Microsoft

Start


# Microsoft® SQL Server™ 2005 Express Edition

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

Copyright © 2007 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Control Number: 2007922584

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Microsoft, Microsoft Press, Excel, Internet Explorer, MSDN, SQL Server, Visio, Visual Basic, Visual C#, Visual Studio, Visual Web Developer, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

**Acquisitions Editor:** Ben Ryan
**Developmental Editor:** Devon Musgrave
**Project Editor:** Victoria Thulman
**Technical Reviewer:** Michael Wachal
**Editorial and Production Services:** Custom Editorial Productions, Inc.

# Contents


Chapter 1
**Introducing Microsoft SQL Server 2005 Express Edition** 1

**What Is a Data Service?** 2
**What Should You Know About Databases?** 4
What's Inside a Database? 4
What Is Normalization? 5
How Do You Maintain Data Integrity? 10
**What Is Microsoft SQL Server 2005 Express Edition?** 14
What Kinds of Databases Can You Build with SQL Server Express? 14
What Are the Key Features You Need to Know About? 15
Where Can You Get Help? 18

Chapter 2
**Installing SQL Server Express** 19

**Preparing to Install SQL Server Express** 20
**Installing SQL Server Express** 20
**Installing SQL Server Management Studio Express** 28
**Going a Few Steps Farther** 31
Downloading the AdventureWorks Sample Database 31
Installing a Local Copy of SQL Server Books Online 34
Configuring SQL Server Express for Remote Access 40

Chapter 3
**Creating Your First Database** 43

**Getting Started with SQL Server Express** 44
Exploring SQL Server Management Studio Express 44
Attaching the AdventureWorks Database 52
**Creating the Home Inventory Database** 57
Creating Your Database 58
Adding Tables to Your Database 62
Adding Relationships 70
Adding Constraints 73
Indexing Your Tables 78
Entering Data into Your Tables 84

Chapter 4
**Working with Transact-SQL** 89

**Working with Data** 90
Adding New Rows to a Table 90
Viewing the Contents of a Table 98
Combining the Contents of Multiple Tables 99
Sorting a Query Result Set 105
Using Aggregations 108
Limiting Rows Retrieved by a Query 114
Changing Data in Tables 119
Deleting Rows from a Table 121

**Working with Database Objects** **124**
Creating Objects 125
Altering Objects 127
Working with Views 129
Using Stored Procedures 131
Working with Scripts 136

*Chapter 5*
*Maintaining Your Database* *145*

**Applying Security** **146**
Creating Logins 148
Assigning Roles 155
Granting Permissions 158
Testing Permissions 164
**Performing Administrative Tasks** **168**
Selecting a Recovery Mode 168
Backing Up Your Database 170
Restoring Your Database 173
Attaching a Database 175

*Chapter 6*
*Managing Your Server* *181*

**Working with Server Instances** **182**
Understanding the SQL Server Express Service 182
**Using Named Instances** **185**
**Working with User Instances** **186**
Enabling User Instances 187
Creating a User Instance 188
Starting User Instances with SQL Server Express Utility 193
Finding Information About User Instances 194
Connecting to a User Instance in Management Studio 196

*Glossary* *199*

*Index* *203*

# Introduction

Microsoft SQL Server 2005 Express Edition is a great product that meets the needs of a wide variety of people, from students just beginning to learn about database technologies to professional developers wishing to use database storage for application data. This product is fully functional and contains the same database, security, and programmability features you can find throughout the SQL Server 2005 family. The Express Edition simply places limits on how large your databases can grow, how much memory can be used by the Express Edition, and the number of CPUs that can be used. You won't be able scale SQL Server 2005 Express Edition across multiple servers, but that's not likely to be your first objective if you're reading this book! This product gives you exactly the functionality you need to build and maintain useful databases with no time limit placed on your usage of the product.

Though this book is focused exclusively on SQL Server 2005 Express Edition, you'll soon discover Microsoft also offers SQL Server 2005 Express Edition with Advanced Services as well as SQL Server 2005 Express Toolkit. The Advanced Services version includes Reporting Services and Full-Text Search. Reporting Services enables you to develop and manage professional reports to present data from your databases, whereas Full-Text Search is a search engine for your databases. It's perfectly fine to use SQL Server 2005 Express Edition to start building databases now, and then switch to Advanced Services later. The SQL Server 2005 Express Toolkit is a collection of resources you can use with SQL Server 2005 Express Edition, such as Business Intelligence Development Studio, which you'll need when building Reporting Services reports. The toolkit also includes Connectivity Components, Management Studio Express, and a Software Development Kit (SDK). These are great tools to obtain and use once you're more comfortable working with SQL Server 2005 Express Edition, which of course you will be once you finish reading this book!

## Who Is This Book For?

This book is for those people who want to get started building databases for personal use or as part of an application, but don't know how to start. If you've tried using spreadsheets to organize information but need a better way to search for information, then a database is a good solution. If you don't have any experience with databases, then this book is for you. In fact, I rather expect you've never worked with a database and are unfamiliar with the terminology and concepts that professional database developers take for granted. But remember, they had to start somewhere once upon a time, too!

My daughter, who's currently attending college far away from home, has gone out of her way to avoid learning anything more technical than checking e-mail. I asked her to read through this book to make sure I never lost sight of the beginner's perspective. To my surprise, she devoured the information in relatively short order. While

she impatiently waited for me to produce another chapter, she downloaded Microsoft Visual Basic 2005 Express Edition and started working on tutorials to learn how to program so that she could eventually link her database to an application. I hope you also are able to use this book to develop your database skills and are inspired to expand your technical skills in other complimentary areas such as programming database applications.

## How This Book Is Organized

This book consists of six chapters, which lead you from an introduction to database technology through installing the product, building a database, and other common activities. In Chapter 1, "Introducing Microsoft SQL Server 2005 Express Edition," you learn many important database concepts, as well as the features that make SQL Server 2005 Express Edition a great place to see how these concepts apply in practice. In Chapter 2, "Installing SQL Server Express," you install SQL Server 2005 Express Edition and SQL Server Management Studio Express to prepare for working through the remaining chapters. Next, in Chapter 3, "Creating Your First Database," you build the Home Inventory database and add tables, relationships, and an index to put into practice the concepts learned in the first chapter. Chapter 4, "Working with Transact-SQL," shows you how to use the Transact-SQL (T-SQL) language to add data to your tables, make changes to that data, find data meeting specific conditions, summarize data, and many other useful tasks. In Chapter 5, "Maintaining Your Database," you learn how to perform important administrative tasks, such as securing your database and backing up or restoring your database. Lastly, Chapter 6, "Managing Your Server," shows you the difference between named and user instances and explains how to work with a user instance of your Home Inventory database. Because each chapter builds on the work performed in previous chapters, be sure to read each chapter in sequence.

## Conventions and Features in This Book

This book presents information by using conventions designed to make the information readable and easy to follow. Before you start the book, read the following list, which explains conventions you'll see throughout the book and points out helpful features in the book that you might want to use.

### Conventions

- Each exercise is a series of tasks. Each task is presented as a series of numbered steps (1, 2, and so on). Each exercise is preceded by a procedural heading that lets you know what you will accomplish in the exercise.
- Notes labeled "Tip" provide additional information or alternative methods for completing a step successfully.
- Notes labeled "Caution" alert you to information you need to check before continuing.
- Text that you type, or items you point to or click appear in bold.
- Menu commands, dialog box titles, and other user interface elements appear with each word capitalized.
- A plus sign (+) between two key names means that you must hold down the first key while you press the second key. For example, *Press* ***Alt+Tab*** means that you hold down the **Alt** key while you press the **Tab** key.

## Other Features

- Sidebars throughout the book provide more in-depth information about the content. The sidebars might contain background information, design tips, or features related to the information being discussed.
- Each chapter ends with an In Summary... section that briefly reviews what you learned in the current chapter and previews what the next chapter will present.

## System Requirements

You'll need the following hardware and software to complete the exercises in this book:

- Microsoft Windows XP with Service Pack 2 (SP2), Microsoft Windows Server 2003 with Service Pack 1 (SP1), Microsoft Windows 2000 with Service Pack 4 (SP4), or Microsoft Vista Home Basic and later
- Microsoft SQL Server 2005 Express Edition
- Microsoft SQL Server Management Studio Express
- Microsoft .NET Framework 2.0
- Microsoft Windows Installer 3.1
- Microsoft Internet Explorer 6.0 SP1 or later
- Computer with a Pentium III–compatible or faster processor, 1 gigahertz (GHz) or faster
- 192 megabyte (MB) random access memory (RAM), 512 MB or more recommended
- Approximately 350 MB hard disk space for installation
- Approximately 9 MB for the database files create as you work through this book
- Video monitor (1024 × 768 or higher resolution)
- Microsoft mouse or compatible pointing device

You'll also need administrator access to your computer to configure SQL Server Express.

## Obtaining SQL Server 2005 Express Edition

To work through the chapters of this book, you'll need to install SQL Server 2005 Express Edition and SQL Server Management Studio Express. If you don't already have these products installed, follow these steps to get the required software downloaded to your computer:

1. Using your browser, navigate to: go.microsoft.com/fwlink/?LinkID=64064.
2. Click the link **Yes, I Want To Register And Receive The Benefits.**
3. Sign in to Microsoft using your Windows LiveID. Or sign up if you don't already have a Windows LiveID.
4. Complete the registration page by providing the requested information in required fields. Continue to the download site.
5. On the download page, click the **Download** link below Install Microsoft SQL Server 2005 Express Edition.
6. After the download completes, click the **Download** link below SQL Server Management Studio Express.

You'll learn how to run the installation procedures for both of these products in Chapter 2.

# About the Author

## Stacia Misner

Stacia Misner is the founder of Data Inspirations, where she delivers global business intelligence consulting and education services. She is a consultant, educator, mentor, and author specializing in business intelligence and performance management solutions using Microsoft technologies. Stacia has more than 23 years of experience in IT and has focused exclusively on Microsoft SQL Server and related technologies since 2000. She wrote both *Microsoft® SQL Server™ 2005 Reporting Services Step by Step* and *Microsoft® SQL Server™ 2000 Reporting Services Step by Step*, and coauthored *Microsoft® SQL Server™ 2005 Administrator's Companion*, *Microsoft® SQL Server™ 2005 Analysis Services Step by Step*, and *Business Intelligence: Making Better Decisions Faster*. Stacia currently lives in Las Vegas, Nevada, with her husband, Gerry, and their flock of parrots. She can be contacted at *smisner@datainspirations.com*.

## Dedication

This book is dedicated to my husband, Gerry, who reminds me everyday how much he believes in me and encourages me to pursue ever greater challenges. As Aristotle said, "A true friend is one soul in two bodies." My husband is my soulmate and truest friend.

## Acknowledgments

The best part of completing a book, in my mind, is the reflection on a job well done by a great team of people. The process of taking a book from an idea to a tangible finished product is not a solo effort. Some people play a more active role than others, but collectively everyone's contribution is vital to bring the project to a successful conclusion. I always enjoy working with the Microsoft Press team and each book project introduces me to someone new that I come to appreciate as much as the others.

On this particular project, I would like to thank Ben Ryan, Devon Musgrave, and Victoria Thulman from Microsoft Press for their respective efforts these past several months. Victoria deserves some extra recognition for acting with grace under pressure as we all worked furiously at the end of the project to meet deadlines while overcoming obstacles. The copy editors and production team also deserve recognition for their transformation of a simple manuscript and some screenshots into a beautiful collection of pages in a book.

I must also thank Mike Wachal from the SQL Server Express team not only for his very thorough technical review of the book, but also for working with me in the early stage of the book to define the type of book we wanted to produce and the overall approach. He also very patiently educated me about several of the nuances of SQL Server Express

that differentiate it from the rest of the SQL Server family. Throughout the development of the book, I was struck by his passion for what he does on a daily basis with SQL Server Express and his desire to help beginners succeed in their early experiences with this product. It was a privilege to work with him on this book.

Finally, my family deserves recognition for their contributions. My husband, Gerry, is incredibly tolerant of the demands of my work and especially my writing projects. His best contribution is allowing me the time and space to do my work, but also making sure I don't forget to eat or ensuring I make some time to sleep. My daughter, Erika Bakse, really stepped up to the plate as my test reader and wildly surpassed my expectations. I didn't expect her to have a problem with the technical content as a relative newbie to databases and technology in general, but I also didn't expect she would find it interesting and fun! Of course, I find it interesting and fun, but that's me. I'm delighted to have shared this experience with her, and appreciate the work she did to test the procedures and to provide feedback from that newbie perspective.

Chapter 1

# Introducing Microsoft SQL Server 2005 Express Edition

*What Is a Data Service?, 2*

*What Should You Know About Databases?, 4*

*What Is Microsoft SQL Server 2005 Express Edition?, 14*

If you're curious about databases and have felt overwhelmed by books and articles that assume you've already got some experience, then this is the right book for you. I'm going to assume that you got this book with nothing but healthy curiosity, a computer that meets the minimum requirements outlined in the Introduction, and enough technical skill to install and run a Windows application. Whether or not you plan to work as a database professional someday, you'll get lots of hands-on practice developing a simple database as you work through this book. When you've reached the end, you'll have the confidence to tackle another database project on your own. Before you start the exercises, you should master a few important concepts first. In this chapter, I'll give you some background information about databases in general, explain how databases store information, and what Microsoft SQL Server 2005 Express Edition does that makes it a great product for people getting started just like you.

Collecting and categorizing information is an activity the human brain does very well, but there are limits on its ability to search and recall specific sets of information when the amount of information collected exceeds a certain threshold. Thankfully, we can use tools such as word processing or spreadsheet programs, for example, Microsoft Office Word or Microsoft Office Excel, to help us manage information more efficiently, but even these tools have their limitations.

As an amateur genealogist, I am continually collecting information about my family history, such as names, dates, and locations of my ancestors and their relatives. If I were to use Word to store all the information I collect, I could search for everyone that lived in Massachusetts and find them one by one, but I'd have to cut and paste a lot if I wanted to create a list of those individuals only. Excel would be a better tool for storing this information because I could use a filter to produce a list of the people from Massachusetts, and I could even use multiple custom filters to produce a list of people born in Massachusetts between 1700 and 1750. However, there is a limit to the number of conditions I can specify for each filter, and there is a limit to the amount of information I can store in Excel. For the greatest flexibility in storing and retrieving my family history information, a database is the best solution.

A *database* is simply a container used to store information. You use *database management system* software to build, maintain, and query a database. Some database management systems are intended for use by a single user to store data locally. This local data store might be created and maintained by the user, or perhaps it's a subset of data from a corporate database the user can access when disconnected from the network. Some desktop applications might even use a database management system instead of the file system to store the user's data, although the distinction is usually transparent to the user.

A database management system that responds to simultaneous requests from *clients*, typically multiple users across a network, is a *data service*. To visualize the relationship between the database management system and clients, refer to Figure 1-1. You can refer back to this figure throughout this chapter as I explain related concepts.

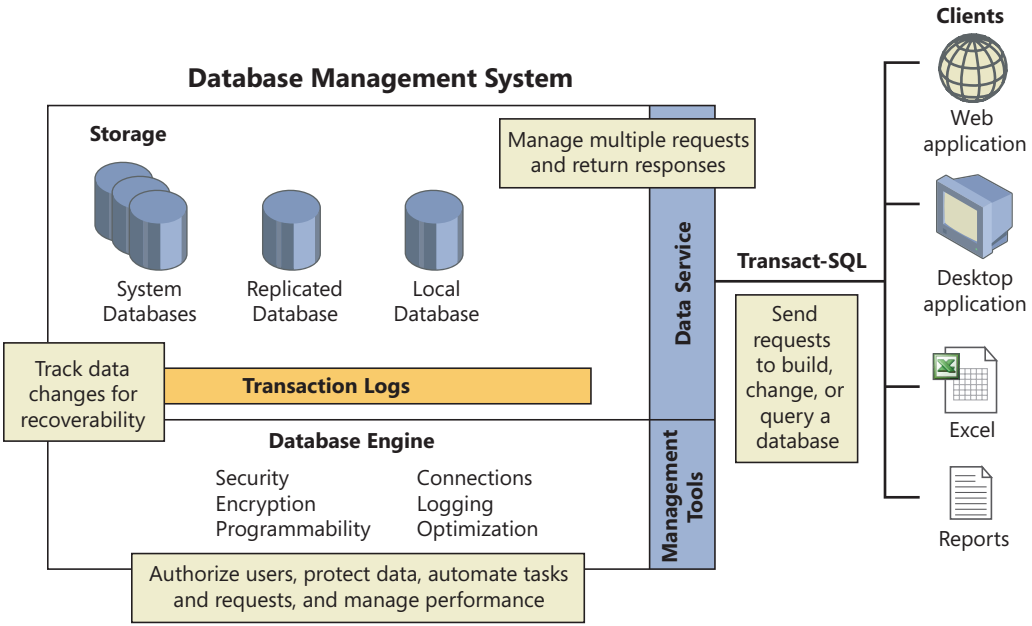



*Figure 1-1*
*A database management system*

A database management system acting as a data service requires different features than one that stores one user's data because it not only has to allow many people to connect to a single database at the same time, but it has to efficiently process each person's *query*, a request for information from the database. In addition to managing connections and queries, the data service must manage requests to add new information to a database, change existing information, or delete information. While all this action is going on, the data service also has to make sure that people see only the data they are allowed to see and make only the changes to data or data structures they are allowed to make.

You should also understand some important differences concerning a data service and the Windows programs installed on your computer. A Windows *service* is a special type of program that you can set up to start when the computer starts without requiring you to log on. Even when you're logged off, the service keeps running. It stops only when you explicitly stop the service or turn off the computer. Because a data service operates as a Windows service, it can run without you, which makes it a better choice than a program because you don't have to be at your computer or leave it logged on if you want someone else to use your database. As long as the other person's computer can connect to yours over a network,

that person can use the database that you are sharing. Of course, this scenario can work only if you set up the database security to let that person use the database.

## What Should You Know About Databases?

As you learned in the previous section, a database is a container for information. Now it's time for you to learn how information is structured within a database. Before you can properly design and build a relational database, you need to understand a few fundamental concepts and how to apply certain rules. In this section, I'll explain the basic components of a database and why applying normalization, data integrity, and referential integrity rules are important.

### What's Inside a Database?

A database these days is often constructed using a *relational model*. A relational model is a database format using a set of related *tables*, which are two-dimensional structures using *rows* and *columns* to store information about something. Each row contains data values separated into multiple columns, each of which stores a specific type of information. A database contains many objects, but the primary object in it is a *table*.

Good database design principles dictate that a table contains information about a single *entity*, which is a generic term used to represent either an object or an occurrence. In general, you can think of an object as a person, place, or thing. So let's see how this applies to a home inventory database. You would use one table to store information about the things you own and another table to store information about rooms in your house, but you wouldn't create one table to hold both types of information. Information about each purchase, which is an example of an occurrence, would be kept in a third table.

Take a look at Table 1-1, which illustrates how rows and columns work together in a table to store data. The purpose of this table is to use a home inventory database to store basic information about items in your house.

| Description | Quantity | PurchaseDate |
|---|---|---|
| Television | 1 | 3/21/2000 |
| Dinette set | 1 | 10/4/2003 |
| Bookshelf | 2 | 5/10/2002 |
| Couch | 1 | 7/28/2005 |

*Table 1-1*
*Item Table*

**NOTE**

**This table is deliberately simpler than the table you'll build later in Chapter 3, "Creating Your First Database," so that you can focus on the key concepts you're learning in this section.**

This Item table has four rows of data. Each row contains data about one and only one item, although an item doesn't have to be literally one thing. If I have several items that are identical and purchased on the same date, I can consolidate the information as a single row and use a Quantity column to keep track of how many I have, as shown in the table's third row for Bookshelf.

Each column in the table describes the type of information it can store. When you add a column to the table, you provide a name so you can refer to it later when adding related objects to the database or when querying the table. You also must specify a *data type*; that is, you must choose whether a column can contain text as shown in the Description column, numbers as shown in the Quantity column, or dates as shown in the PurchaseDate column. Each row of data you add to a table must contain columns using the same data type defined for the corresponding column in the table or else you won't be able to add it to the table. As an example, the database management system won't let you accidentally put description data into the quantity column.

**NOTE**

**Many more data types are available, which you'll learn about later in Chapter 3.**

## What Is Normalization?

In addition to the rule limiting a table to a single entity, another rule, called *normalization*, is used to optimize the database structure and thereby make it easier for you to manage the database, to ensure data is consistent, and to get queries answered as quickly as possible. Normalization is actually a series of steps that you use to break data down into the smallest possible units and to distribute data across multiple tables to improve data performance and minimize redundancy. Minimal redundancy in a database is one way to make sure your data stays in good shape. I'll explain more about the potential problems of redundant data later in this section.

When you complete the first step of normalization, the table structure is described as first *normal form*. The table structure resulting from the second and third steps, as you can probably guess, is called second and third normal form, respectively. Several more possible levels of normal forms exist, but many relational databases are built using third normal form.

It's not important to know which rules apply to which normal form at this stage of your learning process. As long as you follow the rules for third normal form shown below, your database design will be in good shape.

- Break down the data values into smallest possible units of information for each column. The home inventory database is pretty simple and doesn't include an example of data that illustrates this point well, so let's consider a different example. Let's say you want to build a table to hold contact information for your friends. You might be inclined to put the entire name of each person into a single column called Name, but the proper thing to do in a normalized database is to create two columns, FirstName and LastName. If you were to store the full name in a single column, you'd find it challenging to sort the list of your friends by first name only or by last name only. It's not impossible, but you can avoid the problem up front by using a better design.
- Make sure you have a way to uniquely identify each row in the database. You might be able to use one of the data values that is already in your data if the values are unique. The column identifying each unique row is called a *primary key*. Sometimes you might have to use a combination of columns to create a unique identifier. This type of key is called a *composite key*.

  In the Item table shown in Table 1-1, there is no good candidate for a primary key. You might have more than one television, so Description wouldn't work. A combination using Description and PurchaseDate might work, but it makes for a rather large and inefficient composite key. If you have a lot of data, database performance improves when your primary key is relatively small in width and numeric, like an integer. Furthermore, you may not always have a PurchaseDate value for each item. Putting in any old date just to make a unique record isn't a good idea either. How would you know which dates are valid and which aren't? Perhaps a serial number column in the table would be a better way to guarantee uniqueness, but not all items have a serial number so a problem still remains.

The solution to these data dilemmas is to create a *surrogate key*, such as the ItemID column shown in Table 1-2. A surrogate key is one which has no meaning to the user and, for this reason, is often kept hidden from the user. More importantly, though, it's easy to set up and maintain when you need to guarantee each row in the table has a unique primary key. In Microsoft SQL Server databases, you can create an *identity column* to use as a surrogate key. An identity column is a column for which you define a starting number and which is automatically incremented each time you add a new row to the table.

| ItemID | Description | Quantity | PurchaseDate |
|---|---|---|---|
| 1 | Television | 1 | 3/21/2000 |
| 2 | Dining table | 1 | 10/4/2003 |
| 3 | Bookshelf | 2 | 5/10/2002 |
| 4 | Couch | 1 | 7/28/2005 |

**Table 1-2**
*Item Table*

- Decompose data into multiple tables to avoid data redundancy. Redundancy is usually considered a bad thing in databases. Though "disk space is cheap" is a phrase commonly tossed about to justify keeping as much data around as possible, database design is all about storing data efficiently in the smallest possible space to achieve the best possible performance.

  Using the home inventory database again as an example, it seems reasonable to add another column to track which room a particular item is in. Unless you have particularly sparse rooms, you will probably have multiple items in a single room, as shown in Table 1-3.

| ItemID | Description | Quantity | Location | PurchaseDate |
|---|---|---|---|---|
| 1 | Television | 1 | Living room | 3/21/2000 |
| 2 | Dining table | 1 | Dining room | 10/4/2003 |
| 3 | Bookshelf | 2 | Living room | 5/10/2002 |
| 4 | Couch | 1 | Living room | 7/28/2005 |

**Table 1-3**
*Item Table*

To normalize this table, you would modify the Item table, as shown in Table 1-4, and create a Location table, as shown in Table 1-5. The overall space savings become a lot more dramatic when you have hundreds or thousands of rows in the Item table as compared to the small number of rows currently shown in the table.

*Table 1-4*
*Item Table*

| ItemID | Description | Quantity | LocationID | PurchaseDate |
|---|---|---|---|---|
| 1 | Television | 1 | 1 | 3/21/2000 |
| 2 | Dining table | 1 | 2 | 10/4/2003 |
| 3 | Bookshelf | 2 | 1 | 5/10/2002 |
| 4 | Couch | 1 | 1 | 7/28/2005 |

*Table 1-5*
*Location Table*

| LocationID | Location |
|---|---|
| 1 | Living room |
| 2 | Dining room |
| 3 | Kitchen |
| 4 | Master bedroom |

In this example, the Location table is called a *lookup table*. The LocationID column stores values that are referenced in the Item table. The LocationID column is thus defined as a *foreign key* column because it contains values stored in the primary key column of the Location table. Some people refer to this two-table structure as a parent-child relationship. The Location table is the *parent table* because the Item table, as the *child table,* is dependent on it for a valid LocationID.

**NOTE**

**There's no rule that the foreign key column must be named the same as the primary key column in the lookup table, but the practice does make it easier to remember the data relationships in your database.**

- Break down calculated values into separate base values. The home inventory database example we've been reviewing doesn't lend itself well to a discussion on this point, so let's think about a table that tracks transactions, such as a sales table. Rather than store the total amount of the sale, including freight and tax, in a single column, you should create separate columns for each value that is used to calculate the sale, as shown in Table 1-6.

| InvoiceID | ProductID | Date | Quantity | SalesPrice | Tax | Freight |
|---|---|---|---|---|---|---|
| 1204 | 1149 | 4/4/2007 | 2 | 9.99 | 1.40 | 0 |
| 1204 | 2865 | 4/4/2007 | 1 | 22.99 | 1.61 | 0 |
| 1205 | 8689 | 4/4/2007 | 4 | 3.99 | 1.12 | 5.00 |
| 1206 | 5347 | 4/5/2007 | 1 | 215.00 | 15.05 | 32.00 |

***Table 1-6***
*Sales Table*

You might think that because this Sales table is using more columns when one could have sufficed this approach is violating the rule I mentioned previously, in which I told you to keep the amount of space required per row at a minimum. However, sometimes you have to consider that benefits can outweigh costs. By separating the base values, you can still calculate the total sales amount, but you also can perform analytical queries to compute average sales by product or by time. Typically, average sales calculations don't include tax and freight because these are charges passed on to the buyer and don't help a sales manager monitor sales performance.

- Don't use multiple columns for similar types of data. To clarify this rule, let's assume that you want to keep track of photos for your more important items in the home inventory database. Perhaps you think you're going to need only one image per item, so you design the Item table to include a column that stores the file name and location of the photo. Later, you realize you should have a photo of some items from the front and from the back. Do you add another column and now have FrontPhoto and BackPhoto columns? What happens when you decide you need three or four photos for a particular item? Rather than impose a limit on the number of photos you track for any given item, or continually increase the size of the table by adding more and more columns, a better approach is to put that data into a separate table. Each row is a separate photo so you can associate multiple photos with a single item in the Item table. Using this method, you can have zero photos from some items, one photo for other items, or as many photos as you like for any given item.

NOTE

**It's possible to store images in a database, but it's not always a good idea. Image files are much larger than other data you store, and you can experience performance problems because the database management can't work with images as efficiently as other types of data. A common practice is to store a URL address or a folder and file name as a pointer to the image file. Your application can then locate and display the image quickly.**

Will the database police come looking for you if you violate any of these normalization rules? Of course not! The amount of data you're likely to add to the home inventory database is probably not enough for you to notice a performance problem as you update or query the database. However, if you develop good habits now while working with a simple database, you'll be prepared to tackle a more complicated database correctly in the future.

NOTE

**Sometimes there are reasons to break the normalization rules. It's another example of weighing the costs against the benefits to come up with the right design. *Denormalization* is the process of combining data from multiple tables to store the same data. Usually when you do this, you keep both the original table and the new table with combined data in the same database. The data might be identical in the new table, or it might be summarized. That is, you might total sales by month and store the results in a monthly sales table. As new sales occur, rows are added to the original sales table, and new monthly values must be calculated and added or updated in the monthly sales table. This technique is useful when you have a high volume of data that is frequently queried to compute summarized values. If you have those values already calculated and stored in a table, queries against that table run faster than queries that have to find and calculate values from the detail rows. But this approach does have downsides. To name a couple, more disk space is required to store your database, and you have to be very careful to keep the summary table consistent with the detail table.**

## How Do You Maintain Data Integrity?

You may have heard the phrase "Garbage in, garbage out" at some point. Certainly, it seems reasonable to expect that bad data will not magically transform when you put it into a database and later come back out in great condition. But did you know it's possible to put good data into your database and get bad data out later? That's what can happen if you don't take the time to design your database with data integrity in mind. That is, you need to apply rules to keep your data clean, consistent, and correct so you don't get garbage when you query the database.

NOTE

**At this point, I'd like you to simply get familiar with the general concepts presented here. Don't worry about the implementation steps right now. We'll get to that in the Chapter 3, but the information will be scattered throughout the chapter. After you finish this book, you can refer back to this chapter section as a data integrity checklist whenever you work on database design tasks.**

Even though an application often performs some sort of validation routine to make sure users input nice clean data, it's considered good practice to use database features to enforce data integrity. Data integrity rules break down into four categories, as detailed in the following list.

**Domain integrity** Does each column have a valid value? You can use the following techniques to enforce domain integrity.

- Data types control what kind of data can be stored in a column. It's your first line of defense in the effort to keep data clean because the database engine won't accept any other data type in this column. For example, if you create a Quantity column with an integer data type, you can't accidentally put text in that column.
- *CHECK constraints* test a data value before it's added to a new row or before it replaces a value in an existing row to make sure it's within the expected range. A data type won't help if someone tries to use a number that's logically out of bounds, but a CHECK constraint will. In a time tracking application requiring daily input, for instance, you could use a CHECK constraint to make sure the value is less than 24 because it's impossible (so I'm told) to work more than 24 hours in a day.
- A *FOREIGN KEY constraint* requires a parent table to contain a value in the new or updated row of the child table. Typically, this means the foreign key value in the child table has a matching primary key in the parent table, but it's not required. Any column (or columns, for that matter) in the parent table can be used in a FOREIGN KEY constraint. Referring back to Tables 1-4 and 1-5, a traditional FOREIGN KEY constraint on the LocationID column in the Item table would prevent you from adding a new row to that table with a LocationID value of 5 because that value doesn't exist yet in the Location table's primary key column. Without the constraint, adding a row with an invalid LocationID would result in an *orphan row* in the Item table, a row that has no corresponding row in the parent table.
- A *DEFAULT definition* for a column guarantees a valid value gets used if no other value is provided for that column when a new row is added. This technique is not only useful for ensuring data integrity of new data, but it can also simplify the data preparation process because you need only provide values for columns that don't have a DEFAULT definition. Perhaps the vast majority of the items in your home inventory have a quantity of 1. You can add a DEFAULT definition to the Quantity column (shown in Table 1-4) to enter a value of 1 automatically whenever you are loading new data into the table. You can, of course, provide a different value in the new rows for those items that you have in multiple quantities.

**NOTE**

**You might be wondering what problems are caused by orphan rows. If you queried the Item table only, you could get a complete list of all items. No problem. But if you tried to filter the query to list only items found in a particular room, you'd miss the orphan rows and never know it.**

- A *NOT NULL constraint* on a column makes sure that you can't use NULL as a value in a column. Using a NOT NULL constraint is recommended whenever possible because it helps improve database performance.

**IMPORTANT**

**NULL as a value is commonly misunderstood when people are just starting to learn about databases. NULL is supposed to mean you don't know what the value really is. This is not the same as zero, which is a distinct numeric value. An empty string ('') is recognized by SQL Server as a string because it uses single quotation marks to indicate the beginning and end of the string, but it contains no other text characters in the string. Even though you can't see a value in that string, empty is a value that is still not the same as a NULL value. Finally, a blank column has no value at all, which also is not the same as a NULL value. What's the big deal? When you try to compare a column containing zero, an empty string, or blank values with a column containing a NULL value, you'll find the values aren't equal, which often takes beginning database developers by surprise. You might think it's best to avoid NULL values altogether, but it's useful for numeric columns for which you want to compute statistics. If you truly don't know a value, use NULL instead of 0 due to the fact that NULL will not skew the statistic (such as an average) because the row will be ignored, whereas 0 will force the row to be incorrectly included in the computation.**

**Entity integrity** Is each row in a table unique? You can use the following techniques to enforce entity integrity.

- A *PRIMARY KEY constraint* guarantees every row in the table is unique. Remember, this is one of the normalization requirements mentioned in the previous section. It also makes sure you don't try to add another row with the same primary key or a row with a primary key containing a NULL value. So, in the case of the Item table shown in Table 1-4, any attempt to insert another record with a primary key value of 1, 2, 3, 4, or NULL will be rejected.
- A *UNIQUE constraint* is similar to the PRIMARY KEY constraint in that its purpose is to guarantee uniqueness, but it can be used on any column or group of columns. In addition, a NULL value can be used in the column with a UNIQUE constraint, but only one row can contain NULL in this column because otherwise the column would not contain unique values. After reviewing the tables in the home inventory database, you might decide it makes sense to put a UNIQUE constraint on the Location column in the Location table (Table 1-5) to prevent the accidental insertion of a duplicate row.

**Referential integrity** Does the data in a child table correctly relate to data in a parent table? You can use the following techniques to enforce referential integrity.

- A *foreign key relationship* is the definition added to a child table to link a foreign key column to a column in the parent table. It is required if you want to apply a FOREIGN KEY

constraint to the relationship as described in the discussion of domain integrity, so they are often created at the same time.

- A FOREIGN KEY constraint is vital for referential integrity. As you've already learned, it makes sure you can't add a value to a foreign key column in a child table if that value isn't in the parent table. However, this constraint really works both ways. That is, the parent table is affected as well by the constraint, which means you can't delete a row from the parent table if the child table has related rows, thereby protecting the referential integrity of both tables.
- A CHECK constraint is another option you can use to enforce referential integrity. It may be that not only do you want to ensure values exist in a particular column in a parent table, but you also need to limit the value to a particular range of possible values.

**User-defined integrity** Does the data conform to other rules required by your application? You can use the following techniques to enforce entity integrity.

- Any of the constraints I described above—PRIMARY KEY constraints, FOREIGN KEY constraints, CHECK constraints, or UNIQUE constraints—can be used to add your own data integrity rules to the database.
- A DEFAULT definition can be used very creatively to apply the additional rules. You are not limited to providing a constant value; you can also use a system function to calculate a value or specify a NULL value as a default.
- A *trigger* is a special type of *stored procedure*, which in turn is a collection of code you build to perform certain functions and store in the database to process when requested by you or an application. A trigger, by contrast, is processed when you try to add new rows to a table or when you try to update or delete existing rows. One handy use for a trigger is to look up values in other tables—which you can't do with a CHECK constraint—but there are many more possibilities.

Whew! Your head must be reeling with all these new terms and concepts. By reading this section, you've been introduced to the most important database fundamentals. There's a lot more to learn, of course, but now you've got a basic understanding of what a database is, why the design of table structures in the database is important, and how database management features help you keep clean information in the database.

**NOTE**

**Interestingly, you can disable this constraint for a foreign key relationship if you need to do some sort of maintenance that would be detected as a violation of the constraint. Obviously, you should use this option with care, or else you're putting at risk the referential integrity you've worked so hard to create. Another situation in which the FOREIGN KEY constraint might be disabled is during the insertion of a lot of new rows into the table. Sometimes the performance benefit gained by faster insertion outweighs the potential cost. Again, you need to be very careful to ensure the incoming rows have been validated for referential integrity before you start adding them to the table. In either case, when you've finished your task, be sure to enable the constraint.**

SQL Server 2005 is a family of database management systems that can run either as a local data store or as a data service. As with many database management systems, SQL Server 2005 stores data in a relational model. SQL Server 2005 is popular because it includes features that optimize data access, especially when large volumes of data are involved, and it includes tools that help database administrators and application developers perform their tasks more productively. Though a very large company needs a database that is available to thousands of users (or more!) at all hours seven days a week, a small business probably doesn't need that much horsepower to function successfully. Similarly, if you want to build a database to store data for your Web site, you can get by with fewer features. For this reason, Microsoft created multiple editions of SQL Server 2005 to allow people to select the set of features that best suits their requirements.

SQL Server 2005 Express Edition was designed as a replacement for Microsoft SQL Server Desktop Engine 2000 (MSDE). As you probably guessed from its name, MSDE was intended for use as a local data store by applications. Software developers were allowed to create these applications and include MSDE with those applications without paying royalties to Microsoft. Like its predecessor, SQL Server Express is also available and redistributable at no charge, but it includes many new features found in the other editions of SQL Server 2005 that overcome many issues some people experienced with MSDE.

**NOTE**

**From this point forward, I'll refer to SQL Server 2005 Express Edition by the commonly used short version, SQL Server Express. When I mention SQL Server 2005, I am talking about all editions of SQL Server collectively.**

Just because professional software developers use SQL Server Express in their applications, you shouldn't be worried that it might be too complicated for you if you're still a beginner. It's a great tool to learn how databases work and even to start organizing your own collection of information. With this book as your guide, you'll discover how easy it is to build useful and powerful databases with SQL Server Express.

## What Kinds of Databases Can You Build with SQL Server Express?

Using this version of SQL Server 2005, you'll be able to create the following types of databases.

**Embedded databases** When an application is intended for a single user and stores its data in a database rather than on the file system, the type of database it uses is called an *embedded database*. The user may not even know a database is being used because the database is managed without the user's active involvement. An example of this type of database might be found in a personal finances application. To build this type of database, you can use Microsoft Visual Basic 2005 Express Edition, a free download available from Microsoft for application developers.

**Application databases** You can use also use Microsoft Visual Basic 2005 Express Edition to develop Windows applications that retrieve data from your databases. For Web applications that display data from a database, you can use Microsoft Visual Web Developer 2005 Express Edition. In either case, an *application database* is typically a centrally located database that is accessed and updated by one or more users as they use an application. An application database is shared, whereas an embedded database is not.

**Distributed client databases** A variation of the application database, which assumes people are connected to a central database while they work, is the *distributed client database,* which is intended to accommodate situations when people still need to access and update information even when they are not connected. These people use a copy of the database on their own computer, which is kept up-to-date with the central database through a mechanism called replication. SQL Server Express can be used as a client database in this scenario, but it cannot be used as the central database responsible for managing replication with each client.

NOTE

**In this book, you'll build a database that could be used as an application database if you were to build an application for it. To learn how to build such applications, you can read *Microsoft Visual Basic 2005 Express Edition: Build a Program Now!* by Patrice Pelland (Microsoft Press, 2006) or *Microsoft Visual Web Developer 2005 Express Edition: Build a Web Site Now!* by Jim Buyens (Microsoft Press, 2006).**

## What Are the Key Features You Need to Know About?

If I were to make a complete list of all the features found in SQL Server Express, that list would be incredibly long and probably most of the items on the list wouldn't make a lot of sense to you yet. Instead, I picked out the key features you should start learning about now to get a basic understanding of what SQL Server Express can do.

**Easy installation** A simple setup wizard hides the advanced configuration features so you can get started quickly. If you want to share your database later, you can create an application to make it easy for your users to install both SQL Server Express and your database application.

**Easy management** A set of graphical tools is available to help you manage the server without learning complex commands.

- SQL Server Management Studio Express Edition (available as a separate download) lets you easily create and view databases and related objects as well as develop and execute *Transact-SQL* statements and scripts. Transact-SQL is the language you use to create and modify a database and the objects it contains as well as to query the database or manage the server.
- SQL Server Configuration Manager lets you configure the properties for the data service and define the methods that clients are allowed to use to communicate with the server.
- SQL Server Surface Area Configuration lets you change the startup property for the data service, start and stop the service, specify whether only local connections or both local and remote connections can access the data service, and enable or disable specific features.

In addition to these tools, SQL Server Express automates several management activities to keep your databases performing at their best, including database tuning, query optimization, and optional software patching to keep SQL Server Express up-to-date with the latest service packs.

**Complete database functionality** The database engine for SQL Server Express is no different from the database engine found in all other editions of SQL Server. In fact, SQL Server Express databases are fully compatible with other editions so you can easily upgrade later to take advantage of another edition's features if you eventually run into limits with what you can accomplish with SQL Server Express. Meanwhile, you get all the same security, data encryption, and programmability features found in the SQL Server 2005 family.

**NOTE**

**Although SQL Server Express shares the same database engine with the rest of the SQL Server 2005 family, you need to understand some of its limitations before deciding it's the right edition for you. SQL Server Express can use only 1 CPU—single or dual core—and 1 gigabyte (GB) of random access memory (RAM). In addition, each database must be 4 GB or smaller, although there is no limit to the number of databases you can create in SQL Server Express. Several services available in other editions are not available in SQL Server Express: SQL Agent for scheduling jobs, Notification Services for alerting, Analysis Services for supporting online analytical processing, and Integration Services for managing data movement. You can use Reporting Services to build reports using data from your database if you install SQL Server 2005 Express Edition with Advanced Services. You should also be aware that SQL Server Express does not include the Database Tuning Advisor for suggesting database improvements, nor does it include SQL Server Profiler for monitoring performance.**

**XML support** In addition to storing data relationally, which you'll learn how to do in Chapter 3, you can also store data hierarchically using an XML format. SQL Server 2005 includes an XML data type for storing a complete XML document or an XML fragment. To extract data from the stored XML, SQL Server Express lets you use a special query language called XQuery.

**NOTE**

**In this book, I'll focus on working with relational data, but if you are interested in learning more about how XML is supported in SQL Server 2005, you can check out this MSDN article, *http://msdn2.microsoft.com/en-us/library/ms345117.aspx.***

**Visual Studio 2005 integration** If you happen to be using Visual Studio 2005 to develop an application that accesses data from a database, you can use features included with Visual Studio 2005 to design your database, add data to your database, and even query that database. On the flip side, you can develop code in Visual Studio 2005 to create Common Language Runtime (CLR) objects in your database when you need to incorporate complex logic.

**NOTE**

**Integrating CLR objects into your database will not be covered in this book because it's an advanced topic. If you're curious, you can learn more by reading this MSDN article, *msdn2.microsoft.com/en-us/library/ms160866.aspx.***

**Security** In case you're not a security expert yet, SQL Server Express is "secure by default," which means your databases and the data they contain will be protected right from the start. As an administrator, which you must be to install SQL Server Express, you have the ability to do much more than anyone else. As you learn more about how your database should be accessed by authorized users, you can make changes to the security settings as needed. In Chapter 5, "Maintaining Your Database," I'll help you learn how to do this.

**TIP**

**After you've completed the exercises in this book, you can use the starter kits available at *msdn.microsoft.com/vstudio/express/sql/starterkit/default.aspx* to review what you've learned and to explore more ideas for using SQL Server Express. Because these starter kits are complete applications, you will need to download and install a language-specific Express Edition for Visual Studio 2005, such as Visual Web Developer, Visual Basic, or Visual C# as applicable to the starter kit you want to try.**

All of these features in a free product? It's amazing, isn't it? And SQL Server Express is easy and fun to use, too!

## Where Can You Get Help?

At some point, you're going to have a question about something that I couldn't cover in this book, or you might run into a problem and need help troubleshooting and fixing it. A lot of online resources can help you in these situations. Here is a list of sites that you might find helpful at one point or another.

- Microsoft SQL Server 2005 Express Edition Books Online: *msdn2.microsoft.com/en-us/library/ms165706.aspx*
- SQL Server 2005 Express Edition Developer Center: *msdn2.microsoft.com/en-us/sql/aa336346.aspx*
- MSDN Forums for SQL Express: *http://go.microsoft.com/fwlink/?LinkId=62430*
- Microsoft SQL Server TechCenter: *www.microsoft.com/technet/prodtechnol/sql/default.mspx*
- SQL Server Express Weblog: *blogs.msdn.com/sqlexpress/default.aspx*

# In Summary...

Now you have a better understanding of the usefulness of databases and the many tasks that a database management system must perform. You also know the difference between a service and a program. Importantly, you have a grasp of the most important database concepts in preparation for building your first database with SQL Server Express. In Chapter 3, you'll get a chance to see these concepts put into practice. Finally, you learned about the SQL Server 2005 family and what makes SQL Server Express different and special. The next step is to install SQL Server 2005 Express Edition, which I'll show you how to do in the next chapter.

Chapter 2

# Installing SQL Server Express


*Preparing to Install SQL Server Express, 20*

*Installing SQL Server Express, 20*

*Installing SQL Server Management Studio Express, 28*

*Going a Few Steps Farther, 31*


In this chapter, I'll show you step by step how to install SQL Server Express and its companion product SQL Server Management Studio Express. I'll explain a few new concepts as we go, or tell you where in the book I tackle that topic. Before you start the installation, I'll let you know how to get yourself and your computer ready. As I mentioned in the previous chapter, installation is very simple to perform and relatively quick, too. Let's get started!

### *The SQL Server Administrator Role and Microsoft Windows Vista*

*Before I explain how Windows Vista affects the installation of SQL Server Express, let me describe what happens during installation if you are using one of the other supported operating systems. As I mentioned in the Introduction, you must have administrator privileges to run setup for SQL Server Express. In this situation, because you're already defined as an administrator, SQL Server Express always recognizes you as an administrator and allows you to perform administrative tasks on the database engine anytime after installation. More specifically, any user who is a member of the Windows group BUILTIN\Administrators is able to perform administrative tasks because the installation process adds this group to the SysAdmin server role for SQL Server. (I'll explain more about the SQL Server Express roles in*

*Continued on next page*

It's important your computer meets the minimum software and hardware requirements I laid out in the Introduction of this book. If you skipped that part because you're so anxious to begin, stop right now and go back to review the Introduction. I explain there exactly what your computer needs and how to download SQL Server Express and SQL Server Management Studio Express if you don't yet have the software. It's a good idea to verify your computer has all the latest and greatest updates from Microsoft Update (*update.microsoft.com*) and, if not, download and install the necessary updates.

I didn't hammer on this point in the Introduction, but in case you overlooked it, let me reiterate there is a requirement that the .NET 2.0 Framework and Microsoft Windows Installer 3.1 are installed *before* you start the SQL Server Express installation. If you don't have these installed now, you absolutely will not be able to start the installation of SQL Server Express. Just like SQL Server Express, the .NET Framework 2.0 is free to download and use. Just go to *go.microsoft.com/fwlink/?LinkId=45988* and follow the instructions there to download and install it. You can locate the link to the correct version of Windows Installer 3.1 for your operating system at *go.microsoft.com/fwlink/?linkId=42467*.

I'm going to assume this installation is the first of any SQL Server edition on your computer. If that's an incorrect assumption, go to the Microsoft SQL Server 2005 Express Edition Readme for the latest service pack (*go.microsoft.com/fwlink/?linkid=68576)* and read section 3.1, "Prepare for a SQL Server Express Installation," for guidance on installation preparation when you already have other editions installed.

## Installing SQL Server Express

You've prepared your machine, and you know where to look online if you run into an issue (which is not very likely). To get underway with the installation of SQL Server Express, follow the steps below.

## TO INSTALL SQL SERVER 2005 EXPRESS EDITION

1. Click the **Start** button and then click **My Computer**.

2. Right-click the drive to which you downloaded the software and select **Explore**.

3. In the Folders list, select the folder in which you saved the software.

4. In the list of files, double-click **SQLEXPR.EXE** and then click **Run**.

5. Before you can install the product, you must read and accept the license agreement (Figure 2-1). Read the agreement carefully to understand how you can use this product. To continue the installation, select the **I accept the licensing terms and conditions** check box and click **Next**.

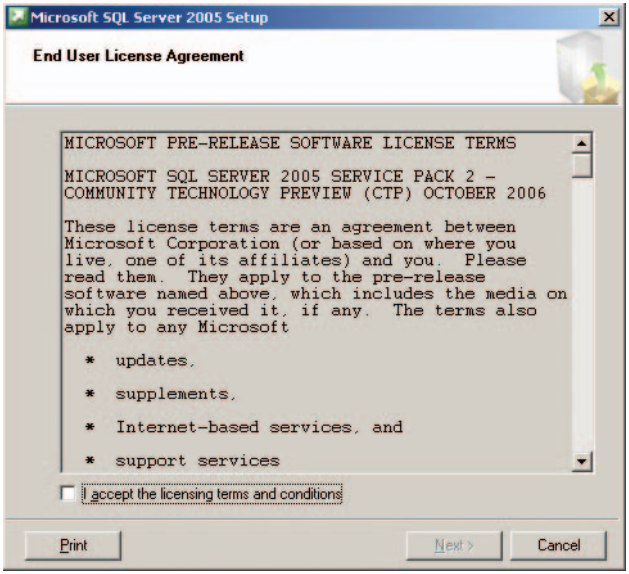



*Figure 2-1*
*License agreement*

**NOTE**

**At the time of this writing, SQL Server 2005 Service Pack 2 Express Edition is available as a Community Technology Preview and will likely be released as the official version of the product as you read this book. Depending on the current status of the product and subsequent service packs released, your screen might reflect a different version of the product than shown here.**

*Continued from page 20*

*Chapter 5, "Maintaining Your Database.") The Administrator user account is also assigned to this role whether or not you are logged in to the computer as Administrator during the installation because this account is a member of the BUILTIN\ Administrators group.*

*Windows Vista handles administrator privileges at the operating system level quite differently. You can be an administrator on the computer, but that's not going to matter. Windows Vista has a feature called User Account Control that effectively lowers your privileges to the status of a regular user. In other words, you may be set up on the computer as an administrator, but Windows Vista won't let you use your administrator powers to do things that regular users cannot. Instead, Windows Vista prompts you to "elevate" your privileges when it detects an attempt to perform an administrative task, such as installing software like SQL Server Express that*

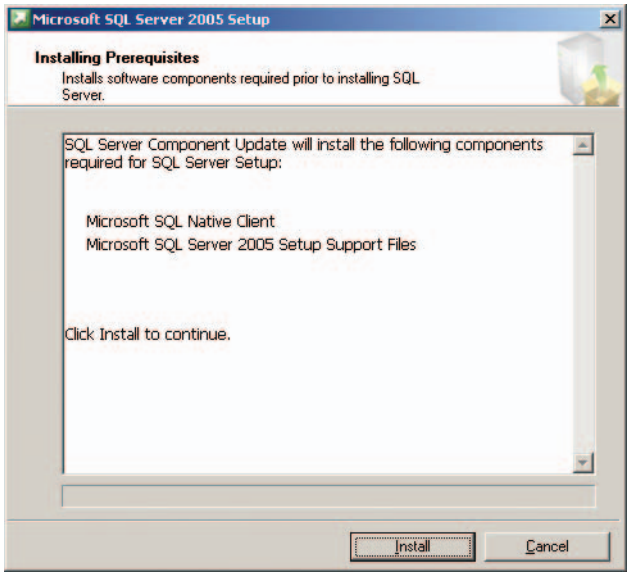
*Continued on next page*

*Continued from page 21*

*requires an administrator to do the deed. After you respond to the prompt to elevate your privileges enough for the setup program to run, you still won't be added to the SysAdmin role and you won't be able to do much. To override this behavior, you must explicitly select the Add User To The SQL Server Administrator Role option on the Configuation Options page of the setup wizard. (I'll remind you of this requirement when you reach that step of the installation.) The setup program will add your user account to the SysAdmin role and you're all set.*

*There's a catch here, though, so be careful. Let's say your own user account doesn't have administrative privileges, but your system administrator stops by to help you with the installation and types in his or her login credentials to allow installation to continue. If you select the option on this page to add the user as a SQL Server*

*Continued on next page*




**6** Next, support files for the setup process must be installed, as shown on the Installing Prerequisites page (Figure 2-2). After these files install successfully, click **Next.**

***Figure 2-2***
*Installing prerequisites*

**7** SQL Server Setup scans your computer's configuration and then displays the Welcome page (Figure 2-3). Click **Next** to continue.

