	$ 25,000.00
5	18PRMG00778	103	Pacific Rim Maritime Insurance	Wise, Larry	Larry.Wise@PRMI.com	206-524-1365	$ 277,500.00	$ 300,000.00
6	MG2018STD00563	104	Portland Maritime General	Evans, Donna	Donna.Evans@PMG.com	503-659-0716	$ 18,000.00	$ 25,000.00
7	18PRMG01108	105	Pacific Rim Maritime Insurance	Wise, Larry	Larry.Wise@PRMI.com	206-524-1365	$ 16,000.00	$ 25,000.00
8								
9								
10								

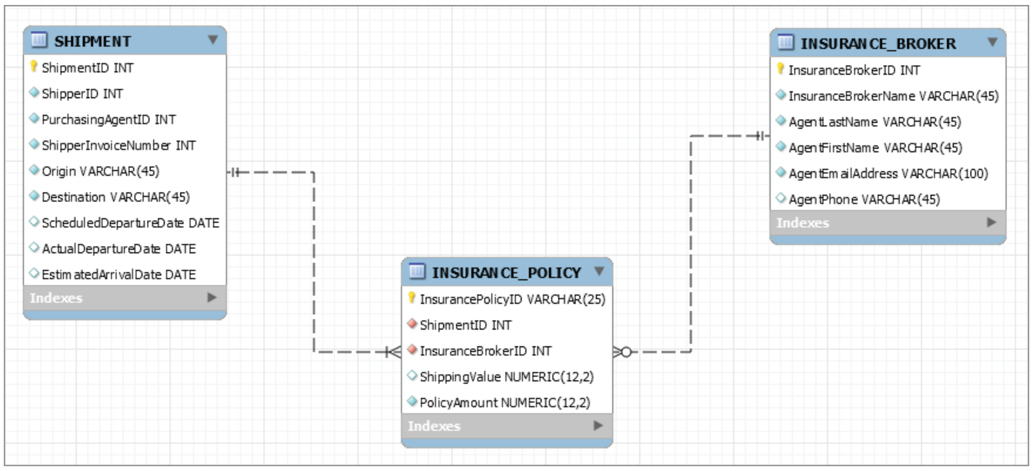


FIGURE 10A-102

Partial Database Design for the Modified MI Database

M. Use the *GetLastNameCommaSeparated* user-defined function you created in step J to populate the AgentLastName column.

N. Use the *GetFirstNameCommaSeparated* user-defined function you created in step K to populate the AgentFirstName column.

O. Create a new table named INSURANCE_BROKER. Use the column characteristics shown in Figure 10A-102, where InsuranceBrokerID is a surrogate key starting at 1 and incrementing by 1.

P. Populate the INSURANCE_BROKER table using the data stored in the INSURANCE$ table. *Hint:* You should insert distinct data into the table, and your final table will have only three records.

Q. Create a new table named INSURANCE_POLICY. Use the column characteristics shown in Figure 10A-102. Note that InsurancePolicyID is not a surrogate key, but rather uses a Varchar (25) character string.

R. Alter the INSURANCE$ table to include an InsuranceBrokerID column (Integer data, allow nulls). By using and comparing the INSURANCE$.InsuranceBrokerName and INSURANCE_BROKER.InsuranceBrokerName columns, populate this column. *Hint:* Assume for this question that no two insurance broker names are the same.

S. Populate the INSURANCE_POLICY table using the data stored in the INSURANCE$ table. *Hint:* You will have one record in the INSURANCE_POLICY table for every record in the INSURANCE$ table, and your final table will have six records.

T. We have completed the modifications of the MI database and are done with the temporary INSURANCE$ table. We *could* delete it if we wanted to, but we will *keep the INSURANCE$ table* in the database.