***Figure 2-3***
*Welcome page*

**8** SQL Server Setup displays the results of the system configuration check which, if you've completed the preparation steps correctly, should be successful, as shown in Figure 2-4. Click **Next** to continue.

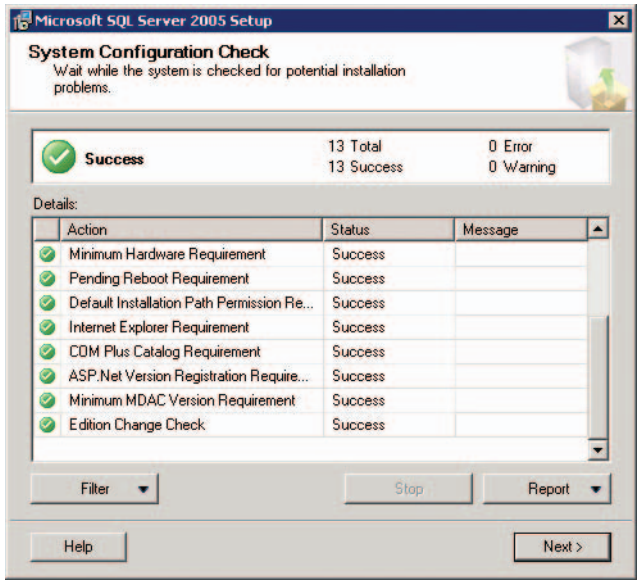



*Figure 2-4*
*System configuration check*

**9** On the Registration Information page (Figure 2-5), type in your name and, optionally, your company's name. Click **Next** to continue.

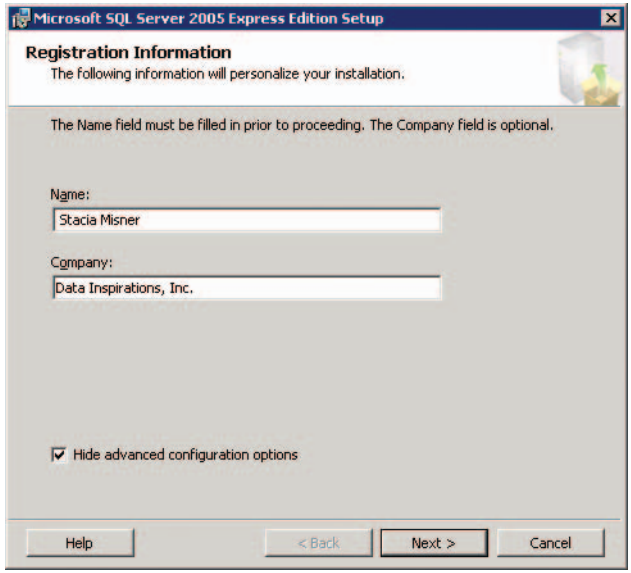



*Figure 2-5*
*Registration information*

**NOTE**

**If a problem is detected at this point, an X symbol will appear next to the requirement that failed during the system configuration check. Click the Report button to view a report that provides a list of the operating system requirements (including service pack level), the compatibility of SQL Server Express with your operating system, and other requirements. The report indicates whether each requirement passed or failed and, for failed items, includes a suggestion for correcting the problem. You can choose to view the report, save it to a file, copy it to the clipboard, or e-mail it.**

*Continued from page 22*

*Administrator, that administrator's user account, instead of your own, is added to the role. You're in the same situation you would be in if you didn't select this option—you won't be able to perform administrative tasks on SQL Server Express.*

*If you miss this step, don't panic. You can use the SQL Server Surface Area Configuration Tool to add your login as a SQL Server Express administrator.*

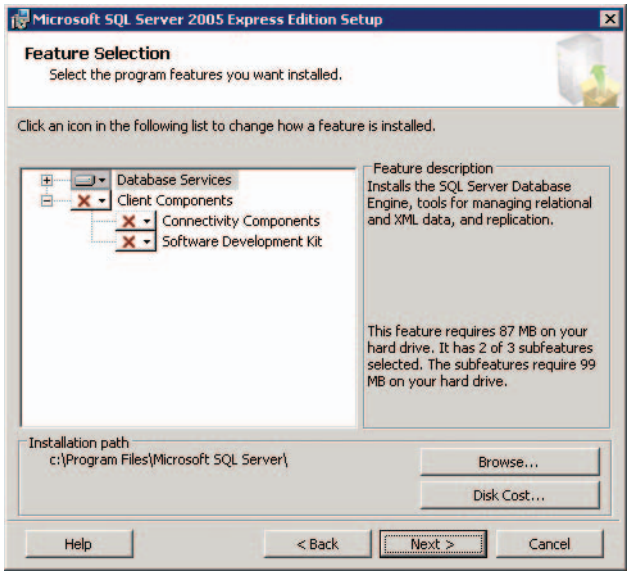
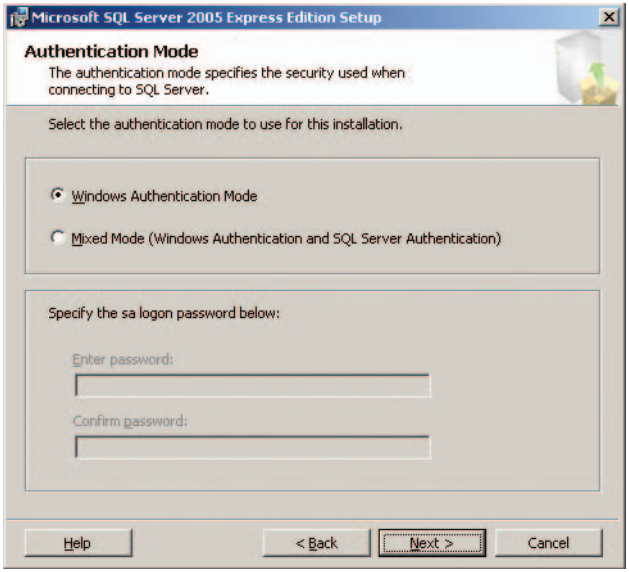



*Figure 2-6*
*Feature Selection page*




**10** The Feature Selection page allows you to select the features you want to install and the installation folder for each feature. I recommend you keep the default settings, as shown in Figure 2-6. Click **Next** to continue.

If you plan to use SQL Server Express for replication with a SQL Server database, you'll need to perform an extra step here because replication is not part of the default installation. Just expand **Database Services**, click **Replication**, and then click **Entire Feature Will Be Installed On Local Hard Drive**. You will also need to click **Client Components** and click **Entire Feature Will Be Installed On Local Drive**. I'm not going to discuss replication in this book, but I do want you to be aware of these extra steps so you can set up replication now if you know you're going to use this feature.

NOTE

**If you already have SQL Server (any edition) installed on your computer, the Instance Name page of the wizard displays instead of the Authentication Mode page as described in the next step. It's okay to have multiple instances on one computer. You just need to give each instance a unique name to create a *named instance*. A named instance runs as a completely separate data service with its own memory requirements and system databases. I recommend you keep the default instance name suggested by the wizard, SQLEXPRESS, if it's not already used by another instance on your computer. When you click Next, the Authentication Mode page displays.**

**11** On the Authentication Mode page (Figure 2-7), you select the type of security required when a user connects to SQL Server. Keep the default selection, **Windows Authentication Mode**, and click **Next** to continue.

NOTE

**It's easy to change this setting later if you change your mind. I explain the difference between Windows Authentication and Mixed Mode Authentication in Chapter 5.**

*Figure 2-7*
*Authentication Mode page*

**12** On the Configuration Options page, you configure the database engine to allow other users to work with the database engine. I recommend you keep the default settings, as shown in Figure 2-8, to work through the examples in this book. Click **Next** to continue.

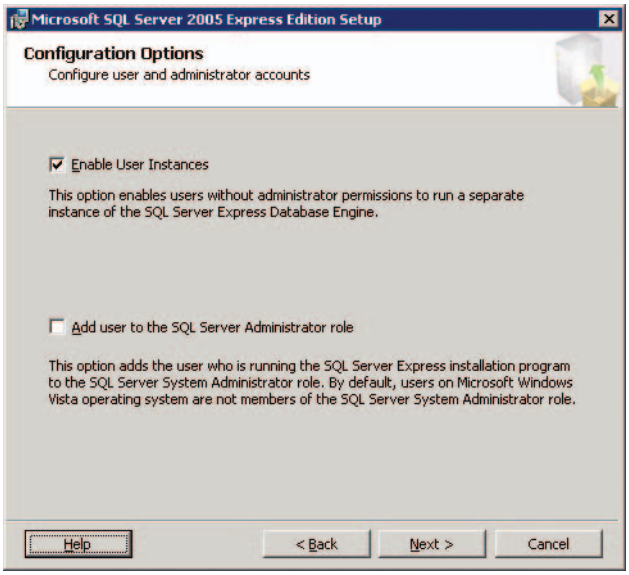



*Figure 2-8*
*Configuration options page*

**IMPORTANT**

**I'll explain all about user instances in Chapter 6, "Managing Your Server," which is the reason I suggest you keep the default selection of the option, Enable User Instances.**

**The second option, Add User To The SQL Server Administrator Role, is required only if you are installing SQL Server Express on a computer running Windows Vista. If you skipped previous pages in this chapter to jump right into the installation steps, and if your computer uses the Windows Vista operating system, turn back now to review the section titled "The SQL Server Administrator Role and Microsoft Windows Vista" earlier in this chapter for a complete explanation. If your computer runs under a different operating system, leave this check box clear because it doesn't apply to you.**

**13** In the final step before installation begins, you decide whether to send error reports and usage data to Microsoft. Make your selections on the Error And Usage Report Settings page (Figure 2-9) by selecting the applicable check boxes. Click **Next** to continue.

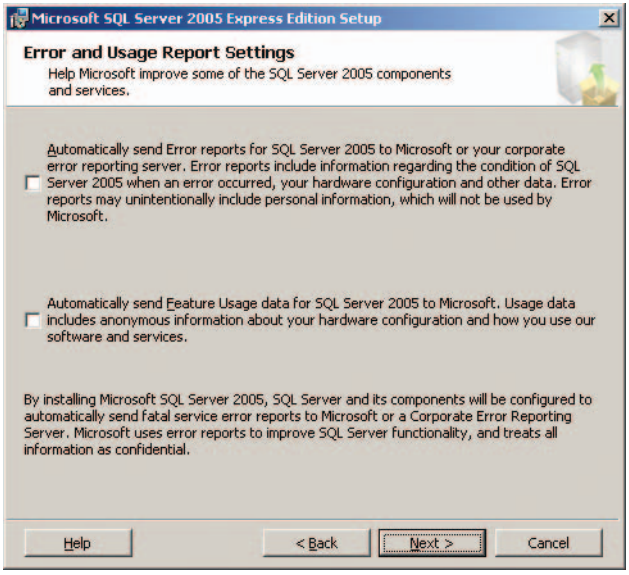



*Figure 2-9*
*Error And Usage Report Settings page*

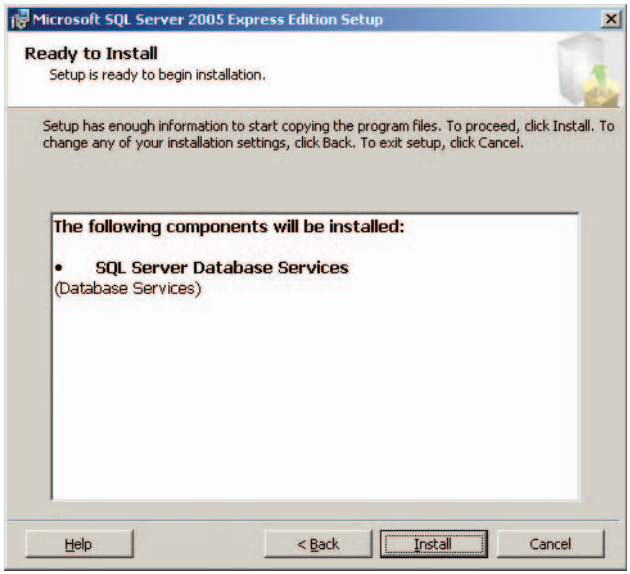



*Figure 2-10*
*Ready To Install page*

**14** At last you've arrived at the Ready To Install page (Figure 2-10). Click **Install** to start the installation.

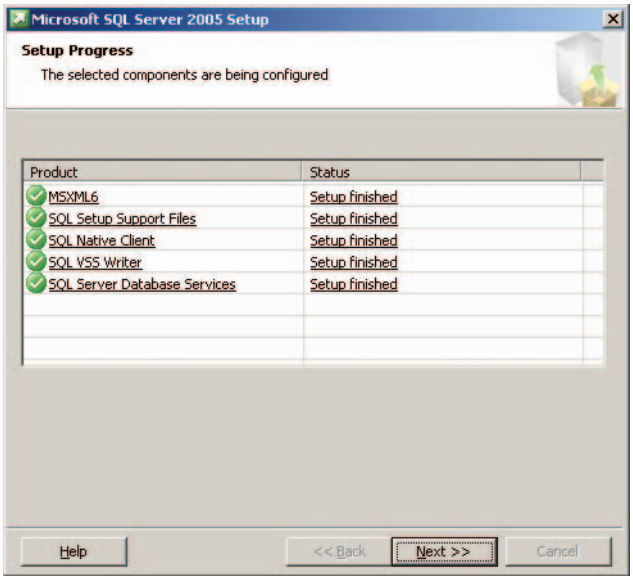



*Figure 2-11*
*Setup progress*

**15** The Setup Progress page displays the current status of the installation by component. This process takes about five minutes. When all components install successfully, as shown in Figure 2-11, click **Next**.

**16** The final page of the wizard, Completing Microsoft SQL Server 2005 Setup (Figure 2-12), provides a link to a Summary Log and provides information about the current configuration. You can change the configuration by using the Surface Area Configuration Tool, which is accessible by a link on this page. Click **Finish**.

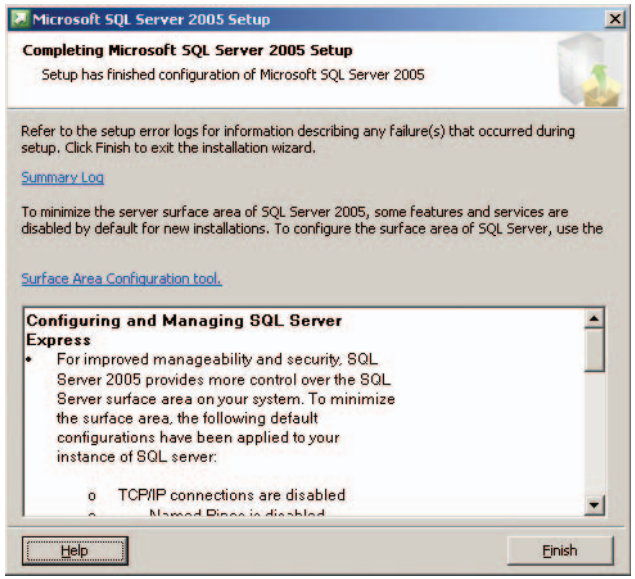



*Figure 2-12*
*Completing Microsoft SQL Server 2005 Setup page*

**NOTE**

**This isn't your only chance to access the SQL Server Surface Area Configuration Tool. Usually, the default configuration is just fine for most people. I'll explain more about this tool in Chapter 6.**

Now you're finished with the SQL Server Express setup. Microsoft SQL Server 2005 is a new program group on your computer, and it includes only another program group for Configuration Tools. Remember from Chapter 1, "Introducing Microsoft SQL Server 2005 Express Edition," that SQL Server Express is a database engine that runs as a data service. It has no interface of its very own. You could use the *sqlcmd* utility on the command line to manage the server and issue queries, but it would be a rather tedious approach to database management. Fortunately, there is SQL Server Management Studio Express to come to your rescue. In the next section, I'll show you how to install it.

# Installing SQL Server Management Studio Express

If you were able to successfully install SQL Server Express, you should have no problem installing SQL Server Management Studio Express. There are fewer decisions to make and fewer steps to perform. To install SQL Server Management Studio Express, follow the steps below.

## TO INSTALL SQL SERVER MANAGEMENT STUDIO EXPRESS

1. Click the **Start** button and then click **My Computer**.

2. Right-click the drive to which you downloaded the software and select **Explore**.

3. In the Folders list, select the folder in which you saved the software.

4. In the list of files, double-click **SQLServer2005_SSMSEE.msi** and then click **Run**.

5. Before you can install the product, you must read and accept the license agreement (Figure 2-13). Read the agreement carefully to understand how you can use this product. To continue the installation, select **I Accept The Terms In The Licensing Agreement** and click **Next**.

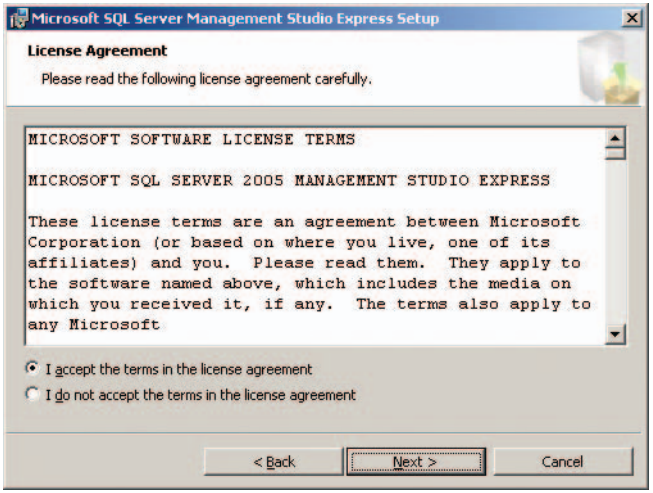



*Figure 2-13*
*License agreement page*

6 Next, on the Registration Information page (Figure 2-14), you must type in your name to continue. If you want, you can also type in your company's name, but it's not required. Click **Next** to continue.

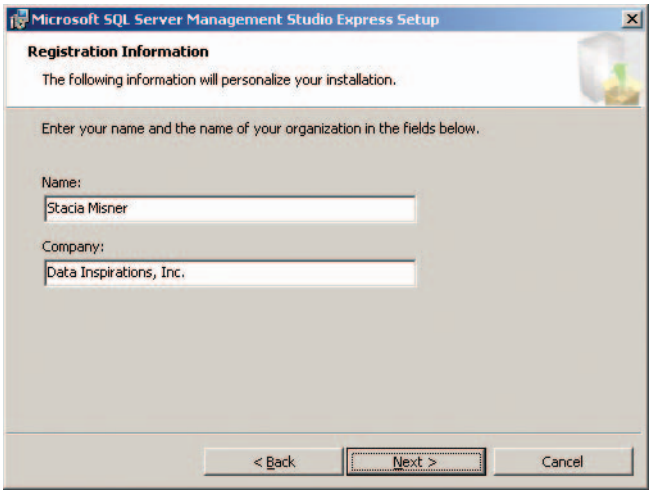



*Figure 2-14*
*Registration information page*

7 On the Feature Selection page, you can change the installation folder, but I suggest you keep the default folder, as shown in Figure 2-15. Click **Next** to continue.

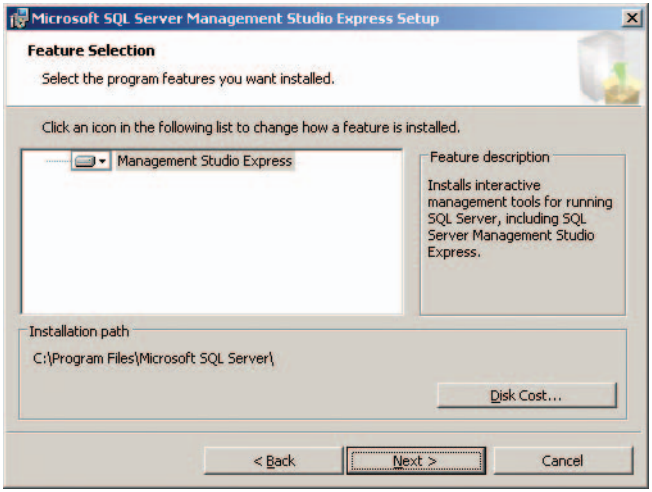



*Figure 2-15*
*Feature Selection page*

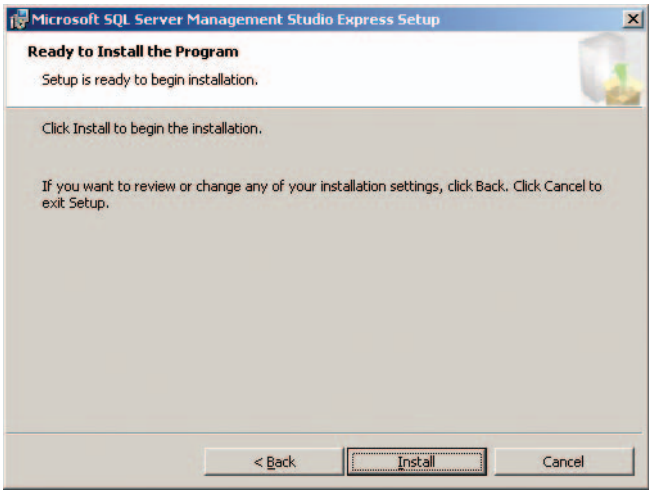



*Figure 2-16*
*Ready To Install The Program page*

**8** On the Ready To Install The Program page (Figure 2-16), click **Install** to start the installation.

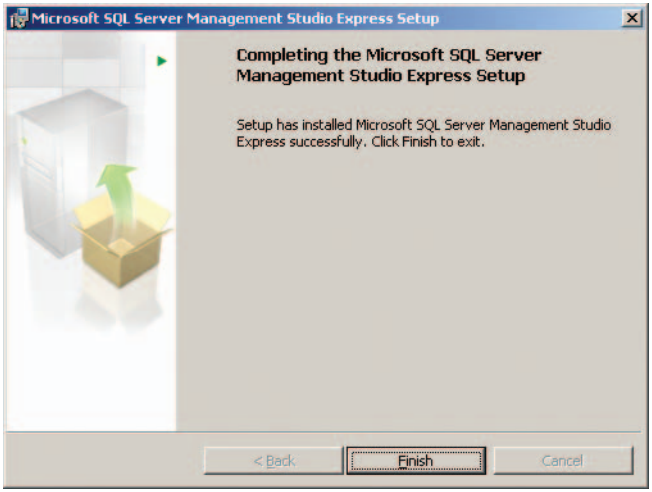



*Figure 2-17*
*Completing setup*

**9** After a few minutes, the final page of the wizard, shown in Figure 2-17, indicates the installation completed successfully. When you click **Finish**, you're all set to start building your first database!

Now the Microsoft SQL Server 2005 program group contains SQL Server Management Studio Express. You'll learn how to use it in Chapter 3, "Creating Your First Database."

# Going a Few Steps Farther

You have everything you need now to work through the exercises in this book. However, you can do a few other things to supplement your SQL Server Express environment that I'll explain in this section. You may want to review a sample database available for SQL Server, access documentation about SQL Server on your computer when you don't have Internet access, or configure SQL Server Express for sharing databases across a network. If so, read on and I'll show you how to do these things. Otherwise, skip ahead to the next chapter, where I'll get you started with your newly installed software.

## Downloading the AdventureWorks Sample Database

The database you'll be building in the next two chapters is very simple compared to databases you'll find in a business, although it has just enough characteristics to help you learn how SQL Server Express works. Microsoft has developed the AdventureWorks sample database as an example of a database structure commonly found in business and frequently references this database in the code examples found in SQL Server 2005 Books Online, the official documentation for the SQL Server family. If you download and install the sample database, you can better follow those code examples. In addition, by studying the structure of AdventureWorks after you finish this book, you'll increase your understanding of database design principles. To obtain and install it, follow the steps below.

### TO DOWNLOAD AND INSTALL THE ADVENTUREWORKS DATABASE

1. Using your browser, go to *go.microsoft.com/fwlink/?LinkID=58560*.
2. Click the link **AdventureWorksDB.msi** in the x86 section of the page.
3. Perform one of the following steps:
   - Click **Run** to install the database right now.
   - Click **Save** to download the software to a folder of your choice to install at your convenience later. You'll need to double-click **AdventureWorksDB.msi** in the folder in which you saved the file to continue with the next step when you're ready.

**NOTE**

**Several other samples are available for download on this page, but some do not work with SQL Server Express. If you're curious about the other samples, download the SQLServerDatabasesAndSamples Overview.htm file on this page for more information.**

*Figure 2-18*
*AdventureWorksDB – InstallShield Wizard*

4 Click **Run** to install. The AdventureWorksDB – InstallShield Wizard displays, as shown in Figure 2-18. Click **Next**.

5 On the License Agreement page, read the license agreement, select **I Accept The Terms In The License Agreement**, as shown in Figure 2-19, and click **Next**.

6 On the Destination Folder page (Figure 2-20), change the default location if you wish (although I don't recommend it) and click **Next**.

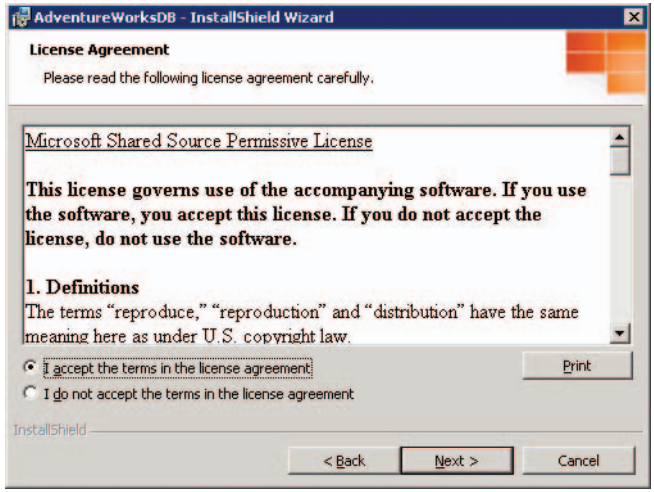



*Figure 2-19*
*License agreement*

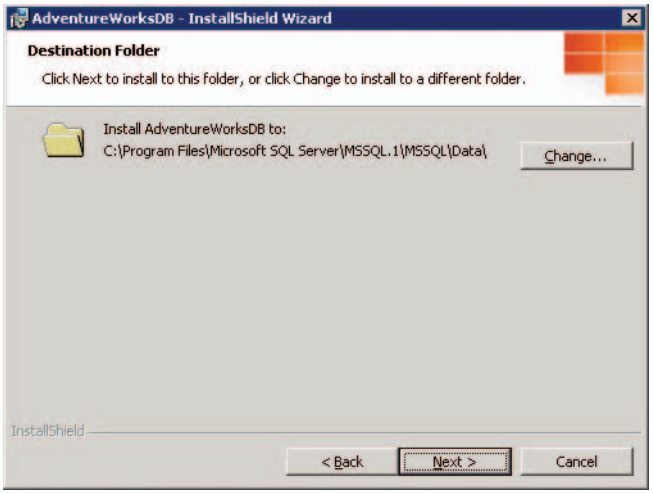



*Figure 2-20*
*Destination Folder page*

7 On the Ready To Install The Program page (Figure 2-21), click **Install**.

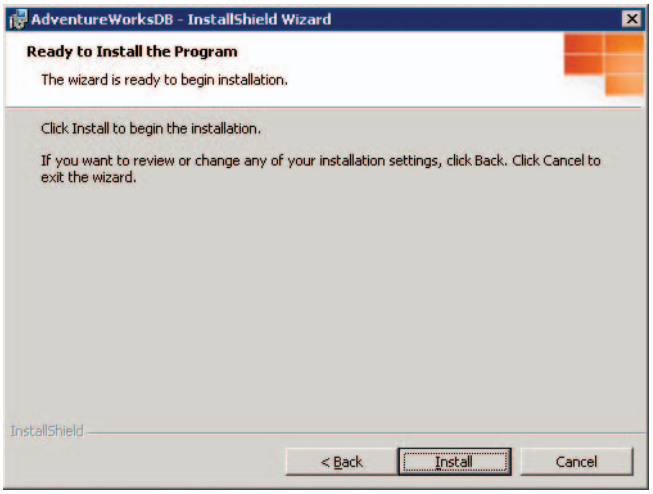



*Figure 2-21*
*Ready To Install The Program page*

8 On the InstallShield Wizard Completed page, shown in Figure 2-22, click **Finish**. Later, in Chapter 3, I'll show you how to take a look at the contents of this database using SQL Server Management Studio Express.

**TIP**

**You might find it easier to explore this database with a little guidance. You can download a diagram of the AdventureWorks database from *www.microsoft.com/downloads/details.aspx?familyid=0F6E0BCF-A1B5-4760-8D79-67970F93D5FF&displaylang=en* in either HTML format or Microsoft Visio format.**

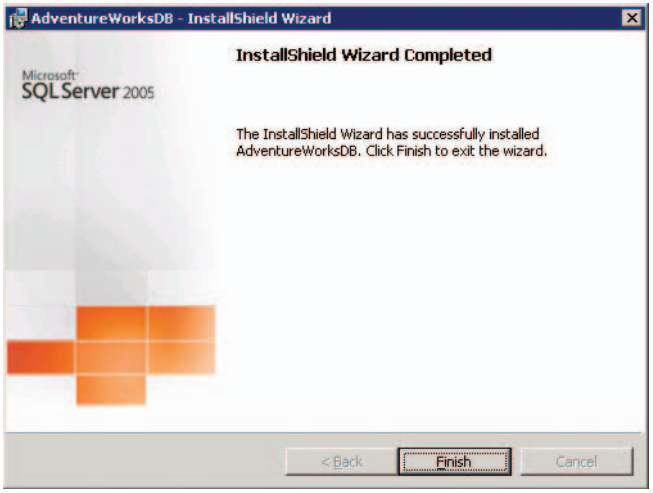



*Figure 2-22*
*InstallShield Wizard Completed page*

## Installing a Local Copy of SQL

## Server Books Online

Many Microsoft products include documentation. In these products, when you need to look up some information to learn how to do something, you can press F1 to get help for the item currently selected in the window, or you can use the Help menu to find the documentation. This is not the case with SQL Server Express or Management Studio Express, for that matter. You can always get the most current documentation online at MSDN (as explained in Chapter 1); for those times you're not connected to the Internet, you can download Books Online. This version of Books Online covers all editions of SQL Server 2005. To download and install Books Online, follow the steps below.

### TO DOWNLOAD AND INSTALL SQL SERVER BOOKS ONLINE

1. Using your browser, go to *go.microsoft.com/fwlink/?LinkID=52287*.

2. On the right side of the page, in the Download Books Online From The Download Center drop-down list, select the language of your choice, and click **Go**.

3. On the SQL Server 2005 Books Online page, click **Download**.

4. Perform one of the following steps:
   - Click **Run** to install Books Online right now.
   - Click **Save** to download the documentation to a folder of your choice to install at your convenience later. You'll need to double-click **SqlServer2K5_BOL_Jul2006.msi** in the folder in which you saved the file to continue with the next step when you're ready.

NOTE

**This MSI file is the most current file at the time of this writing. The file you download may be more current, and therefore the name may differ slightly.**

5 The Microsoft SQL Server 2005 Books Online Setup Wizard displays. You'll have to wait a minute or two as it prepares for installation. When the Next button is enabled, as shown in Figure 2-23, click **Next**.

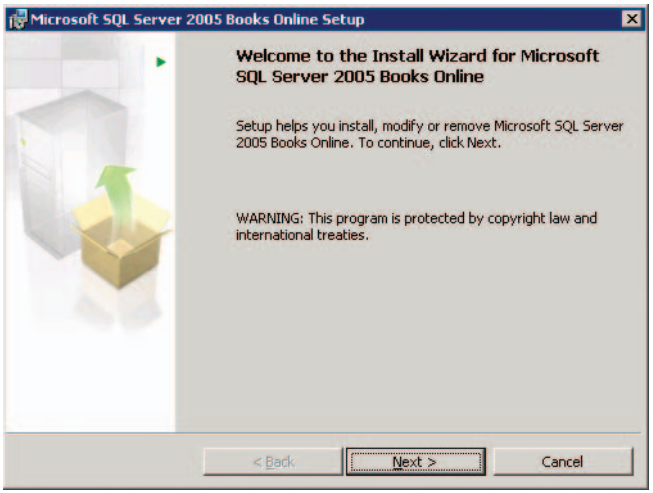



*Figure 2-23*
*Microsoft SQL Server 2005 Books Online Setup Wizard*

6 On the License Agreement page, read the license agreement, select **I accept the terms in the license agreement**, as shown in Figure 2-24, and click **Next**.

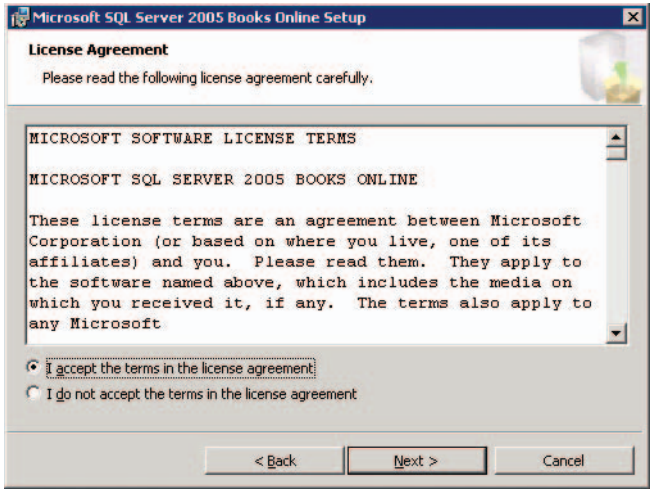



*Figure 2-24*
*License agreement page*

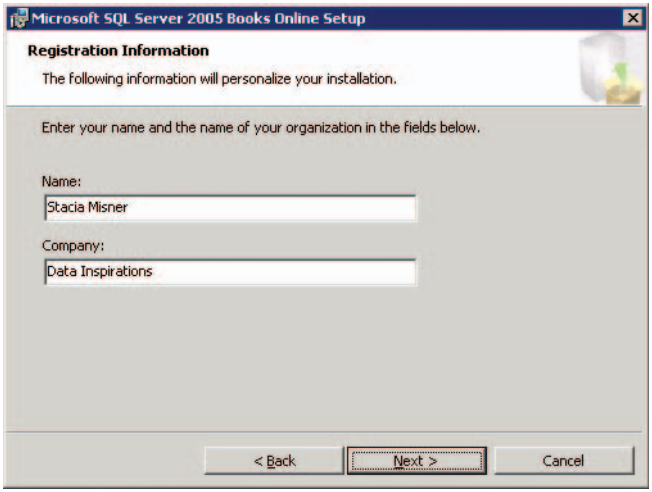



*Figure 2-25*
*Registration Information page*

**7** On the Registration Information page (Figure 2-25), type in your name and, optionally, your company's name. Click **Next** to continue.

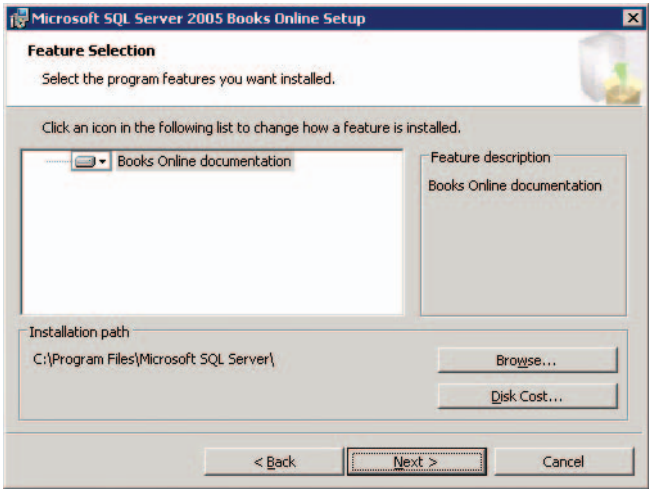



*Figure 2-26*
*Feature Selection page*

**8** Keep the default selection on the Feature Selection page, shown in Figure 2-26. You can change the installation folder here, but I recommend you keep the default setting. Click **Next** to continue.

9 On the Ready To Install The Program page (Figure 2-27), click **Install**.

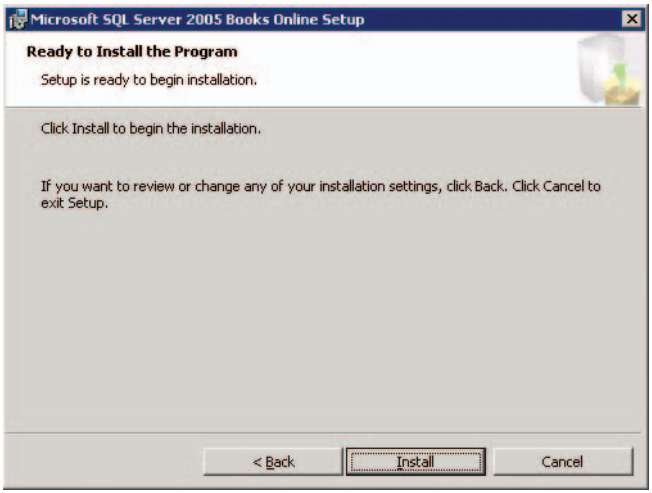



*Figure 2-27*
*Ready To Install The Program page*

10 Each step of the installation displays in the wizard. When installation is complete, as shown in Figure 2-28, click **Finish**.

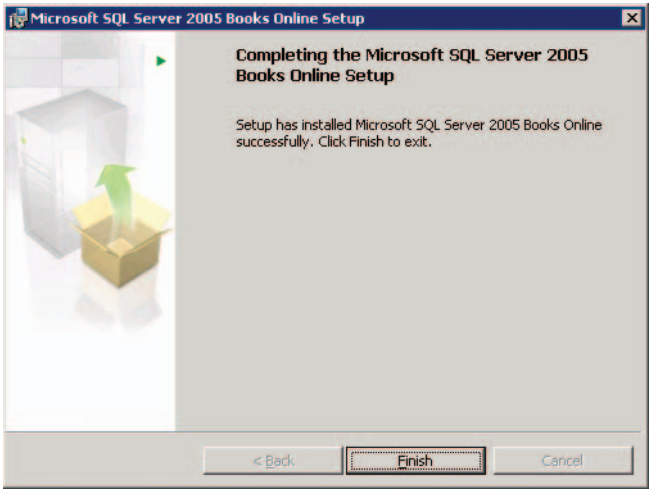



*Figure 2-28*
*Completing The Microsoft SQL Server 2005 Books Online Setup page*

To make it easier for you to find information applicable to SQL Server Express, you can use the SQL Server Express filter. To apply this filter, follow the steps below.

## TO APPLY THE SQL SERVER EXPRESS FILTER TO SQL SERVER BOOKS ONLINE

1. Click **Start**, point to **All Programs**, point to **Microsoft SQL Server 2005**, point to **Documentation And Tutorials**, and click **SQL Server Books Online**.

   If you are connected to the Internet when you start Books, the program connects to Books Online at the MSDN Web site. However, as you select topics to view, the content displays from your local copy.

2. In the Filtered By drop-down list, select **SQL Server Express**. Books Online contents are now filtered to display topics available specifically for SQL Server Express, as shown in Figure 2-29.

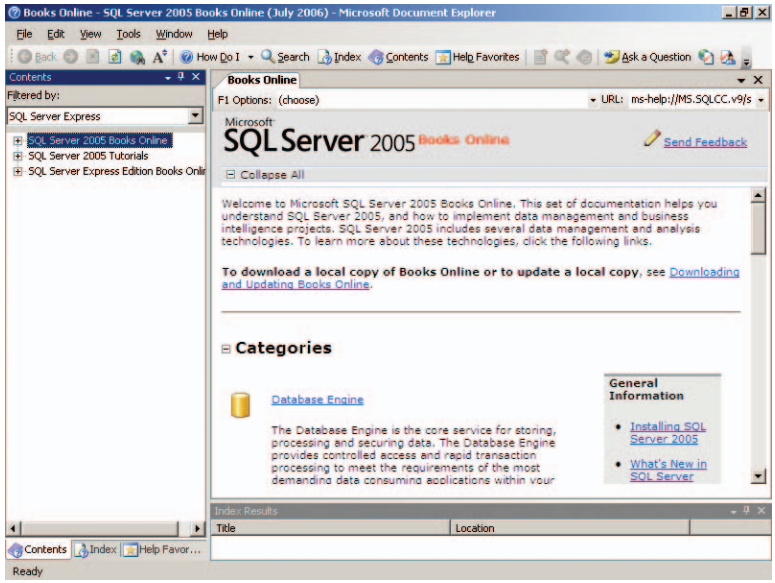



*Figure 2-29*
*SQL Server Books Online filtered by SQL Server Express*

3 You can find information by expanding topics in the Contents window and browsing around until you find what you need. A more direct method is to click the **Index** tab at the bottom of the Contents window, type in a keyword or phrase in the **Look For** box, such as **creating databases**, and press **Enter**. The main window displays the topic, as shown in Figure 2-30.

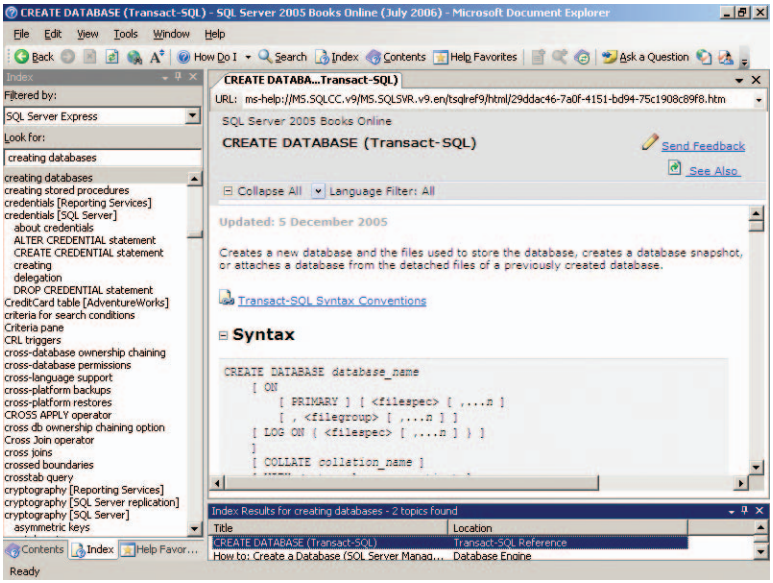



*Figure 2-30*
*SQL Server Books Online displaying a topic and index results*

If multiple topics match your criteria, the Index Results box displays at the bottom of the page. You can click on a different topic in the list of results to change the contents of the main window.

## Configuring SQL Server Express for Remote Access

By default, SQL Server Express installs for local use only. That means the people who have login access to your computer are the only ones who can use SQL Express, assuming you have given them the correct privileges, which I discuss in Chapter 5. Even if you are the only one who needs to connect to SQL Server Express from a different computer, you still need to take some extra steps to enable remote access. To configure SQL Server Express correctly for remote usage, follow the steps below.

### TO CONFIGURE SQL SERVER EXPRESS FOR REMOTE ACCESS

1. Click the **Start** button, point to **All Programs**, point to **Microsoft SQL Server 2005**, point to **Configuration Tools**, and select **SQL Server Configuration Manager**. The SQL Server Configuration Manager window displays, as shown in Figure 2-31.

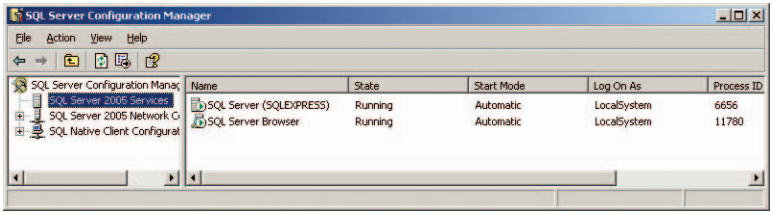



*Figure 2-31*
*SQL Server Configuration Manager*

2. Expand **SQL Server 2005 Network Configuration** and then click **Protocols For SQLEXPRESS**, as shown in Figure 2-32. Notice the TCP/IP protocol is disabled.

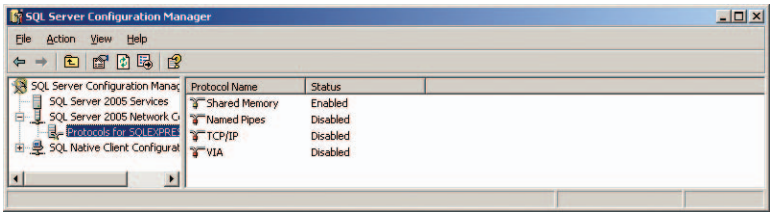



*Figure 2-32*
*Protocols for SQLEXPRESS*

3. Right-click **TCP/IP** and select **Enable**. A warning message (Figure 2-33) displays to let you know the change will go into effect only when you restart the service. Click **OK** to continue.

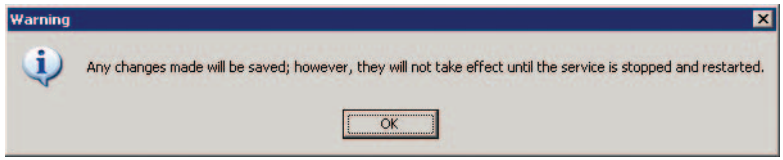
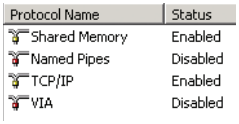



*Figure 2-33*
*Warning message*

The status of the TCP/IP changes to show the protocol is now enabled, as shown in Figure 2-34. But you still need to restart the service, so let's do that next.



| Protocol Name | Status |
|---|---|
| Shared Memory | Enabled |
| Named Pipes | Disabled |
| TCP/IP | Enabled |
| VIA | Disabled |

*Figure 2-34*
*TCP/IP enabled*

4 In the list on the left side of the window, click **SQL Server 2005 Services**. Right-click SQL Server (SQLEXPRESS) and select **Restart**. A message box (Figure 2-35) shows the progress while SQL Server Configuration Manager attempts to stop and then restart the service.

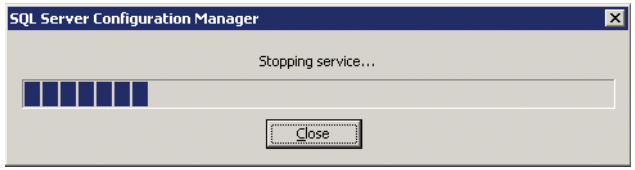



*Figure 2-35*
*Stopping the SQL Server (SQLEXPRESS) service*

When the service starts successfully, you'll see a green arrow icon on the service to show it's running and the State column displays Running, as shown in Figure 2-36.

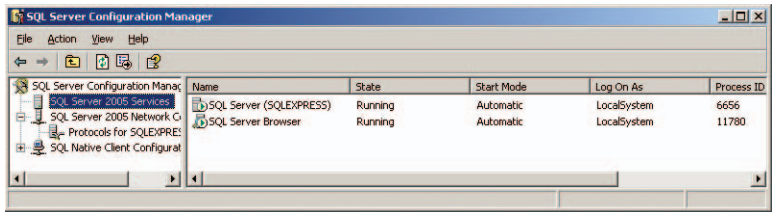



*Figure 2-36*
*SQL Server (SQLEXPRESS) service is started*

5 On the **File** menu, select **Exit** to close SQL Server Configuration Manager.

**IMPORTANT**

**If you are using third-party hardware or software firewalls, please skip the remaining steps and refer to that product's documentation for instructions to configure your firewall to allow exceptions for SQL Server Express and for SQL Server Browser.**

6. Next, you'll need to configure Windows Firewall to permit requests to use SQL Server Express through the firewall. Click **Start**, point to **Control Panel**, and double-click **Windows Firewall**.

7. In the Network Tasks box, click **Change Windows Firewall Settings**.

8. In the Windows Firewall dialog box, click the **Exceptions** tab and review the list of programs and services. Then do one of the following steps:

- If SQL Server is in the list, make sure the check box to the left of SQL Server is selected. If it isn't, go ahead and select it.
- If SQL Server is not on the list, click **Add Program**, click **Browse**, and navigate to the folder you specified during the SQL Server Express installation. By default, this folder is C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\BINN. When you open the folder, double-click **sqlservr.exe**. In the Add A Program dialog box, with **sqlservr.exe** highlighted, click **OK.**

9. Next, you need to repeat this process to create an exception for SQL Server Browser. Review the list of programs and services, and then do one of the following steps:

- If SQL Server Browser is in the list, make sure the check box to its left is selected. If it isn't, go ahead and select it.
- If SQL Server Browser is not on the list, click **Add Program**, click **Browse,** and navigate to the folder you specified during the SQL Server Express installation. By default, this folder is C:\Program Files\Microsoft SQL Server\90\Shared. When you open the folder, double-click sqlbrowser.exe. In the Add A Program dialog box, with **sqlbrowser.exe** highlighted, click ***OK.***

10. Click **OK** again to close the Windows Firewall dialog box.

## In Summary...

By completing this chapter, you have both SQL Server 2005 Express Edition and SQL Server Management Studio Express installed. Although you weren't required to do so, you may have even downloaded and installed the sample database or Books Online, or configured SQL Server Express for remote users. At this point, there is nothing left to do but move on to the next chapter and start building your database.

Chapter 3

# Creating Your First Database

*Getting Started with SQL Server Express, 44*

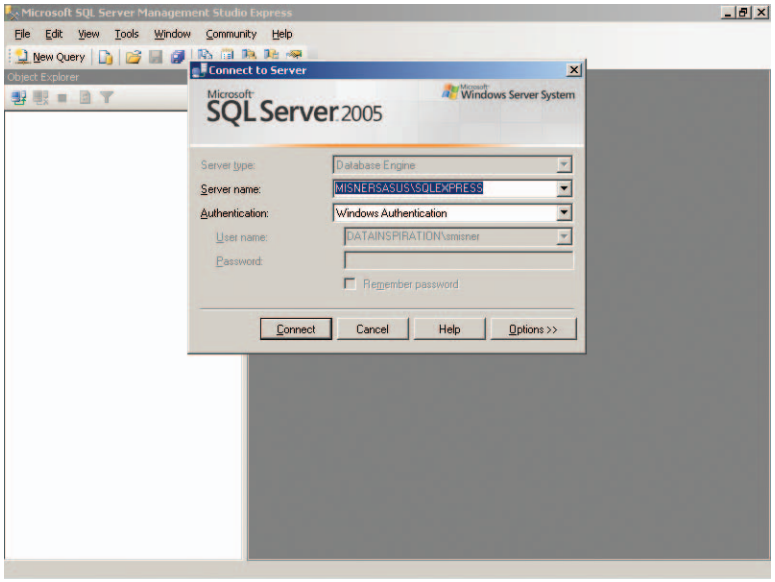
*Creating the Home Inventory Database, 57*

SQL Server Management Studio Express is the tool you use to create and maintain your database as you work through the exercises in this book. Now that you have it installed, it's time to learn how it works in general by exploring its interface. Once you're comfortable using this program to explore existing databases, I'll show you how easy it is to quickly and easily create your first database and how to add some data to it.

There are several ways to interact with SQL Server Express. If you know a programming language, you can write an application to send Transact-SQL commands to the database engine. You can also write and execute Transact-SQL scripts in SQL Server Management Studio Express, which I'll teach you in Chapter 4, "Working with Transact-SQL." However, the simplest way to interact with SQL Server Express without writing any code is to use the menus and designer windows in SQL Server Management Studio Express, as you'll learn in this section.

## Exploring SQL Server Management Studio Express

Before you can start building a database and adding tables it, you should get familiar with the main features of SQL Server Management Studio Express, which I'll refer to as Management Studio from now on. If you ever decide to upgrade to a different version of SQL Server 2005, you'll already know how to use its management interface, known as SQL Server Management Studio, and you won't have to spend time learning a brand new tool. To help you get comfortable with Management Studio, I'll show you how to look at the system databases that install automatically when you install SQL Server Express. To explore the main features of Management Studio, follow the steps below.




### TO EXPLORE MANAGEMENT STUDIO

1. Click **Start**, point to **All Programs**, point to **Microsoft SQL Server 2005**, and click **SQL Server Management Studio Express**. The first time you start this program, you'll see a message telling you the environment needs to be configured. This process might take a few minutes. When it completes, the Connect To Server dialog box displays, as shown in Figure 3-1.

*Figure 3-1*
*Connect To Server dialog box*

2 In the Connect To Server dialog box, the default values for the server name and authentication are already provided, so you can simply click **Connect** to continue to the main window in Management Studio, shown in Figure 3-2.

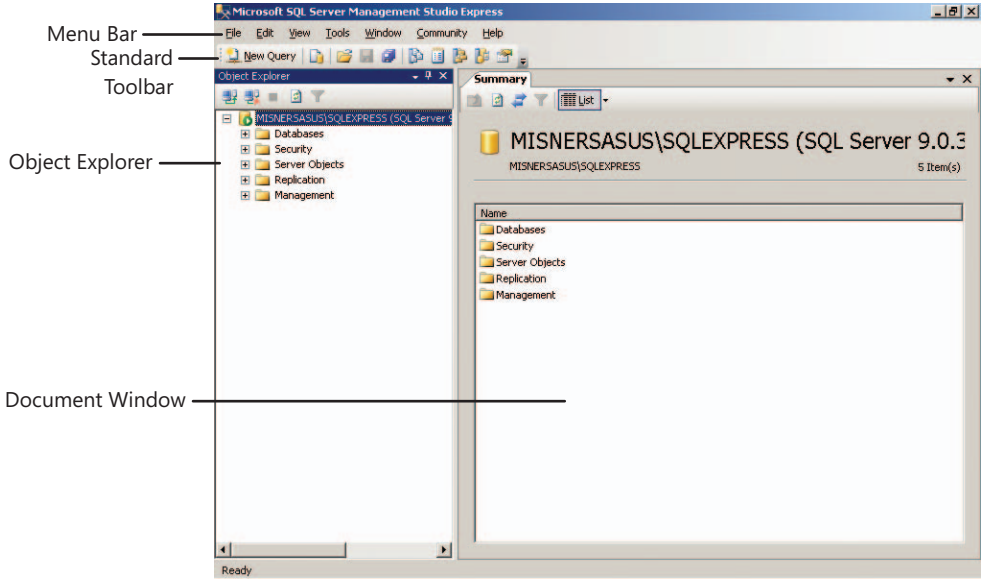



*Figure 3-2*
*SQL Server Management Studio Express main window*

**NOTE**

**Because SQL Server Express installs as a named instance, SQLEXPRESS, when you connect to the server using any program, including Management Studio, you must combine the name of your computer with \SQLEXPRESS. The Connect To Server dialog box requires you to pick an instance, even if you have only one instance on your computer. If you're on a network, you can also connect to other instances running on a different computer if you have been given permission to do so. Management Studio lets you access all servers from one convenient spot.**

Let's take a moment to get familiar with the layout of Management Studio when the program first opens after connecting to SQL Server Express.

**Menu bar** Just like any other Windows program, a menu bar is available for you to select commands that help you interact with SQL Express. The specific set of commands available at any given point depends on which object you currently have selected in one of the windows. These are not the only commands you'll be using, however, because each object has its own context menu that displays when you right-click the object.

**Standard toolbar** You can use the toolbar buttons as an alternative to selecting commands from the menu bar or the context menus. Point to a toolbar to display the command for the button, as shown in Figure 3-3.

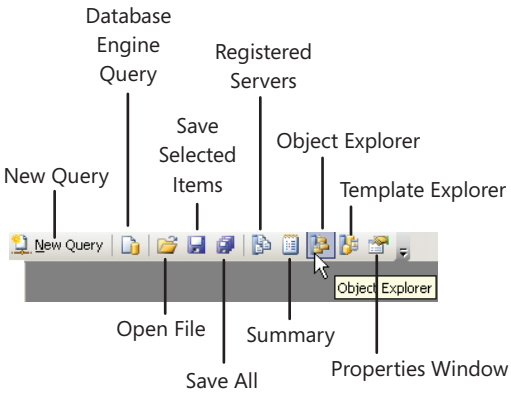



*Figure 3-3*
*Management Studio's standard toolbar*

**Object Explorer** This window displays all of the database objects managed by the SQL Server Express instance organized by type: Databases, Security, Server Objects, Replication, and Management. Notice that Object Explorer has its own toolbar (Figure 3-4) to connect to other instances, to disconnect from an instance, to stop the display of folder contents if it's taking too long, to refresh the contents of the window, and to filter a long list of folder contents to include only those objects you want to see. You'll spend a lot of time working with Object Explorer as you build, secure, manage, and monitor your database.

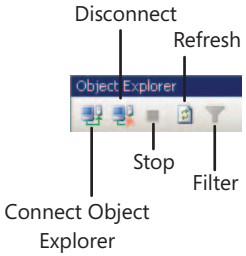



*Figure 3-4*
*Object Explorer's toolbar*

**Document window** By default, the Document window appears to the right of Object Explorer and displays the Summary window when Management Studio opens. When you click an object in Object Explorer, summary information about that object displays in the Summary window. While you work, you can add more windows to the Document window to browse tables, write queries, or work with an object's properties.

**3** In Object Explorer, click the plus sign (+) to expand the Databases folder and then expand the System Databases folder to view a list of databases installed with SQL Server Express, as shown in Figure 3-5.

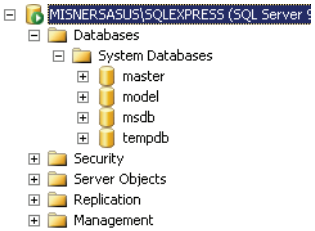



***Figure 3-5***
*System databases*

System databases are used by SQL Server Express to keep track of information it needs to manage certain tasks, such as starting up the server, keeping operations running smoothly, and making temporary space available for users and applications. Here's a brief explanation of each one.

***master*** The *master* database stores all the information SQL Server Express needs to run the server, which makes it pretty important. If you accidentally delete this database, SQL Server Express can't function. In addition to configuration information used to initialize the server, this database contains information about login accounts, linked servers, and database file locations.

***model*** The *model* database is used as a starting point for all new databases. That is, if you add a table to the *model* database and then create a new database, that table will be added automatically to your new database. You aren't limited to tables, though. You can also add other objects to the *model* database, such as stored procedures, or set properties, such as database options or permissions. Even if you plan never to use this feature, you need to keep this database on your server because it's used each time the server starts to create the *tempdb* database.

***msdb*** The *msdb* database stores historical information about backup and restore activities and other server management information required by SQL Server Express.

***tempdb*** The *tempdb* database is used by all users of the same SQL Server Express instance to store objects temporarily, such as tables or variables created only for a particular operation. When the SQL Server Express service restarts, a new copy of *tempdb* is created, so your applications don't have to worry about deleting these temporary objects. SQL Server Express also uses this database when it needs to perform operations like sorting or tracking row versions when data is modified.

SQL Server Express takes care of managing these databases, so you don't really need to worry about what's in them right now. However, they are important to the proper functioning of your database server, so you should be sure to perform regular backups of these databases (with the exception of *tempdb*), which I'll explain how to do in Chapter 5, "Maintaining Your Database." For now, let's continue getting familiar with Management Studio by taking a peek inside the *master* database.

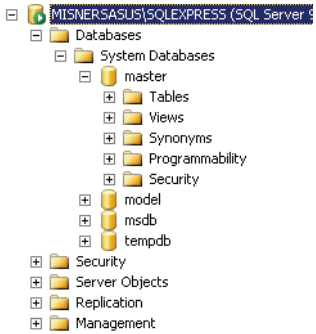



*Figure 3-6*
*Database folders*

**4** In Object Explorer, expand the ***master*** folder. All databases contain the same set of folders shown in Figure 3-6 whether or not you create objects for these folders (except ***tempdb***, which includes an additional folder for Temporary Tables).

**5** In Object Explorer, expand the Views folder and then click, but don't expand, the System Views folder. The Summary page now shows the list of ***views*** found in the System Views folder, as shown in Figure 3-7.

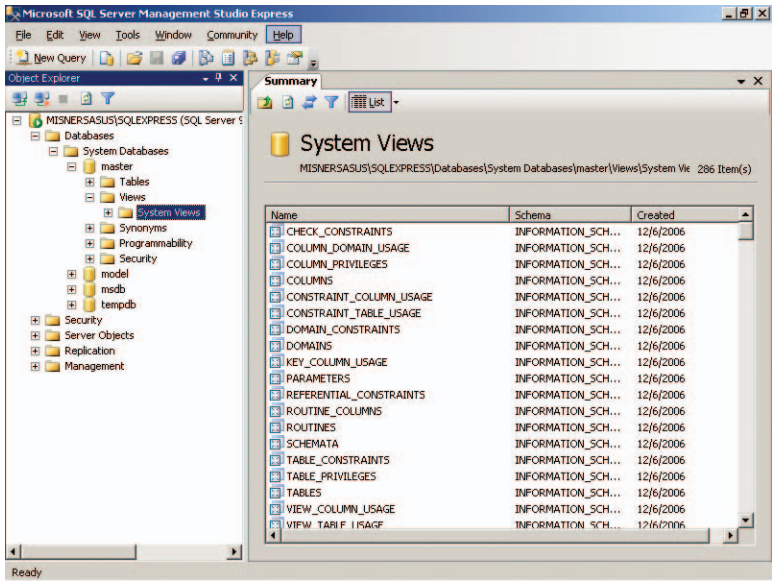



*Figure 3-7*
*System views in the master database*

A view looks like a table because it shows data in rows and columns, but it is really a virtual representation of a table and doesn't physically contain data. You'll learn more about views in Chapter 4.

You can click the column headings to change the sort order of the listed views. Alternatively, you can click the **List** button in the Summary toolbar and select **List** to switch to a simpler display that shows only the name of the views. To switch back to the display that includes the name, schema, and create date of each view, click the **List** button again and select **Details**.

6 Now, go ahead and expand the System Views folder. Use the scroll bar in Object Explorer to locate the *sys.databases* view. Expand this view and click the **Columns** folder to view the columns defined for this view, as shown in Figure 3-8.

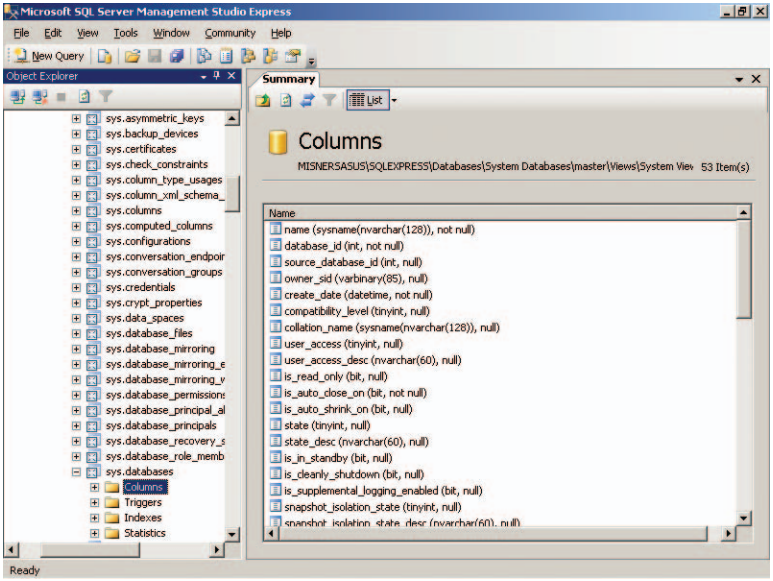



***Figure 3-8***
*Columns in sys.databases view*

In addition to the column name, the Summary window displays the data type and nullability of each column. When you build your own database, you'll be able to explore similar ***metadata*** about your database objects. Metadata is commonly known as data about data. Right now, you're viewing metadata about the ***sys.databases*** columns, while earlier you were viewing metadata about the database's views. Next, let's see how to display the data in a system view, which is a little different than the method you'll use to display data in your own database tables or views.

7 In the standard toolbar, click **New Query** (the first button). The New Query Editor window is added to the Document window. Type in the following query, as shown in Figure 3-9:

```
select * from sys.databases;
```

All you have to do to see the view data is execute the query, which you'll do in the next step.

**TIP**

**You can access each window in the Document window by clicking the applicable tab. Sometimes you'll find it helpful to keep multiple windows open. You'll use some windows for browsing metadata and other windows for writing queries. Just click the X in the upper right corner of the Document window when you're ready to close the window currently in view. If you open a lot of windows, you might not be able to see all of the tabs. In that case, just click the arrow to the left of the X in the window's corner. You'll be able to select a new window in the list that displays.**

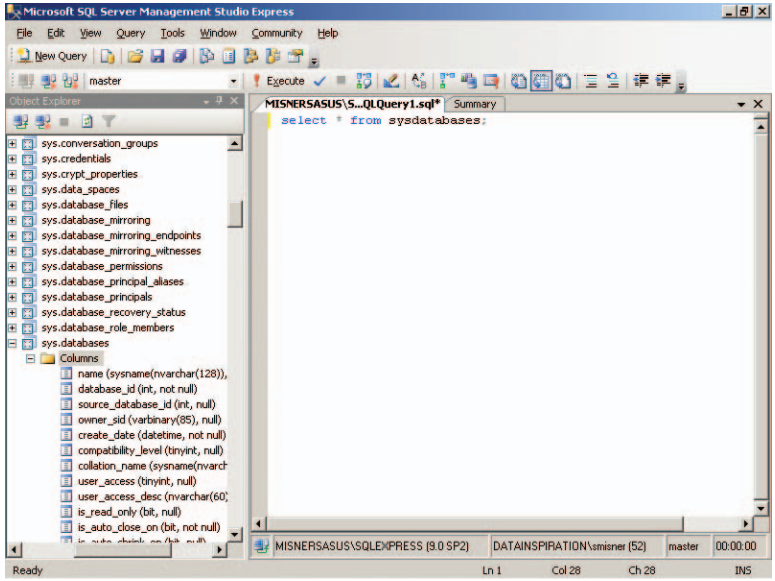
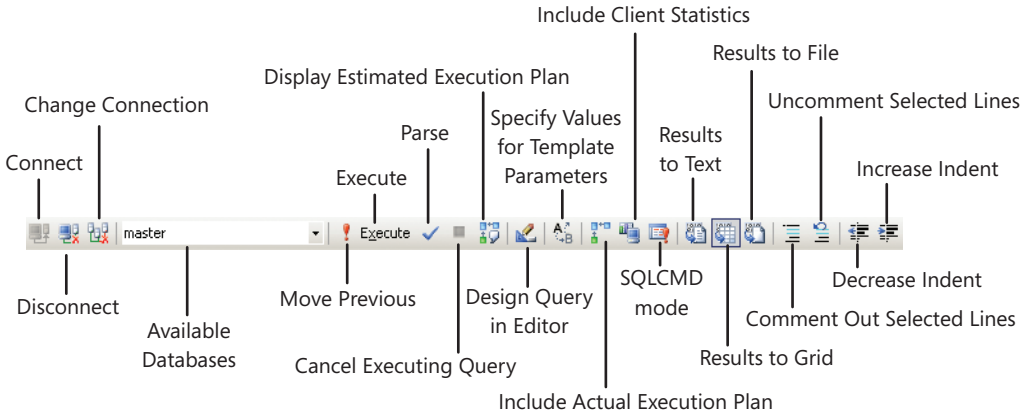



*Figure 3-9*
*New Query Editor window with a Transact-SQL statement to display sys.databases data*

You use the New Query Editor window to type in a query. You'll learn more about working with queries in Chapter 4, so don't worry about the details of this query now. Your objective at this point is simply to get some experience using Management Studio.

Notice the SQL Editor toolbar (Figure 3-10) that displays when you open the New Query Editor window. As with other toolbars, just point to a button to see the tool tip explaining what command the button carries out.

*Figure 3-10*
*SQL Editor toolbar*




**8** On the SQL Editor toolbar, click **Execute** to view the query results, as shown in Figure 3-11. When you execute a query, the data service receives the request for data, locates the database and tables required to answer the query, finds the rows that match the criteria specified in the query, and sends the results back to the requesting application—Management Studio, in this case. Actually, there's a lot more to the process than I described, but that explanation should suffice for now.

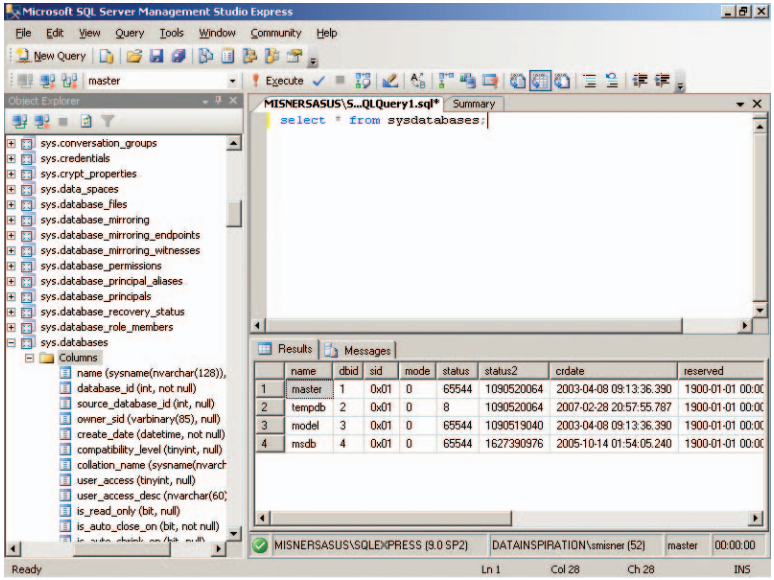



*Figure 3-11*
*Query results*

There's a more straightforward way to view data in your own databases, which you'll learn later in this chapter. System tables, however, cannot be opened quite the same way as tables you create, so you have to query the system tables when you want to view their contents. The column names are included in the Results pane, shown at the bottom of the Query Editor window, and then each row of data is listed. The Results pane includes a number for each row, which isn't actually stored in the table. It's included simply to make it easier for you to keep track of particular rows as you scan the data.

At the bottom of the Results pane, all the way to the right, you can see the total number of rows in the query results, which in this case is four rows. To the left of this number is the amount of time the query required to execute in hours, minutes, and seconds (HH:MM:SS). On my machine, this query executed extremely fast—under one second! But then, four rows is not a lot to ask from a database.

Queries that are more complex and return many more rows typically take longer. One factor that could potentially impact query performance is the number of client applications performing operations on the same SQL Serve Express instance. There are so many factors that can affect performance that some people specialize in SQL Server performance tuning. Unless you intend to develop an application database to be used by many users, you don't need to know as much as these experts. However, that doesn't mean you don't need to know anything. As you progress through this book, I'll provide some recommendations that should help you get satisfactory performance from your databases.

## Attaching the AdventureWorks Database

If you downloaded and installed the AdventureWorks database as described in the previous chapter, you might be surprised to find it doesn't appear in Object Explorer. The install program just put the database files into the data folder used by SQL Server Express, but it didn't *attach* the database to your SQL Server Express instance. When you attach a database, you tell the server instance where to find the database files so the database can be accessed and managed by that instance. You can *detach* a database to stop the server from continuing to use the database in preparation for moving the files to a different instance. It's a fast way to move databases between servers. To attach the AdventureWorks database, follow the steps below.

## TO ATTACH THE ADVENTUREWORKS DATABASE

1. In Object Explorer, right-click the **Databases** folder and select **Attach**. The Attach Databases dialog box displays, as shown in Figure 3-12.

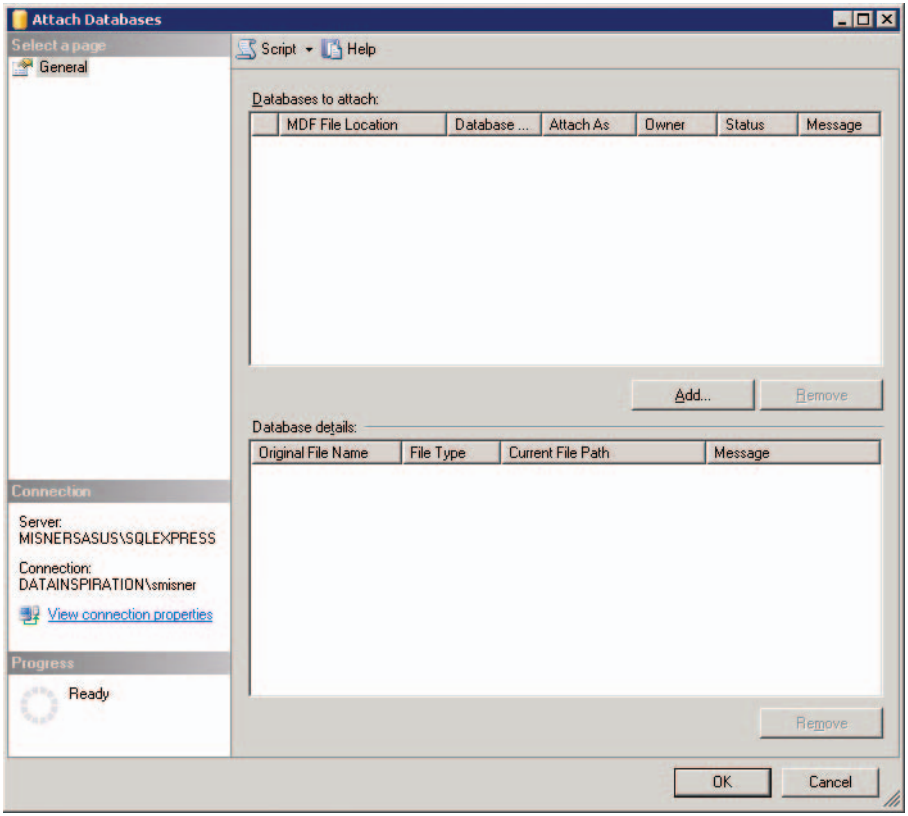
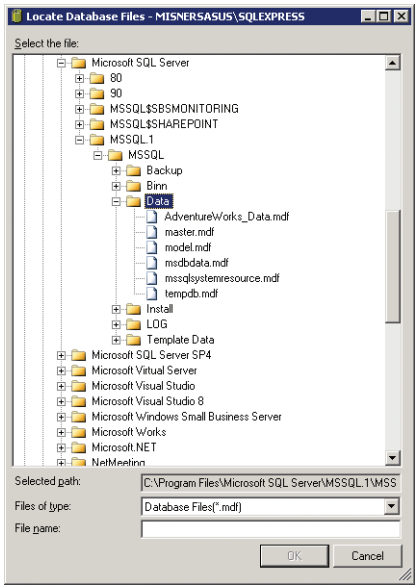



*Figure 3-12*
*Attach Databases dialog box*

**NOTE**

**I assume you have downloaded and installed the AdventureWorks sample database as instructed in Chapter 2, "Installing SQL Server Express." However, this procedure was optional and isn't required for you to successfully work through the remainder of this book. If you skipped installing the AdventureWorks sample database, you can jump ahead to the next section of this chapter to start working on your own**

2 Click **Add**. The Locate Database Files dialog box displays the data folder associated with your SQL Server instance and applies a filter to show only database files that have the file extension *mdf*, as shown in Figure 3-13.




3 In the Locate Database Files dialog box, select **AdventureWorks_Data.mdf** and click **OK**. The Attach Database dialog box displays the database and its details, as shown in Figure 3-14. (I've expanded the dialog box and resized the columns to make it easier for you to read.)

I'll explain the database files and different file types when you create your own database.

***Figure 3-13***
*Locate Database Files dialog box listing the default data folder*

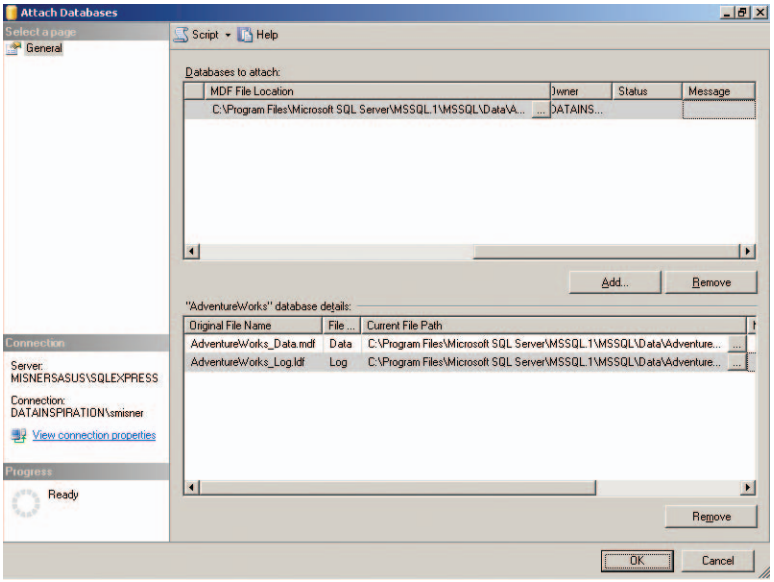



*Figure 3-14*
*Attach Databases dialog box displaying the AdventureWorks database files*

**NOTE**

**SQL Server Express expects to find database files in the data folder configured for the server, which by default is C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data. You can store databases almost anywhere else without a problem. You simply need to navigate through the file tree to find the folder that contains the database files, open that folder, and select your database. You shouldn't place database files in your My Document folder because SQL Server Express can't access that folder.**

**4** In the Attach Databases dialog box, click **OK**. Object Explorer now includes the AdventureWorks database, as shown in Figure 3-15.

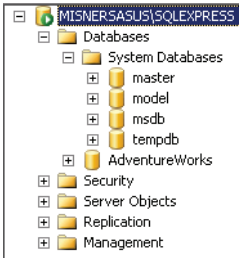



*Figure 3-15*
*Object Explorer displaying the AdventureWorks database*

Table - Production.Product

| ProductID | Name | ProductNumber | MakeFlag | FinishedGoodsF... | Color | SafetyStockLevel | Re |
|---|---|---|---|---|---|---|---|
| 1 | Adjustable Race | AR-5381 | False | False | NULL | 1000 | 75 |
| 2 | Bearing Ball | BA-8327 | False | False | NULL | 1000 | 75 |
| 3 | BB Ball Bearing | BE-2349 | True | False | NULL | 800 | 60 |
| 4 | Headset Ball Bea... | BE-2908 | False | False | NULL | 800 | 60 |
| 316 | Blade | BL-2036 | True | False | NULL | 800 | 60 |
| 317 | LL Crankarm | CA-5965 | False | False | Black | 500 | 37 |
| 318 | ML Crankarm | CA-6738 | False | False | Black | 500 | 37 |
| 319 | HL Crankarm | CA-7457 | False | False | Black | 500 | 37 |
| 320 | Chainring Bolts | CB-2903 | False | False | Silver | 1000 | 75 |
| 321 | Chainring Nut | CN-6137 | False | False | Silver | 1000 | 75 |
| 322 | Chainring | CR-7833 | False | False | Black | 1000 | 75 |
| 323 | Crown Race | CR-9981 | False | False | NULL | 1000 | 75 |
| 324 | Chain Stays | CS-2812 | True | False | NULL | 1000 | 75 |
| 325 | Decal 1 | DC-8732 | False | False | NULL | 1000 | 75 |
| 326 | Decal 2 | DC-9824 | False | False | NULL | 1000 | 75 |
| 327 | Down Tube | DT-2377 | True | False | NULL | 800 | 60 |
| 328 | Mountain End Caps | EC-M092 | True | False | NULL | 1000 | 75 |
| 329 | Road End Caps | EC-R098 | True | False | NULL | 1000 | 75 |
| 330 | Touring End Caps | EC-T209 | True | False | NULL | 1000 | 75 |

1 of 504 Cell is Read Only.

*Figure 3-16*
*Table window*

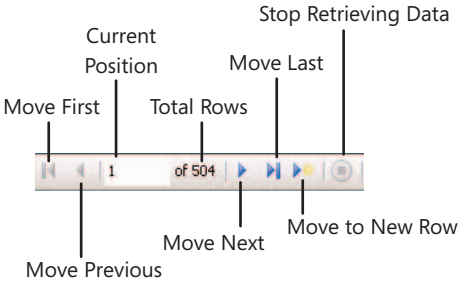



*Figure 3-17*
*Query Result Pane navigation bar*

**5** Let's see how to view data in this database. Expand the AdventureWorks folder, expand the Tables folder, right-click **Production.Product**, and select **Open Table**. The Table window displays, as shown in Figure 3-16.

Management Studio queries the table and displays the results in the Table window. You can use the scroll bars to view columns or rows that aren't visible on the initial display. At the bottom of the screen, you can see the Query Result Pane navigation bar (Figure 3-17).

In the Query Result Pane navigation bar, you can see the table has a total of 504 rows, with the first row specified as the current row. If you look at the first row in the table, you can see an arrow next to it to indicate it's the current row. You can type in a new row number in the Current Position box if you want to jump to a particular row or use the Move buttons to reposition the current row indicator one row at a time or to jump to the beginning or end of the table. When a row is current, you can type in new values. If you want to add a new row to the table, click the Move To New Row button and start typing in values. Use the Stop Retrieving Data button when you try to open a large table and you don't need to see all rows.

**6** You now know enough to continue exploring the AdventureWorks database on your own. When you're finished, close all Table windows so you've got a clean slate before starting the next exercise.

## Creating the Home Inventory Database

Now that you know how to use Management Studio to browse databases managed by SQL Server Express, it's time to create a database of your own. In this section, you'll learn how to use Management Studio features to do just that. In Chapter 4, you'll learn how to use Transact-SQL to do some of these tasks.

In Chapter 1, "Introducing Microsoft SQL Server Express," you learned a lot about how information is structured in a database using tables and columns and how data integrity is protected using primary keys, foreign keys, relationships, and constraints. To illustrate those concepts in that chapter, I introduced a simple version of a home inventory database. If you take a look at the database diagram in Figure 3-18, you'll see that I've added a few more columns to the Item table and added the Photo table to create a more useful database.

In addition to the table and column names, this database diagram shows the primary key columns, foreign key relationships, and data types for each column. All data types have a *length* that tells SQL Server Express how much storage space to allocate to the column. Some data types, like int, have a fixed length while other data types, like varchar, require you to specify a length. The database diagram includes the data length specification where it's applicable. You'll use this diagram as a reference while building tables for your database.

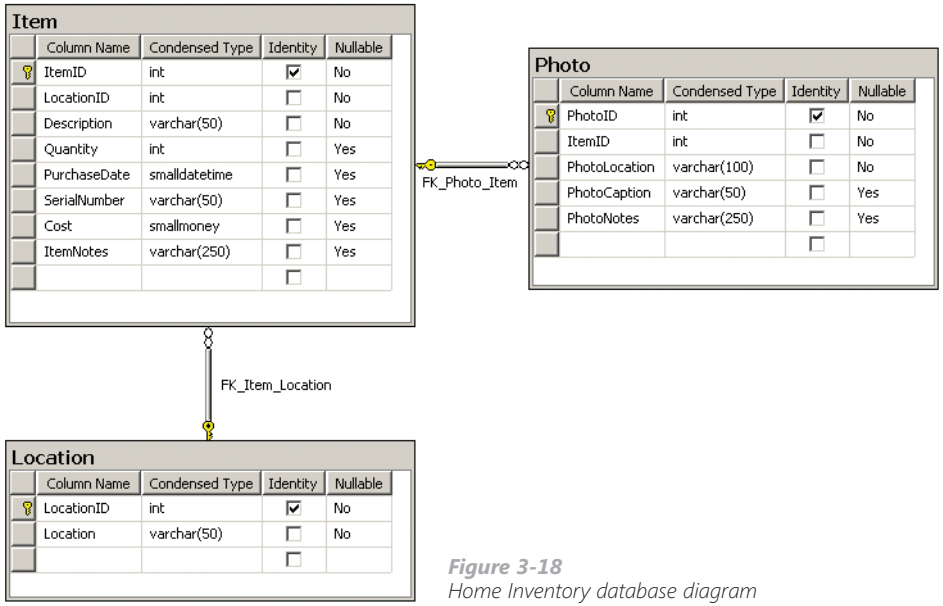


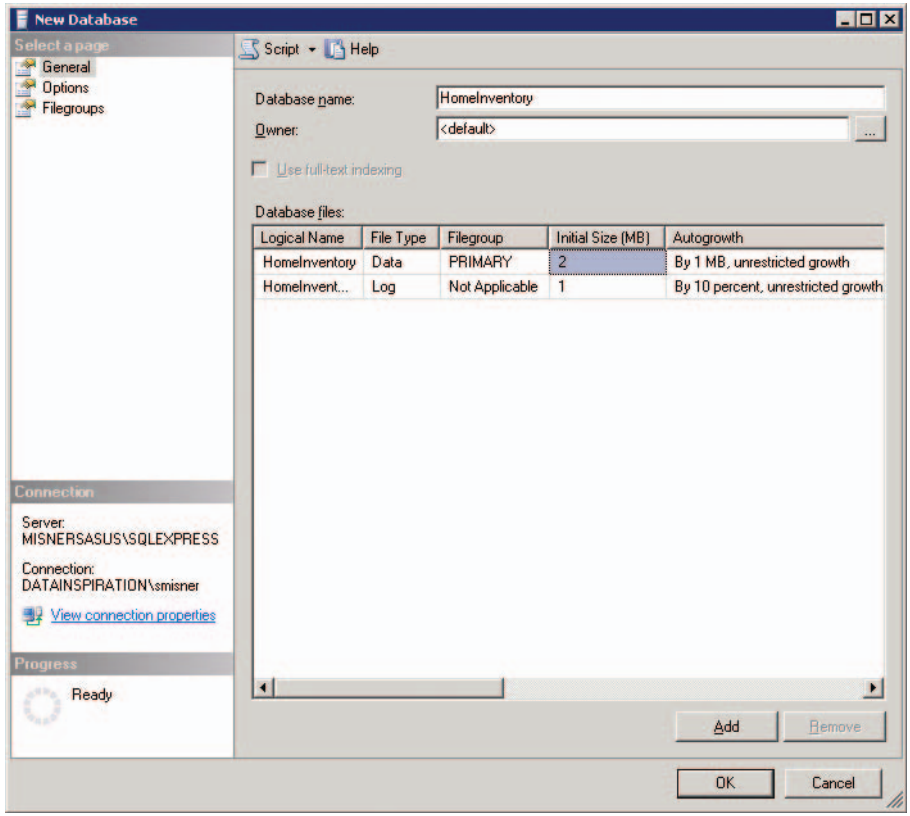

*Figure 3-18*
*Home Inventory database diagram*

## Creating Your Database

We've spent a lot of time preparing you for this moment by reviewing database theory and learning how to use Management Studio. From now on, you'll spend most of your time working on your database to really reinforce what you've been learning so far. To get started creating a database, follow the steps below.

### TO CREATE A DATABASE

1. If it's not already open, start Management Studio. In Object Explorer, right-click the Databases folder and select **New Database**.




2. In the New Database dialog box, in the Database Name box, type **HomeInventory**, as shown in Figure 3-19.

**NOTE**

**You're allowed to use spaces in a database name, but I prefer to name a database with one word because it's easer to reference in Transact-SQL statements. When a database name includes a space, you have to enclose it in brackets or double quotation marks. In fact, this rule applies to any *identifier*, the name of any database or database object—a table, column, or stored procedure to name a few.**

As you type the database name, the logical names for the database files generate automatically. The logical name is the name of the database that you see in Object Explorer. Any time you want to work with this database, you'll refer to it by this name. Under the covers, though, the logical name is a placeholder for the file where SQL Server Express actually stores the data.

*Figure 3-19*
*New Database dialog box*

Notice the File Type column in the Database Files list. Each database has at least two files—a database file with an *mdf* extension and a *transaction log file* with an *ldf* extension. The database file is the physical storage for the data values in your tables. The transaction log file records database activity in case something bad happens to your database so you can reproduce transactions later after restoring a database backup.

It's important to give some thought to the values defining Initial Size MB and Autogrowth properties. If you don't allocate enough space to the database when you create it, SQL Server Express can automatically increase the size of each file as needed. When the file is full, SQL Server Express locks everything down to prevent new data from being added. To avoid being caught by surprise when this happens, you should try to estimate a size a bit larger than you expect to need. For the home inventory database, you don't need to worry about managing file size, but while we're here, let's take a look at the options for Autogrowth settings.

**3** Click the ellipsis button in the **Autogrowth** box in the first row to display the Change Autogrowth For HomeInventory dialog box (Figure 3-20). If you don't see the ellipsis button, you may need to use the scroll bar near the bottom of the New Database dialog box to locate the button.

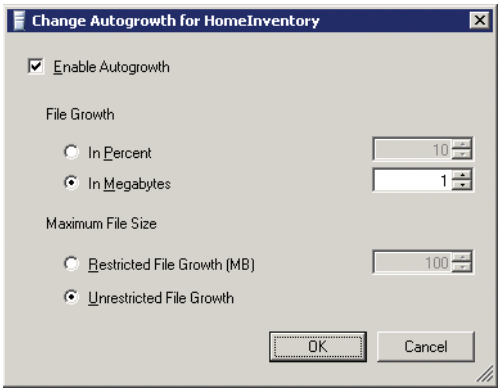



*Figure 3-20*
*Change Autogrowth for HomeInventory dialog box*

You don't have to use the Autogrowth feature. You can clear the Enable Autogrowth check box to turn it off, but if you do this, you run the risk of running out of space in the file eventually. If, on the other hand, you decide to use Autogrowth, you need to specify how much the file is allowed to grow. This growth rate can be calculated as a percentage of the database size, or it can be a specific number of megabytes.

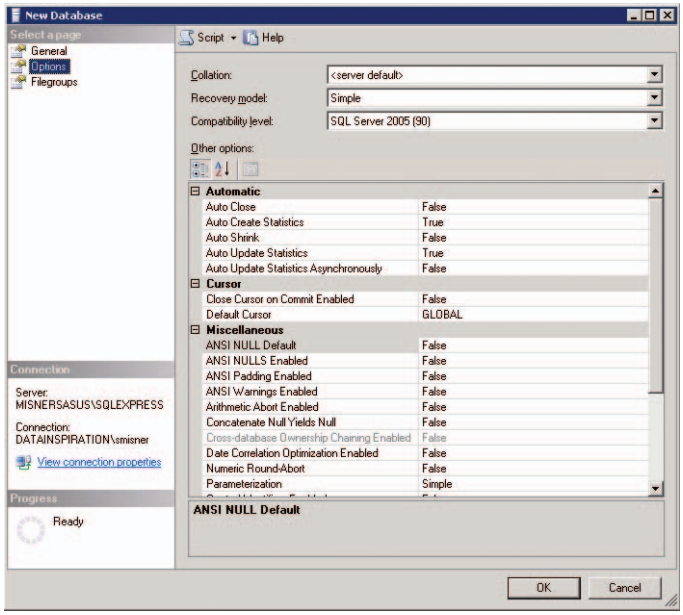
You can also put a cap on how big the file can get if you select the Restricted File Growth (MB) option and specify a value. By default, the Autogrowth feature places no limit on file growth, but the 4-gigabyte (GB) limit on the database will always be enforced.

4. Because we're here just to see what your options are, click **Cancel** to keep the default settings.

5. Click **Options** in the left pane to review the default database options, as shown in Figure 3-21.

Usually the default settings are exactly what you need. When you start building your own databases, you should give some thought to the Recovery Model selection, which tells SQL Server Express how much logging it needs to do. The default recovery model, Simple, keeps logging to a minimum. If something goes wrong with the database, you can restore a backup. But anything that occurred in the database after the last backup was created is lost because it wasn't stored in the transaction log file. You'll have to find some other way to recreate any changes to the database since the last backup.

For a database that stores critical information, you should use the Full recovery model, which has the most complete log. If you need to restore a database, you can use the transaction log to reproduce activity occurring between the last backup and the point of failure. The Bulk-logged recovery model is somewhere between the other two options. It keeps a complete log for normal transactions but only minimally logs a ***bulk load*** or the creation of indexes. A bulk load is a fast way to import data from a file and insert it as new rows into a table.




You can set a lot of other database options here, but at this point there is no particular need to do so. After you've gained some more experience with databases, you can learn more about these options by searching SQL Server Books Online for the topic Setting Database Options.

*Figure 3-21*
*Default database options*

6 Click **Filegroups** in the left pane to review the default filegroup, as shown in Figure 3-22.

Filegroups are applicable only to the database data files. You can create multiple data files for your database by using the Add button on the General page of the New Database dialog box. Then you can assign the data files to different filegroups. If you have a large database, spreading data across different filegroups allows you to place each filegroup on a separate hard drive so that data can be read from all drives simultaneously. You must always have a PRIMARY filegroup, but any other data file you add can be either in the PRIMARY filegroup or in a secondary filegroup you create here. The additional files you create using this feature will have an *ndf* file extension.

7 Click **OK** to officially create the database. It usually takes a few seconds, but when the process completes, you'll see the new database in Object Explorer (Figure 3-23). You can also find the physical files in the installation folder, C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data.

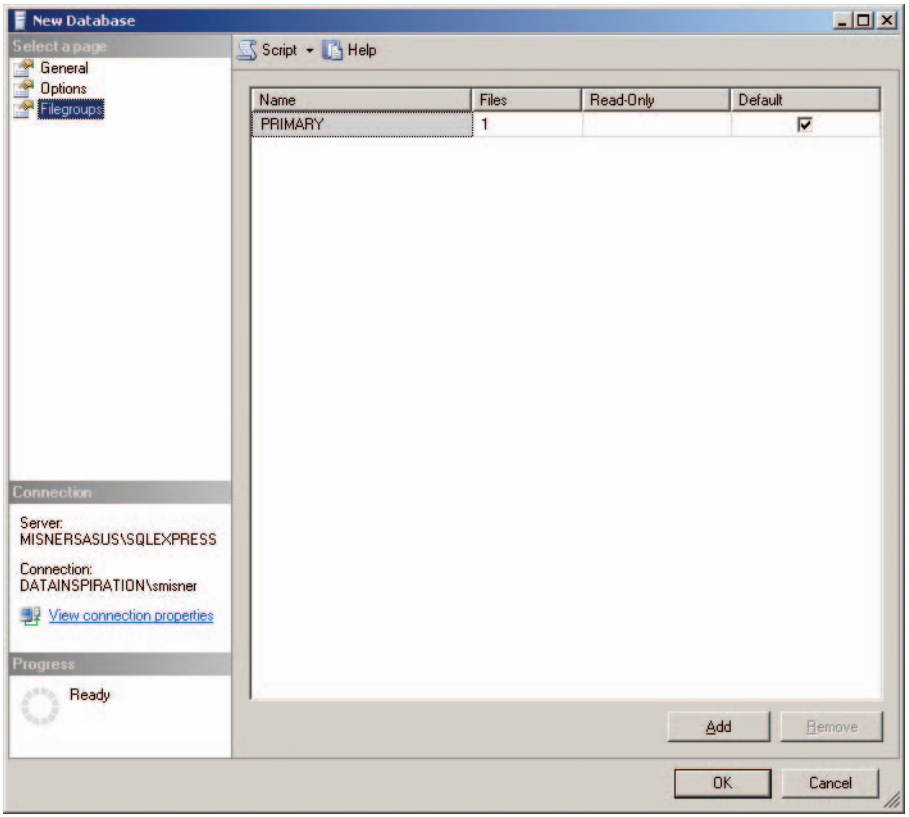



*Figure 3-22*
*Default filegroup*

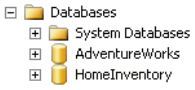



*Figure 3-23*
*Updated Object Explorer*

You have now created the physical storage for your database. The next step is to define the structure of the data, which you'll do in the following section.

## Adding Tables to Your Database

Creating the database is pretty straightforward and quick, but setting up the tables takes a bit more time. It's still easy to do using Management Studio, but you have a lot more details to look after. For each table, you need to define the columns by providing an identifier, selecting a data type, and setting other properties as applicable. You can have a whopping 1,024 columns in any table, although you might find it tedious to set up that many. Even more mind-boggling is the two billion table limit within a database. Can you imagine the amount of work required to build a database that large?

In Chapter 1, I made a big deal about thinking about space usage in your tables and encouraged you to aim for a smaller size whenever possible. The reason for this recommendation is the 8,060-byte limit on each row. Total table size is determined not only by how many rows are in the table (depth) but also by the size of the rows (width). As I pointed out earlier in this chapter, the data type you select for each column determines the amount of storage space the database allocates for the row. You should keep in mind a couple of general rules of thumb. Numeric data types take up less space than non-numeric types, and the SQL Server database engine can work more efficiently with smaller data types than larger data types.

Usually, you store data using a system-defined data type. You can also create a user-defined data type by using a system-defined data type as a base, but, for the purposes of your home inventory database, the system-defined data types will suffice. As you build other databases, you'll find that you use some data types more frequently than others. However, it's a good idea to review your available options so you can select a data type that's most applicable to your needs. In Table 3-1, you can review the system-defined data types you can use:

*Table 3-1*
*Data Type Table*

| Data Type | Storage Size | Description |
|---|---|---|
| **Exact Numerics** | | |
| Bigint | 8 bytes | An integer ranging from $-2^{63}$ (-9,223,372,036,854,775,808) to $2^{63} - 1$ (9,223,372,036,854,775,807) |
| Int | 4 bytes | $-2^{31}$ (-2,147,483,648) to $2^{31} - 1$ (2,147,483,647) |
| Smallint | 2 bytes | $-2^{15}$ (-32,768) to $2^{15} - 1$ (32,767) |
| Tinyint | 1 byte | 0 to 255 |
| Bit | 1 byte for up to 8 bit columns, 2 bytes for 9 to 16 bit columns, and so on | An integer value of 0, 1, or NULL |
| Decimal or numeric | 5 bytes for precision 1–9<br>9 bytes for precision 10–19<br>13 bytes for precision 20–28<br>17 bytes for precision 29–38 | A value ranging from $-10^{38} + 1$ through $10^{38} - 1$ if using the maximum precision of 38 |
| Money | 8 bytes | A monetary value ranging from -922,337,203,685,477.5808 to 922,337,203,685,477.5807 |
| Smallmoney | 4 bytes | A monetary value ranging from -214,748.3648 to 214,748.3647 |
| **Approximate Numerics** | | |
| Float | 4 bytes for 7-digit precision and 1–24 bits for *n*<br>8 bytes for 15-digit precision and 25–53 bits for *n* | A floating point numeric value ranging from -1.79E + 308 to -2.23E − 308, 0 and 2.23E − 308 to 1.79E + 308. The number of bits (*n*) used to store the mantissa of the float number in scientific notation determines both precision and storage size. |

*Continued on next page*

*Continued from page 63*

| Data Type | Storage Size | Description |
|---|---|---|
| Real | 4 bytes | A floating point numeric value ranging from -3.40E + 38 to -1.18E − 38, 0 and 1.18E − 38 to 3.40E + 38 |
| **Date and Time** | | |
| Datetime | 8 bytes<br>` | A date and time value ranging from January 1, 1753, through December 31, 9999 with precision to 3.33 milliseconds |
| Smalldatetime | 4 bytes | A date and time value ranging from January 1, 1900, through June 6, 2079 with precision to 1 minute |
| **Character Strings** | | |
| Char | 1 byte per character specified regardless of actual data stored | Fixed length non-Unicode characters for which you specify a length from 1 to 8,000 |
| Varchar or Varchar(max) | Actual length of data stored plus 2 bytes | Variable length non-Unicode character data for which you specify a length from 1 to 8,000; use varchar(max) for character data exceeding 8,000 bytes up to a maximum storage of $2^{31} - 1$ bytes |
| **Unicode Character Strings** | | |
| Nchar | 2 bytes per character specified regardless of actual data stored | Fixed length Unicode characters for which you specify a length from 1 to 4,000 |
| Nvarchar or Nvarchar(max) | Twice the actual length of data stored plus 2 bytes | Variable length Unicode character data for which you specify a length from 1 to 4,000; use nvarchar(max) for character data exceeding 4,000 bytes up to a maximum storage of $2^{31}-1$ bytes |

*Continued on next page*

*Continued from page 64*

| Data Type | Storage Size | Description |
|---|---|---|
| **Binary Strings** | | |
| Binary | Specified length | Fixed length binary data for which you specify a length from 1 to 8,000 |
| Varbinary or Varbinary(max) | Actual length of data stored plus 2 bytes | Variable length non-Unicode character data for which you specify a length from 1 to 8,000; use varbinary(max) for character data exceeding 8,000 bytes up to a maximum storage of $2^{31}-1$ bytes |
| **Other Data Types** | | |
| Sql_variant | Depends on data type stored with maximum of 8,016 bytes | Any type of value except varchar, nvarchar, varbinary, xml, timestamp, and user-defined data types used when the data type in a column can change from row to row |
| Timestamp | 8 bytes | A unique binary number generated automatically when a row is inserted or updated |
| Uniqueidentifier | 16 bytes | A binary data value used as a globally unique identifier (GUID) |
| Xml | Actual length of data up to 2 GB | XML data in the form of an XML document or XML fragments |

**NOTE**

**In the preceding table, you may not be familiar with some terms:**
***Precision*** **is the number of significant digits that can be used to the left and to the right of the decimal point.**
***Scale*** **limits the number of significant digits to the right of the decimal point. The minimum scale is 0, but the maximum depends on precision, which the maximum scale cannot exceed.**
***Unicode*** **is used to support international characters in a database.**

If you refer back to the database diagram in Figure 3-18, you can see the home inventory database uses only four data types. Integers are used for primary keys because we can be certain we won't be exceeding the 2,147,483,647 limit. Even if you have a lot of stuff and are really diligent about entering *all* of it into the database, I'd be very surprised if you could generate enough rows to go over this limit. Several columns use the varchar data type to conserve space where necessary, but to allow space for enough characters for what we anticipate to be the largest value for a particular column. If you find later that you need a larger column, you can always increase the length then. For the PurchaseDate column, the smalldatetime data type is a good choice because you probably didn't acquire anything you have prior to January 1, 1900. Even if you have inherited an antique from a relative, you should use the date you actually took possession of the item and not the original purchase date. Finally, use the smallmoney data type to store an item's cost. This should be adequate unless you have an item worth more than $214,748. If that's really the case, I'm impressed and recommend you use the money data type instead. Now, let's get on with creating these tables by following the steps below.

## TO ADD TABLES TO A DATABASE

1. Let's start with the Item table. In Object Explorer, in the Databases folder, expand the HomeInventory folder, right-click the **Tables** folder, and select **New Table**. The Document window now displays the Table Designer window and the Properties window appears next to it, as shown in Figure 3-24.

2. Now, you'll add columns to the table in sequence. From the database diagram in Figure 3-18, you can see the first column is ItemID. In the Table Designer, in the Column Name box, type **ItemID**. In the Data Type drop-down list, select **Int** and clear the Allow Nulls check box. Remember from Chapter 1 that a primary key column can never be null when you need to protect data integrity.

**NOTE**

**In the Data Type drop-down list, you may notice some additional data types that I didn't include in the data types table: ntext, text, and image. These data types will be removed in future versions of SQL Server so it's best not to get in the habit of using them.**

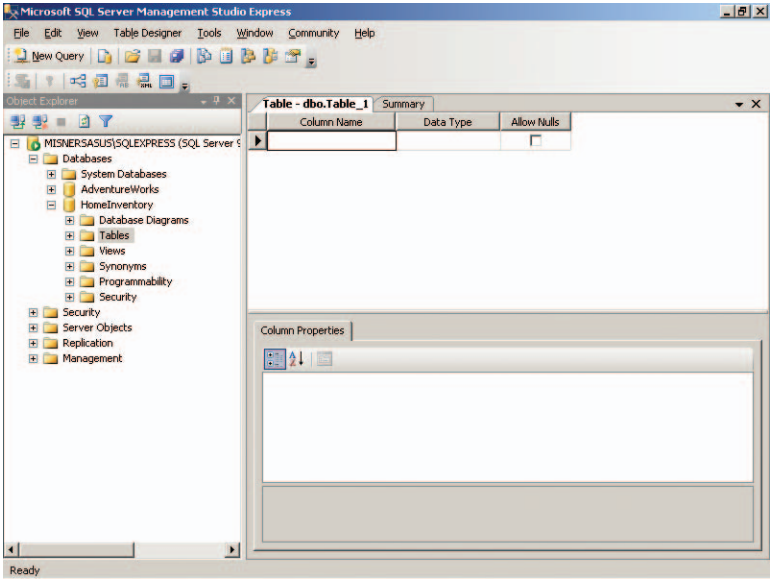



*Figure 3-24*
*Creating a new table*

**3** To set the ItemID column as a primary key, click the **Set Primary Key** button on the Table Designer toolbar (Figure 3-25). A key icon appears next to the column so you know at a glance this is a primary key column.

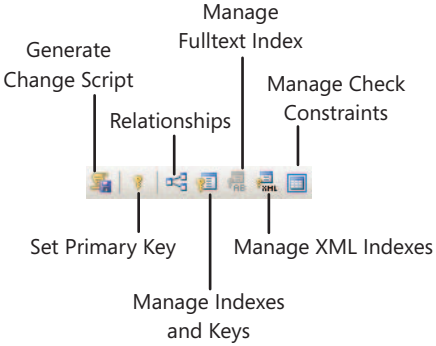



*Figure 3-25*
*Table Designer toolbar*

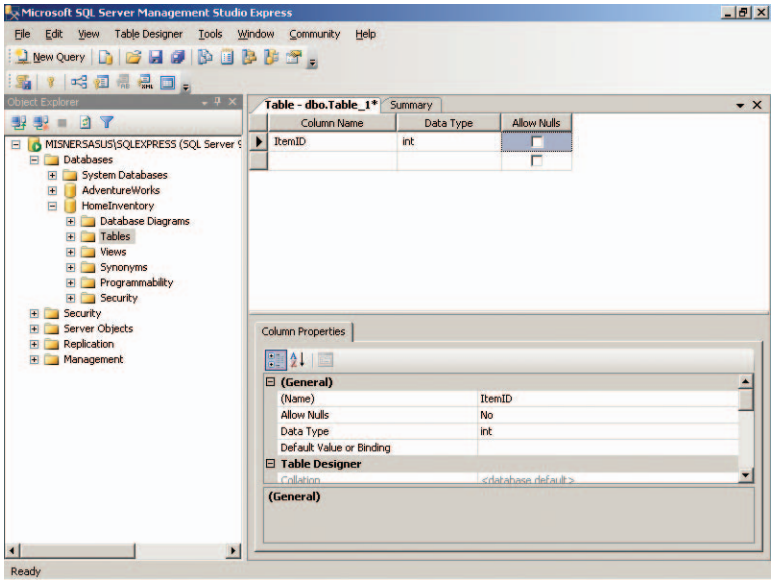



*Figure 3-26*
*Table Designer window displaying primary key and Column Properties pane*

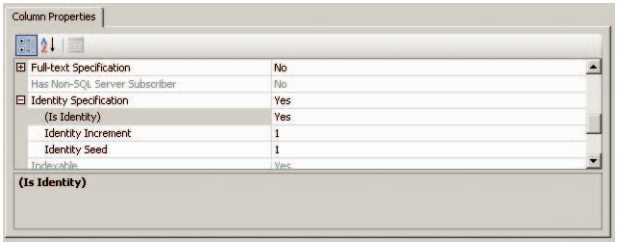



*Figure 3-27*
*Column properties pane with identity specification*

The Column Properties pane at the bottom of the Table Designer now displays property values, as shown in Figure 3-26.

**4** There's one more task to complete before you move on to the next column. Remember the ItemID column is also an identity column. The only place to configure this is in the Column Properties pane. Scroll down to locate the ***Identity Specification*** property, click the plus sign next to it, and then, in the (Is Identity) drop-down list, select **Yes**. Keep the default values for Identity Increment and Identity Seed, as shown in Figure 3-27. That's it! One column done and seven to go for this table.

**5** To add the next column, just click in the Column Name box below ItemID, type **LocationID**, and set the Data Type to the value shown in the database diagram in Figure 3-18. Clear the Allow Nulls check box as well. Continue adding the remaining columns for this table. It's okay to allow nulls for the remaining columns except Description.

**TIP**

**Variable-length data types, such as Varchar, default to a specific length that you can override simply by typing a new value in place of the default value. For example, if you select varchar(50) in the Data Type drop-down list, you can replace 50 with 250 to define a larger column.**

When you're finished, the Table Designer should look like the one shown in Figure 3-28.

**TIP**

**If you skip a column accidentally, you can insert a column between two existing columns to put it where it belongs. Just right-click the column that is supposed to appear below the missing column and select Insert Column. Now, you've got the space to add the new column.**

Table - dbo.Table_1*

| Column Name | Data Type | Allow Nulls |
|---|---|---|
| ItemID | int | ☐ |
| LocationID | int | ☐ |
| Description | varchar(50) | ☐ |
| Quantity | int | ☑ |
| PurchaseDate | smalldatetime | ☑ |
| SerialNumber | varchar(50) | ☑ |
| Cost | smallmoney | ☑ |
| ItemNotes | varchar(250) | ☑ |
| | | ☐ |

*Figure 3-28*
*Table Designer with all columns for the Item table*

**6** The columns in the table are added, but the table doesn't exist yet. To save your work and officially add the table to the database, click the **Save Table_1** button in the standard toolbar or press **Ctrl+S**. (This button was labeled Saved Selected Items in Figure 3-3 when you were first introduced to it earlier in this chapter, but it's context-sensitive and changes its name according to what you're doing.) In the Choose Name dialog box, shown in Figure 3-29, type **Item** to name the table and then click **OK**.

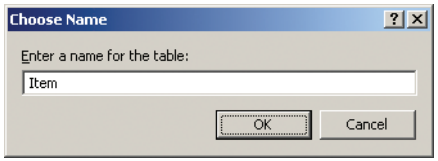



*Figure 3-29*
*Choose Name dialog box*

**7** In Object Explorer, expand the Tables folder, expand dbo.Item, and then expand the Columns folder, as shown in Figure 3-30. You can see all the columns you just added in the table, and you can see that ItemID is a primary key by the key icon next to it and by the inclusion of the abbreviation PK in parentheses.

**8** Click the **X** in the upper right corner of the Table Designer window to close it.

**9** Using the database diagram in Figure 3-18 as a guide, go ahead and add the remaining tables to the database. Double-check your work to make sure you included all of the columns in each table. Do not allow nulls in the following columns: LocationID, Location, PhotoID, ItemID, and PhotoLocation. Just focus on adding the tables and columns right now. In the next section, you'll add relationships between the tables. Don't forget to set the primary key column as an identity column in each table.

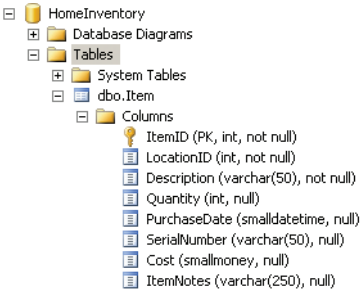



*Figure 3-30*
*Object Explorer displaying the Item table's columns*

## Adding Relationships

Remember from Chapter 1 that foreign key relationships help you prevent data integrity problems from creeping into your database. Just because the primary keys and foreign keys shown in the database diagram in Figure 3-18 are identical, SQL Server Express doesn't automatically detect the foreign key relationships. You need to explicitly define them. Besides, there is no rule that says the related column names must match; it's just easier to keep track of the relationships if you do give them the same name. To add a foreign key relationship to a table in your database, follow the steps below.

### TO ADD A FOREIGN KEY RELATIONSHIP TO A TABLE

1. In Object Explorer, in the Databases folder, expand the HomeInventory folder, expand the Tables folder, expand dbo.Item, right-click **Keys**, and select **New Foreign Key**. The Foreign Key Relationships dialog box displays, as shown in Figure 3-31.

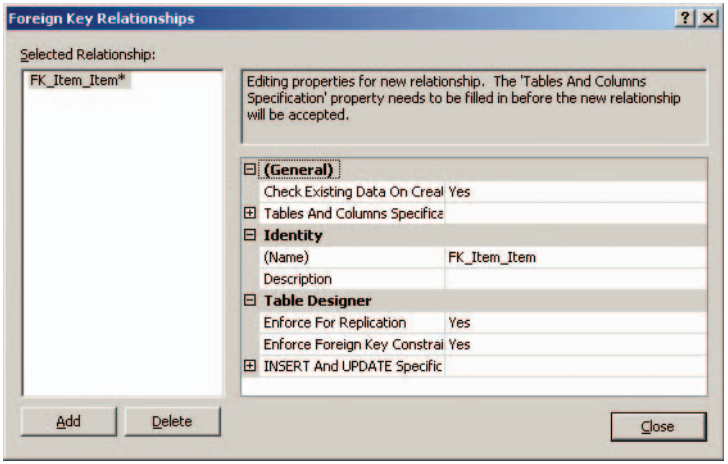



*Figure 3-31*
*Foreign Key Relationships dialog box*

2. Click in the Tables And Columns Specifications box and then click the ellipsis button to display the

Tables And Columns dialog box (Figure 3-32).

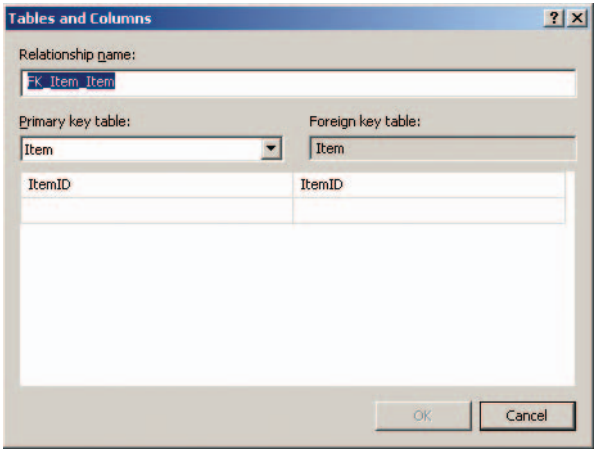



*Figure 3-32*
*Tables And Columns dialog box*

**3** Type a new name in the Relationship Name box, **FK_Item_Location**. In the Primary Key Table drop-down list, select **Location** and then, in the drop-down list on the first row below Location, select **LocationID**. In the first row below Item, in the drop-down list, select **LocationID**. The Tables And Columns dialog box on your screen should look like Figure 3-33. Click **OK** to close the Tables And Columns dialog box.

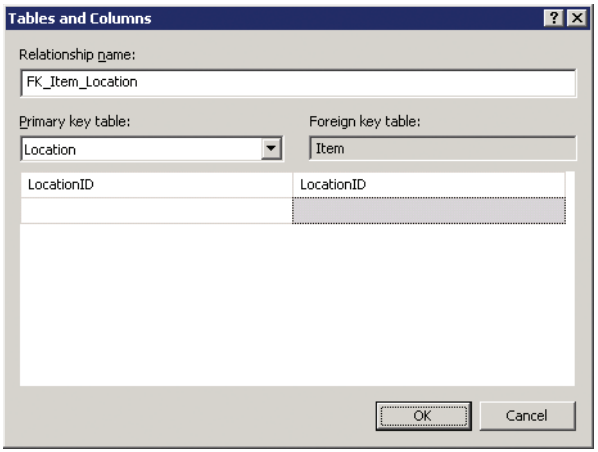



*Figure 3-33*
*Tables And Columns dialog box with a foreign key defined*

**TIP**

**You can name the relationship anything you like, but it's standard practice to use a naming convention that includes FK to indicate it's a foreign key relationship and also includes the related tables.**

Let's take a look at the properties in the Foreign Key Relationships dialog box:

**Check Existing Data On Creation Or Re-Enabling** The default value is Yes, which validates the data to ensure there are no orphan rows if you add a foreign key relationship between tables after data is already entered into the table or if you enable the relationship after previously disabling it.

**Description** This property is optional. When many people are responsible for maintaining a database, it's helpful to use this property to include more information about the foreign key relationship, such as specific business rules that must be enforced. You actually have quite a lot of space here for a description. Just click the ellipsis button to open the Description Property dialog box and type away.

**Enforce For Replication** If your database will be used in a replication scenario, and this property's value is Yes, any new rows, changes to existing rows, or attempts to delete a row will be performed only when the result of the activity does not violate referential integrity between the tables.

**Enforce Foreign Key Constraint** The default value is Yes, which is the whole point of creating a foreign key relationship. As mentioned in Chapter 1, there may be occasions to disable the constraint temporarily by changing the value to No.

**INSERT And UPDATE Specification** The Delete and Update rules here show you what happens if a user or application tries to delete or update a row that has an existing foreign key relationship. If the rule is No Action, an error message displays and the activity is prevented from continuing. For deletes, the Cascade rule deletes all related rows in a child table and in the parent table regardless of which table the row you are deleting is in. The Cascade rule for updates propagates a change to a primary key in a parent table to the foreign key columns in the child tables. The Set Null rule for deletes and updates changes the foreign key columns to null in the child tables, though the Set Default rule uses the default definition of the column in the child tables to replace the value in the foreign key column.

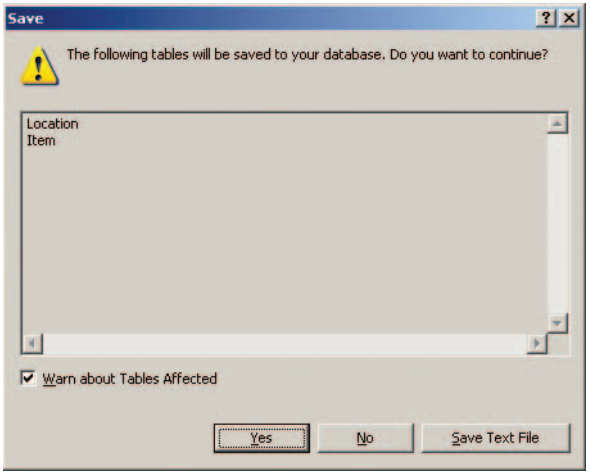



*Figure 3-34*
*Save changes to the affected tables*

4. Click **Close** to close the Foreign Key Relationships box. You'll learn another way to add a foreign key relationship when you add FK_Item_Photo in Chapter 4.

5. Click the **Save** button. You will be prompted to save changes to the affected tables, as shown in Figure 3-34.

6. Click **Yes** and then close the Table Designer window.

## Adding Constraints

When you identified primary key columns in your tables, you automatically created a PRIMARY KEY constraint, which guarantees each row in the table will have a unique primary key value. Adding foreign key relationships added another constraint to ensure referential integrity between related tables. In addition to these constraints, you can use a few other types of constraints to enforce data integrity, as explained in Chapter 1. To review, use a CHECK constraint to make sure data values in a column stay within a specified range. To prevent a NULL value from existing in a column, use a NOT NULL constraint. A UNIQUE constraint is similar to the PRIMARY KEY constraint in that it guarantees values in a column remain unique in all rows of the table, except it permits one row to have a NULL value though the PRIMARY KEY constraint does not.

Using the home inventory database, you can learn how to implement these additional constraints. You already implemented the NOT NULL constraint when you left the Allow Nulls check box cleared as you added columns. Now, consider that the Cost column in the Item table is defined with the smallmoney data type, which enables you to use negative values. However, it doesn't make sense to allow negative values in this column, so you'll add a CHECK constraint to make sure only positive values are entered. Next, it seems reasonable to expect locations to be unique, so you'll add a UNIQUE constraint to enforce that requirement. To add these constraints, follow the steps below.

### TO ADD CONSTRAINTS TO TABLE COLUMNS

1. In Object Explorer, in the Databases folder, expand the HomeInventory folder, expand the Tables folder, expand dbo.Item, right-click **Constraints**, and select **New Constraint**. The Check Constraints dialog box displays, as shown in Figure 3-35.

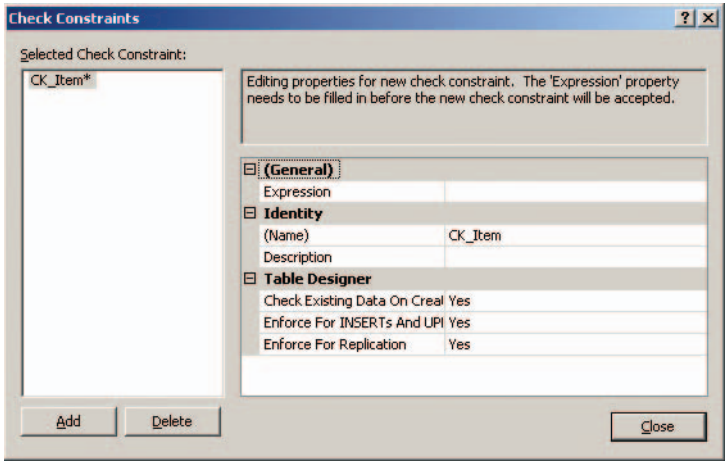



*Figure 3-35*
*Check Constraints dialog box*

**NOTE**

**An expression needs to evaluate as TRUE or FALSE to be valid. Often an expression in a CHECK constraint compares the data value in a column to a constant value or to a data value in another column using operators like greater than (>), less than (<), equals (=), and so on. You can also build more complex conditions that use two or more expressions connected with AND and OR.**

**NOTE**

**Just like the name for a foreign key relationship, you can name a CHECK constraint whatever you like, but standard practice is to use CK as a prefix followed by the table name and column name with an underscore (_) as a separator.**

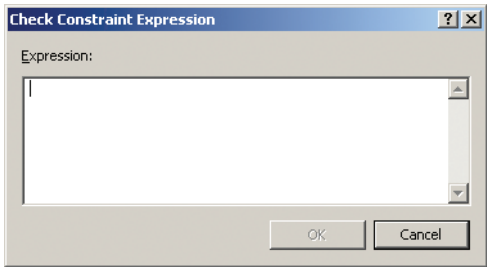



*Figure 3-36*
*Check Constraint Expression dialog box*

2. Click in the Expression box and then click the ellipsis button to display the Check Constraint Expression dialog box, as shown in Figure 3-36.

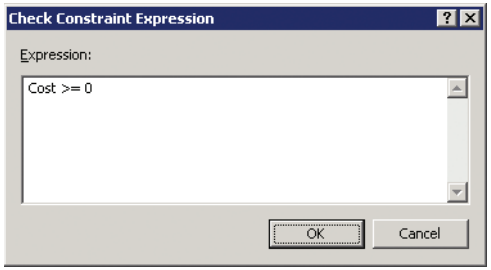
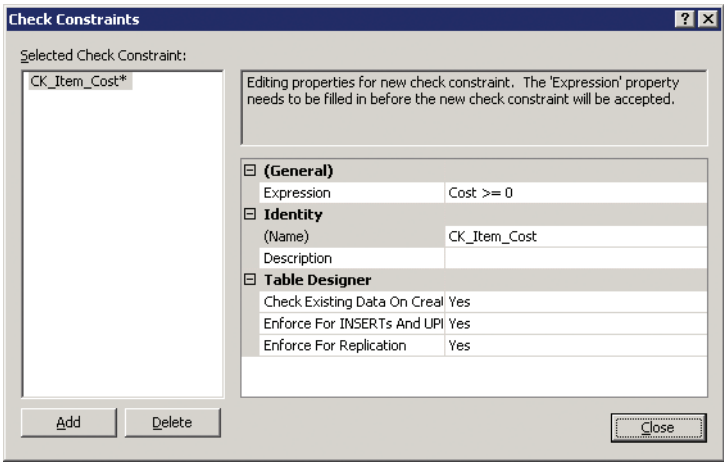



*Figure 3-37*
*CHECK Constraint Expression to require a positive or zero value for the Cost column*

3. Type the following expression in the Expression box to match Figure 3-37:

```
Cost >= 0
```

4. Click **OK** to close the Check Constraint Expression dialog box.




5. In the (Name) box, type **CK_Item_Cost**, as shown in Figure 3-38.

*Figure 3-38*
*CHECK constraint for the Cost column in the Item table*

While we're here, let's review the other properties in the Check Constraints dialog box:

**Description** This property is optional. You can use this property to document the business rules for the constraint, which is helpful when you're not the only one maintaining the database. Click the ellipsis button to open the Description Property dialog box and type as much text as you need to describe the constraint.

**Check Existing Data On Creation Or Re-Enabling** Keep the default value of Yes when you create a constraint on a table that already has data or when you re-enable the constraint after temporarily disabling it. SQL Server will validate the data as it creates the constraint.

**Enforce For INSERTs And UPDATEs** It doesn't make much sense to create a constraint that isn't enforced when you add new rows or update existing rows, so you should keep the default value as Yes. However, as with FOREIGN KEY constraints, some situations may require you to disable the constraint temporarily by changing the value to No.

**Enforce For Replication** This property behaves similarly to the same property you saw in the Foreign Key Relationships dialog box. If your database will be used in a replication scenario, and this property's value is Yes, any data changes will be performed only when the constraint expression evaluates as TRUE.

6 Click **Close** to close the Check Constraints dialog box. The CHECK constraint is now added to the table. It would seem reasonable to expect the constraint to exist now, but you have to save the change to the table as a final step. Click **Save** in the standard toolbar and then close the Item table.

7 Let's continue by adding the UNIQUE constraint to the Location table. In Object Explorer, right-click the **Location** table and select **Modify**.

8 In the Table Designer toolbar, click **Manage Indexes and Keys**. The Indexes/Keys dialog box displays, as shown in Figure 3-39. It shows the existing primary key definition.

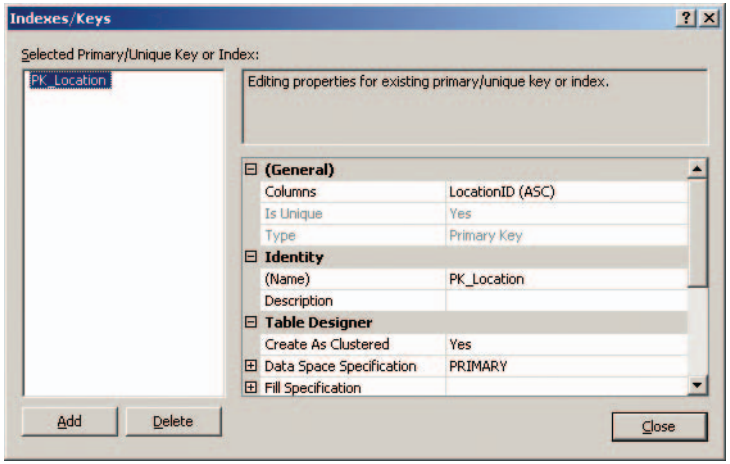



*Figure 3-39*
*Indexes/Keys dialog box*

9 Click **Add**. Click in the Columns box and then click the ellipsis button to open the Index Columns dialog box (Figure 3-40). The default column is the primary key column, which you'll replace in the next step.

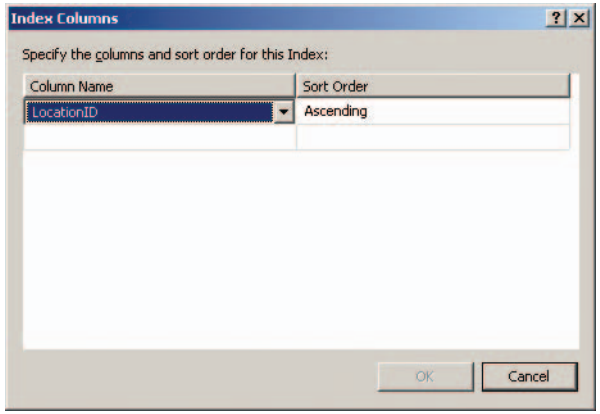



*Figure 3-40*
*Index Columns dialog box*

10 In the first Columns drop-down list, select **Location**, as shown in Figure 3-41, and then click **OK**.

**NOTE**

**Remember that a UNIQUE constraint can be defined by multiple columns. Just continue adding columns to subsequent rows when you need this type of constraint.**

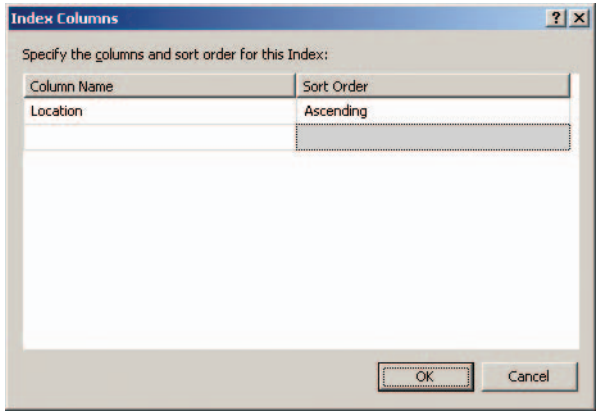



*Figure 3-41*
*Index Columns dialog box with Location selected*

11 In the Type drop-down list, select **Unique Key**. In the (Name) box, type **Uniq_Location**, as shown in Figure 3-42.

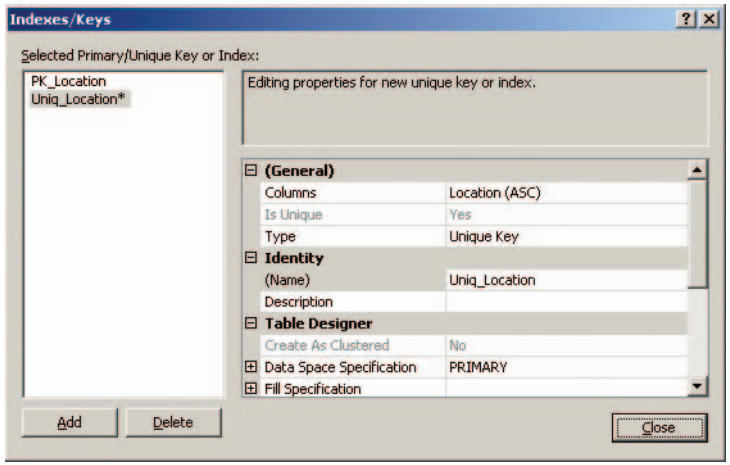



*Figure 3-42*
*Indexes/Keys dialog box with the UNIQUE constraint defined for the Location column*

**NOTE**

**Notice the list of keys and indexes displays Uniq_Location*. You don't need to type the asterisk(*) as this symbol is added automatically by Management Studio as a visual cue that this is a new index or unique key.**

**TIP**

**You can name a UNIQUE constraint whatever you like, but consider using Uniq_ as a prefix to the column name as a naming convention for this type of constraint.**

This dialog box has several properties. Let's review them now:

**Description** This property, as you've learned by now, is optional. Use this property to document the business rules for the constraint. Click the ellipsis button to type descriptive text in the Description Property dialog box.

**Data Space Specification** If you have multiple filegroups for database storage, you can select the filegroup used to store the index used to manage uniqueness for this column. Expand Data Space Specification to access the Filegroup Or Partition Scheme Name property, which you use to identify the storage location.

**Fill Specification** Expand Fill Specification to access the Fill Factor property. Here you can specify a value from 1 to 100 to control how much extra space should be factored into the amount allocated by default to avoid splitting up data later and causing performance problems. You'll probably find the default value of 0 is suitable most of the time because SQL Server Express will use the space allotment efficiently and allow a bit extra for expansion as you load data into the table.

12 Click **Close** to close the Indexes/Keys dialog box. Click **Save** in the standard toolbar and then close the Location table.

## Indexing Your Tables

The index at the back of this book helps you locate specific information much more quickly than if you were to search page by page beginning with page one. Similarly, an index on a table is useful for speeding up query performance because the database engine can zero in on the requested data instead of examining the table row by row. Developing a good index strategy can be tricky, though. Though speeding up query performance is a good thing, the down side is that more indexes take up more disk space.

To create an index, you select one or more *index keys* to define which column or columns will be indexed. An index with a single column is a *simple index*. An index is better when it is as unique as possible, which might require a *composite index* consisting of several key columns.

TIP

**Don't get carried away with adding key columns, though, because an index with too many columns takes up more space than needed without necessarily delivering the expected benefit of faster query performance. On top of that, it takes longer to add new data and to change data when a table has a lot of indexes.**

You probably won't add enough data to your home inventory database to notice much of a difference between queries before and after you add an index, but learning how to create an index now will prepare you for the time when adding an index is necessary. When deciding which table column or columns would be useful in an index, you should think about how you'll query the table and the criteria you'll use to filter the query results to a specific row or set of rows. With this in mind, a good candidate for an index is the SerialNumber column in the Item table. To add an index for this column, follow the steps below.

### TO ADD AN INDEX TO A TABLE

1. In Object Explorer, in the Databases folder, expand the HomeInventory folder, expand the Tables folder, expand dbo.Item, right-click **Indexes**, and select **New Index**. The New Index dialog box displays, as shown in Figure 3-43.

2. In the Index Name box, type **Item_SerialNumber_IX**.

**TIP**

**A naming convention to consider for an index is the name of the table, the name of the column or columns used as an index key, and the letters IX to represent an index. Each component of the name is then separated with an underscore (_).**

When creating a new index, you can choose from three index types:

**Clustered** In a clustered index, the table data is stored in sorted order to avoid the overhead of rearranging the index later as data changes. In fact, the data is stored right in the index itself, which forces all queries to use the index to get at the data. It's important to design the clustered index correctly to get the best performance. By the way, you can have only one clustered index in a table.

**Nonclustered** You can have as many nonclustered indexes as you like in a table. In contrast to a clustered index, a nonclustered index does not store table data in the index. Instead, it stores a pointer to data in a clustered index or to a particular row in a table when a clustered index hasn't been created.

**Primary XML** This type of index is used to speed up queries on columns storing an XML data type.

**3** Click **Add**. In the Select Columns From 'dbo. Item' dialog box, select the check box to the left of SerialNumber, as shown in Figure 3-44, and click **OK**.

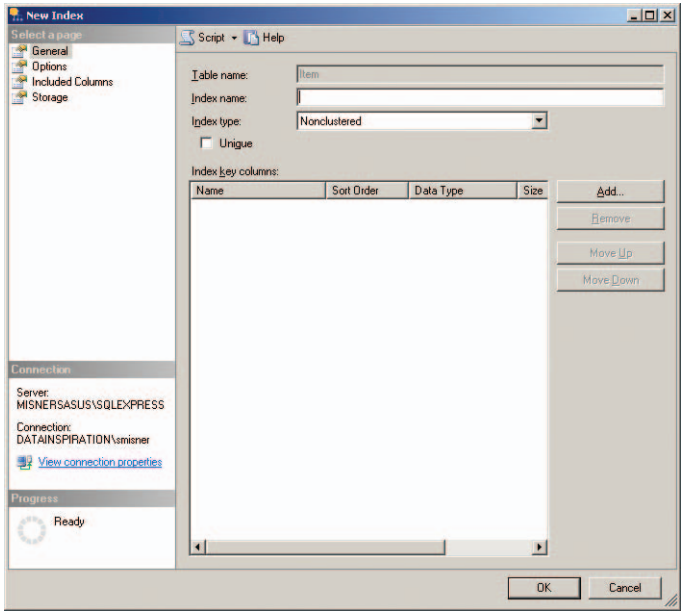



*Figure 3-43*
*New Index dialog box*

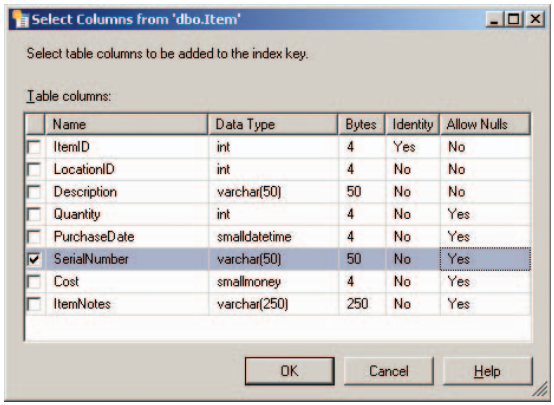



*Figure 3-44*
*Select Columns From 'Dbo.Item' dialog box with selection of SerialNumber as index key*

In the New Index dialog box (Figure 3-45), you can see the column added to the list.

**4** Let's explore the other pages in this dialog box. Click **Options** to view the page, as shown in Figure 3-46.

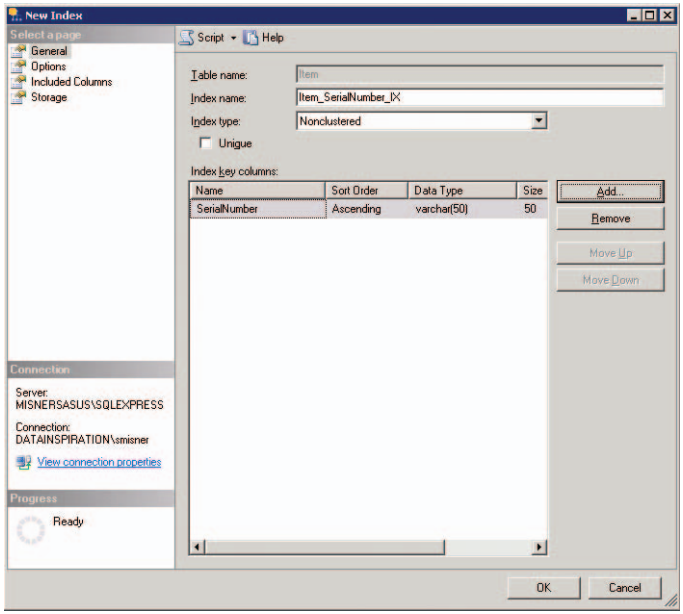



*Figure 3-45*
*New Index dialog box with a column added*

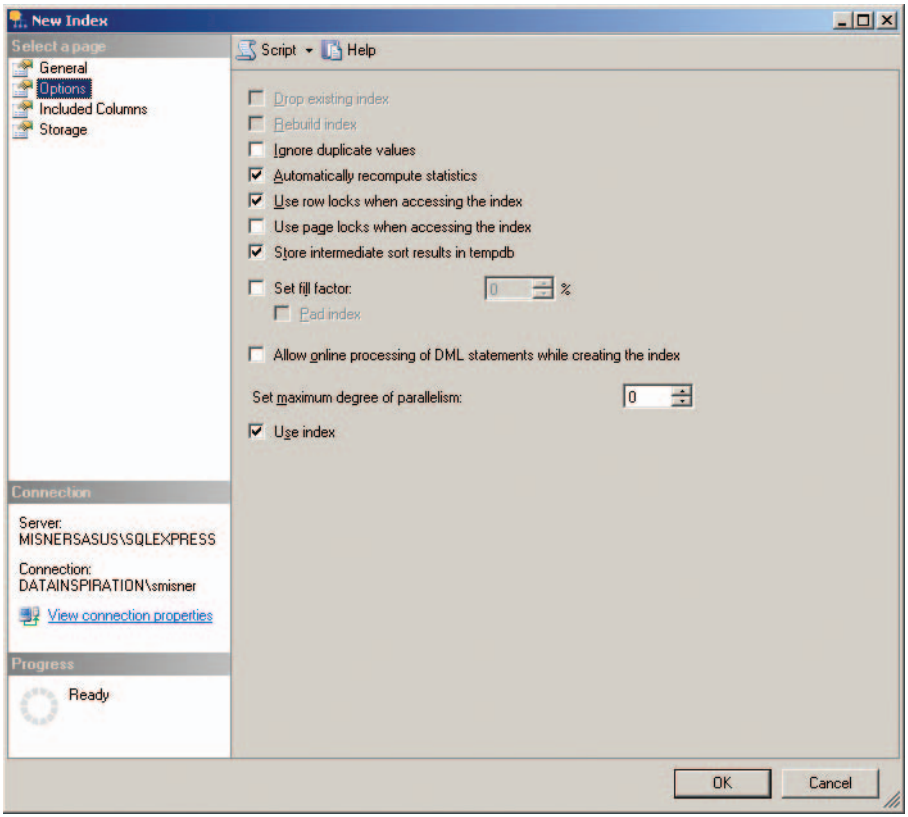



*Figure 3-46*
*The Options page of the New Index dialog box*

As with database options, the defaults for index options can usually be left as they are. These options affect the server resources by controlling the resources available during index creation and during queries that use the index.

**5** Click **Included Columns** to view the page, as shown in Figure 3-47.

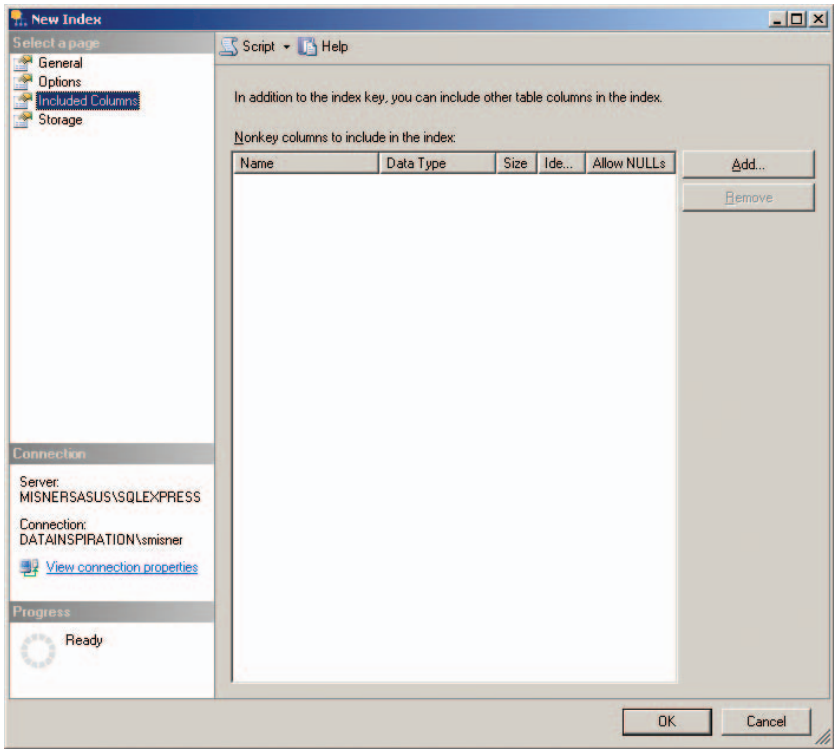



*Figure 3-47*
*The Included Columns page of the New Index dialog box*

When you use a nonclustered index, you can also specify other columns to include in the index in addition to the key columns you've already added. This approach is a good idea when you can anticipate the columns you expect to request in a query. If all of these columns are in the index, the database engine doesn't have to read the table and can return the query results much more quickly. Let's add the Description column as an included column because your queries based on serial number are likely to return the item description.

6 Click **Add**. In the Select Columns From 'Dbo. Item' dialog box, select the check box to the left of Description, as shown in Figure 3-48, and click **OK**.

7 Click **Storage** to view the page, as shown in Figure 3-49.

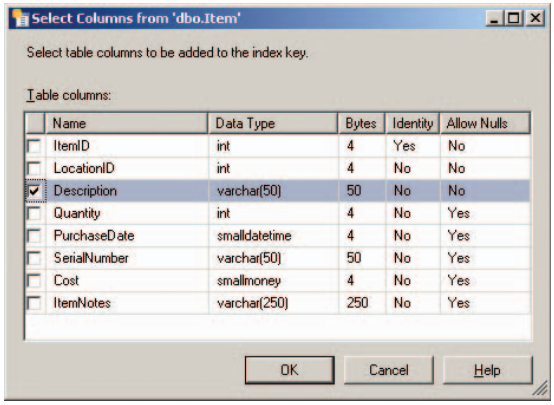



*Figure 3-48*
*Select Columns From 'Dbo.Item' dialog box with selection of Description as included column*

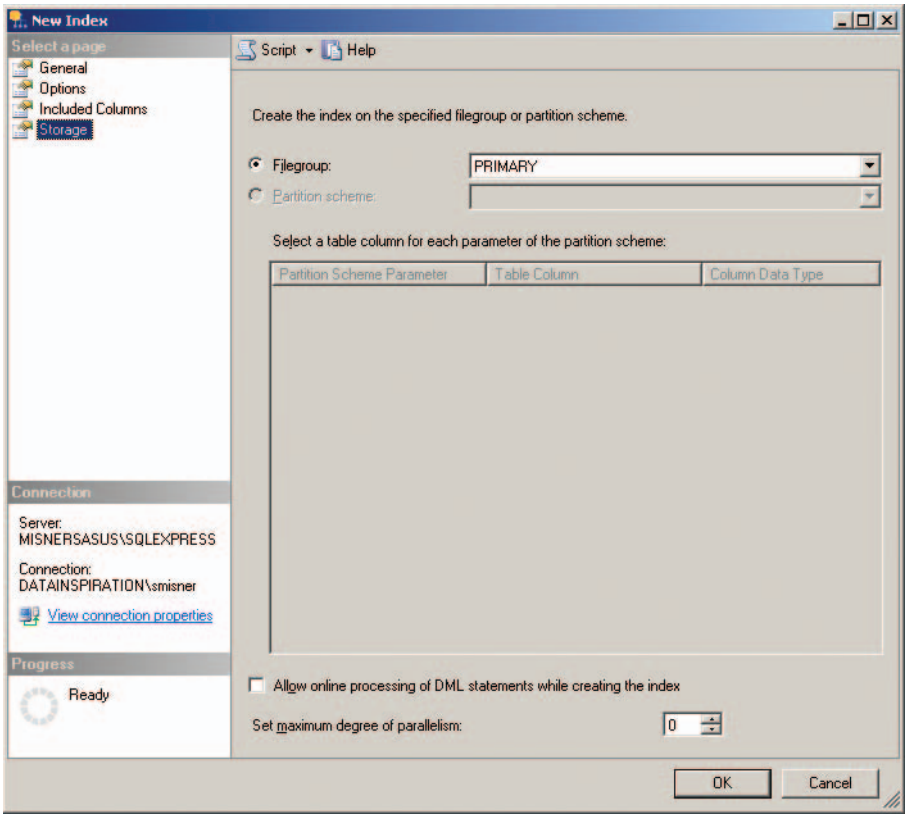



*Figure 3-49*
*The Storage page of the New Index dialog box*

If your database has more than one filegroup, you can select the filegroup in which the index data will be stored.

**8** Click **OK** to close the dialog box.

## Entering Data into Your Tables

Of course, the whole reason to build a database is to store data. Now that you've got the database built, you're ready to start entering data. In this section, I'll show you how to use Management Studio for data entry. In the next chapter, I'll show you how to use Transact-SQL to load data in your tables. As you enter data, you'll be able to test the constraints you've added and see what happens when data violates a constraint. To add data to your tables, follow the steps below.

### TO ADD DATA TO TABLES

1. Let's test out the foreign key relationship between the Item and Location tables first. Because the Location table is currently empty, you shouldn't be able to add a row to the Item table. In Object Explorer, in the Databases folder, expand the HomeInventory folder, expand the Tables folder, right-click **dbo.Item**, and select **Open Table**. The Table window displays, as shown in Figure 3-50.

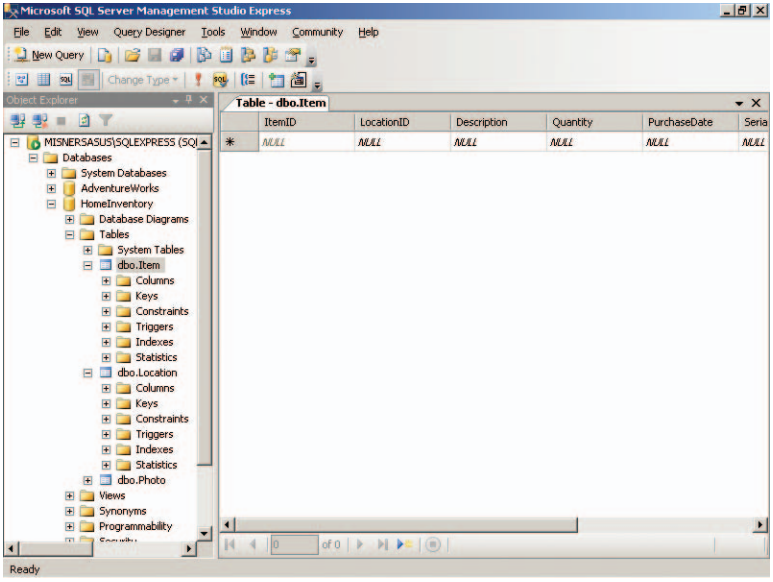



*Figure 3-50*
*Table window*

2. Notice the ItemID column is grayed out to let you know that you can't type a value in this column. Click in the LocationID column, type **1**, and then press the **Tab** key to move the cursor to the next column. Don't be alarmed by the exclamation point in a red circle that displays. It's just an indicator that data has changed in the grid, but nothing has actually been stored in the database yet. You can hover the cursor over the icon to view its message, as shown in Figure 3-51.

Table - dbo.Item

| | ItemID | LocationID | Description | Quantity | PurchaseDate |
|---|---|---|---|---|---|
| | NULL | 1 | NULL | NULL | NULL |
| * | NULL | NULL | | | |

This Cell has changed.
The change has not been commited to the database.
The original Data is NULL.

*Figure 3-51*
*The LocationID column indicating data is changed*

**3** In the **Description** column, type **Television**. Press **Tab** to continue to the next column. Continue typing data into each column until the grid looks like the one shown in Figure 3-52. In the PurchaseDate column, just type **3/21/2000**; the time is added automatically for you.

Table - dbo.Item

| | LocationID | Description | Quantity | PurchaseDate | SerialNumber | Cost | ItemNotes |
|---|---|---|---|---|---|---|---|
| | 1 | Television | 1 | 3/21/2000 12:0. | XJ1394A2 | 500 | NULL |
| * | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

*Figure 3-52*
*Table window with one row of data*

**4** When you're finished, press **Tab** to go to the next row. An error message displays, as shown in Figure 3-53.

*Figure 3-53*
*FOREIGN KEY constraint error from INSERT*

**5** To fix the problem, you need to add a row to the Location table. First, click **OK** to close the message box. You can leave the Item table open while you fix the Location table. In the HomeInventory folder, under the Tables folder, right-click **dbo.Location** and select **Open Table**. Type data into the grid to match the one shown in Figure 3-54. Press **Tab** to save the current row and move to the next row.

Because LocationID is defined as an identity column, a value is assigned to this column when you save the row.

Table - dbo.Location

| | LocationID | Location |
|---|---|---|
| | 1 | Living Room |
| * | NULL | NULL |

*Figure 3-54*
*The Location table with one row of data*

**6** Switch back to the Item table and press **Tab** to save the current row and move to the next row. The row is added successfully and the ItemID is assigned a value of 2.

When you attempted to save the item in a previous step, the identity value was 1. Even though the row was not saved, the identity value originally assigned to the row is no longer available. When you defined the identity column in this table, you accepted the default Identity Increment of 1, which means each attempt to insert a row increases the next identity value by 1 and ignores any identity values that couldn't be used previously. Remember that the identity column is used to keep each row in the table unique, so don't worry if you wind up with gaps in the series of identity values in your table.

**7** Now, you're going to see what happens when you try to delete a row in a parent table when related rows exist in a child table. Switch to the Location table and click the box to the left of the LocationID column in the first row to highlight the entire row. Now, right-click the row and select **Delete**, as shown in Figure 3-55.

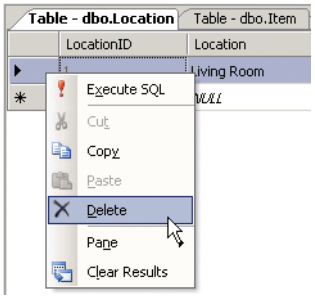



*Figure 3-55*
*Deleting a table row*

**8** Click **Yes** in the message box to confirm the deletion. An error resulting from the FOREIGN KEY constraint displays, as shown in Figure 3-56. Click **OK** to close the message box.

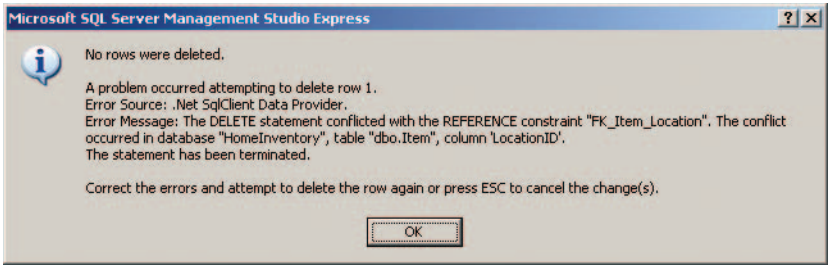



*Figure 3-56*
*FOREIGN KEY constraint error from DELETE*

You now have hands-on experience with FOREIGN KEY constraints. You can't add a row to a child table if the parent table has no corresponding row, and you can't delete a row in the parent table if a child table has related rows. The FOREIGN KEY constraint is upheld whether you use Management Studio to enter data or use an application to add or delete rows.

**9** Let's test the CHECK constraint now. Switch to the Item table. On the second row, type data into the grid so your screen looks like the one shown in Figure 3-57.

Table - dbo.Item

| ItemID | LocationID | Description | Quantity | PurchaseDate | SerialNumber | Cost | ItemN |
|---|---|---|---|---|---|---|---|
| 2 | 1 | Television | 1 | 3/21/2000 12:0... | XJ1394A2 | 500.0000 | NULL |
| NULL | 1 | Couch | 1 | 7/28/2005 12:0... | NULL | -2500 | NULL |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

*Figure 3-57*
*The Item table with a negative value in the Cost column*

**10** Now, press **Tab** twice to save the row. Because the Cost column contains a negative value in violation of the CHECK constraint you added earlier, an error message (Figure 3-58) displays. Click **OK** to close the message box. Press **Esc** to clear the data you have entered and then close the Item and Location tables.

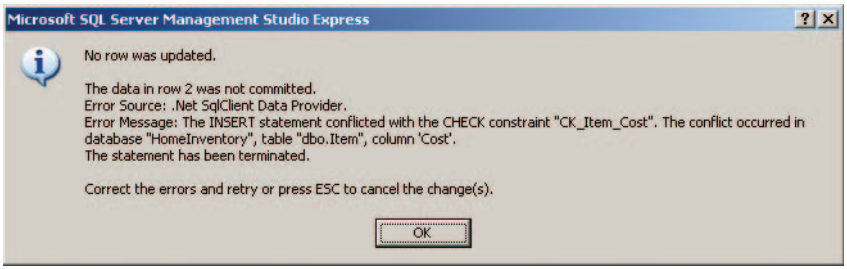



*Figure 3-58*
*CHECK constraint from INSERT*

You've now run the gamut of things you commonly need to do when creating a new database, including adding data to tables and testing the constraints. In the next chapter, you'll see how to do many of these same tasks by writing Transact-SQL statements instead of using the Management Studio interface.

## In Summary...

You have covered a lot of ground in this chapter. You started by learning how to use Management Studio to view database metadata and table contents. As a result of your exploration, you now know SQL Server uses system databases to manage server and database operations. You also learned how to attach an existing database to your SQL Server instance.

Throughout most of the chapter, you concentrated on using the Management Studio interface features to build a database and database objects. After creating the database, you added three tables to your home inventory database and learned how to set the column data type (and why selecting the right data type is important), configure an identity column, and specify a primary key. Then you added a foreign key relationship, a CHECK constraint, and a UNIQUE constraint to protect data integrity in your database. To top off the construction of the database, you added an index to the database to enhance query performance. In Chapter 4, you'll learn how to modify these objects and how to use Transact-SQL to add other database objects.

Usually an application is used to add, update, or delete data, but you learned it's pretty easy to use Management Studio to work with table data. To test out the constraints you added, you entered erroneous data and confirmed the constraints work as expected. In the next chapter, you'll learn other ways to make data changes to your tables.

**TIP**

**You can find videos covering SQL Server 2005 Express Edition for beginners at *msdn.microsoft.com/vstudio/express/sql/learning/default.aspx*. These videos are a great way to reinforce the concepts you're learning in this book. Lesson 8 shows you how applications can interact with databases and might inspire you to build your own application as an interface for your home inventory database.**

Chapter 4

# Working with Transact-SQL

*Working with Data, 90*

*Working with Database Objects, 124*

So far, so good, right? Working with databases in SQL Server Express is pretty easy. At this point, you're able to do quite a bit. From now on, you could successfully create and work with database objects by using Management Studio exclusively, but you might find it easier to perform certain tasks if you know how to use Transact-SQL (T-SQL). Even if you plan to try your hand at programming an application later to work with your database, you'll still need to learn T-SQL because it's the only language the SQL Server database engine understands. T-SQL is Microsoft's version of *structured query language* (SQL), a language used to work with data in relational databases. Microsoft created T-SQL to support more functionality in the SQL Server family than the SQL standard adopted by the American National Standards Institute (ANSI). Most popular database products use a variant of the SQL standard, so learning T-SQL now can give you a head start on other types of databases you might work with in the future.

In this chapter, I'll introduce you to the basics of T-SQL by showing you how to use it as a query language to get data into and out of a database. I'll also show you how to make changes to data already in the database and how to use script files to save the queries you develop so you can use them again later without having to rewrite them from scratch. T-SQL does much more than query a database. With it, you can also create database objects, such as databases, tables, or constraints, to name only a few things you can do with the language. Of course, you can also make changes to existing objects or delete those you no longer need.

**NOTE**

**You might be curious to know how to pronounce SQL. Officially, you should say each letter individually, but most developers I know pronounce it like *sequel*.**

All sorts of languages are used in the computer world, each of which is specialized for certain types of activities. Programming languages like Microsoft Visual Basic .NET or Microsoft Visual C# .NET are object-oriented languages that describe how to send a message to an object, how an object receives messages, and how to process data or to perform calculations. T-SQL is different from these languages because it's a *data manipulation language*, which means you use it to add, change, or delete data in a database and to retrieve data from a database.

**TIP**

**You can use SQL Server 2005 Books Online to learn more about T-SQL. Start with the topic Transact-SQL Reference at *msdn2.microsoft.com/en-us/library/ms189826.aspx* and use the associated table of contents to locate more specific topics.**

Using T-SQL statements, you issue commands that start with a verb to the database engine, such as *INSERT*, *UPDATE*, *DELETE*, or *SELECT*. The type of verb you use determines the *syntax* required for the statement. Syntax is the set of rules defining the keywords you can use in T-SQL and the way you can combine these keywords to form a valid statement. Unlike most spoken languages, like English, which give you some flexibility in selecting and organizing words in sentences, T-SQL requires you to construct statements according to specific rules. The best way to learn syntax is to practice creating and executing statements, so let's get started.

## Adding New Rows to a Table

At this point, you don't have much data in your tables, so you'll start learning T-SQL by adding more rows to each table. The T-SQL command to add data to a table, you might have guessed, is *INSERT*. The syntax for the INSERT statement looks like this:

```
INSERT INTO table (column1, column2, ...) VALUES (value1, value 2, ...)
```

You aren't required to capitalize words in this statement. It's just easier to distinguish the T-SQL keywords in a statement when you capitalize them. When you use this statement, you need to specify the table to store the new rows, a list of columns in the table, and a list of values that corresponds to the column list.

**NOTE**

**The list of columns isn't really required in the INSERT statement, but it is easier for you to remember later what you're trying to accomplish if you include it. If you omit the column list, be sure the list of values is in the correct order.**

**If you do include the column list, which I highly recommend, you should know the order of columns in this list doesn't have to match the order of columns in the table. However, if you decide to change the column order in the INSERT statement, you need to be sure the list of values follows the same order you use for the column list.**

## TO USE THE INSERT STATEMENT

1. If Management Studio is not already open, click **Start**, point to **All Programs**, point to **Microsoft SQL Server 2005**, and click **SQL Server Management Studio Express**.

2. You can take advantage of the Management Studio Express INSERT template to jump-start a new INSERT statement rather than typing it yourself from scratch. Because you have defined a foreign key constraint between the Location and Item tables, you'll need to start inserting rows into Location first, as you learned when adding rows in the previous chapter. To use the template, expand the **Tables** folder in the HomeInventory database folder, right-click **dbo.Location**, point to **Script Table As**, point to **INSERT To**, and click **New Query Editor Window**.

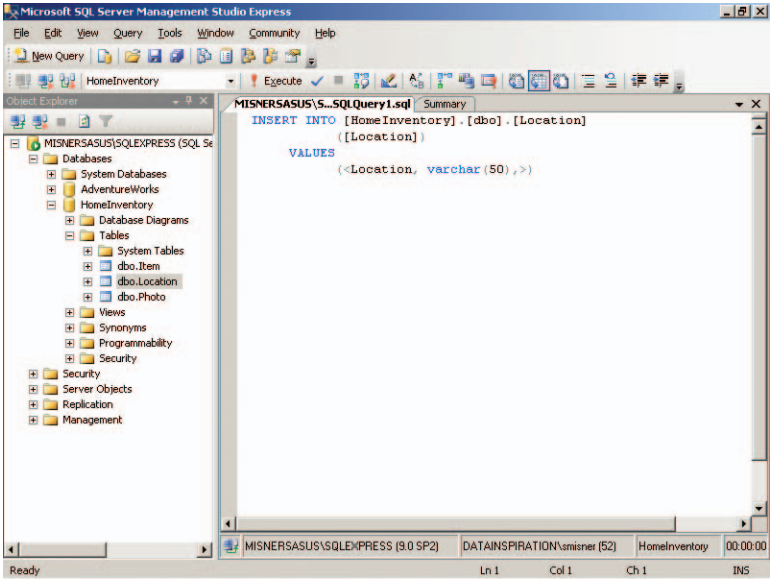



*Figure 4-1*
*INSERT template*

The INSERT statement is partially completed for you. The name of the table and the table columns are provided for you.

Did you notice the primary key column is missing from the list of columns created by the template? Because LocationID is an identity column, a value is created when the row is saved. Therefore, there's no need to name the column in the INSERT statement.

3 All you need to do now is to type in the value of the column for the row you want to add. The brackets located inside the parentheses tell you that the text between the brackets is simply a placeholder for a column's value. The data type for that column is also included with the placeholder to help you input a value of the correct data type now before you find out there's a problem later when you try to execute the statement. Modify the text inside the parentheses to match the following statement:

**NOTE**

**You can type a T-SQL statement entirely on one line, or you can separate the statement into multiple lines as shown in the example to the right. If you use multiple lines, the statement can be easier to read.**

```
INSERT INTO [HomeInventory].[dbo].[Location]
      ([Location])
   VALUES
      ('Dining Room')
```

This form of the INSERT statement allows you to insert only one row at a time. At this point, you could execute this statement to insert the row for dining room, but you'll find it easier to include multiple statements in the Query Editor to execute each statement consecutively.

Notice the table name created by the template isn't simply the table name. It also includes the database name, HomeInventory, and the schema name, dbo. This method of naming a table ensures you insert a row into the intended table. SQL Server lets you create multiple tables with the same name in the same database only if you use different schema names for each table. This capability is useful when you need the same table structure for different purposes, such as when you want to provide a separate table to individual users within the same database. In this example, each user's table would have the same name, but the schema name associated with the table would be unique for each user. If you don't create schemas explicitly, the default schema is dbo.

**NOTE**

**Brackets to enclose the database, schema, or table names are required only under certain conditions. For complete details, search for the topic "Delimited Identifiers" in SQL Server 2005 Books Online.**

When you use a string value in a T-SQL statement, such as Dining Room in this example, you must enclose the string in single quotes. Numeric values don't require any special treatment.

**NOTE**

**There is a variation of the INSERT statement to insert multiple rows when you can retrieve data from an existing table. To learn how to use this variation, search for the topic "Adding Rows by Using INSERT and SELECT INTO" in SQL Server 2005 Books Online.**

4 Right-click **dbo.Location**, point to **Script Table As**, point to **INSERT To**, and click **Clipboard**. Now click behind the end of the statement in the Query Editor, press **Enter**, and then press **Ctrl+V** to paste the clipboard contents into the window. Then change the text inside the VALUES clause to match the following statement:

```
INSERT INTO [HomeInventory].[dbo].[Location]
      ([Location])
   VALUES
      ('Kitchen')
```

5 Repeat step 4 to add a row for the master bedroom and the garage, as shown in Figure 4-2.

```
INSERT INTO [HomeInventory].[dbo].[Location]
            ([Location])
      VALUES
            ('Dining Room')
INSERT INTO [HomeInventory].[dbo].[Location]
            ([Location])
      VALUES
            ('Kitchen')
INSERT INTO [HomeInventory].[dbo].[Location]
            ([Location])
      VALUES
            ('Master Bedroom')
INSERT INTO [HomeInventory].[dbo].[Location]
            ([Location])
      VALUES
            ('Garage')
```

*Figure 4-2*
*INSERT statements for all location rows*

**TIP**

**You can, of course, simply copy and paste the INSERT statement to make another copy or you can type the INSERT statement yourself. Personally, I like to minimize the amount of typing I do to ensure I don't introduce an error from typing the statement incorrectly.**

6 Now you're ready to execute all INSERT statements at once. Click **Execute** in the SQL Editor toolbar. The INSERT statements execute sequentially, beginning with the first statement and continuing through the last statement. A message to indicate the success of each insertion displays in the Messages pane of the Query Editor (Figure 4-3).

```
Messages

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)
```

*Figure 4-3*
*Successful query execution of INSERT statements*

7 In Chapter 3, "Creating Your First Database," you learned how to use Management Studio to view the contents of a table. Open the Location table now to double-check the results of the INSERT statements (Figure 4-4).

Table - dbo.Location | MISNERSASUS\...

| | LocationID | Location |
|---|---|---|
| ▶ | 2 | Dining Room |
| | 5 | Garage |
| | 3 | Kitchen |
| | 1 | Living Room |
| | 4 | Master Bedroom |
| * | NULL | NULL |

*Figure 4-4*
*Viewing rows in the Location table*

**NOTE**

**In the next section of this chapter, you'll learn how to use T-SQL to read data in a table as an alternative to opening the table in Management Studio.**

8 Close the Location table and the Query Editor and, when prompted to save your changes, click **Yes**. In the File Name box, type **Insert Location Rows** and click **Save**.

Your T-SQL statements are now saved to a file as a T-SQL *script*. By using T-SQL scripts, you can avoid retyping statements later when you want to perform the same tasks.

9 To start a new query, click **New Query** in the standard toolbar. Because rows now exist in the Location table for each room, you can add rows to the Item table without violating the foreign key constraint as long as you reference valid locations in the LocationID column of each item row. Use the steps you learned in this section to add rows to the Item table, as shown in the following series of INSERT statements:

```
INSERT INTO [HomeInventory].[dbo].[Item]
      ([LocationID]
      ,[Description]
      ,[Quantity]
      ,[PurchaseDate]
      ,[SerialNumber]
      ,[Cost]
      ,[ItemNotes])
   VALUES
      (2
      ,'Dining Table'
      ,1
      ,'10/4/2003'
      ,NULL
      ,1000
      ,NULL)
```

```
INSERT INTO [HomeInventory].[dbo].[Item]
       ([LocationID]
       ,[Description]
       ,[Quantity]
       ,[PurchaseDate]
       ,[SerialNumber]
       ,[Cost]
       ,[ItemNotes])
   VALUES
       (1
       ,'Bookshelf'
       ,2
       ,'5/10/2002'
       ,NULL
       ,200
       ,NULL)
INSERT INTO [HomeInventory].[dbo].[Item]
       ([LocationID]
       ,[Description]
       ,[Quantity]
       ,[PurchaseDate]
       ,[SerialNumber]
       ,[Cost]
       ,[ItemNotes])
   VALUES
       (1
       ,'Couch'
       ,1
       ,'7/28/2005'
       ,NULL
       ,2500
       ,NULL)
```

**NOTE**

**If, for some reason, the identity values in your Location table don't match the identity values shown in Figure 4-4, you'll need to modify the INSERT statements for the Item table. Specifically, be sure to replace the LocationID for the dining table in the first INSERT statement with the LocationID for Dining Room shown in your table. In the remaining statements, use the LocationID for Living Room in your table.**

**10** Next, click **Execute**. If you performed all previous steps correctly, you should see the message *(1) row(s) affected)* display three times in the Messages pane of the Query Editor window.

**11** Double-check the results of the previous INSERT statements by opening the Item table (Figure 4-5). Take note of the identity values assigned to each item now, because you'll need to reference these values when you add rows to the Photo table.

Table - dbo.Item

| | ItemID | LocationID | Description | Quantity | PurchaseDate | SerialNumber | Cost | ItemNotes |
|---|---|---|---|---|---|---|---|---|
| | 2 | 1 | Television | 1 | 3/21/2000 12:0... | XJ1394A2 | 500.0000 | NULL |
| | 4 | 2 | Dining Table | 1 | 10/4/2003 12:0... | NULL | 1000.0000 | NULL |
| | 5 | 1 | Bookshelf | 2 | 5/10/2002 12:0... | NULL | 200.0000 | NULL |
| | 6 | 1 | Couch | 1 | 7/28/2005 12:0... | NULL | 2500.0000 | NULL |

*Figure 4-5*
*Viewing rows in the Item table*

**12** Because the script currently includes only three of the four items you now have in the Item table, you should update the script to include the item you added manually by appending the following statement:

```
INSERT INTO [HomeInventory].[dbo].[Item]
       ([LocationID]
       ,[Description]
       ,[Quantity]
       ,[PurchaseDate]
       ,[SerialNumber]
       ,[Cost]
       ,[ItemNotes])
   VALUES
       (1
       ,'Television'
       ,1
       ,'3/21/2000'
       ,'XJ1394A2'
       ,500
       ,NULL)
```

**13** Close the Item table and the Query Editor. When you're prompted to save your changes, click **Yes** and save to the **Insert Item Rows.sql** file.

**IMPORTANT**

**Don't skip step 13! In the section titled "Deleting Rows from a Table" later in this chapter, you'll delete the items you just inserted to practice using the DELETE statement. You'll be able to insert rows back in to the Item table without retyping your work if you save the T-SQL statements as a script now.**

**14** Click **New** to start a new query. You're ready now to create INSERT statements that add rows to the Photo table in your database, as shown in Table 4-1. Don't forget to use NULL for PhotoNotes where the value is unknown and not provided in the table.

| ItemID | PhotoLocation | PhotoCaption | PhotoNotes |
|---|---|---|---|
| 2 | C:\HomeInventory\P100001.JPG | Television – Front | |
| 2 | C:\HomeInventory\P100002.JPG | Television – Back | Notice permanent etching of identification number on back of TV |
| 4 | C:\HomeInventory\P126010.JPG | Side view of dining table | |
| 5 | C:\HomeInventory\P110021.JPG | Bookshelf units | |

*Table 4-1*
*Photo Table*

**NOTE**

**Remember from Chapter 1, "Introducing Microsoft SQL Server 2005 Express Edition," that database management systems don't work efficiently with images, so it's better to use URLs or a combination of folder and file name to reference the location of an image. In this example, you don't really need to have images stored in the locations shown in the PhotoLocation column because SQL Server Express won't try to confirm the images are indeed there.**

**15** After creating an INSERT statement for each Photo row, click **Execute**. You should see the message *(1) row(s) affected)* display four times in the Messages pane of the Query Editor window.

**16** Open the Photo table (Figure 4-6) to verify the results. I've widened some of the columns to get a better view of the value in each column.

Table - dbo.Photo

| PhotoID | ItemID | PhotoLocation | PhotoCaption | PhotoNotes |
|---|---|---|---|---|
| 1 | 2 | C:\HomeInventory\P100001.JPG | Television - Front | NULL |
| 2 | 2 | C:\HomeInventory\P100002.JPG | Television - Back | Notice permanent etching of identification number on back of TV |
| 3 | 4 | C:\HomeInventory\P126010.JPG | Side view of dining table | NULL |
| 4 | 5 | C:\HomeInventory\P110021.JPG | Bookshelf units | NULL |

*Figure 4-6*
*Viewing rows in the Photo table*

**NOTE**

**If the identity values in your Item table don't match the identity values shown in Figure 4-5, use the PhotoCaption column in Table 4-1 to identify an item and locate the correct ItemID for that item in your Item table.**

**17** Close the Photo table and the Query Editor. When you're prompted to save your changes, click **Yes** and save to the **Insert Photo Rows.sql** file.

## Viewing the Contents of a Table

When you want to view the data in a table, you use the SELECT statement. In fact, even if you use Management Studio to open a table, a SELECT statement is generated and executed for you. The SELECT statement is the most versatile of all T-SQL statements you'll use because you can choose to view data in so many different ways. We'll examine many of these variations in the following sections of this chapter, but first let's start with the basic syntax of the SELECT statement, which looks like this:

```
SELECT (column1, column2, ...) FROM table
```

### TO USE THE SELECT STATEMENT

**TIP**

**By selecting the database you plan to query, you are preparing the Query Editor to work with a specific database and saving yourself a step later.**

1. In Object Explorer, expand the **Databases** folder and click **HomeInventory**.

2. To start a new query, click **New Query**. In the Query Editor, type the following statements:

```
SELECT * FROM Location
SELECT * FROM Item
```

**NOTE**

**Notice the toolbar displays HomeInventory in the Available Databases drop-down list. HomeInventory appears here because you selected that database in the previous step. You can also select a different database in the drop-down list, but this selection changes the context of the queries. In that case, the *EXECUTE* command would return an error if the tables used in the SELECT statements aren't found in the selected database. Another way to ensure the database context is correct without including the name of the database with the table name, such as HomeInventory.dbo. Location, is to precede the SELECT statements with a USE statement, like this: `USE HomeInventory`.**

Instead of typing in a column list, you can use an asterisk to indicate you want to retrieve all columns in the table. It's considered best practice, however, to list each column if you plan to save the query in a script or build the query into an application. Using this approach, you can ensure the structure of the query results remains consistent even if new columns are added to the table later. Because we're not saving this query, it's okay to use the asterisk.

3. Even when you have multiple statements in a single Query Editor, you can execute each one individually by highlighting a complete statement before you use the *EXECUTE* command. Let's try it now. Highlight the first query, `SELECT * FROM Location`, and then click **Execute** to see the rows for the Location table only (Figure 4-7), even though two SELECT statements are in the Query Editor. The rows returned for the query are also known as a *query result set*.

Isolating statements for execution makes it easier to work with multiple queries in a single Query Editor without having to execute each one every time.

4 Now clear the highlighting so that you can execute both statements consecutively. Click **Execute** to view the contents of both tables in the Results pane (Figure 4-8).

The Results pane can display more than one result set. If the result set has more rows than can fit on the screen, a vertical scroll bar displays to enable you to view each result set independently.

## Combining the Contents of Multiple Tables

Although you can use multiple SELECT statements to retrieve data from several tables and display the results on a single page, often you'll prefer to combine the results so that the data looks like it comes from a single table. To achieve this sleight of hand, you use a single SELECT statement and add a JOIN clause to link the tables together. It's very common to join tables that have a foreign key relationship, but it's not necessary if there is a way to compare values from columns in each table. A SELECT statement that includes a JOIN clause looks like this:

```
SELECT (column1, column2, ...) FROM table1
join_type table2 ON join_condition
```

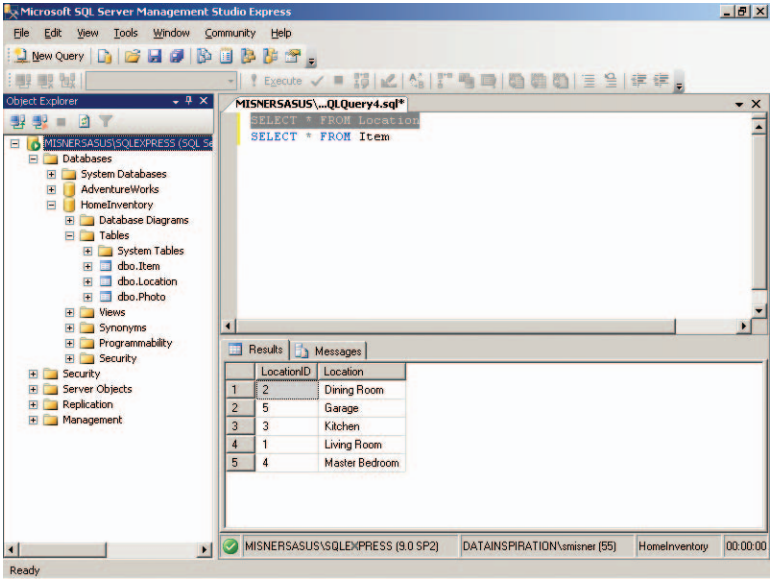



*Figure 4-7*
*Executing a single T-SQL statement*

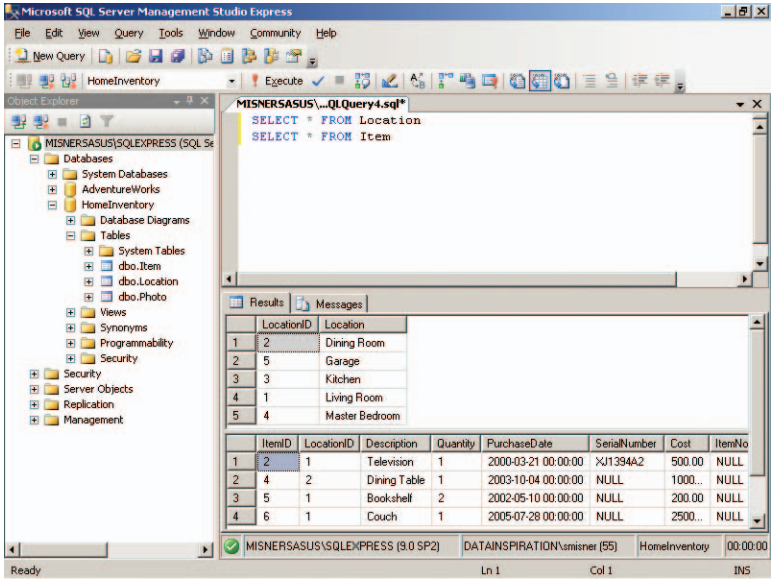



*Figure 4-8*
*Viewing results of multiple SELECT statements*

You can use different types of joins: inner joins, outer joins, full joins, and self-joins. Understanding the differences between these join types is easier when you see the results. In this book, I won't review the intricacies of each join type, but I will use the following procedure to help you explore the joins you're most likely to use with the home inventory database.

## TO USE THE JOIN CLAUSE

**1** Even though the number of rows in the Location and Item tables is small enough to see the query result set of both SELECT statements in the same window, you'll find it useful to combine related data into a single result set by using a join. Replace the two SELECT statements in the Query Editor window with the following statement:

```
SELECT * FROM Location JOIN Item ON Location.LocationID = Item.LocationID
```

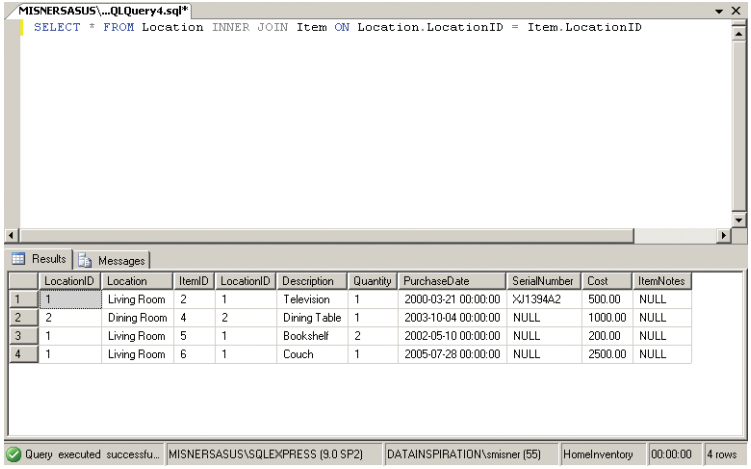
This example illustrates the simplest type of join, which matches up rows in two tables when the specified join condition is true. The join condition shown here requires the LocationID in the Location table to match the LocationID in the Item table. This type of join is also known as an inner join and can be written like this:

```
SELECT * FROM Location INNER JOIN Item ON Location.LocationID = Item.LocationID
```

**2** Click **Execute** to see the results shown in Figure 4-9.

**TIP**

**You can also abbreviate the table names in the join condition by using an alias, which is handy when you have long table names. An alias can be a complete word, a single character, or a combination of letters and numbers. For example, I can rewrite the SELECT statement shown in the first step like this: `SELECT * FROM Location l JOIN Item i on l.LocationID = i.LocationID`.**




| | LocationID | Location | ItemID | LocationID | Description | Quantity | PurchaseDate | SerialNumber | Cost | ItemNotes |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Living Room | 2 | 1 | Television | 1 | 2000-03-21 00:00:00 | XJ1394A2 | 500.00 | NULL |
| 2 | 2 | Dining Room | 4 | 2 | Dining Table | 1 | 2003-10-04 00:00:00 | NULL | 1000.00 | NULL |
| 3 | 1 | Living Room | 5 | 1 | Bookshelf | 2 | 2002-05-10 00:00:00 | NULL | 200.00 | NULL |
| 4 | 1 | Living Room | 6 | 1 | Couch | 1 | 2005-07-28 00:00:00 | NULL | 2500.00 | NULL |

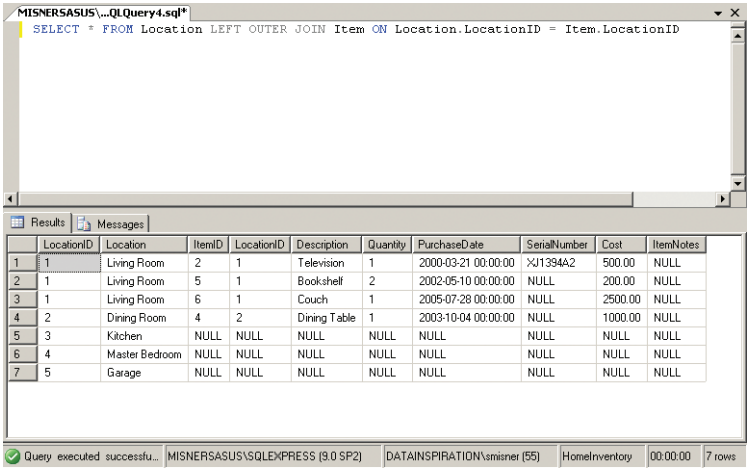
*Figure 4-9*
*Combining tables by using a join*

The database engine compares the LocationID in each row in the Location table to the LocationID in each row in the Item table. When the two values are equal, the join condition evaluates as true, and the matched rows are combined to appear as if they were a single row. Because the Location table contains two columns and the Item table contains eight columns, the query result contains ten columns.

Notice that rows from the Location table can appear more than once in the query result set, but rows in the Item table appear only once. In all cases, the LocationID columns in both tables are identical because the join condition required equality. You can use other operators in a join condition, such as less than (<), less than or equal to (<=), greater than (>), or greater than or equal to (>=), but these are used in advanced queries that we won't cover in this book.

**3** Did you also notice the locations missing from the query result set—the ones that don't have items associated with them yet? Specifically, Kitchen and Master Bedroom were excluded from the query result because no items contained a matching LocationID. If you want the query result set to include all locations, even those without associated items, you can use another type of join—the left outer join. Replace the statement in the Query Editor with the following statement and execute the statement to see the results as shown in Figure 4-10:

```
SELECT * FROM Location LEFT OUTER JOIN Item ON Location.LocationID = Item.LocationID
```




| | LocationID | Location | ItemID | LocationID | Description | Quantity | PurchaseDate | SerialNumber | Cost | ItemNotes |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Living Room | 2 | 1 | Television | 1 | 2000-03-21 00:00:00 | XJ1394A2 | 500.00 | NULL |
| 2 | 1 | Living Room | 5 | 1 | Bookshelf | 2 | 2002-05-10 00:00:00 | NULL | 200.00 | NULL |
| 3 | 1 | Living Room | 6 | 1 | Couch | 1 | 2005-07-28 00:00:00 | NULL | 2500.00 | NULL |
| 4 | 2 | Dining Room | 4 | 2 | Dining Table | 1 | 2003-10-04 00:00:00 | NULL | 1000.00 | NULL |
| 5 | 3 | Kitchen | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 6 | 4 | Master Bedroom | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 7 | 5 | Garage | NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

*Figure 4-10*
*Using a left outer join*

Knowing when there are no rows in the second table related to the first table can be just as informative as knowing when there are one or more related rows. By using a left outer join, the query result set will include all rows in the first table listed in the FROM clause of your SELECT statement.

**TIP**

**Less commonly used are the right outer join and full outer join. You use the right outer join to display rows in the second table that don't match up with rows in the first table. Because referential integrity is currently enforced in your tables, you should never have a situation in which an Item exists without a corresponding Location. Going one step further, a full outer join combines the functionality of the left outer join and the right outer join. That is, all rows in both tables are included in the query result set regardless of a match in the other table. This technique is useful when you're trying to uncover and fix data integrity issues in tables. However, as I just noted, because your tables should be nice and clean at this point, you don't need to use a full outer join.**

4 All columns from both tables in the FROM clause are included in the query result set. Unless you specify otherwise, the columns from the first table display first, and then the columns from the second table display. You can control both the inclusion of columns and the order in which they display by explicitly defining the column list in the SELECT statement instead of using an asterisk. For example, including LocationIDs isn't necessary because they don't have any meaning outside of the tables containing them. Revise the statement as shown here and then execute it to see the query results shown in Figure 4-11:

```
SELECT Location, Description, Quantity, PurchaseDate, Cost FROM Location LEFT OUTER JOIN Item
ON Location.LocationID = Item.LocationID
```

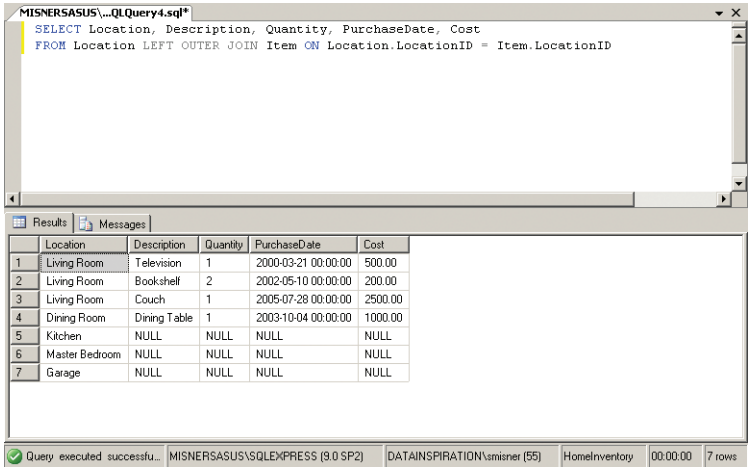



*Figure 4-11*
*Using a column list in a SELECT statement*

**TIP**

**If you have the same column name in both tables, you must include the table name with the column name. You can use either the full table name or the alias if you have created one. For example, if you want to include the LocationID from the Location table in the query result set, you must include Location.LocationID or l.LocationID in the column list. Otherwise, you'll get a message about an ambiguous column, and the query will not execute.**

5 The order of the columns in the query set is determined by the order you specify when naming the columns in your query. You don't have to show columns in the same order they exist physically in the table, and you don't have to keep columns from the same table together. Try executing the following statement to see how you can change the column order of the query results, as shown in Figure 4-12:

```
SELECT Location, PurchaseDate, Quantity, Description, Cost FROM Location LEFT OUTER JOIN Item
ON Location.LocationID = Item.LocationID
```

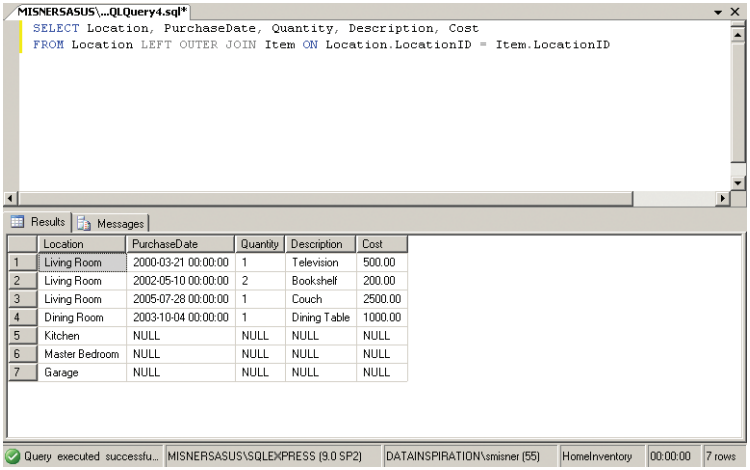



*Figure 4-12*
*Rearranging columns in a SELECT statement*

**6** Whereas a join can only be defined between two tables, it's perfectly fine to use multiple joins in a single SELECT statement. When working with multiple joins, you really need to consider whether using inner joins or outer joins will give you the results you want. Execute the following three statements to compare the query results:

```
SELECT Location, PurchaseDate, Quantity, Description, Cost, PhotoLocation, PhotoCaption
FROM Location
LEFT OUTER JOIN Item ON Location.LocationID = Item.LocationID
LEFT OUTER JOIN Photo ON Item.ItemID = Photo.ItemID

SELECT Location, PurchaseDate, Quantity, Description, Cost, PhotoLocation, PhotoCaption
FROM Location
JOIN Item ON Location.LocationID = Item.LocationID
JOIN Photo ON Item.ItemID = Photo.ItemID

SELECT Location, PurchaseDate, Quantity, Description, Cost, PhotoLocation, PhotoCaption
FROM Location
JOIN Item ON Location.LocationID = Item.LocationID
LEFT OUTER JOIN Photo ON Item.ItemID = Photo.ItemID
```

The first query uses two left outer joins to combine rows in the Location, Item, and Photo tables, as shown in Figure 4-13. Use of the left outer joins ensures all rows from the Location table are included in the query result set, even if no items are associated with a location. In addition, all rows from the Item table are included, even items for which no photos exist.

| | Location | PurchaseDate | Quantity | Description | Cost | PhotoLocation | PhotoCaption |
|---|---|---|---|---|---|---|---|
| 1 | Dining Room | 2003-10-04 00:00:00 | 1 | Dining Table | 1000.00 | C:\HomeInventory\P126010.JPG | Side view of dining table |
| 2 | Garage | NULL | NULL | NULL | NULL | NULL | NULL |
| 3 | Kitchen | NULL | NULL | NULL | NULL | NULL | NULL |
| 4 | Living Room | 2000-03-21 00:00:00 | 1 | Television | 500.00 | C:\HomeInventory\P100001.JPG | Television - Front |
| 5 | Living Room | 2000-03-21 00:00:00 | 1 | Television | 500.00 | C:\HomeInventory\P100002.JPG | Television - Back |
| 6 | Living Room | 2002-05-10 00:00:00 | 2 | Bookshelf | 200.00 | C:\HomeInventory\P110021.JPG | Bookshelf units |
| 7 | Living Room | 2005-07-28 00:00:00 | 1 | Couch | 2500.00 | NULL | NULL |
| 8 | Master Bedroom | NULL | NULL | NULL | NULL | NULL | NULL |

***Figure 4-13***
*Using left outer joins to combine three tables*

The second query uses two inner joins to combine rows in all three tables, as shown in Figure 4-14. This query result set is the most restricted set possible because only locations related to items that in turn are associated with photos are included. For example, even though items related to Living Room display in the query result set, Couch is not one of those items because there are no photos for it.

| | Location | PurchaseDate | Quantity | Description | Cost | PhotoLocation | PhotoCaption |
|---|---|---|---|---|---|---|---|
| 1 | Living Room | 2000-03-21 00:00:00 | 1 | Television | 500.00 | C:\HomeInventory\P100001.JPG | Television - Front |
| 2 | Living Room | 2000-03-21 00:00:00 | 1 | Television | 500.00 | C:\HomeInventory\P100002.JPG | Television - Back |
| 3 | Dining Room | 2003-10-04 00:00:00 | 1 | Dining Table | 1000.00 | C:\HomeInventory\P126010.JPG | Side view of dining table |
| 4 | Living Room | 2002-05-10 00:00:00 | 2 | Bookshelf | 200.00 | C:\HomeInventory\P110021.JPG | Bookshelf units |

***Figure 4-14***
*Using inner joins to combine three tables*

Finally, the third query uses an inner join to limit rows from the Location table to those related to items, but it uses a left outer join to include those items whether or not a photo exists. To continue the previous example, Couch is now in the query result set shown in Figure 4-15. Put another way, the inner join acts as a filter to restrict the rows that wind up in the result set. Only locations associated with items will be in the result set, such as Living Room and Dining Room. The left outer join simply attempts to match up items with photos and doesn't filter out an item if it isn't related to a photo, such as the couch.

| | Location | PurchaseDate | Quantity | Description | Cost | PhotoLocation | PhotoCaption |
|---|---|---|---|---|---|---|---|
| 1 | Living Room | 2000-03-21 00:00:00 | 1 | Television | 500.00 | C:\HomeInventory\P100001.JPG | Television - Front |
| 2 | Living Room | 2000-03-21 00:00:00 | 1 | Television | 500.00 | C:\HomeInventory\P100002.JPG | Television - Back |
| 3 | Dining Room | 2003-10-04 00:00:00 | 1 | Dining T... | 1000... | C:\HomeInventory\P126010.JPG | Side view of di... |
| 4 | Living Room | 2002-05-10 00:00:00 | 2 | Bookshelf | 200.00 | C:\HomeInventory\P110021.JPG | Bookshelf units |
| 5 | Living Room | 2005-07-28 00:00:00 | 1 | Couch | 2500... | NULL | NULL |

***Figure 4-15***
*Using an inner join and a left outer join to combine three tables*

Based on your understanding of joins, can you guess what the query result set would look like if you used a left outer join on the Location and Item tables and an inner join on the Item and Photo tables? Test your knowledge by changing one of the SELECT statements to see what happens when you execute it.

**7** Close the Query Editor and, when prompted, click **No** to decline saving your work as a script file.

## Sorting a Query Result Set

By adding an ORDER BY clause to a SELECT statement, you can provide very specific instructions for sorting the rows returned in the query result set. By default, sort order is assumed to be ascending. That is, words are sorted in the same direction you would find in a dictionary, and numbers are usually sorted in ascending numerical order. I say usually because if you use a numeric value in a character field, it loses its numeric characteristics and sorts like any other string character. For example, let's say a numeric column contains the following values in no particular order: 20, 100, 1, 15, 10. If this column were sorted normally in ascending order, those values would sort like this: 1, 10, 15, 20, 100. If these values were stored in a varchar column, however, an ascending sort would produce the following list: 1, 10, 100, 15, 20.

A simple SELECT statement that includes an ORDER BY clause looks like this:

```
SELECT (column1, column2, ...) FROM table1 ORDER BY column1, column2, ...
```

As I already pointed out, the sort order is assumed to be ascending, but you can reverse the sort order if necessary. Just add the *DESC* keyword after the column name. Although the *ASC* keyword exists and can be used to explicitly define an ascending sort, you don't really need to include it. The ascending sort just happens automatically.

### TO USE THE ORDER BY CLAUSE

1. Click **New Query** and make sure **HomeInventory** is selected in the Available Databases drop-down list. In the Query Editor, type the following statement:

```
SELECT Location, PurchaseDate, Quantity, Description, Cost
FROM Location LEFT OUTER JOIN Item ON Location.LocationID = Item.LocationID
ORDER BY Description
```

The query results are shown in Figure 4-16. Notice that NULL descriptions are ordered in the result set before valid descriptions are sorted.

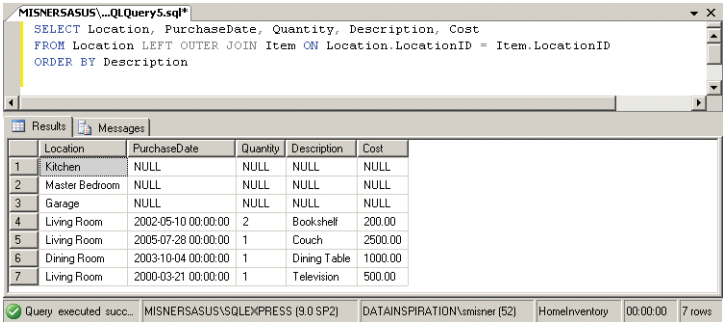



*Figure 4-16*
*Using the ORDER BY clause*

**2** Sometimes you might want to view your items with the most expensive item listed first and the least expensive item listed last. When you want to reverse the sort order, just add DESC next to the column name like this:

```
SELECT Location, Description, Quantity, PurchaseDate, Cost
FROM Location LEFT OUTER JOIN Item ON Location.LocationID = Item.LocationID
ORDER BY Cost DESC
```

Even when working with numeric data, rows with a NULL value in the column used for sorting are sorted separately. In an ascending sort, the NULL values would appear first, whereas they appear last when a descending sort is applied, as shown in Figure 4-17.

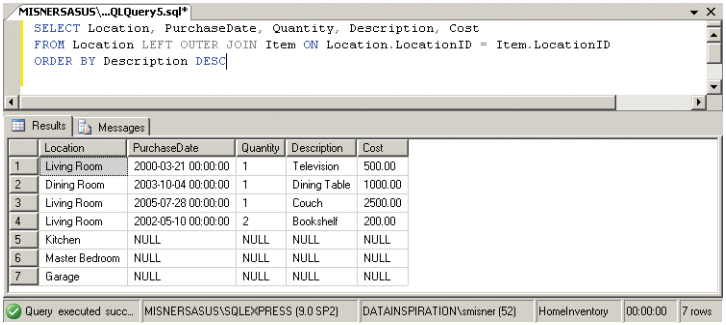



*Figure 4-17*
*Reversing the sort order in a SELECT statement*

3 You're not limited to sorting a single column. The database engine sorts each column separately, beginning first by sorting rows according to the order defined for the first column. Then, within each set of rows that share the same value in the same column, rows are sorted by the order defined for the second column. This process continues for each subsequent column listed in the ORDER BY clause. Try the following statement to verify the query results shown in Figure 4-18:

```
SELECT Location, Description, Quantity, PurchaseDate, Cost
FROM Location LEFT OUTER JOIN Item ON Location.LocationID = Item.LocationID
ORDER BY Location, Cost DESC
```

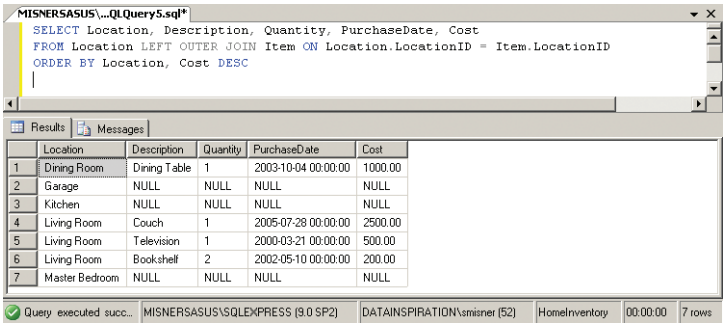



*Figure 4-18*
*Using multiple sorts in a SELECT statement*

4 Close the Query Editor and, when prompted, click **No** because you don't need to save your work.

## Using Aggregations

Sometimes you don't really need to see the individual rows in a table and instead need to see summarized values. For instance, you might want to add together the costs of all items when you want to know the total value of your household inventory. An aggregate function combines values from one or more rows and returns a single value. T-SQL includes many aggregate functions, as shown in Table 4-2, which includes the aggregate functions you're most likely to use.

*Table 4-2*
*Transact-SQL Aggregate Functions*

| Use This Function | To Do This |
|---|---|
| *AVG* | Divide the sum of values in the specified column by the number of rows (excluding rows with NULL values in the specified column) |
| *COUNT* | Count the number of rows in a table or group as data type int |
| *MAX* | Find the highest value in the specified column |
| *MIN* | Find the lowest value in the specified column |
| *SUM* | Sum the non-null values in the specified column |

**NOTE**

**Several more aggregate functions are available in T-SQL, but their usage tends to be specialized to certain types of analysis. For more information on all aggregation functions supported by T-SQL, search for the topic "Aggregate Functions (Transact-SQL)" in SQL Server 2005 Books Online.**

A SELECT statement that includes aggregate functions looks like this:

```
SELECT (AVG(column1) alias1, SUM(column2) alias2, ...) FROM table1
```

This form of the SELECT statement returns only one row, no matter how many rows are actually in the table. You can include as many aggregate functions as you need, and you can even reference the same column in different aggregate functions. Typically, you add an alias for the column name to provide a header caption in the query result set, but an alias is not required.

When you want to display subtotals, or intermediate aggregations, for groups of rows in a table, you add a GROUP BY clause like this:

```
SELECT (column3, column 4, AVG(column1) alias1, SUM(column2) alias2, ...) FROM table1 GROUP BY
column3, column4, ...
```

The columns included in the SELECT statement's column list that are not used with an aggregate function must be specified in the GROUP BY clause. Otherwise, an error message will display, and the query will not execute.

## TO USE AGGREGATE FUNCTIONS

1. Click **New Query** and check that **HomeInventory** is selected in the Available Databases drop-down list. In the Query Editor, type the following statement to compute a grand total of costs:

```
SELECT SUM(Cost) [Total Cost] FROM Item
```

Execute the statement to view the query results, as shown in Figure 4-19. The alias Total Cost is used to provide a column name for the result computed by the aggregate function. Brackets are required to enclose the alias because it includes a space.

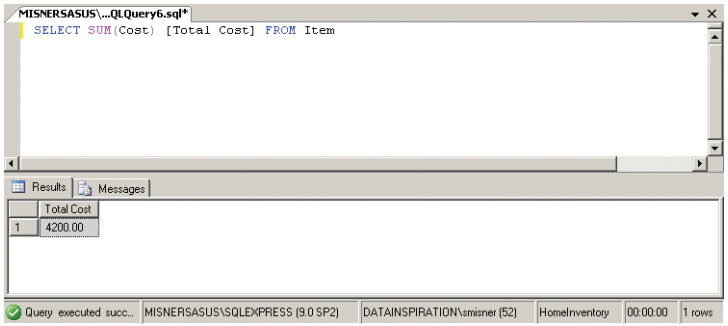



*Figure 4-19*
*Using the SUM function*

2. The Total Cost calculated in the previous step has only one problem. It is a sum of the per-unit costs in the Item table and doesn't factor in the Quantity column. You need to adjust the statement to correctly compute the value of your household inventory, like this:

```
SELECT SUM(Quantity * Cost) [Total Cost]
FROM Item
```

Execute the statement to view the results shown in Figure 4-20.

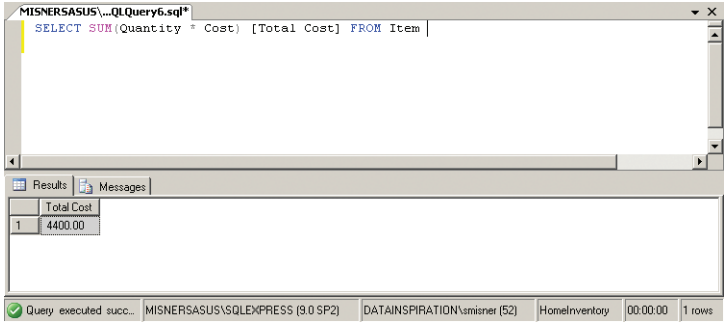



*Figure 4-20*
*Summing a calculated value*

In this case, you need to multiply the quantity by the cost on a row by row basis to calculate the extended cost, which is then added for all rows, so it's okay to perform the calculation inside the SUM function. If, on the other hand, you needed to compute a ratio such as Average Value, you would need to aggregate the ratio's numerator and denominator separately and then perform the division operation, like this: `SUM(value) / SUM(Count)`. This alternative calculation method is necessary because you cannot calculate an aggregated ratio by adding ratios computed row by row in a table.

**3** You can also compute subtotals by creating groups. Try executing the following statement to view the query results shown in Figure 4-21:

```
SELECT Location, SUM(Quantity * Cost) [Total Cost]
FROM Location LEFT OUTER JOIN Item ON Location.LocationID = Item.LocationID
GROUP BY Location
```

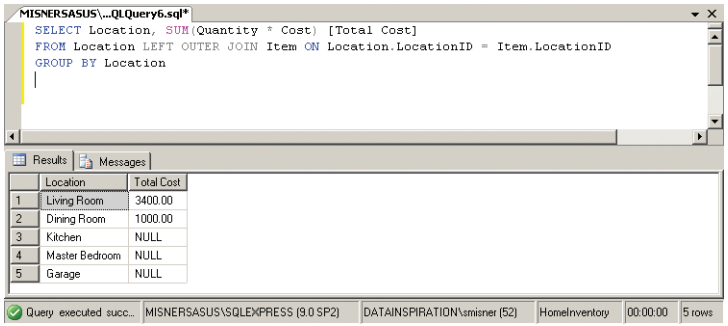



*Figure 4-21*
*Summing groups of rows using the SUM function and GROUP BY*

Notice the column used for grouping doesn't have to come from the same table containing the column used in the aggregate function. The key requirement here is that if you include a column in the column list that isn't aggregated, you must reference that column in the GROUP BY clause. Using this technique, you can create multiple levels of grouping by adding more columns to the column lists in the SELECT and GROUP BY clauses of the statement.

4 An aggregate function is not limited to the SELECT clause. When you want to compute group subtotals and then sort the groups based on the subtotal, you can use a T-SQL statement like this:

```
SELECT Location, SUM(Quantity * Cost) [Total Cost]
FROM Location LEFT OUTER JOIN Item ON Location.LocationID = Item.LocationID
GROUP BY Location
ORDER BY SUM(Quantity * Cost)
```

Figure 4-22 shows the query result set for this statement. Notice also that the alias for the aggregate function's result is used only in the SELECT clause. Keep in mind that when you use the GROUP BY and ORDER BY clauses in the same statement, you must specify the GROUP BY clause first. To specify a descending sort, just add the DESC keyword after the aggregate function in the ORDER BY clause, like this: `ORDER BY SUM(Quantity * Cost) DESC.`

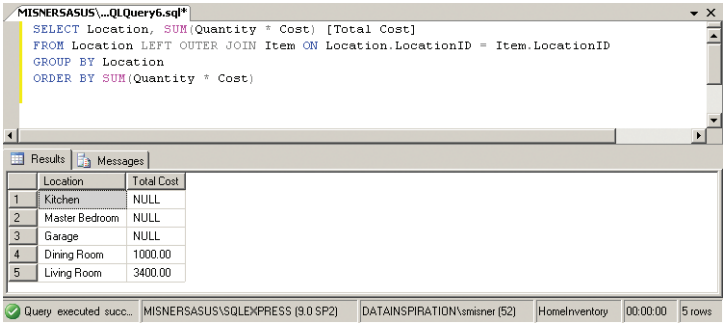
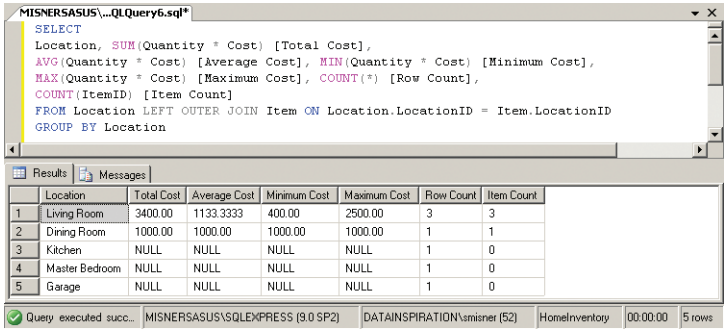



*Figure 4-22*
*Using the SUM function in the ORDER BY clause*

5 So far, the statements you've been using contain only a single aggregate function in the query, but you can certainly use as many aggregate functions in the same query as you need. Consider the following statement:

```
SELECT
Location, SUM(Quantity * Cost) [Total Cost],
AVG(Quantity * Cost) [Average Cost], MIN(Quantity * Cost) [Minimum Cost],
MAX(Quantity * Cost) [Maximum Cost], COUNT(*) [Row Count],
COUNT(ItemID) [Item Count]
FROM Location LEFT OUTER JOIN Item ON Location.LocationID = Item.LocationID
GROUP BY Location
```

Execute this statement to view the results, as shown in Figure 4-23. Notice in particular the difference in the results for `COUNT(*)` and `COUNT(ItemID)`. When you use the `COUNT(*)` function, all rows in the group are counted. As you learned previously in this chapter, the left outer join returns every row from the Location table even if no corresponding rows are in the Item table. As a result, there is one record for Kitchen and one for Master Bedroom, but you would see a NULL value in the ItemID column of your T-SQL statement if it included detail rows. Refer back to Figure 4-10 for a reminder of how this looks. When you use the COUNT(ITEMID) function, the database engine counts only those rows with a non-null value in the specified column, which is ItemID in this case.




| | Location | Total Cost | Average Cost | Minimum Cost | Maximum Cost | Row Count | Item Count |
|---|---|---|---|---|---|---|---|
| 1 | Living Room | 3400.00 | 1133.3333 | 400.00 | 2500.00 | 3 | 3 |
| 2 | Dining Room | 1000.00 | 1000.00 | 1000.00 | 1000.00 | 1 | 1 |
| 3 | Kitchen | NULL | NULL | NULL | NULL | 1 | 0 |
| 4 | Master Bedroom | NULL | NULL | NULL | NULL | 1 | 0 |
| 5 | Garage | NULL | NULL | NULL | NULL | 1 | 0 |

*Figure 4-23*
*Using multiple aggregate functions in a SELECT statement*

**NOTE**

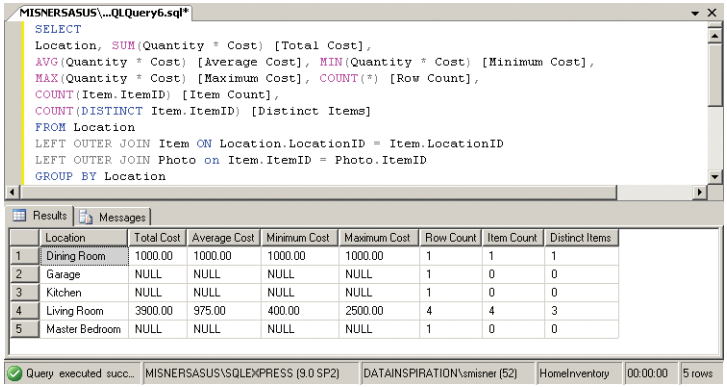
**It's important to remember that NULL values will affect row counts used by the AVG function as well. If, for example, two of the three items related to Living Room had a cost and the third item had a NULL cost, then the sum of the two items' costs would be divided by two to compute the average cost. The row with a NULL cost would be ignored.**

**6** Sometimes you want an aggregate function to consider only unique values. In these situations, you can include the DISTINCT keyword in front of the column name, as shown in the following statement, which adds one more column to count distinct items, updates the ItemID to Item.ItemID to avoid the ambiguous column error, and joins to the Photo table to potentially return more than one row per item in the query result set:

```
SELECT
Location, SUM(Quantity * Cost) [Total Cost],
AVG(Quantity * Cost) [Average Cost], MIN(Quantity * Cost) [Minimum Cost],
MAX(Quantity * Cost) [Maximum Cost], COUNT(*) [Row Count],
COUNT(Item.ItemID) [Item Count],
COUNT(DISTINCT Item.ItemID) [Distinct Items]
```

```
FROM Location
LEFT OUTER JOIN Item ON Location.LocationID = Item.LocationID
LEFT OUTER JOIN Photo ON Item.ItemID = Photo.ItemID
GROUP BY Location
```

Notice in the query results (Figure 4-24) the difference between the Item Count and Distinct Items columns. The Row Count and Item Count for the Living Room group have both increased by one because there are four detail rows for this query, but Distinct Items is only three. In other words, three distinct items are related to Living Room, but one of these items has two photos, which creates four detail rows that are then summarized by the COUNT functions.




| | Location | Total Cost | Average Cost | Minimum Cost | Maximum Cost | Row Count | Item Count | Distinct Items |
|---|---|---|---|---|---|---|---|---|
| 1 | Dining Room | 1000.00 | 1000.00 | 1000.00 | 1000.00 | 1 | 1 | 1 |
| 2 | Garage | NULL | NULL | NULL | NULL | 1 | 0 | 0 |
| 3 | Kitchen | NULL | NULL | NULL | NULL | 1 | 0 | 0 |
| 4 | Living Room | 3900.00 | 975.00 | 400.00 | 2500.00 | 4 | 4 | 3 |
| 5 | Master Bedroom | NULL | NULL | NULL | NULL | 1 | 0 | 0 |

***Figure 4-24***
*Using the DISTINCT keyword with the COUNT function*

**7** As you can see, using aggregate functions can be quite useful for summarizing your inventory records. You can use many other T-SQL functions to manipulate the values returned in the query result set. In fact, a whole book could be devoted to exploring how to use all these functions. Instead, let me simply introduce you to a date function you might find useful. As an example, you might want to summarize the costs of your household inventory by the year of purchase. You can use the YEAR function with a datetime or smalldatetime column to display only the four-digit year instead of the entire date, as shown in the following statement:

```
SELECT
Year(PurchaseDate) [Purchase Year], SUM(Cost) [Total Cost]
FROM Item
GROUP BY Year(PurchaseDate)
```

You can use the YEAR function in the SELECT, GROUP BY, and ORDER BY clause to organize the query result set (Figure 4-25) just the way you want it.

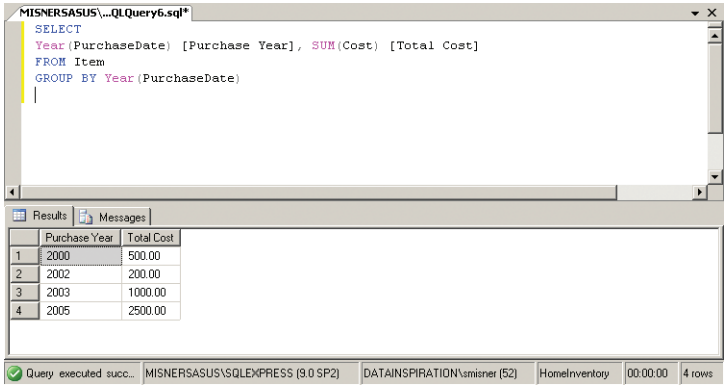



*Figure 4-25*
*Using the YEAR function to define a group*

**TIP**

**Related to the YEAR function are other date-related functions, such as MONTH and DAY. You might also be interested in the DATEPART, DATEADD, and DATEDIFF functions when working with dates. There are also functions you can use to manipulate strings or to perform mathematical operations. For more information about all the functions you can use, search for the topic "Functions (Database Engine)" in SQL Server 2005 Books Online.**

**8** Close the Query Editor and, when prompted whether to save the file, click **No**.

## Limiting Rows Retrieved by a Query

You've already discovered that joins can limit the number of rows in a query result set, but you have another option to use. By adding a WHERE clause to a SELECT statement, you can add another filter to retrieve only the rows in which you're interested. You can use a WHERE clause whether or not you're joining tables, but it would be the only way to limit rows when you're working with a single table. The ability to combine different clause types in a single SELECT statement is the power behind T-SQL, but you do have to be sure to place each clause in its correct place. A statement that includes all the clauses you've learned so far looks like this:

```
SELECT (column3, column 4, AVG(column1) alias1, SUM(column2) alias2, ...) FROM table1
WHERE condition1 AND (condition2 OR condition3)
GROUP BY (column3, column4, ...)
ORDER BY (column3, column4, …)
```

The condition you use in the WHERE clause is evaluated for each row in the table (or the result of joined tables). If the condition evaluates as true, then the row is kept in the query result set. Otherwise, it is discarded. You can also create complex conditions by using the keywords *AND* and *OR* and by forcing a specific order of evaluation by using parentheses.

## TO USE THE WHERE CLAUSE

1 One way you might want to filter data in a table is to retrieve all items related to a particular room. To prepare to retrieve this data, click **New Query** and verify **HomeInventory** is selected in the Available Databases drop-down list. Then add this statement to the Query Editor and execute it to view the results shown in Figure 4-26:

```
SELECT Location, Description
FROM Location JOIN Item ON Location.LocationID = Item.LocationID
WHERE Location = 'Living Room'
```

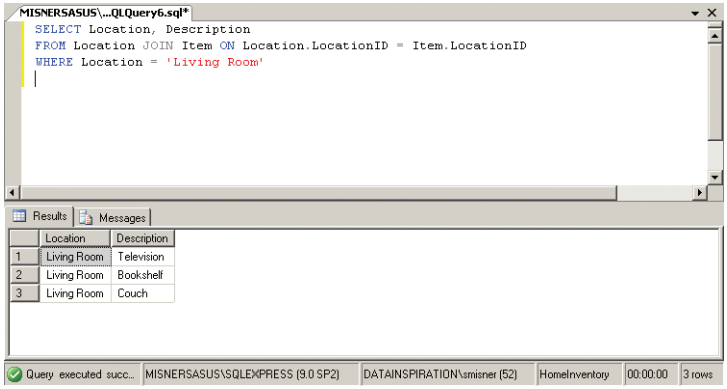



*Figure 4-26*
*Filtering rows based on a single column value*

2 You can use any operator in the WHERE clause, such as less than or greater than, but that type of operator is more commonly used for conditions that compare numeric values. You can use other operators that you didn't learn in mathematics class in the WHERE clause. For example, you can filter a query result set to retrieve items that are found in a list of rooms, using the statement here, which produces the query result set shown in Figure 4-27.

```
SELECT Location, Description
FROM Location JOIN Item ON Location.LocationID = Item.LocationID
WHERE Location IN ('Living Room', 'Dining Room')
```

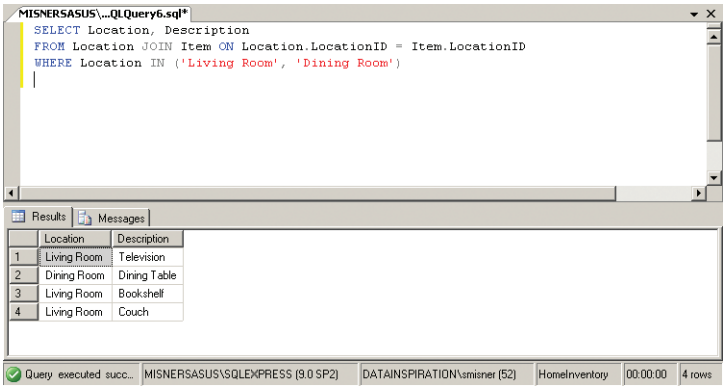



*Figure 4-27*
*Filtering rows based on a list of column values*

3 You can use the ***BETWEEN*** operator to compare a value to a low and a high value. In the case of dates, you can compare a date to a start date and an end date, like this:

```
SELECT Description, PurchaseDate FROM Item
WHERE PurchaseDate BETWEEN '1/1/2003' and '12/31/2005'
```

When you execute this statement, you'll see a query result set like the one shown in Figure 4-28.

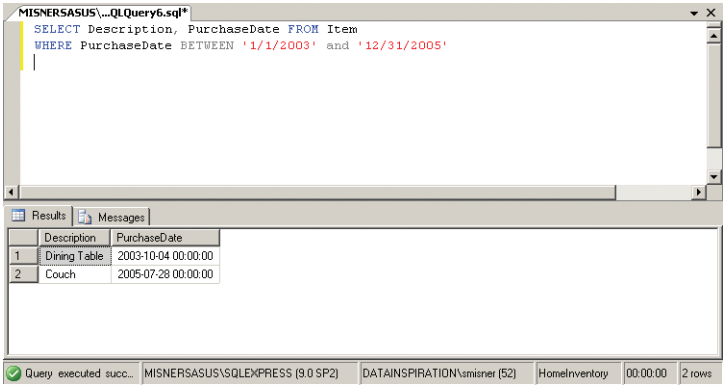



*Figure 4-28*
*Using dates in the WHERE clause with the BETWEEN operator*

4 Using the filter techniques described in the previous three steps can go a long way toward satisfying your reporting requirements, but there are still more options! Sometimes data gets entered into a database in a way we don't anticipate, so trying to locate that data by matching the name exactly will not always work. Fortunately, you can create a filter that uses pattern matching, like the statement shown here, to get valid query results like the ones shown in Figure 4-29.

```
SELECT Description, SerialNumber, Location
FROM Item JOIN Location ON Item.LocationID = Location.LocationID
WHERE SerialNumber LIKE '%4A2'
```

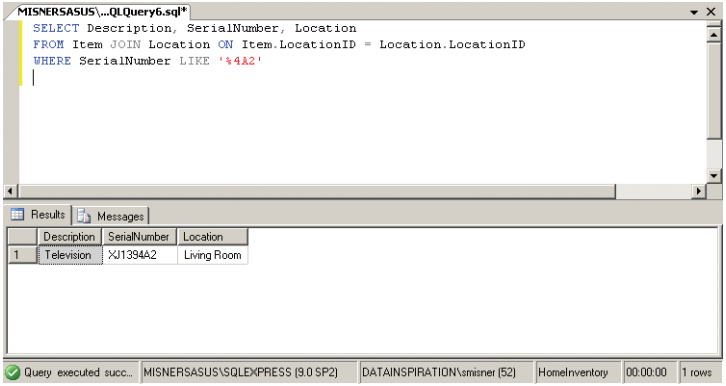



*Figure 4-29*
*Filtering with pattern matching*

With pattern matching, you use wildcard characters as placeholders for one or more characters. Table 4-3 provides examples of how you might use wildcard characters in a WHERE clause.

*Table 4-3*
*Wildcard Characters for Pattern Matching*

| Use This WHERE Clause | To Do This |
|---|---|
| WHERE SerialNumber LIKE '%4A2' | Find all items with serial numbers that end with the characters 4A2 |
| WHERE SerialNumber LIKE '%4A%' | Find all items with serial numbers that contain the characters 4A |
| WHERE SerialNumber LIKE '_4A%' | Find all items with serial numbers that contain the characters 4A in the second and third position |
| WHERE SerialNumber LIKE '[XY]J4A%' | Find all items with serial numbers that begin with either XJ4A or YJ4A |
| WHERE SerialNumber LIKE 'X[^K]%' | Find all items with serial numbers that begin with the letter X and do not contain the letter K in the second position |

**5** Of course, you're not limited to a single condition in the WHERE clause. By adding more conditions, you can be very specific about the data you want to retrieve. You can also connect multiple conditions by using the keywords *AND* or *OR*, as shown in this example for which the query result set is shown in Figure 4-30.

```
SELECT Location, Description, PurchaseDate, Cost
FROM Location JOIN Item ON Location.LocationID = Item.LocationID
WHERE Year(PurchaseDate) >= 2003 AND Cost >= 1000
```

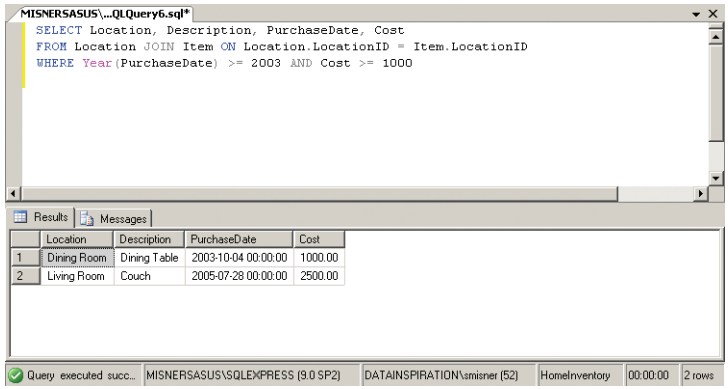



*Figure 4-30*
*Using multiple conditions in the WHERE clause*

**6** Close the Query Editor and, when prompted whether to save the file, click **No**.

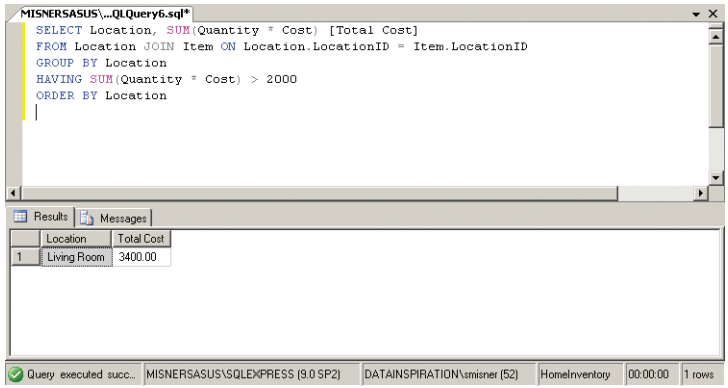



*Figure 4-31*
*Using the HAVING clause*

## *Filtering with the HAVING Clause*

*If your T-SQL statement includes aggregate functions, the filter applies to each row before the aggregation is performed. There is another type of filter you can use that evaluates after the aggregation. Let's say you want to determine which room has items that total more than $2,000. To find the answer, you use a statement that includes the HAVING clause, like this:*

```
SELECT Location, SUM(Quantity
* Cost) [Total Cost]
FROM Location JOIN Item ON
Location.LocationID = Item.
LocationID
GROUP BY Location
HAVING SUM(Quantity * Cost)
> 2000
ORDER BY Location
```

*The results for this query are shown in Figure 4-31.*

## Changing Data in Tables

It's a rare database that never requires updates after data is inserted into its tables. Sometimes an update is required because the original data was flawed in some way. Other times an update is required because information has a way of changing over time. Regardless of the reasons you have for making a change to existing data, you'll be glad to know T-SQL provides an UPDATE statement to help you perform this task. You can use Management Studio to open a table and type in new values directly, but T-SQL is the best way to make changes to many rows at a time.

The syntax for the UPDATE statement is different from the SELECT statement. Let's review the basic syntax for updates, which looks like this:

```
UPDATE table
SET (column1 = value, column2 = value, ...)
WHERE condition_expressions
```

Use of the WHERE clause is optional in the UPDATE statement. If you omit it, then the values you specify for each column are updated in every row of the table. You probably don't need to apply such a broad update, so be very careful about defining an appropriate filter to update only those rows that require a change. Once you execute the command, the deed is done without any way to undo the change short of restoring a backup of the database.

### TO USE THE UPDATE STATEMENT

1 The presence of NULL values in a column is often a source of consternation for database developers, so you might want to use the UPDATE statement to set a common value in a column, such as N/A, to replace any rows containing a NULL value. To prepare to update a table, click **New Query** and confirm **HomeInventory** is selected in the Available Databases drop-down list. Use the WHERE clause to make sure the update applies only to the rows you intended to update, as shown in the following statement.

```
UPDATE Item
SET ItemNotes = 'N/A'
WHERE ItemNotes IS NULL
```

When you execute this statement, the ItemNotes column is tested for a NULL value. Think back to Chapter 1, where I explained about the idiosyncrasies of NULL and the problem with testing for equality between a NULL value and another value. If you try to use a conditional expression like ItemNotes = NULL, the condition will not evaluate as true, and the rows will not be updated. By using IS NULL instead, you ensure the condition evaluates as true, the rows are updated, and the number of rows updated by the statement displays in the Messages pane, as shown in Figure 4-32.

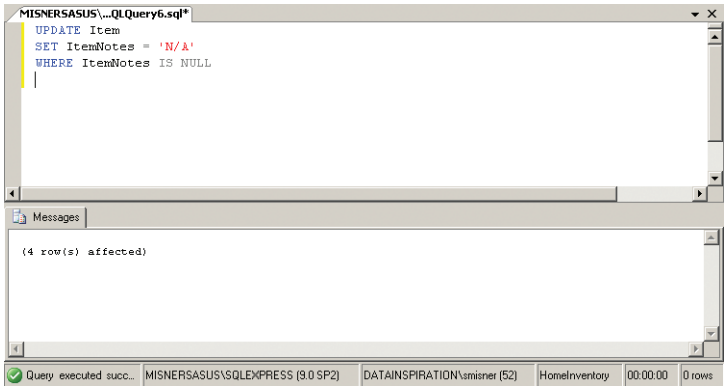



*Figure 4-32*
*Using the UPDATE statement to replace NULL values*

**TIP**

**I designed this example to give you practice with the UPDATE statement. As an alternative to performing updates on this table periodically, you might consider setting a default value for the ItemNotes column to keep NULL values out of this column in the first place.**

**2** When you need to update multiple columns, just set up a list of column name and value pairs in the SET clause, as shown in the statement here, and then execute it to ensure the update is performed successfully. Only one row should be updated.

```
UPDATE Item
SET
Item.LocationID =
   (SELECT LocationID FROM Location WHERE Location = 'Garage'),
ItemNotes = 'Preparing item for garage sale or disposition'
FROM Item JOIN Location ON Item.LocationID = Location.LocationID
WHERE Location = 'Dining Room'
```

This particular statement also introduces a couple of new concepts. First, the value assigned to Item. LocationID is a ***subquery***. A subquery is a query that evaluates separately from the main query. You can use a subquery to select a single value from a table or to select multiple rows that you use for comparison in a WHERE clause. In this example, the LocationID for Garage is retrieved from the Location table and assigned to any record in the Item table that is currently assigned to Dining Room. The result is the same as if you had typed the LocationID for Garage directly into the query like this:

```
UPDATE Item
SET
Item.LocationID = 5,
ItemNotes = 'Preparing item for garage sale or disposition'
FROM Item JOIN Location ON Item.LocationID = Location.LocationID
WHERE Location = 'Dining Room'
```

The second concept introduced in this query is the use of a join to filter rows before the update is performed. The Location column is not present in the Item table, so to find the items related to a specific location, the UDPATE statement includes an inner join between the Item and Location tables. All items are matched up with their respective locations, and the results are then filtered to limit the query result set to those rows that contain Dining Room in the Location column. Once these rows are identified, the update is performed to replace the old LocationID with the new LocationID retrieved by the subquery.

By performing a lookup to the Location table, you don't have to know the LocationIDs to update the correct rows in the Item table with the correct LocationID for Garage. Because the UPDATE statement will fail if the update will violate constraints, it's always a good idea to build a lookup like this into your UPDATE statements when foreign key relationships exist.

3 Close the Query Editor and, when prompted whether to save the file, click **No**.

## Deleting Rows from a Table

Just as you might need to update rows in a table from time to time, you might also find yourself with a table that has obsolete data, which should be purged. The syntax for the statement to delete rows is pretty simple and looks like this:

```
DELETE FROM table WHERE condition_expressions
```

Using the WHERE clause is optional with the DELETE statement as it was in the UPDATE statement; not using it is equally dangerous. If you execute a DELETE statement without a WHERE clause, all rows in the table are removed. Period.

## TO USE THE DELETE STATEMENT

1. First, prepare the Query Editor. Click **New Query** and select **HomeInventory** in the Available Databases drop-down list if necessary. Now let's say you changed your mind about including Garage as a location. You can delete this one row from the table easily, but what about the record in the Item table that you just associated with Garage? Try executing the following statement:

```
DELETE FROM Location WHERE Location = 'Garage'
```

As shown in Figure 4-33, the DELETE statement fails.

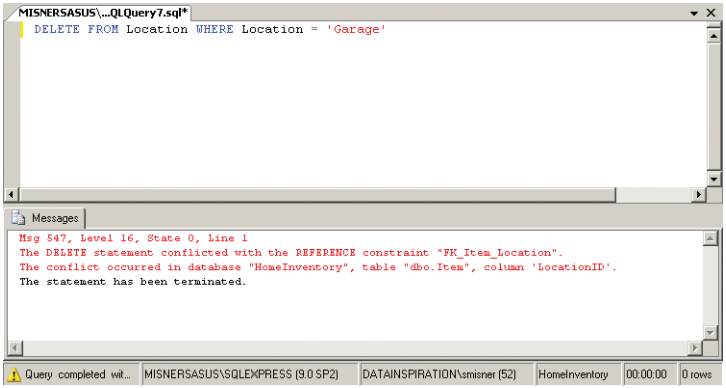



*Figure 4-33*
*Violating a foreign key constraint with the DELETE statement*

2. Now let's try something really drastic. To delete all records from the Item table, execute the following statement:

```
DELETE FROM Item
```

No questions asked. The rows are now gone, although if we had the foreign key constraint defined for the Photo table, this attempt to delete items would fail also. Hopefully, you followed my advice about saving the script to insert rows into the Item table so that you can quickly recover the table. Real-life data isn't quite so easily salvaged, so making regular backups is important for just such an occasion when you didn't really mean to delete the records. I'll explain backups in more detail in Chapter 5, "Maintaining Your Database."

**NOTE**

**Your Item table size is small enough that you wouldn't notice a difference, but when working with larger tables, you might consider using the TRUNCATE TABLE statement instead of the DELETE statement. For example, to remove all rows from the Item table, you would execute the following statement: TRUNCATE TABLE Item. This technique removes the rows from the table much faster because it doesn't log the deletion of each row. Ignoring logging is fine when you're dealing with prototype data and don't care about reproducing from the transaction log.**

3. Now let's do something about those missing records. If you saved the script file correctly in the section titled "Adding New Rows to a Table" you can retrieve that file now to reinsert rows into the table. On the **File** menu, point to **Open** and then click **File**. In the Open File dialog box, select **Insert Item Rows.sql** and click **Open**.

4. Click **Connect** when the Connect To Database Engine dialog box displays. The script displays in the Query Editor, ready for you to make changes if needed or to go ahead and execute it as is. Click **Execute**.

You're all set. Of course, the dining table is once again associated with the dining room, but that's okay. Moving it to the garage was simply an intellectual exercise and didn't require any actual heavy lifting on your part.

**NOTE**

**Using script files can be useful for more than just recreating your work, for example, when inserting rows. You'll discover later in this chapter how script files can be used to recreate database objects as well.**

5. The rows in the Photo table are now orphaned. That is, the ItemIDs stored in the Photo table no longer relate to rows in the Item table. Because the file references are fictitious anyway, go ahead and execute a DELETE statement to remove all rows from the Photo table.

6. Close the Query Editor.

**TIP**

**Deleting rows is a pretty drastic step, so you should be very certain this is really what you want to do. Another option to consider is a "soft delete." That is, instead of physically removing rows from the table, you could have a column to track a disposition date in case you sell, give away, or destroy an item. If you do this, you can keep historical information about each item if there's ever a need for it. You'll just need to modify queries to filter out records that have a disposition date when you want the query results to include only items you still own.**

Not only is T-SQL a data manipulation language, it is also a *data definition language* (DDL). Using a data definition language, you can create, modify, or delete a database and the structures it contains. Similar to the T-SQL statements you've learned so far, The DDL commands in T-SQL also start with verbs, like *CREATE*, *ALTER*, *DROP*, and *TRUNCATE*. The specific syntax you use for these commands depends, of course, on your specific goals. In this chapter, I'll introduce you to the basics and tell you how to find more information in SQL Server 2005 Books Online.

Anything you can create in a database by using Management Studio you can build using T-SQL. If it's so easy to do in Management Studio, you might wonder why bother using T-SQL to do the same thing? Well, one reason is to easily reproduce a database and its objects on another computer. For example, a friend might like to use the home inventory database on a home computer. To save your friend from all the steps you followed to build the database, even though those steps were pretty easy, you could simply provide a script containing the necessary *CREATE* commands to set up all the tables, including all the constraints and indexes. In short, T-SQL enables you to save your work so you can reproduce it as needed later.

You can use DDL for a variety of scenarios. You can, of course, build the database objects you're already using, such as a database, table, or index. You can also build security objects, such as a login or an application role. (You'll learn more about security in Chapter 5.) The best way to become familiar with DDL is to try it out, so I'll use the remainder of this chapter to show you how to enhance your home inventory database by adding another table and making changes to existing tables. I'll also introduce you to some database objects—views and stored procedures—you can use to save yourself the effort of running a new query every time you want to retrieve data from the database. I'll wrap up the chapter by showing you how to use scripts to package up your work so that you can share the database structure with others or rebuild your database on another computer.

## Creating Objects

To build new objects, you use the CREATE statement. The syntax for this statement requires you to define the object you want to create, such as a database, as well as the name to assign to the database. Some objects require a bit more information from you to build the object, for example, column names and data types if you're building a table.

The CREATE statement for many objects gives you the option to supply additional arguments to configure the object to meet more specific requirements. If you omit these arguments, SQL Server uses the default values it normally uses for each type of object. You reviewed some of these default values when you explored the Options page of the New Database and New Index dialog boxes in Chapter 3. As I mentioned in that chapter, the default values usually work just fine, so we won't review them in this book. If you're curious, just go to SQL Server 2005 Books Online and look up the specific statement you want to learn more about, such as CREATE DATABASE, to determine whether you should use the statement's arguments.

In this section, I'll show you how to create a new table in your database. As it stands now, you can execute a lot of queries to determine the value of items you own and can even produce a list of items you purchased since the last time you provided updated information to your insurance agent. Sometimes insurance policies impose limitations on the amount of coverage provided for certain types of household goods unless you acquire a policy rider or some other type of policy. If your insurance agent were to ask how much you own in consumer electronics or jewelry, how would you come up with the answer? Unless you used the ItemNotes column in the Item table to store this information, you probably wouldn't be able to query the database to come up with the total value of these types of items. Even if you did include this information in the ItemNotes column, it's not likely this column is used exclusively for that purpose, and you'll find it challenging to separate category information from other types of information you might store in this column. The solution is to add another table to your database so that you can store category information. You'll also need to add a column to the Item table to associate each item to a category, which I'll show you how to do later in this chapter. To add the new category table, follow these steps.

## TO USE THE CREATE TABLE STATEMENT

1. Review the updated diagram shown in Figure 4-34 that now includes the new Category table. Think about how you would use this information and Management Studio to add a new table.

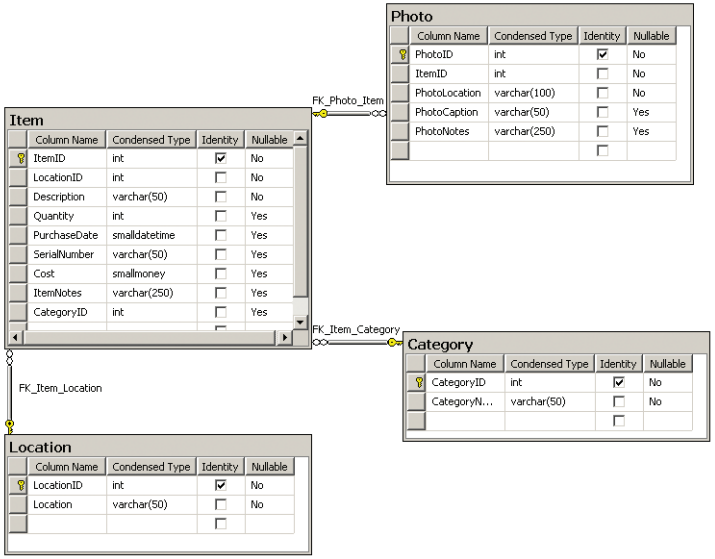



*Figure 4-34*
*Updated home inventory diagram*

2. Click **New Query** and check that **HomeInventory** is selected in the Available Databases drop-down list. Execute the following statement:

```
CREATE TABLE [dbo].[Category]
(
        [CategoryID] [int] IDENTITY(1,1) NOT NULL,
        [CategoryName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Category] PRIMARY KEY CLUSTERED
 ( [CategoryID] ASC )
)
```

As you can see, the information you provide in the CREATE TABLE statement is very similar to the information you provide when using Management Studio to add a new table. At a minimum, you need to include a name for the table and specify names and data types for each column in the table. In this particular example, the default schema, dbo, is included in the statement, but this is not required. Because you're an administrator, the dbo schema will automatically be used for any table you create.

As you can see in the preceding CREATE TABLE statement, you still provide a name and data type for each column, but instead of clearing the Allow Nulls check box as you would in Management Studio, you use the keywords *NOT NULL* to ensure null values aren't allowed in the columns. To configure the identity specification, you add IDENTITY(1,1) to the identity column to configure the identity property for that column with the seed value of 1 (the first argument), which will be assigned to the first row added to the table and incremented by 1 (the second argument) each time a row is added to the table. Lastly, the PRIMARY KEY constraint you see in this CREATE TABLE statement performs the equivalent task as when you click Set Primary Key in the Management Studio toolbar.

3. Using either Management Studio or Query Editor, add rows to the Category table, as shown in Table 4-4. Remember that CategoryID is an identity column, so you don't need to supply that value when you add records. I included these IDs in the table so that you can see the order in which I added categories and can relate the foreign keys I add later to the Item table to each category.

| CategoryID | Category |
|---|---|
| 1 | No Category |
| 2 | Electronics |
| 3 | Jewelry |
| 4 | Furniture |
| 5 | Computer and Peripherals |

*Table 4-4*
*Category Table*

**TIP**

**In Object Explorer, you'll need to right-click the Tables folder and click Refresh to view the newly added table.**

4. Close Query Editor and any table windows you may have open.

## Altering Objects

To associate each item with a category, you need to make a change to the Item table by adding a new column. In addition, you might want to add a disposition date to the Item table to track when you removed an item from inventory, as I suggested in the "Deleting Rows from a Table" section, earlier in this chapter. Finally, you'll need to add a foreign key constraint to the Photo table because we purposely skipped that step when creating the Photo table in Chapter 3.

For each CREATE statement for database objects, there is a corresponding ALTER statement. So, for example, you can use ALTER DATABASE to add or remove the files or filegroups associated with an existing database, among other tasks. Similarly, you can use ALTER TABLE to add or remove columns or constraints in an existing table.

## TO USE THE ALTER TABLE STATEMENT

1. Click **New Query** in the standard toolbar and make sure **HomeInventory** is selected in the Available Databases drop-down list. Execute the following statements:

```
ALTER TABLE [dbo].[Item] ADD [CategoryID] [int]

ALTER TABLE [dbo].[Item] ADD [DispositionDate] [smalldatetime] NULL
```

   Because the CategoryID column will have a foreign key relationship to the Category table, you would ordinarily define the column using NOT NULL if the column were included at the time you created the table. You could even include those keywords in the ALTER TABLE statement if the table were empty. However, neither of those conditions is true right now, so you have to let the column be nullable for the time being. After you add data to this column, you'll be able to fix the column's nullability later.

2. Replace the statements in the Query Editor with the following statements:

```
ALTER TABLE [dbo].[Item]
ADD CONSTRAINT [FK_Item_Category] FOREIGN KEY([CategoryID])
REFERENCES [dbo].[Category] ( [CategoryID] )

ALTER TABLE [dbo].[Photo]
ADD CONSTRAINT [FK_Photo_Item] FOREIGN KEY([ItemID])
REFERENCES [dbo].[Item] ( [ItemID] )
```

   As you can see, making changes to tables is still pretty easy even when using DDL.

3. Using either Management Studio or Query Editor, update the Item table as shown in Figure 4-35.

Table - dbo.Item

| | Ite... | LocationID | Description | Quantity | PurchaseD... | SerialNumber | Cost | ItemNotes | CategoryID | DispositionDate |
|---|---|---|---|---|---|---|---|---|---|---|
| | 7 | 2 | Dining Table | 1 | 10/4/2003 ... | NULL | 1000.0000 | N/A | 4 | NULL |
| | 8 | 1 | Bookshelf | 2 | 5/10/2002 ... | NULL | 200.0000 | N/A | 4 | NULL |
| | 9 | 1 | Couch | 1 | 7/28/2005 ... | NULL | 2500.0000 | N/A | 4 | NULL |
| | 10 | 1 | Television | 1 | 3/21/2000 ... | XJ1394A2 | 500.0000 | N/A | 2 | NULL |

***Figure 4-35***
*Item table with updated CategoryID values*

4. Now you can update the table to set the nullability of the CategoryID column. Replace the statements in the Query Editor with the following statement and then execute it:

```
ALTER TABLE [dbo].[Item] ALTER COLUMN [CategoryID] int NOT NULL
```

5. Close any open Query Editor and table windows.

## Working with Views

Rather than write a SELECT statement each time you want to see data retrieved from the database in a certain way, you can create a view. Think of a view as a reusable SELECT statement that you save to the database. After you create the view, you can use the SELECT statement with the view as if it were really a table.

Views are popular with database administrators because they can hide the technical details of a table from users by using friendly column names, hiding joins between tables, filtering data in advance, or summarizing data. In some cases, views can be used to speed up performance or to provide a security layer by controlling exactly what the user can see. You can, of course, use Management Studio to create a view, but because you're learning how to use T-SQL to create objects, I'll show you how to use the CREATE VIEW statement instead. To create a view that summarizes the value of your household inventory by purchase year, follow these steps.

## TO USE THE CREATE VIEW STATEMENT

1. Click **New Query** in the standard toolbar and make sure **HomeInventory** is selected in the Available Databases drop-down list. Execute the following statement:

```
CREATE VIEW [dbo].[vTotalCostByPurchaseYear]
AS
SELECT   YEAR(PurchaseDate) AS [Purchase Year], SUM(Cost) AS [Total Cost]
FROM     dbo.Item
GROUP BY YEAR(PurchaseDate)
```

**TIP**

**It's common to use a naming convention that prefixes the view name with the letter *v*.**

Creating a view requires only a few words more than creating a query, but the benefit is that you create it once and don't need to type the details of the SELECT statement each time you need the information provided by the query. This view is obviously quite simple and not too tedious to reproduce later as a query when needed, but imagine if your query were to retrieve multiple columns from four different tables that you join together. You don't need to create too many queries like this on a frequent basis before you realize creating a view saves you a lot of time and effort.

2. To query the view, replace the statement in the Query Editor with the following statement and execute it:

```
SELECT * FROM vTotalCostByPurchaseYear
```

As you can see by the query results in Figure 4-36, you can query views by using exactly the same syntax you use to query tables. You can join a view with other views or tables, sort results by adding an ORDER BY clause, use aggregate functions to summarize a value in a view's column, perform calculations on view columns to derive a new column in the query results, and filter results from the view by using a WHERE clause. Your query skills really pay off when you add views to your repertoire.

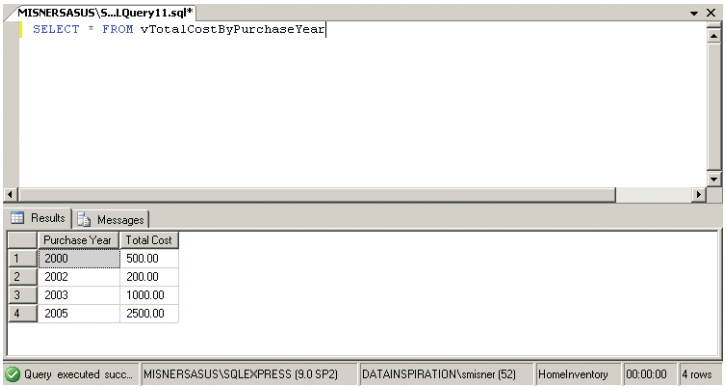



*Figure 4-36*
*Using a SELECT statement with a view*

**3** Close the Query Editor window.

## Using Stored Procedures

You can use a stored procedure to simplify user access to data, add a security layer, or potentially improve query performance. Like a view, a stored procedure can be as simple as a single SELECT statement or as complex as a query that joins multiple tables. However, by using a stored procedure, you can combine multiple T-SQL statements into one stored procedure, and you aren't limited to SELECT statements as you are within a view.

Stored procedures are precompiled by SQL Server, which means the T-SQL statements included in the stored procedure might be optimized by the database engine to execute more quickly than if you executed them individually in the Query Editor window or using a custom application. You can also add error handling to a stored procedure to control what happens if something should go wrong during execution. Finally, you can use parameters with a stored procedure to make it more flexible for different situations. Of course, the best way to understand the benefits of a stored procedure is to build one yourself by following these steps.

## TO CREATE AND EXECUTE A STORED PROCEDURE

1. Click **New Query** in the standard toolbar and make sure **HomeInventory** is selected in the Available Databases drop-down list. In Object Explorer, expand **Programmability**, right-click **Stored Procedures**, and click **New Stored Procedure**. The CREATE PROCEDURE template (Figure 4-37) displays in the Query Editor.

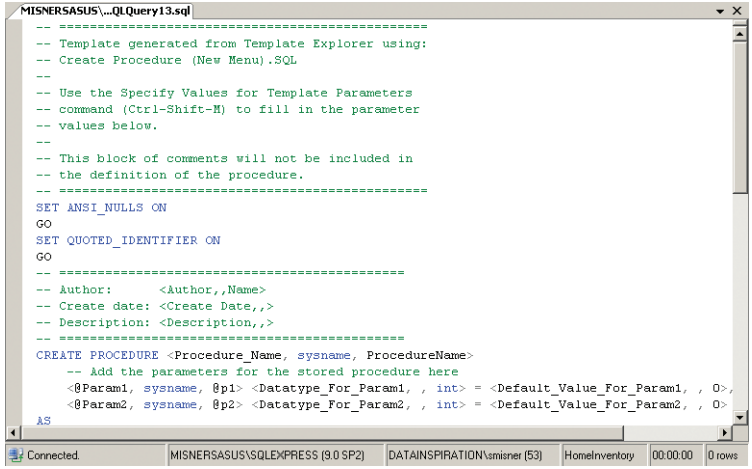



*Figure 4-37*
*The CREATE PROCEDURE template*

2. Locate the line that begins with CREATE PROCEDURE and modify it to look like this:

```
CREATE PROCEDURE spGetItemCostSummary
```

3. Locate and delete the following lines:

```
<@Param1, sysname, @p1> <Datatype_For_Param1, , int> = <Default_Value_For_Param1, , 0>,
<@Param2, sysname, @p2> <Datatype_For_Param2, , int> = <Default_Value_For_Param2, , 0>
```

4 Now let's start with a simple stored procedure. Locate and delete the following line:

```
SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
```

5 In place of the line just deleted, add the following statement, to match Figure 4-38:

```
SELECT Year(PurchaseDate) AS PurchaseYear, CategoryName, Location, SUM(Cost) as TotalCost
FROM Location
JOIN Item ON Location.LocationID = Item.LocationID
JOIN Category ON Category.CategoryID = Item.CategoryID
GROUP BY Year(PurchaseDate), CategoryName, Location
```

6 Execute the CREATE PROCEDURE statement, as shown in Figure 4-38.

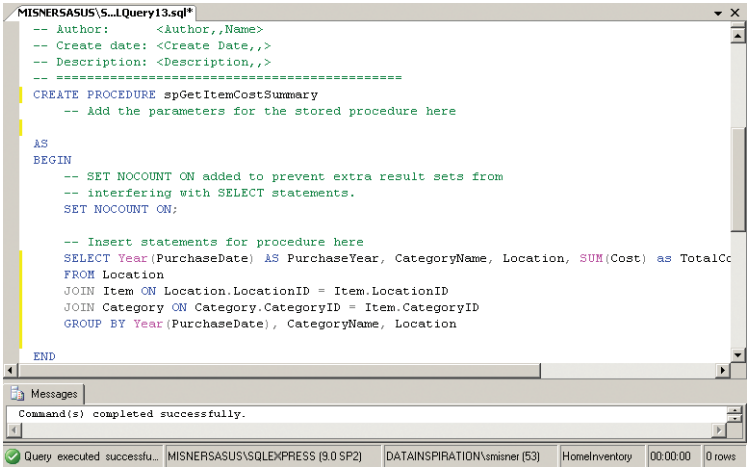



*Figure 4-38*
*Successful execution of CREATE PROCEDURE statement*

7 To execute the stored procedure, click **New Query** and execute the following statement:

```
EXEC spGetItemCostSummary
```

As you can see in Figure 4-39, the costs of your home inventory have been summarized by year, by category, and by location.

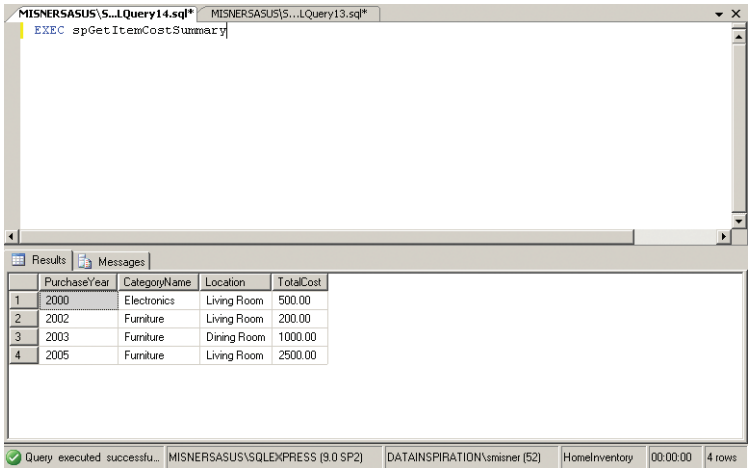



*Figure 4-39*
*Executing a stored procedure*

**8** Now let's alter the stored procedure to use a parameter to filter the query results based on user input. To create an ALTER PROCEDURE template, right-click **Stored Procedures**, click **Refresh**, right-click **spGetItemCostSummary**, point to **Script Stored Procedure As**, point to **ALTER To**, and click **New Query Editor Window**. The script to alter the procedure displays in the Query Editor (Figure 4-40).

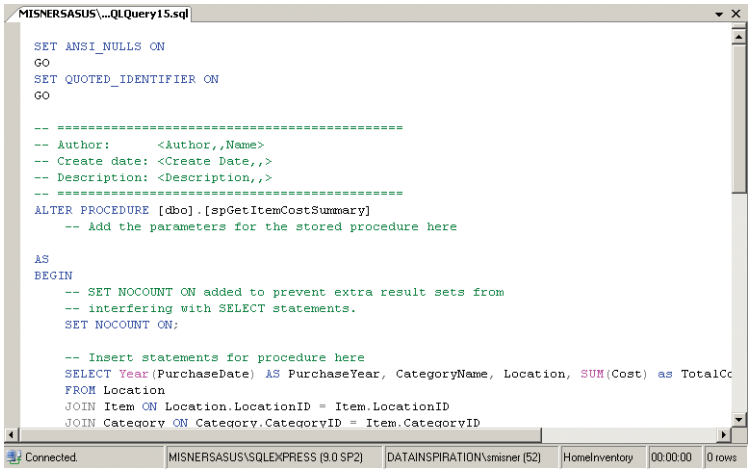



*Figure 4-40*
*The ALTER PROCEDURE template*

9 Under the instruction Add The Parameters For The Stored Procedure Here, insert a new line and type the following lines:

```
@Category varchar(50),
@PurchaseYear int
```

You provide values for these parameters, @Category and @PurchaseYear, when you execute the stored procedure.

10 Above the GROUP BY clause in the SELECT statement of the stored procedure, insert the following WHERE clause:

```
WHERE CategoryName = @Category AND Year(PurchaseDate) = @PurchaseYear
```

The values you specify for each parameter when you execute the stored procedure will replace the placeholders in this WHERE clause.

11 Execute the ALTER PROCEDURE statement to update your stored procedure, as shown in Figure 4-41.

**TIP**

**Alternatively, you could define a default value here for each parameter, but we'll keep things simple for now. To learn more about using parameters in stored procedures, see "CREATE PROCEDURE (Transact-SQL)" in SQL Server 2005 Books Online.**

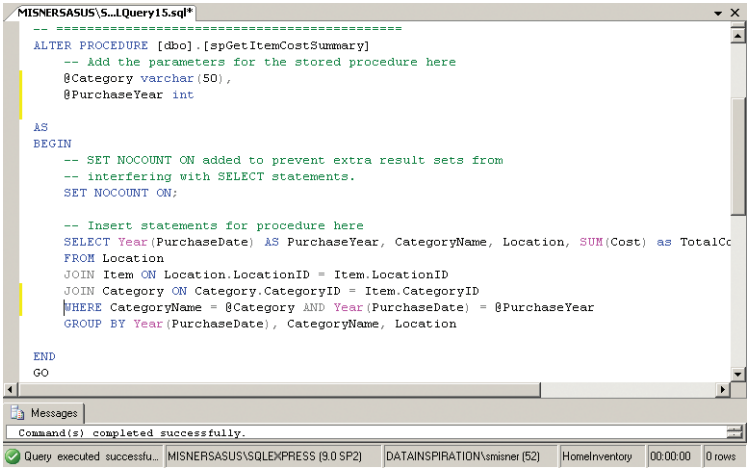



*Figure 4-41*
*Successful execution of the ALTER PROCEDURE statement*

12 Now you're ready to execute the stored procedure to retrieve results for furniture purchased in 2005. Click **New Query** and execute the following statement to view the results shown in Figure 4-42:

```
EXEC spGetItemCostSummary 'Furniture', 2005
```

An easy way to execute a stored procedure with parameters is to specify the parameter values in the same sequence as declared in the stored procedure. In this example, you specify a value for CategoryName and then a value for PurchaseYear. When the parameter value is a string, be sure to enclose the value in single quotation marks.

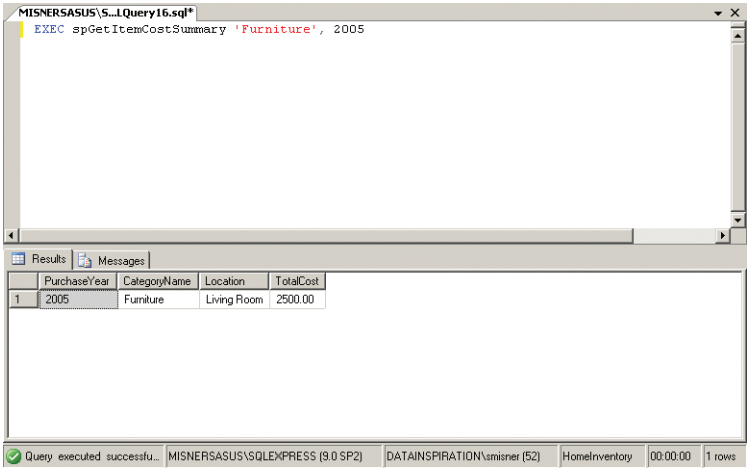



*Figure 4-42*
*Executing a stored procedure with parameters*

13 Close all Query Editor windows.

## Working with Scripts

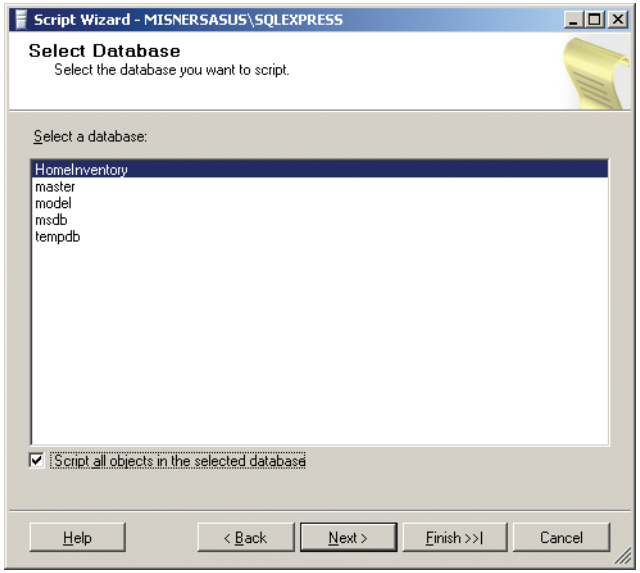
Now you have a complete database, you've even added some enhancements like views and stored procedures, and you're ready to share your database with a friend. You already know you can use Management Studio to generate scripts, save scripts as SQL files to the file system, and later execute those scripts from the Query Editor window in Management Studio.

You do have other options, though. Instead of using Management Studio, you can use the SQLCMD utility to execute your scripts. Using this utility, you can execute T-SQL statements or system procedures that you enter directly into the utility, or you can provide the name of a script file for the utility to execute. The SQLCMD utility can be a flexible option because you can use scripting variables to take advantage of one script in a variety of scenarios. For example, if you create a script to build the home inventory database on your friend's computer, you can use a scripting variable to create the database with a different name than HomeInventory.

## TO CREATE AND TEST A DATABASE SCRIPT

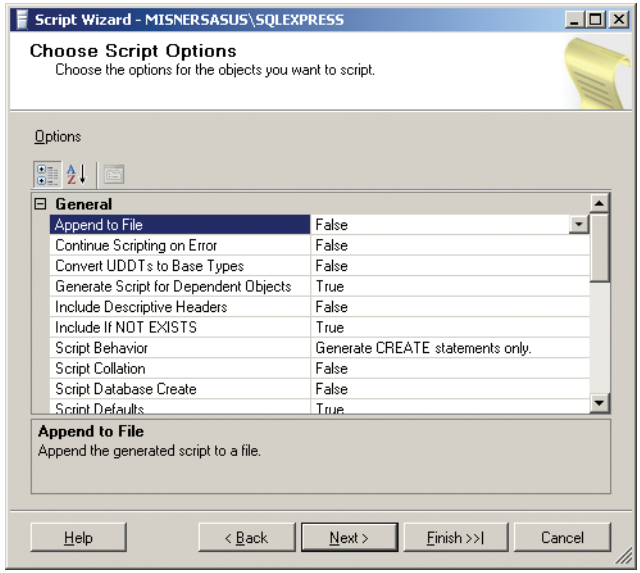
1. In Object Explorer, right-click **HomeInventory**, point to **Tasks**, and click **Generate Scripts**. The welcome page of the Generate SQL Server Scripts Wizard displays, as shown in Figure 4-43. Click **Next**.

*Figure 4-43*
*Welcome page of the Generate SQL Server Scripts Wizard*




2 On the Select Database page of the wizard, select **HomeInventory** and then select the **Script all objects in the selected database** check box, as shown in Figure 4-44. Click **Next**.

*Figure 4-44*
*Select Database page of the Generate SQL Server Scripts Wizard*




3 On the Choose Script Options page of the wizard (Figure 4-45), you can review the various options that determine how the script will generate. I recommend you keep all default settings. Click **Next**.

*Figure 4-45*
*Choose Script Options page of the Generate SQL Server Scripts Wizard*

4 On the Output Option page of the wizard, select **Script to file**. In the File Name box, you can type a different folder path if you like, but I recommend you change the file name from script.sql to something more meaningful, such as **CreateHomeInventory.sql**. You can change the folder and keep all other default selections, as shown in Figure 4-46. Click **Next**.

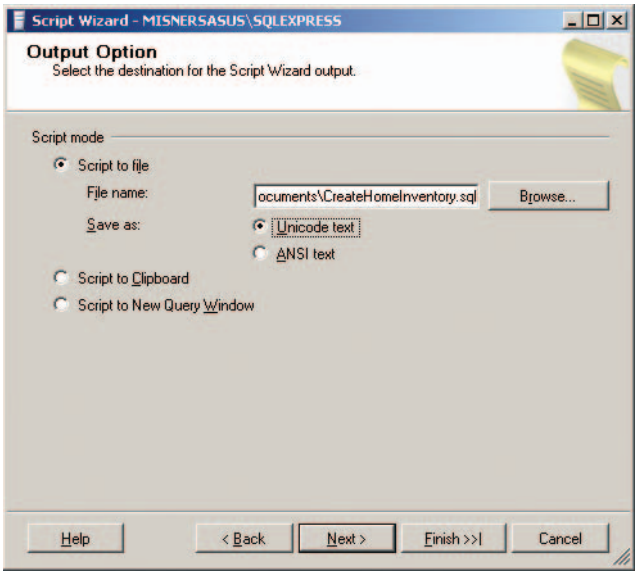



*Figure 4-46*
*Output Option page of the Generate SQL Server Scripts Wizard*

5 On the Script Wizard Summary page of the wizard (Figure 4-47), click **Finish**.

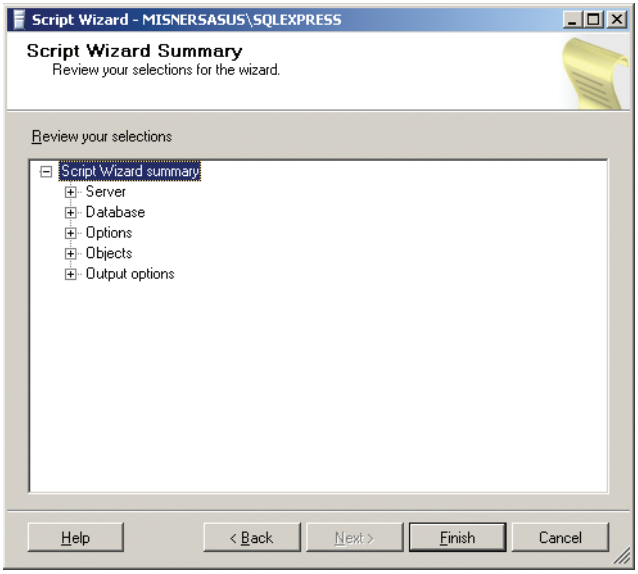



*Figure 4-47*
*Script Wizard Summary page of the Generate SQL Server Scripts Wizard*

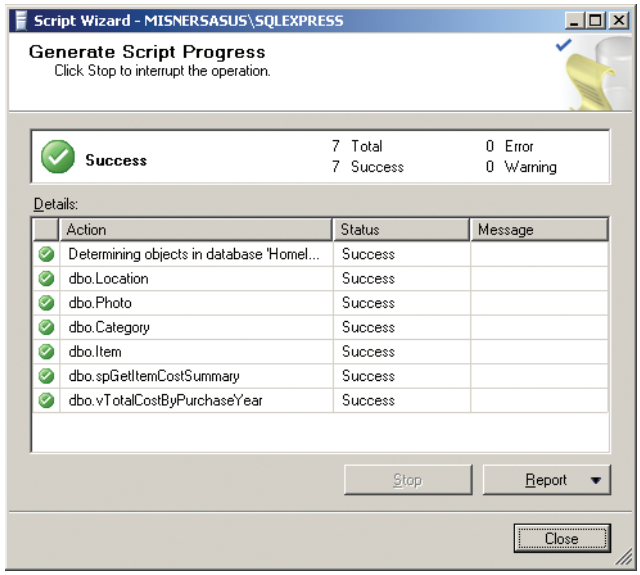



*Figure 4-48*
*Generate Script Progress page of the Generate SQL Server Scripts Wizard*

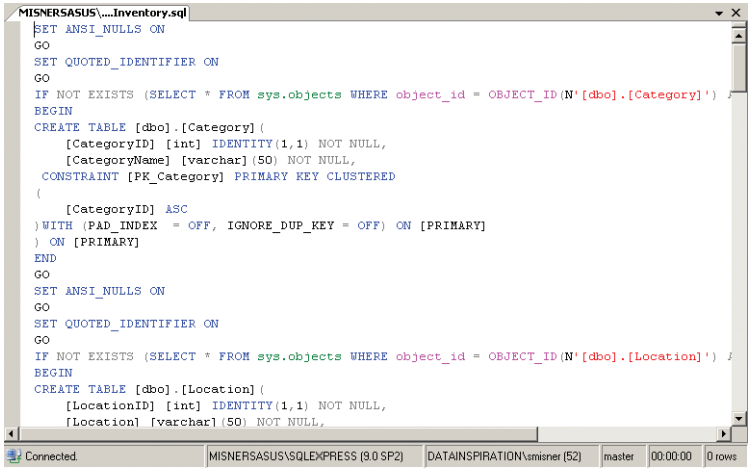



*Figure 4-49*
*The CreateHomeInventory script*

**6** On the Generate Script Progress page of the wizard (Figure 4-48), when the script generation completes successfully, click **Close**.

**7** In Management Studio, on the **File** menu, point to **Open** and then click **File**. Navigate to the folder in which you saved the database script, select **CreateHomeInventory.sql** (or the file you created in step 4 if you opted to use a different name), and click **Open**. In the Connect to Database Engine dialog box, click **Connect**. The script displays in the Query Editor, as shown in Figure 4-49. Scroll through the script to review the CREATE and ALTER statements that were used to reproduce your database and its objects.

This script assumes you have already created the database. In later steps, you'll modify the script to include the CREATE DATABASE statement. As you review the remainder of the script, you should recognize many of the statements. There are four CREATE TABLE statements to build the Location, Category, Item, and Photo tables. The sequence of these statements is important because of the foreign key constraints There are a few additional arguments you probably don't recognize that set table options. Notice also that the script repeats the *SET* command several times. In general, these commands tell SQL Server to follow a standard known as SQL-92 when handling things like nulls, identifiers, and values smaller than a defined column size. The *GO* command tells the Query Editor to send the preceding set of statements to the database engine as a batch.

Your script is now ready for execution. However, it's not particularly flexible in its current state. If you try to run it now on your machine, the script will fail because you already have the HomeInventory database. Let's make one more change to allow the script to use a scripting variable for the database name. Then, you can create another copy of the database on your machine for testing, or you can share the database with a friend who can choose to personalize the database name.

**NOTE**

**Alternatively, you could script each object individually and combine the scripts into a single file. As you can see, the Generate Script Wizard simplifies this process for you.**

**8** Now let's modify this script to include the CREATE DATABASE statement. Insert the following lines into the beginning of the script, as shown in Figure 4-50:

```
CREATE DATABASE $(DatabaseName)
GO
USE $(DatabaseName)
GO
```

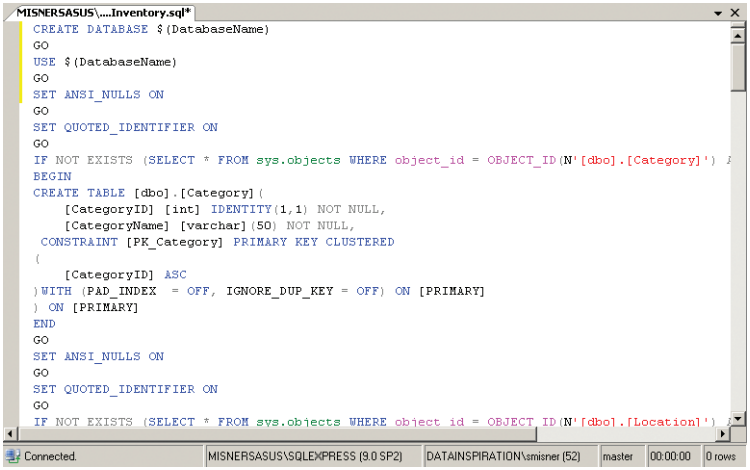



*Figure 4-50*
*Updated CreateHomeInventory script*

The value for the scripting variable, ***$(DatabaseName)***, will be set when the script is executed using the SQLCMD utility. By using scripting variables, you can take advantage of the same script to produce uniquely named copies of the same database.

**9** Close Query Editor and click **Yes** when prompted to save the script.

*Figure 4-51*
*Enabling SQLCMD mode*

**10** Click **New Query** and click **SQLCMD Mode** in the SQL Editor toolbar, as shown in Figure 4-51.

In SQLCMD mode, you can write and edit SQLCMD scripts. By using SQLCMD scripts, you can use variables in your T-SQL scripts for ultimate flexibility.

**NOTE**

**The SQLCMD utility offers many other benefits, but unfortunately this book doesn't have room to discuss them all. You can learn more by searching for SQLCMD in SQL Server 2005 Books Online.**

**11** In Query Editor, type the following statements and then click **Execute**.

```
:setvar DatabaseName "MyHomeInventory"
:r CreateHomeInventory.sql
```

**NOTE**

**This example assumes that you saved the CreateHomeInventory.sql file in your user directory, C:\Documents and Settings\<your name>\My Documents. If you saved the script file to a different location, in step 11, you'll need to type the full path to the script. For example, if you saved to a folder called C:\My Scripts, type the following command: "*C:\My Scripts\CreateHomeInventory.sql*". The double-quotation marks are required if your folder path contains spaces.**

**12** Close the Query Editor window. In Object Explorer, right-click **Databases** and click **Refresh**. Expand the **MyHomeInventory** folder and review the objects created in the database, as shown in Figure 4-52. Congratulations! You've successfully used scripting to build an entire database!

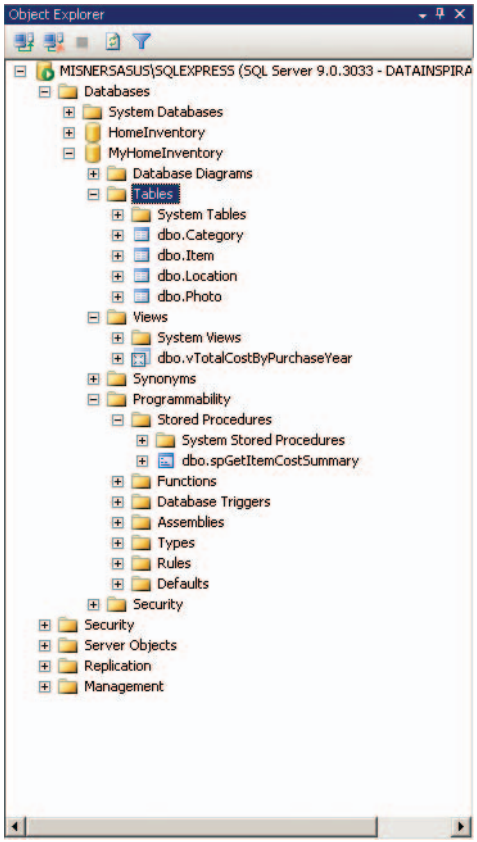



*Figure 4-52*
*Reviewing objects in MyHomeInventory*

## In Summary...

Once again, you're learned a lot of material. In this chapter, you started working directly with your database, using T-SQL statements to add, change, and delete data. You also learned many different ways to retrieve data from your database with T-SQL. Lastly, you learned how to create new objects and make changes to existing objects in your database as an alternative to using the graphical interface in Management Studio. The benefit to using T-SQL as a DDL, as you now know, is to store scripts for reuse later when you want to quickly reproduce your efforts or share your work with others.

At this point, you know everything you need to successfully add your own data to your database, to keep it up-to-date, and to retrieve information when you need it. There's more to taking care of a database than managing the data, however. In the next chapter, you'll learn how to keep your database secure and how to protect your data from disaster by implementing a backup process.

Chapter 5

# Maintaining Your Database

*Applying Security, 146*

*Performing Administrative Tasks, 168*

You're now done building your database, and you know everything you need to know to continue adding data to the database and to build queries that retrieve information from it. After all your hard work so far, the next thing you should think about is how to take good care of your database to ensure it not only lasts as long as you need it, but it also keeps all your data stored safely. First, you want to be sure you control who is able to use your database and what they can do with it. Second, you want to be sure you're able to make a copy of your work in case your computer fails or you want to relocate the database to another machine. In this chapter, I'll show you how to perform these administrative tasks.

I'm going to assume the only activities you've performed with SQL Server Express since you downloaded and installed it are the procedures described in the previous chapters in this book. If that's a correct assumption, then at this point you're the only person who can work with the HomeInventory database or any other database currently stored in SQL Server Express.

If you don't intend to allow anyone else to enter or use the data in the SQL Server Express databases, then you're all set. You don't need to do anything more to configure security in SQL Server Express because everyone else is locked out of your databases by default. There is a special login added to SQL Server Express called BUILTIN\Users that allows anyone that can log onto your computer with a Windows login name to connect to the SQL Server Express instance. They can see databases in SQL Server Express, but they can't open up the databases, nor can they see database objects or change data by using the Management Studio interface or by executing T-SQL statements. So, for all practical purposes, users are locked out of SQL Server Express until you change their security level.

If instead you plan to enlist your family's help to maintain the HomeInventory database, you might want to set up some simple security measures to protect the data from accidental changes or loss. Or you might later decide to build a database for a Web application, and you'll need to be prepared to implement security properly when that time comes. Before you start the process of setting up security, you should understand exactly what it means to secure a database.

Security has its own vocabulary, which can make the concepts seem too confusing for the average person to bother with, but security is very important and is therefore very much worth the effort to understand. Refer to Figure 5-1 as you read through this section to better understand security concepts. The basic purpose of database security is to ensure that only requests—attempts to connect, Transact-SQL (T-SQL) statements, and so on—from valid *principals* are honored by SQL Server Express. A principal can be a person, a Windows service, or a group that includes people, services, or both. When you secure a database, you specify how you're going to *authenticate* and *authorize* each principal. Authentication confirms a principal is using a valid login with the correct password, whereas authorization defines what that principal can and cannot do with each database. For the purposes of

further discussion about security in this database, I'm going to assume the only principals to use your database will be individual people rather than services or groups, but keep in mind that the concepts apply equally to all principals.

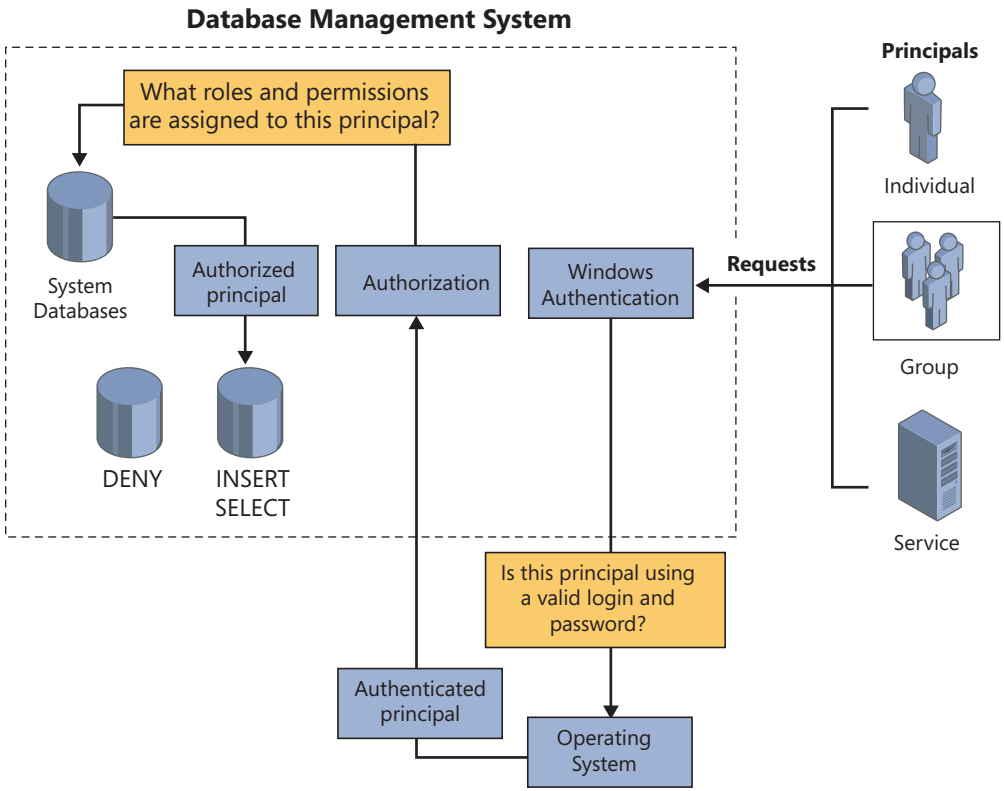



*Figure 5-1*
*Database security*

To dig a little deeper into these concepts, authentication ensures the person requesting information from your database is actually the person you intended to interact with your database. Of course, this method of authentication is only as good as the security of the password. If your user divulges the password to someone else or leaves the password where prying eyes can find it, your system can be compromised. You could add advanced security like biometrics or a smartcard system to avoid this situation, but I'll assume your environment doesn't require these stricter security measures at this time.

Authorization spells out which databases an authenticated individual can access. Even more specifically, authorization details which objects in a database someone can see and whether that person can change the structure of database objects or the data contained in those objects. As part of the process you follow to implement security in SQL Server Express, you use roles and permissions to define the authorization rules that apply to each person. I'll explain these two concepts in more detail later in this chapter.

## Creating Logins

SQL Server Express, like other editions in the SQL Server family, is well-integrated with Windows security, which means both user and groups you set up in the operating system to access your computer are recognized by SQL Server Express. When you installed SQL Server Express, if you followed the instructions in Chapter 2, "Installing SQL Server Express," you instructed SQL Server Express to use Windows authentication. This method is an easy way to implement authentication, and in fact is the only method Microsoft recommends. To use Windows authentication, you only need to give each user one *login*, a combination of user name and password, and then you add that login to SQL Server to define the authentication and authorization for that login.

**IMPORTANT**

**It's easy to get confused about logins. Each user of your computer must have a Windows login, even if you have many users sharing the same login (as might be the case when you set up a Web application that retrieves data from your SQL Server Express database). You could also set up a Windows group to combine Windows logins that share common characteristics, such as tasks you'll allow these users to perform. Either a Windows login or a Windows group can be a SQL Server login. If you don't explicitly add a SQL Server login for a user, or if you don't assign the user to a Windows group for which a SQL Server login exists, that user won't be able to access your database.**

**NOTE**

**You could change SQL Server Express to use Mixed Mode Authentication, which enables you to use both Windows authentication and SQL Server authentication. In other words, either the operating system authenticates the principal or SQL Server performs the authentication. In the latter case, SQL Server will store the login name and a password to validate the identity of the principal when a connection is attempted by that principal. SQL Server authentication is included for backward compatibility with earlier editions of SQL Server and is not currently considered to be a best practice, so I won't explain the details in this book.**

In this section, I'm going to assume you'd like to create a Windows login for a member of your family, and I'll show you exactly how to do that. To get started, follow these steps.

### TO ADD A WINDOWS LOGIN TO SQL SERVER EXPRESS

1. First, you need a user account on your computer for the person you're allowing to use your database. If you're using Windows XP Home Edition, continue to step 2. If a user account already exists, skip ahead to step 6. Otherwise, open the Computer Management console. There are several ways to open this console, but my preferred method is to right-click **My Computer** (or, for Windows Vista users, **Computer**) and click **Manage** to view the Computer Management console shown in Figure 5-2.

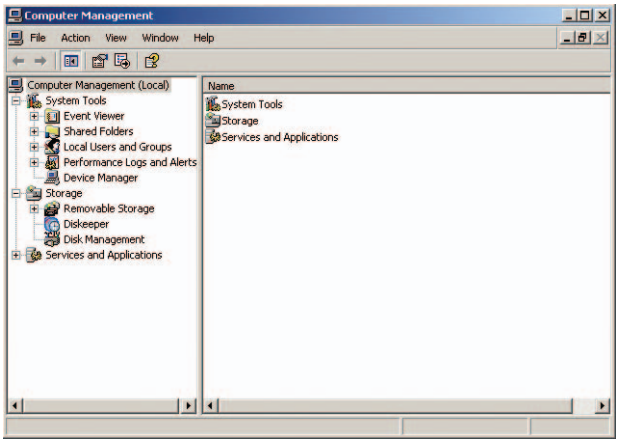



*Figure 5-2*
*Computer Management console*

**NOTE**

**The method you use to open the Computer Management console depends on the version of Windows you're using. If you're not familiar with this console and can't find it using the method I describe, use the Help And Support Center on your computer to search for the topic "Using Computer Management." There, you'll find step-by-step instructions for opening this utility.**

**2** Follow the instructions applicable to your operating system:

### Windows Server and Windows Vista

Expand **Local Users And Groups**, right-click **Users**, and click **New User**. The New User dialog box displays, as shown in Figure 5-3.

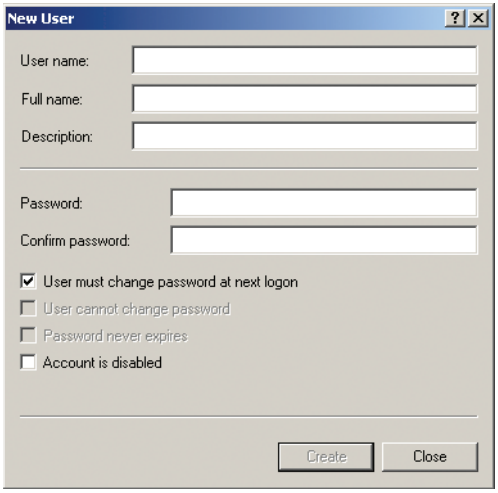



*Figure 5-3*
*New User dialog box*

In the User Name box, type a name for the new user and then type a password in the Password and Confirm Password boxes. By selecting the applicable check boxes in this dialog box, you can indicate whether you want to allow the user to change the password right away and whether the password will expire on a periodic basis. Click **Create** when you're finished.

**Windows XP Home**

Click **Start**, click **Control Panel**, and double-click **User Accounts**.

Click **Create A New Account**.

In the Type A Name For The New Account box, type a name for the new user and click **Next**.

On the Pick An Account Type page of the User Accounts Wizard, select **Limited** and click **Create Account**.

**Windows XP Professional**

To complete this chapter, you need to disable Simple File Sharing so that you can add user accounts. To disable Simple File Sharing, click **Start**, click **My Computer**, and then, on the **Tools** menu, click **Folder Options**.

Click the **View** tab, scroll to locate Use Simple File Sharing (Recommended), and clear the check box next to it. Click **OK**.

In the Computer Management console, expand **System Tools**, expand **Local Users And Groups**, right-click **Users**, and click **New User**.

In the User Name box, type a name for the new user and then type a password in the Password and Confirm Password boxes. Click **Create** when you're finished.

3. Continue by adding more users if you want to allow other people to work with your database. After you add the last user, click **Close** in the New User dialog box.

4. Close the Computer Management console.

5. Open SQL Server Management Studio Express and connect to your SQL Server Express instance.

6. Expand **Security** and click **New Login** to display the Login dialog box, as shown in Figure 5-4.

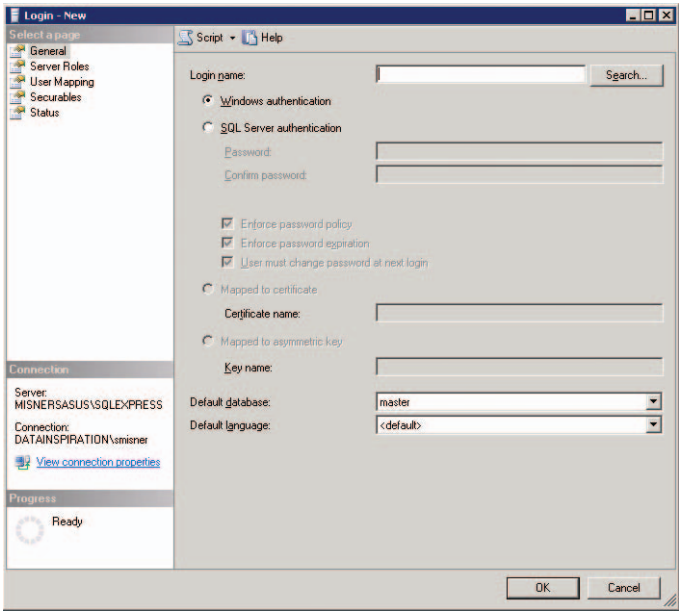



*Figure 5-4*
*Login dialog box*

7 You must correctly specify the login in the Login Name box. To be absolutely safe, it's a good idea to click **Search** rather than type the login directly here. The Select User Or Group dialog box displays, as shown in Figure 5-5.

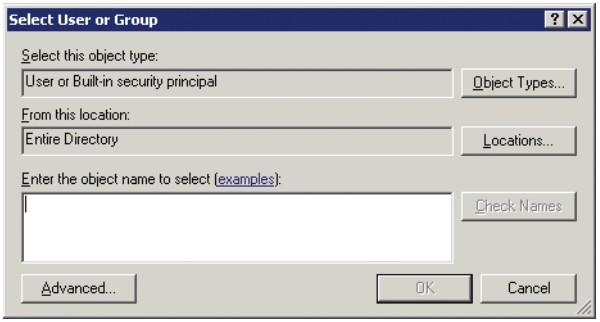



*Figure 5-5*
*Select User Or Group dialog box*

8 Again, it's a good idea to let the operating system find the available names for you, so click **Advanced**. The Select User Or Group dialog box expands to allow you to specify search parameters, which is useful for administrators in a business environment who must sift through a lot of users or groups. I assume your computer isn't configured with quite so many users, so I suggest you click **Find Now** to display all users and groups rather than define a filter. Click the user or group you want to add to SQL Server Express and click **OK**. The user or group name displays in the Enter The Object Name To Select box, as shown in Figure 5-6, where you can see my fictitious user, Test Misner.

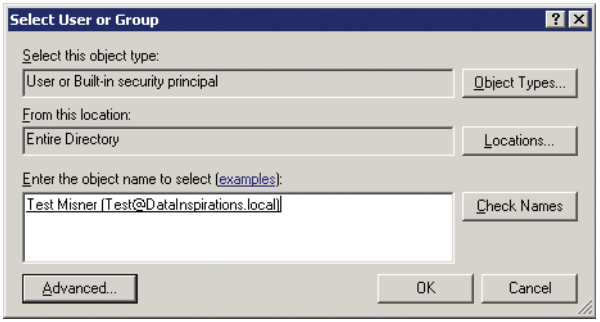



*Figure 5-6*
*Select User Or Group dialog box with a Windows account selected*

9 Click **OK** to add the selected user or group name to the Login Name box, as shown in Figure 5-7.

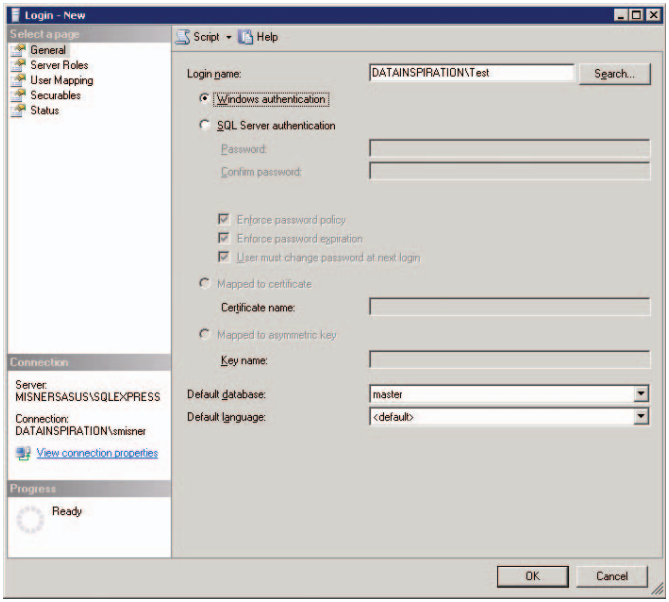



*Figure 5-7*
*Login Name dialog box with a Windows account selected*

**NOTE**

**If your computer is not in a domain, you won't see the domain name preceding the user name as you see in Figure 5-7. The Windows account name is sufficient when your computer is in a workgroup or home network in which no domain has been set up.**

10 In the Default Database drop-down list, select **MyHomeInventory**, as shown in Figure 5-8. Leave the dialog box open for the next procedure.

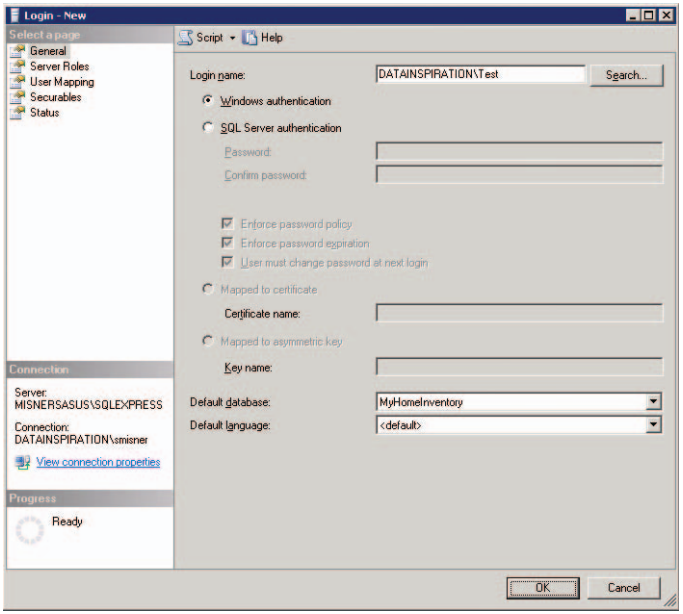



*Figure 5-8*
*Login Name dialog box with a Windows account selected*

You can leave <default> as the selection for Default Language because it will simply use the language currently used by the operating system of your computer. You'll notice this dialog box has other pages, which I'll cover in the following sections of this chapter.

**NOTE**

**To save yourself the trouble of deleting rows from each table before you share it, use the HomeInventory database as your "master copy" of the database to share with others. Use the MyHomeInventory database to add your own data and manage it over time.**

At this point, the user you just added will be able to log onto the server, but that's all. To be more specific, if that person opens Management Studio, the Connect To Database dialog box displays, Windows authenticates the user, and SQL Server Express authorizes the user to the view the existing databases in Object Explorer, as shown in Figure 5-9. However, you haven't yet told SQL Server Express what this user can do, so having a connection to the server at this point isn't enough. This user cannot expand the databases you created to view the tables, view, or stored procedure you created. You'll fix this situation in the next section of this chapter.

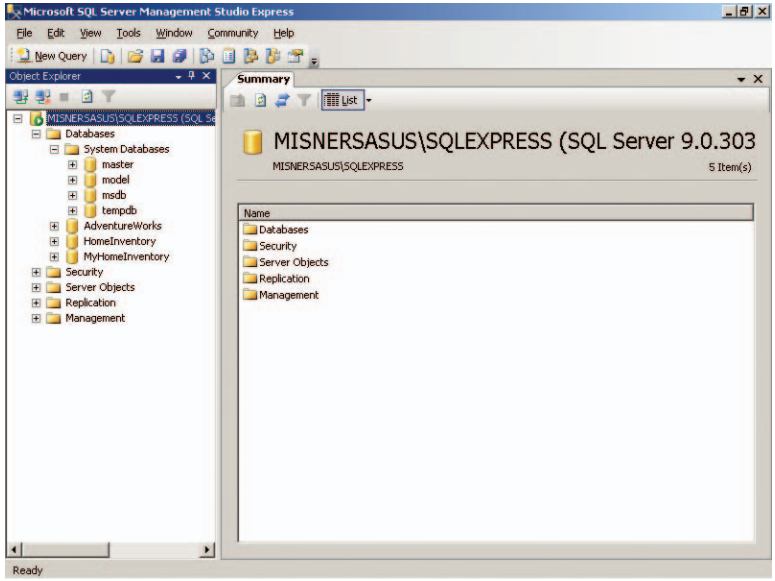



*Figure 5-9*
*Object Explorer as it appears to an authenticated user with no authorization*

## Assigning Roles

The concept of *roles* in the SQL Server family is much like that of groups in Windows. You can assign one or more logins to a role just like you can assign one or more accounts to a Windows group. When you want multiple people to work with a database in the same way, you'll find it's easier to put them all into a single role and then assign *permissions* to that role. Permissions are the explicit access privileges that a principal is given for each database and its objects, which could limit the principal to viewing data or allow the principal to change data or even to modify or delete database objects. If you don't assign a role, you must set the permissions for each login individually, which could easily turn into a tedious process.

All SQL Server editions come with four built-in types of roles. There is a fixed set of server roles that you can optionally assign the users you want to perform specific tasks on the server, such as create databases or manage security settings. There is also a fixed set of database roles that you use to specify what each user can do within each database, such as write data or back up the database. You can't modify the fixed server roles or database roles in any way. They are simply there for you to use or not. By contrast, you can create your own database roles or even application roles. It's quite common in business environments to use combinations of roles to get just the right set of permissions assigned to logins.

Because server roles apply to the entire server, you'll find them located in the Security folder for the server instance. Database roles, as you might expect, affect each database separately. Within each database folder, you'll find a Security folder, which in turn contains a Database Role folder. You'll find all the fixed database roles here, and you can add your own as well.

In the database's Security folder, you'll also find an Application Roles folder to store the application roles you might define. You create an application role when you have an application that connects to your database, and you want to limit permissions to the database for users of that application.

To review, the very minimum you need to do when you want a new user to access your database is to create a Windows account. If you have set up a SQL Server login for a Windows group, add this Windows account to that group or create a new SQL Server user login that uses Windows authentication. You then need to add the user to a role that has been granted permissions to your database or explicitly grant the user login access to the database. To learn how to assign a user to a role, follow these steps.

## TO ASSIGN A ROLE

1 This step assumes you are continuing from the previous section and still have the Login dialog box open. If you took a break, you can return to this point by expanding the Security and Logins folders in the Object Explorer window of Management Studio. Right-click the user login, click **Properties**, and click **Server Roles**. The Server Roles page of the Login dialog box appears, as shown in Figure 5-10.

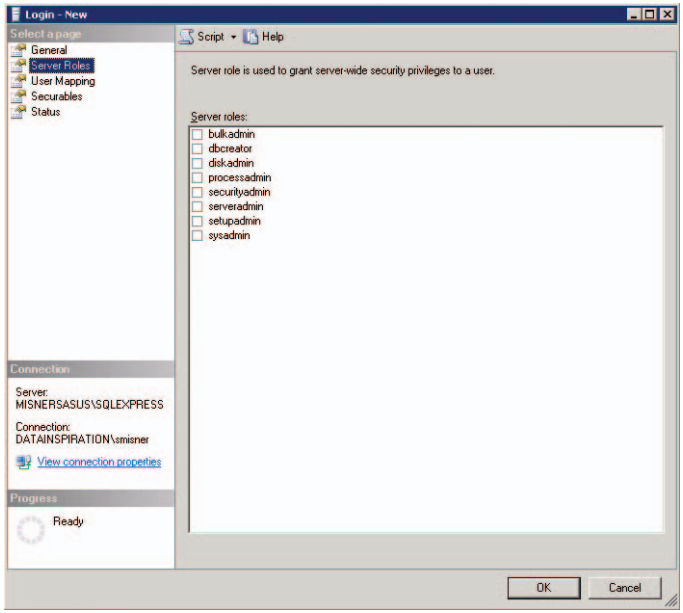



*Figure 5-10*
*Server Roles page of Login dialog box*

Server roles are primarily intended for use by business organizations that divide database responsibilities among many people. In your situation, we'll assume you are the sole administrator, and you don't intend to delegate any of the tasks that occur at the server level. If you do want to assign a server role, you simply select the check box to the left of the role you want to assign.

2 Click the User Mapping page, shown in Figure 5-11.

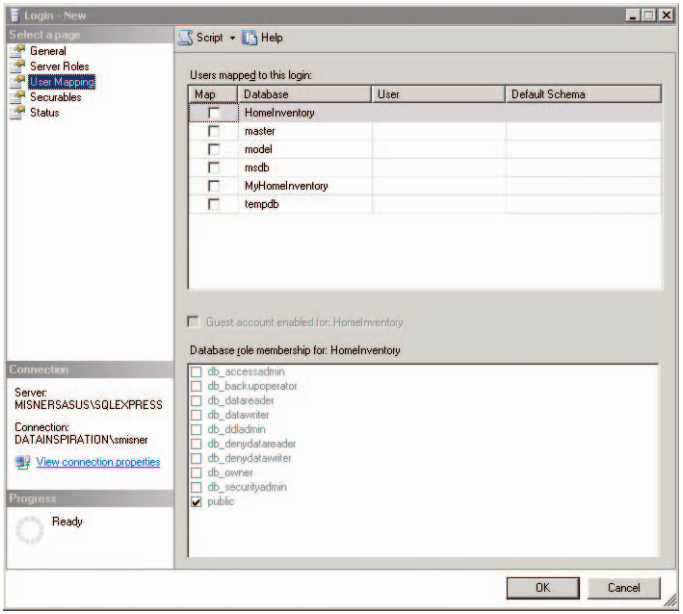



*Figure 5-11*
*User Mapping page of Login dialog box*

The User Mapping page is important. Here you'll not only specify what the user can do with the database in regards to administrative tasks, such as backups or security configuration, but also what data the user can change or even see.

For this example, we'll assume your family member should be able to add and change data, but you don't want this person to delete data. The db_datawriter role fits these requirements most closely because it allows someone to read, change, or delete data. But remember, you don't want to grant delete permissions. That's okay, because you can assign this user to the db_datawriter role and then change the role to remove the delete permissions. Let's also assume for this example that you want to delegate responsibility for backing up the database periodically, so you'll assign this user to the db_backupoperator role as well.

3 To assign the roles described in the preceding step, select the **Map** check box to the left of MyHomeInventory and then select the **DB_Backupoperator** and **DB_Datawriter** check boxes in the Database Role Membership section of the Login dialog box, as shown in Figure 5-12.

**NOTE**

**As you learned in Chapter 4, "Working with Transact-SQL," anything you can do using the graphical interface in Management Studio can be done with T-SQL commands. You can choose either of these tools to perform security-related tasks as well. To learn more about using T-SQL to configure security for logins and roles, refer to SQL Server 2005 Books Online and search for the topics "Create Login" and "Create Role."**

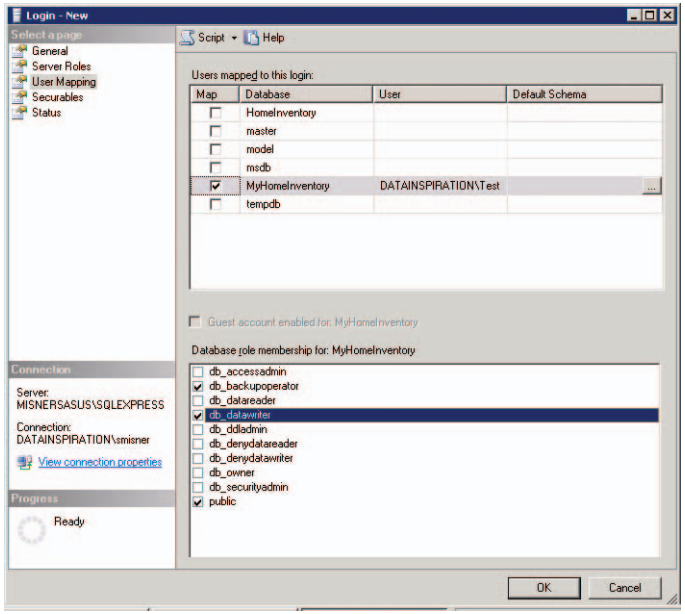



*Figure 5-12*
*User Mapping page of Login dialog box with database role assignments*

Notice the public database role is already assigned to this user by default. If your security needs are very simple, you could define permissions for the public role in this database, and then all new logins you add to SQL Server Express are assigned the same authorization level without any further effort on your part.

4 Click **OK** to close the Login dialog box.

## Granting Permissions

Permissions are defined automatically for each role, which you can use to manage multiple principals easily when those principals should have the same access privileges. Alternatively, you can define permissions for an individual login when the access privileges for that login should be distinctly different from the larger group of authorized principals. As I mentioned earlier in this chapter, you use permissions to tell SQL Server Express exactly what a principal can and can't do in a database. For example, you can explicitly block someone from using

any stored procedures in a database by denying permissions. Or you might allow someone to read your data but prevent them from making any changes. Not only can you control access to specific objects and to data, you can also use permissions to specify whether a principal is allowed to create, change, or delete database objects such as tables, views, or stored procedures, to name a just a few options.

Remember that, in the example we've been working on in this chapter, we've already decided your family member should be able to add or change data, but not delete data. You have already assigned the login to the db_datawriter role, but now you want to override the default permissions granted to that role. To create more explicit permissions, follow the steps below.

## TO GRANT PERMISSIONS

1. To override default permissions, you must configure security at the database level by explicitly granting or denying permissions. In Object Explorer, expand the **Databases** folder, expand **MyHomeInventory**, expand **Security**, expand **Users**, right-click the login you added, and click **Properties**. The Database User dialog box, shown in Figure 5-13, displays.

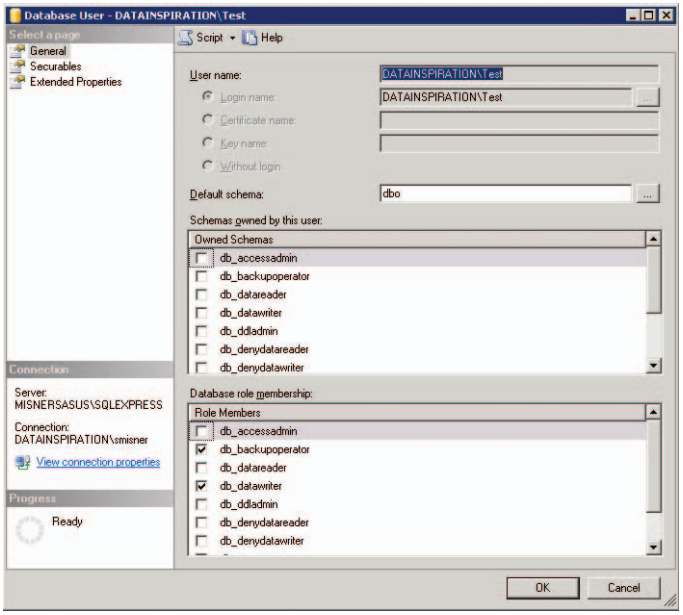



*Figure 5-13*
*Database User dialog box*

Notice the database roles you assigned in the previous section are visible here. These roles are just fine, so there's no need to make a change.

2 Click **Securables** to display the Securables page of the Database User dialog box, as shown in Figure 5-14. The page is always empty when first opened. You need to identify specific objects to review the explicit permissions configured for each one.

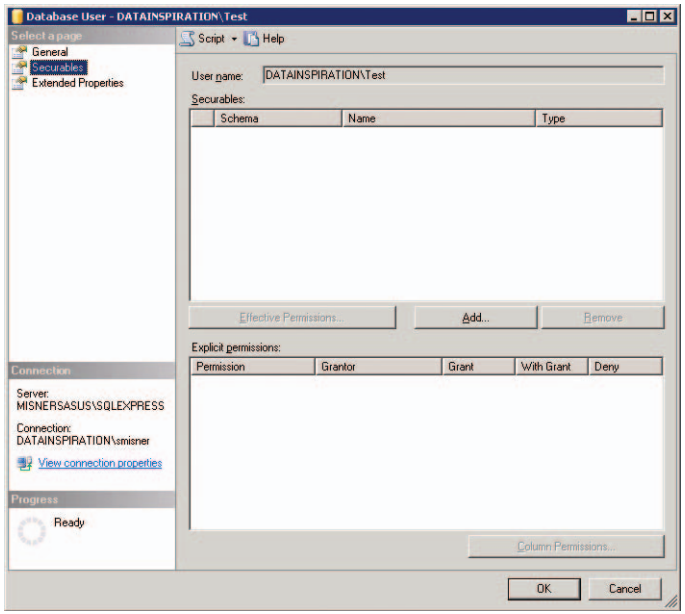



*Figure 5-14*
*Securables page of the Database User dialog box*

3 Because we do want to change explicit permissions, let's find the securable object. Click **Add** to display the Add Objects dialog box, as shown in Figure 5-15.

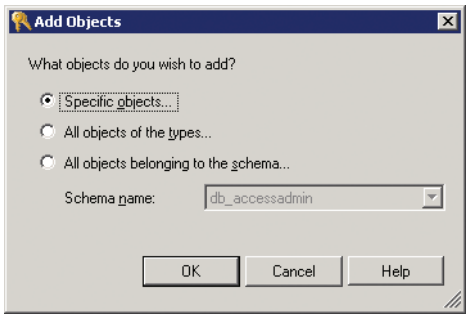



*Figure 5-15*
*Add Objects dialog box*

Your selection here determines the action performed when you click OK. By selecting Specific Objects, you indicate that you want to see a list of available object types to use as a filter for eventually selecting a specific table or view, for example. The All Objects Of The Types selection enables you to simply select the object types, and then all database objects with the type, such as all tables, are added to the list of securables. Lastly, the option All Objects Belonging To The Schema enables you to select a specific database role, and then all objects associated with that role are added to the list of securables.

4 We'll keep things simple by choosing only those object types we're interested in—tables, views, and stored procedures—and then choosing specific objects from the resulting list of available objects. With Specific Objects selected, click **OK** to display the Select Objects dialog box.

5 Click **Object Types** to display the Select Object Types dialog box. Select the **Stored procedures**, **Tables**, and **Views** check boxes, as shown in Figure 5-16.

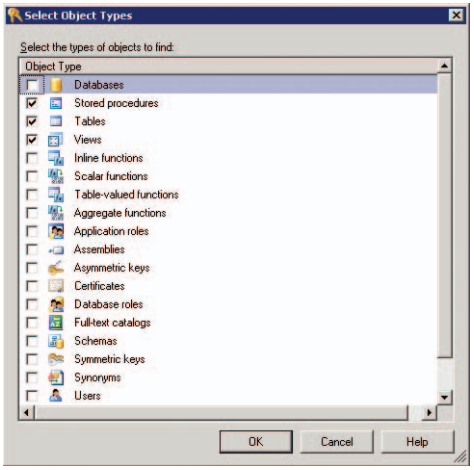



*Figure 5-16*
*Select Object Types dialog box*

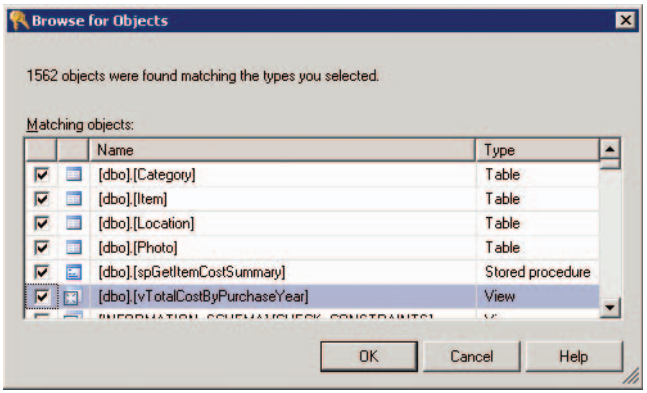



*Figure 5-17*
*Browse For Objects dialog box*

**6** Click **OK** and then click **Browse** to view the Browse For Objects dialog box, which displays a list of the objects in the MyHomeInventory database. Let's refine the requirements to limit the new login's access to all tables in the database, the spGetItemCost-Summary stored procedure, and the vTotalCostByPurchaseYear view. Select the applicable check boxes, as shown in Figure 5-17.

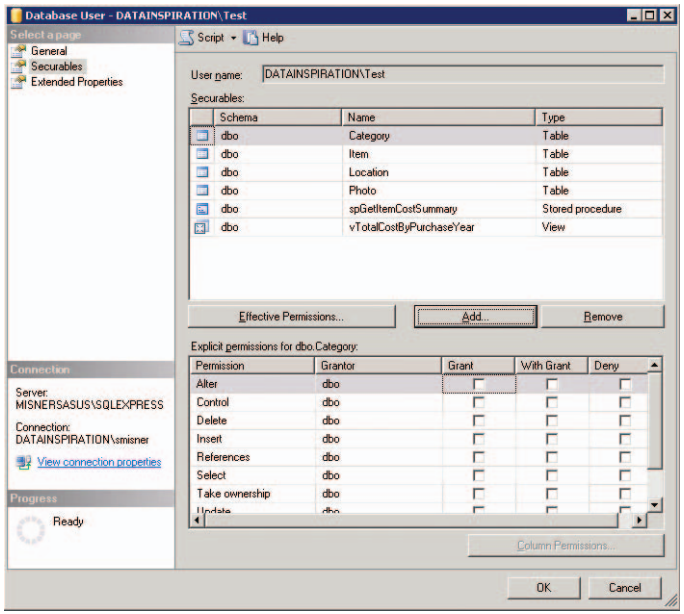



*Figure 5-18*
*Securables page of the Database User dialog box with database objects selected*

**7** Click **OK** twice to close each dialog box until you return to the Database User dialog box, as shown in Figure 5-18.

At this point, the database permissions for the assigned roles remain unchanged. Collectively, the permissions result in a set of effective permissions that resolves the interaction of all permission settings to determine what the principal can and can't do. If you're ever unsure of the current access a principal has, you can double-check the principal's effective permissions, as we'll do in the next step.

**8** Click **Effective Permissions** to view the permissions for the Category table, as shown in Figure 5-19.

You can see here that the principal can delete rows, insert rows, and make modifications to rows without restriction. Click OK to close the Effective Permissions dialog box.

**9** You can override the effective permissions by specifying exactly the permissions you want to grant or deny. On the Securables page of the Database User dialog box, select the check box in the Deny column of the Delete row and select the check boxes in the Grant column of the Insert and Select rows, as shown in Figure 5-20.

**10** Continue by granting Insert and Select permissions and denying Delete permissions to the remaining tables. Notice we're going to ignore the view for now.

**11** Check the effective permissions on the stored procedure and notice there are none. You must explicitly grant Execute permissions to allow the login to use this stored procedure. Click **OK** to finish. Your new login is now configured to use the MyHomeInventory database, albeit in a limited way.

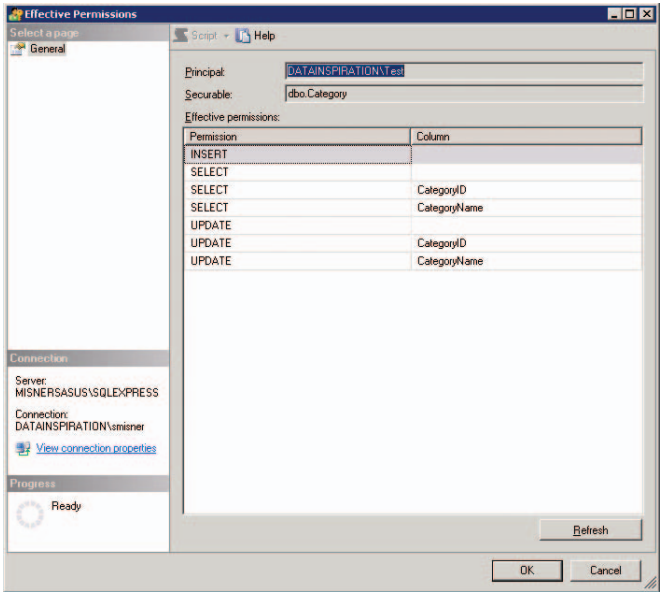



*Figure 5-19*
*Effective permissions for the Category table related to the selected principal*

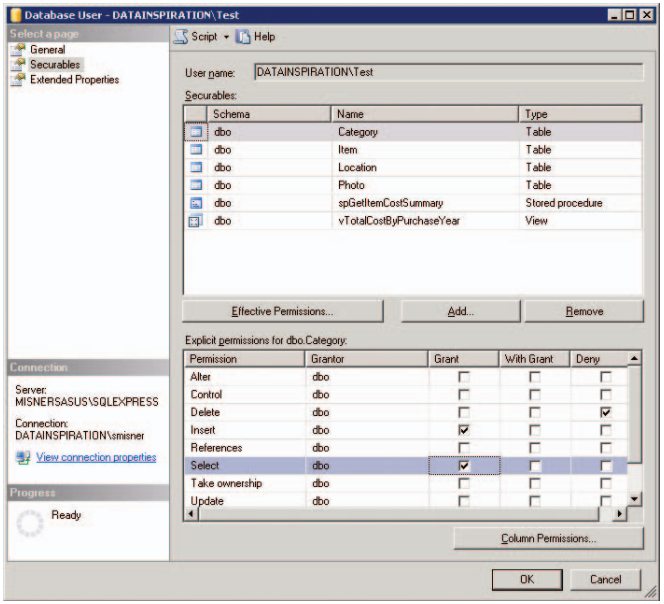



*Figure 5-20*
*Creating explicit permissions in the Category table*

**NOTE**

**When you select a table in the Securables list, the Column Permissions button is enabled when you click a row that allows you to set column-specific permissions. If you click this button, you can grant or deny permissions to selected columns for an even more detailed degree of security.**

## Testing Permissions

The best way to understand the effect of the steps you've completed so far in this chapter is to try to perform specific tasks. That is, you should try to view database objects that the login was not authorized to view by impersonating the login you created. You should also make sure the login can see objects that were authorized explicitly. Furthermore, you should ensure you can execute only those stored procedures authorized through explicit permissions and you cannot perform unauthorized operations, such as DELETE.

Remember that, in the example we've been working with in this chapter, the login is able to add or change data but not to delete it from the database. This login can view definitions of authorized objects but cannot make changes to these objects. In addition, you allowed Execute permissions on the stored procedure. Let's test security by using the following steps to impersonate this principal.

### TO TEST PERMISSIONS

1. Though it's not absolutely necessary to close Management Studio to test user permissions, you'll probably find it less confusing if you shut down that application now because you're going to open a new instance of Management Studio that you'll run in the context of the login you created. Having one instance open at a time will ensure you are absolutely working with the right instance of Management Studio.

2. To open a new instance of Management Studio, point to **Start**, point to **All Programs**, point to **Microsoft SQL Server 2005**, and right-click **SQL Server Management Studio**. Then do one of the following steps.

- If you're running Windows Server or Windows XP, click **Run As** to open the Run As dialog box, shown in Figure 5-21.

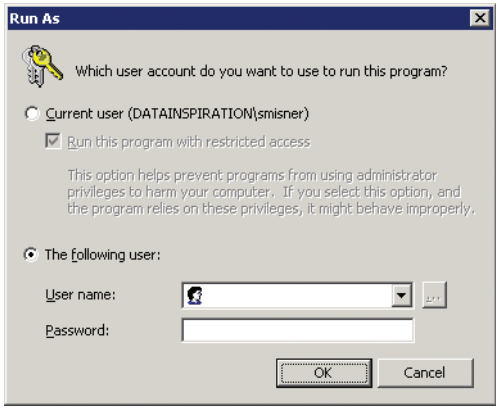



*Figure 5-21*
*Run As dialog box*

- If you're running Windows Vista, click **Run As Administrator**.
  - If you're not a member of the Administrators group on your computer, a prompt displays for you to provide a user name and password. Proceed to step 3.
  - Otherwise, if you are an administrator, you need to open the program from the command prompt window. Click the **Start** button, type **cmd** in the Start Search box, and select **cmd.exe** in the Programs list. In the command prompt window, type **runas.exe /user:*domain\username* "C:\Program Files\Microsoft SQL Server\90\Tools\Binn\VSShell\Common7\IDE\ssmsee.exe"** and press **Enter.** Supply a password at the prompt and proceed to step 4.

**NOTE**

**Replace *domain\username* with the user account you created earlier in this chapter. Omit the domain name and backslash if your computer is not part of a domain.**

3. Change the value in the User Name box to the login for your family member. Don't forget to include the domain name if your computer is part of a domain. For example, type MyDomain\MyUser for a login MyUser in the domain MyDomain. If your computer is not part of a domain, simply type the login name. Either way, be sure to type the password and then click **OK**.

4. The Connect To Server dialog box displays. Verify the Server Name box displays the correct instance name for SQL Server Express. If not, in the Server Name drop-down list, select **Browse For More**. In the Browse For Servers dialog box, expand Database Engine, select your SQL Server Express instance, and click **OK**.

5. With the SQL Server Express instance displayed in the Connect To Server dialog box, click **Connect**.

6. First, let's verify this login can view only the MyHomeInventory database. Expand the **System Databases** folder and expand **HomeInventory**. A message box, as shown in Figure 5-22, displays to let you know you cannot access the HomeInventory database. Click **OK** to close the message box. Now try opening MyHomeInventory. You should now be able to expand the various folders for this database and explore their contents.

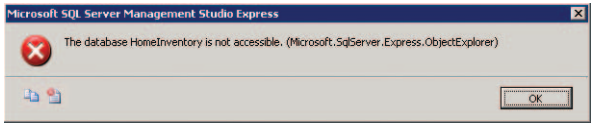



*Figure 5-22*
*Error resulting from attempting to access a database for which the login is not authorized*

7 Next, let's try to view the stored procedure. Expand **MyHomeInventory**, expand the **Programmability** folder, expand the **Stored Procedures** folder, right-click **dbo.spGetItemCost-Summary**, and click **Execute Stored Procedure**. In the Value box for the @Category parameter, type **1** and then, for the @PurchaseYear, provide a value of **2003**, as shown in Figure 5-23.

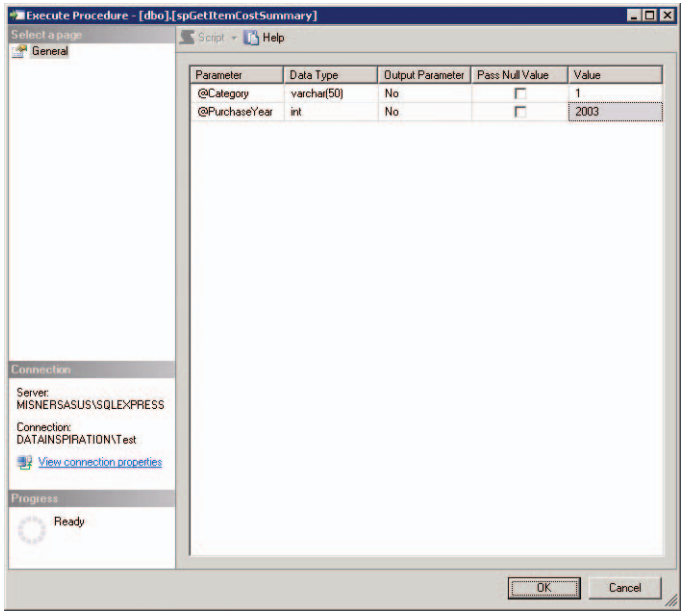



*Figure 5-23*
*Specifying parameters in the Execute Procedure dialog box*

Click **OK** to continue. The Query Editor displays the script generated by SQL Server Express to execute the stored procedure and the results of executing the stored procedure, as shown in Figure 5-24. The successful completion of these steps shows that this login has been authorized to use this stored procedure. The result set is empty because this database currently has no data. Remember, it's just a copy of the structure of the original database and not a copy of that database's data.

8 Click **New Query** and type the following statement:

```
Select * from vTotalCostByPurchaseYear
```

Click **Execute** to see the execution results, as shown in Figure 5-25. Remember, we didn't specify any explicit permissions, and there weren't any effective permissions, so you can't use this database object at all with this login. You'll also receive an error if you try to open the view by using Management Studio.

9 Lastly, let's see how this login can interact with the tables. Remember from the previous two chapters, you can either use Management Studio to open a table or write a SELECT statement to view the table contents. You should be able to open all tables in the database. Try inserting a row into the Location table and then deleting it.

**TIP**

**If you ever discover behavior you don't expect, such as the display of an error message when you believe you have permissions to access an object, double-check the effective permissions of the object and make any necessary corrections on the Securables page to reset the explicit permissions.**

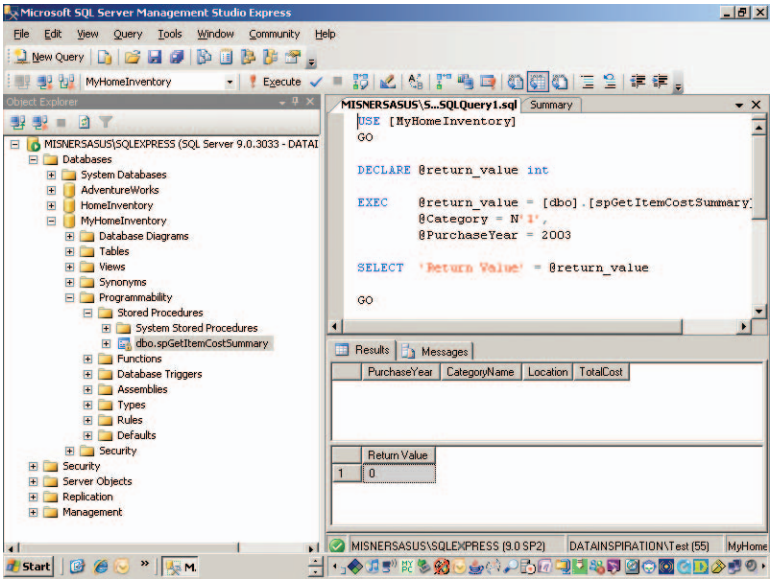



*Figure 5-24*
*Executing the spGetItemCostSummary stored procedure*

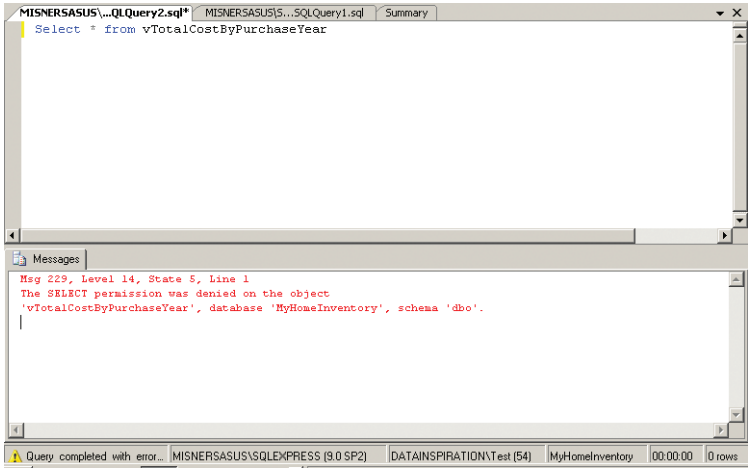



*Figure 5-25*
*Executing a SELECT statement on a view for which the login is not authorized*

In addition to setting security, several other administrative tasks are commonly performed. First and foremost, creating a regular backup is vital. In this chapter, I'll explain the differences between the types of backup you can perform and will show you how to create a backup. Of course, you'll need to know how to restore your backup as well, so I'll lead you through the applicable steps and discuss some reasons you might restore your database for reasons other than recovering from disaster. Lastly, I'll show you an alternative to working with copies of databases by helping you detach and attach your database.

## Selecting a Recovery Mode

Quite simply, a backup is a copy of your database. However, SQL Server Express enables you to create three different types of backups, which means you can create different types of copies of your database. The type of backup you can create is determined by how additions, changes, and deletions in your database are logged. The level of detail logged, in turn, is determined by the recovery mode you specify for the database. If you have multiple databases, you could use a different recovery mode for each database. Before we plunge into creating a backup, let's review the available recovery modes.

**Full** Using this recovery mode, you have database backups and transaction log backups. If you were to restore a backup for a database configured to use the Full recovery mode, you would be able to recreate a copy of the database as it existed at a point in time, and you would be able to restore subsequent transaction logs to update the database with activity that occurred after the database backup, up to a different point in time. Backing up transaction logs frequently might run faster than backing up an entire database.

**Bulk-Logged** All activity except bulk operations is logged the same way activity is logged in Full recovery mode. Bulk operations are only minimally logged to enable a very fast backup. You must have a backup of bulk data kept elsewhere if you need to reproduce the bulk operations after restoring a backup for a database using the Bulk-Logged recovery mode.

**Simple** In Simple recovery mode, the backup process truncates the transaction log. The

NOTE

**Bulk operations are a special type of import, export, or copy of data that is optimized for very fast performance. I haven't brought up the concept of bulk transactions previously in this book because it's not likely to be relevant until you're working in a business environment that works with huge volumes of data.**

backup itself does not include transaction log data, and you cannot create a backup of the transaction log as you can with a database using the Full recovery mode. It simply represents a point in time. The Simple recovery mode makes it easy to manage backup and restore processes.

The differences between these recovery modes generally matter most in business environments where the volume of data changing and the speed at which it is changing must be balanced against the amount of time required to do the backup. Your home inventory database isn't likely to become so large that you must carefully consider which recovery mode works best for you. For this reason, I recommend you change your database settings to use the Simple recovery mode, which is the default in SQL Server Express. Though you won't change the recovery mode, follow the steps below to learn where this setting is configured.

## TO CHECK THE RECOVERY MODE

1. Right-click **MyHomeInventory** database and select **Properties**. The Database Properties dialog box, shown in Figure 5-26, displays.

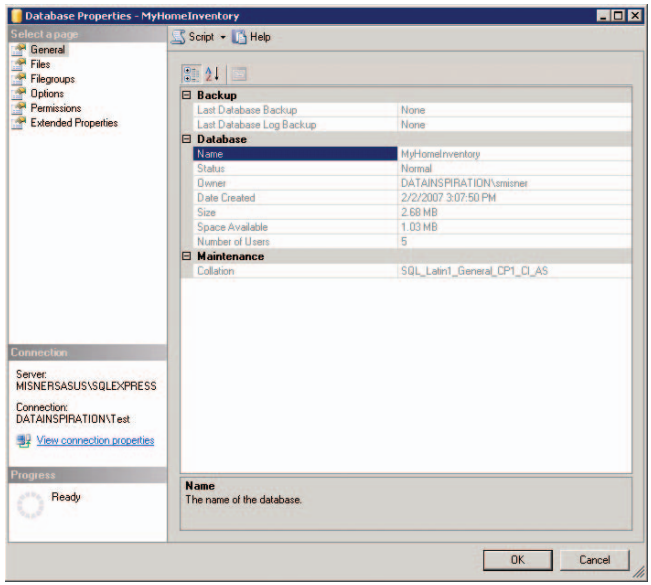



*Figure 5-26*
*The Database Properties dialog box*

2 Click **Options** to view the next page of this dialog box, shown in Figure 5-27.

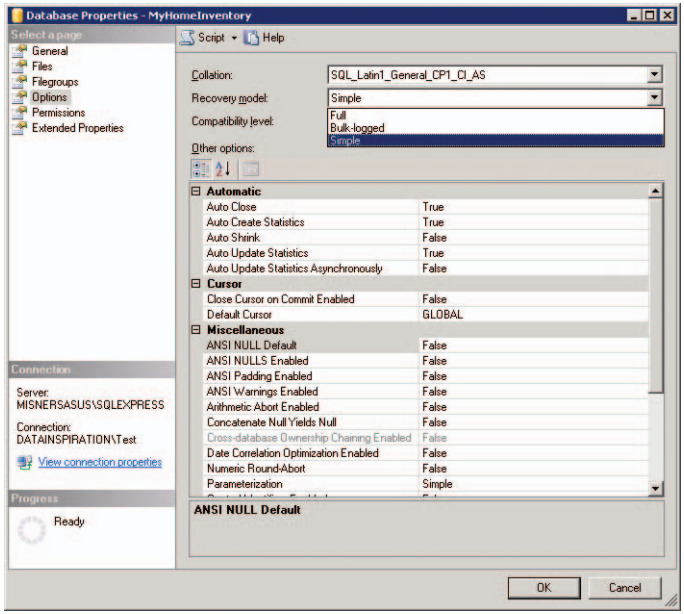



*Figure 5-27*
*The Options page of the Database Properties dialog box*

3 Examine the selections available in the Recovery Model drop-down list. Click **Cancel** to close the Database Properties dialog box.

## Backing Up Your Database

Please, please, please—back up your database and store the backup file in a safe place. If you ever need your home inventory database because your home is damaged, you'll be better prepared to work with your insurance agent if you stored your backup copy of the database in a safe deposit box at your bank. This is the most important database administration task you'll ever perform. Hopefully, you'll never need to restore your backup, but you'll be glad you have one if the time ever comes.

## TO BACK UP YOUR DATA

1. Right-click **MyHomeInventory**, select **Tasks**, and click **Back Up** to display the dialog box shown in Figure 5-28.

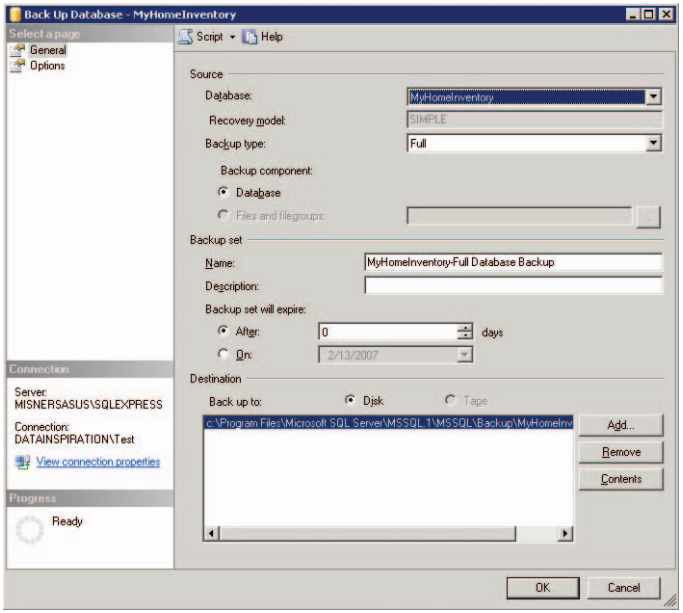



*Figure 5-28*
*The Back Up Database dialog box*

Notice the Backup Type option, which is currently set to Full. There are two ways you can perform a backup. Your first choice is a full backup, which copies the entire database and, if you are using Full recovery mode for the database, the transaction log. The Differential backup copies only data that has changed since the last backup was performed. This option is good when you have lots of activity in the database because it gives you a great deal of flexibility about what to restore, but it isn't required for your home inventory database.

Notice also the default name for the backup. You can certainly provide your own name. You can set an expiration date for the backup after which you can overwrite the file as part of a new backup cycle. Lastly, you can see the file to which the backup data will be stored. I recommend you keep the default values.

2 Click **Options** to view the other page in this dialog box, as shown in Figure 5-29.

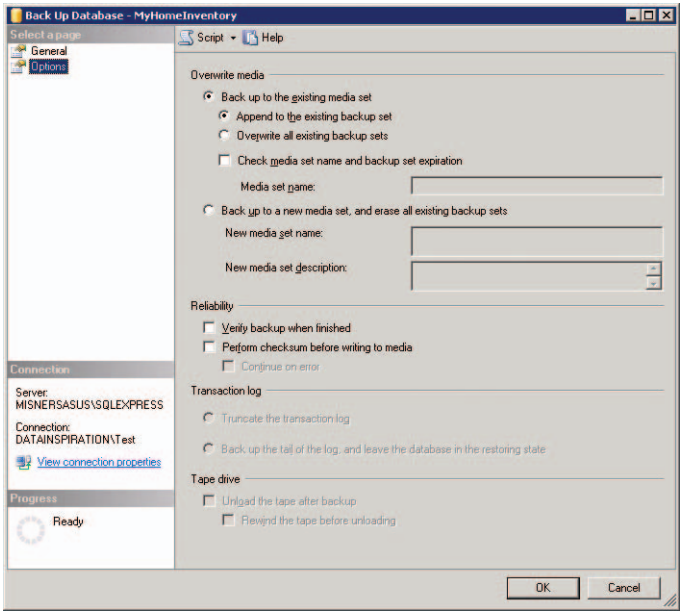



*Figure 5-29*
*The Options page of the Back Up Database dialog box*

Essentially, these options give you more control than just using the expiration date does to enable overwriting data for reusable backup media. There's no need to change anything here.

3 Click **OK** when finished. The backup process begins. When it's complete, a message box, shown in Figure 5-30, displays to inform you of the backup status. Click **OK** to close the message box.

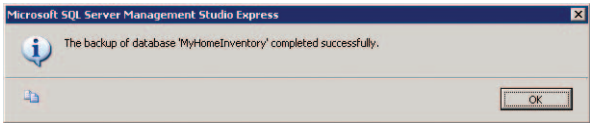



*Figure 5-30*
*Confirmation of successful backup*

## Restoring Your Database

Beyond disaster recovery, you should know how to perform other database administrative tasks. For example, you might want to experiment with changes to your database design with actual data already there without damaging your data in case you decide you're not ready to implement your new ideas. You can restore your database as a separate copy in the same SQL Server Express instance to use as your experimental playground. As another option, you might need to move a database from one server to another. Again, restoring your backup to another server is an easy way to accomplish this task.

### TO RESTORE A BACKUP AS A SEPARATE COPY

**1** Because your family member's login doesn't have permissions to restore a database, you must open Management Studio using your own identity, or else you'll run into trouble later in this procedure. Right-click **MyHomeInventory**, select **Tasks**, point to **Restore**, and click **Database**. The Restore Database dialog box displays, as shown in Figure 5-31.

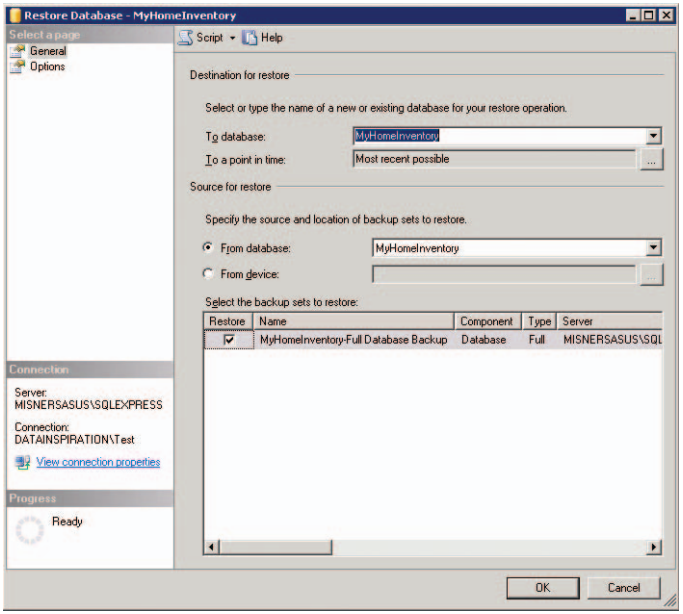



*Figure 5-31*
*The Restore Database dialog box*

2. In the To Database box, type a new name, such as **MyHomeInventory_Copy**. By specifying a unique name here, you are adding another copy of the database to this instance of SQL Server Express rather than overwriting an existing copy.

If you were using Full recovery mode on the database, you could also set the point in time to which the restore process should add activity from the transaction logs to the data captured in the backup. Of course, because you're using Simple recovery mode, point-in-time restoration won't work for you.

3. Click **Options** to review the additional settings you can configure for the restore process, as shown in Figure 5-32, and click **OK**.

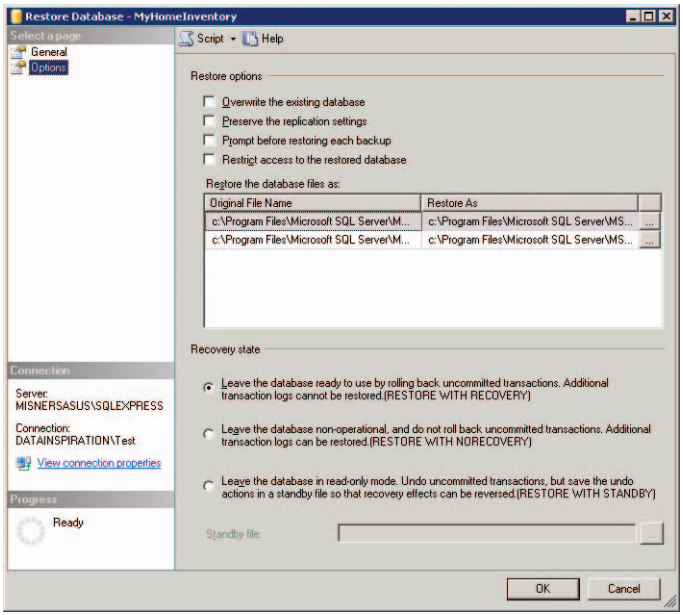



*Figure 5-32*
*The Options page of the Restore Database dialog box*

4. When the database restores successfully, a message box, as shown in Figure 5-33, displays. Click **OK** to close the message box.

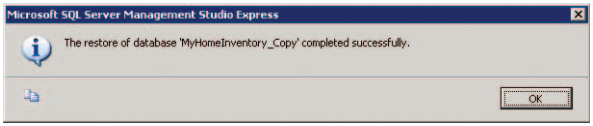



*Figure 5-33*
*Successful restoration of the database*

## Attaching a Database

A quicker way to move a database is to simply detach the database from one SQL Server instance and attach the database to another instance. In this final section of the chapter, you'll learn how to detach a database and then practice attaching it again into the same SQL Server Express instance.

### TO DETACH AND ATTACH A DATABASE

1. Right-click **MyHomeInventory**, point to **Tasks**, and click **Detach**. The Detach Database dialog box displays, as shown in Figure 5-34.

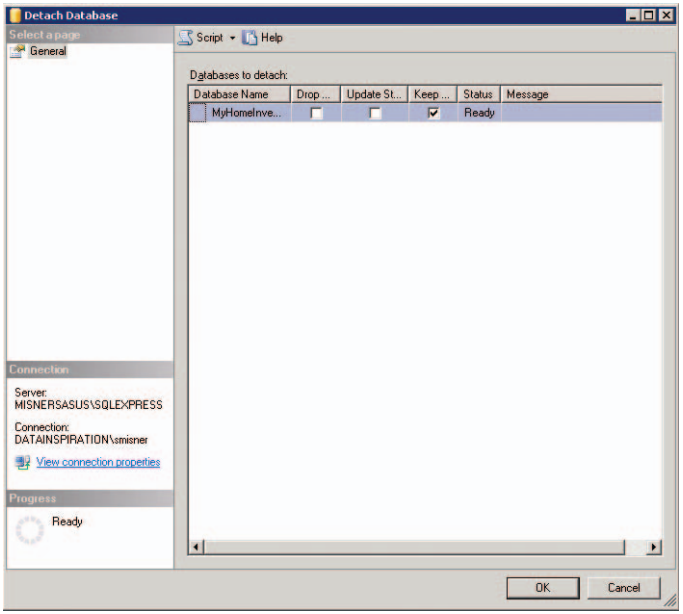



*Figure 5-34*
*Detach Database dialog box*

Notice you have several options here. You could detach the database and drop any existing connections. This is the simplest option. The next option is to update statistics in the detached version of the database. SQL Server Express uses statistics to optimize query performance. Because this database is empty, it won't do any good to update the statistics now. The third option is Keep Full Text Catalogs. This option keeps the files containing data for the full text search engine. You can find all files related to the database in the installation directory, which by default is C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data. The path on your computer might be different if you changed the location during installation, or if you are running multiple editions of SQL Server 2005 on the same computer.

**NOTE**

**Because you don't have full text search enabled in your instance of SQL Server Express, the option to keep the files has no effect. You can clear the check box or keep the selection with no impact on the detach task.**

2 Click **OK** to detach the database. Notice the Object Explorer window no longer displays MyHomeInventory, as shown in Figure 5-35.

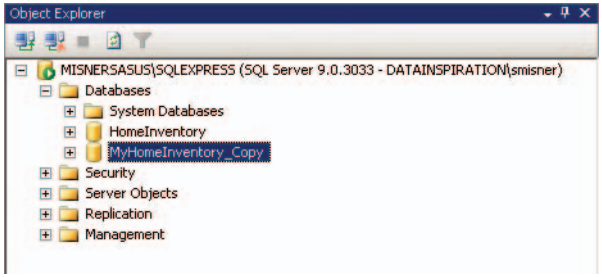



*Figure 5-35*
*Object Explorer shows detached MyHomeInventory is no longer visible in the window*

At this point, you could copy the files to another server that you might want to use to host your database. You could then attach the database to any SQL Server instance available. However, I'll assume you don't really have another machine configured to run SQL Server at the moment, so we'll simply attach the database to the same instance we detached it from.

3 Continue to step 4 if you're running Windows Server or Windows XP. If you're running Windows Vista with User Account Control (UAC) enabled (which it is by default), you don't have permissions to the database file any longer, so you'll need to give yourself Full Control permission first. To do this, follow these steps.

- Click **Start**, click **Computer**, and navigate to **C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data**. You may have to respond to the prompts to elevate your privileges to reach this folder.
- Right-click **MyHomeInventory.mdf** and click **Properties**.
- In the MyHomeInventory.mdf Properties dialog box, click the **Security** page.
- Click **Advanced**.
- In the Advanced Security Settings For MyHomeInventory.mdf dialog box, click **Edit**.
- A new instance of the Advanced Security Settings For MyHomeInventory.mdf dialog box displays. This time, click **Add**.
- In the Select User Or Group dialog box, click **Advanced**.
- Click **Find Now**.
- Locate your user name in the Search Results, select your name, and click **OK**.
- Click **OK** again to close the Select User Or Group dialog box.
- In the Permission Entry For MyHomInventory.mdf dialog box, select the **Full Control** check box in the Allow column and click **OK**.
- In each instance of the Advanced Security Settings For MyHomeInventory.mdf dialog box, click **OK**. That is, click **OK** twice to close both dialog boxes.
- Finally, in the MyHomeInventory.mdf Properties dialog box, click **OK**.
- Repeat this sequence of steps to give yourself Full Control permission for the log file, MyHomeInventory_log.ldf. You now have the necessary permissions to proceed with the next step.

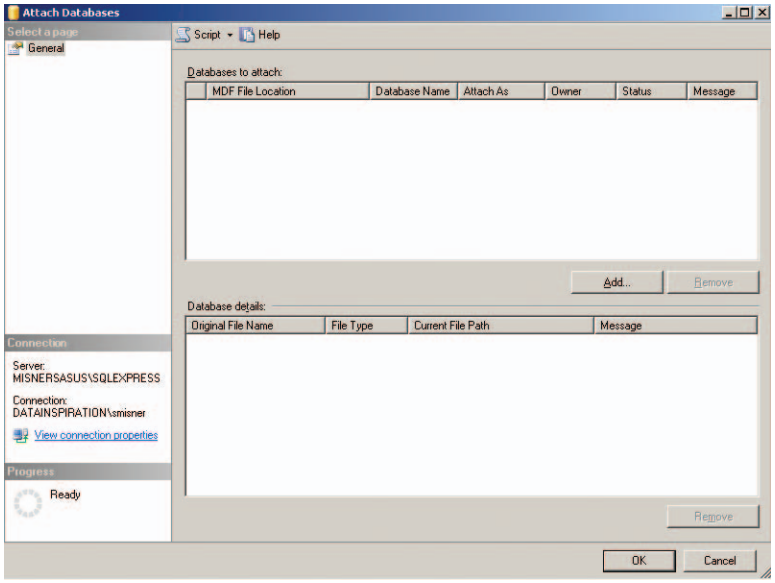



*Figure 5-36*
*Attach Databases dialog box*

**4** Right-click the **Databases** folder and click **Attach**. The Attach Databases dialog box, shown in Figure 5-36, displays. Click **Add**.

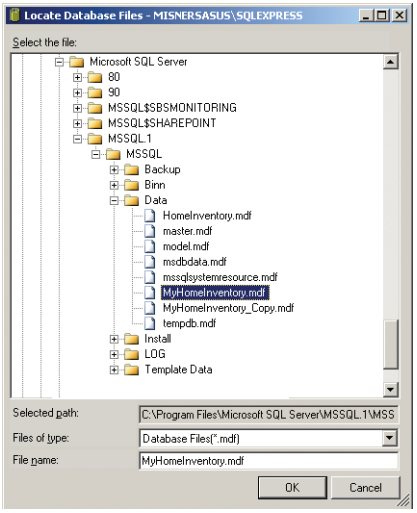



*Figure 5-37*
*Locate Database Files dialog box*

**5** In the Locate Database Files dialog box, navigate to the folder containing the MDF file representing your database and select the file, as shown in Figure 5-37. Click OK.

6 Now the Attach Databases dialog box displays the database file to attach in addition to the transaction log file associated with that database, as shown in Figure 5-38. You could take this opportunity to rename the file by clicking in the Attach As box and typing a new name, but let's leave the name in its original state for now. Click **OK**.

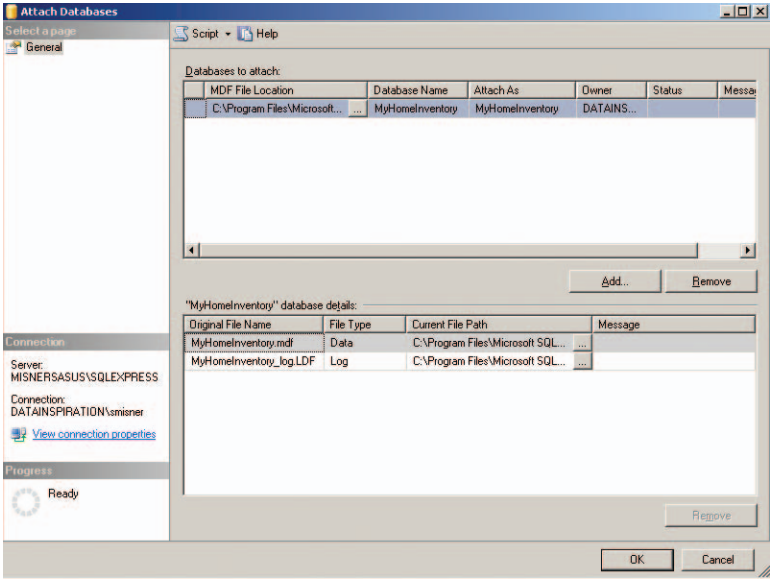



*Figure 5-38*
*Attach Databases dialog box with database files to attach*

7 Object Explorer once again includes the MyHomeInventory database, as shown in Figure 5-39. The database is ready for use by you and anyone you designate.

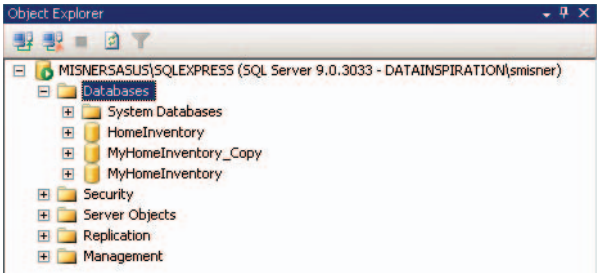



*Figure 5-39*
*Object Explorer shows attached MyHomeInventory*

## In Summary...

Do you remember what you knew about databases before you started reading the first chapter? By completing all of the steps in each subsequent chapter, you've incrementally built up your skills such that you can now create a database, load it with data, and query it. In this chapter, you extended these skills by learning the most important administrative tasks that protect your efforts. You spent a significant portion of this chapter learning about security and how to configure logins for authentication and authorization. You even learned how to test security to ensure you configured authorization correctly.

The final part of the chapter focused on ensuring your work would not be lost due to hardware failure or other disaster by showing you how to back up your data and how to later restore it when needed. Finally, you learned how attach and detach as alternatives for moving files between SQL Server instances.

By now, you have a solid foundation of skills to successfully work with SQL Server Express databases. There's only one more topic to cover before we end our time together. I've used the term *instance* quite frequently in this chapter, perhaps more so than in any previous chapter. Now it's time to explore that concept in more depth. In the next chapter, I will provide a much more complete explanation of SQL Server Express instances than I have given you so far.

Chapter 6

# Managing Your Server

*Working with Server Instances, 182*

*Using Named Instances, 185*

*Working with User Instances, 186*

At last, you've arrived at the last chapter. To round out our exploration of SQL Server Express, we're going to shift our attention to services, systems processes, and instances. Here you'll discover how SQL Server instances use your computer's resources. By developing a clear understanding of the characteristics of services and server instances, you'll be better prepared to learn later in this chapter how user instances are different and how to work with a user instance if you create one.

I first introduced the concept of Windows services in Chapter 1, "Introducing Microsoft SQL Server 2005 Express Edition," and SQL Server instances in Chapter 2, "Installing SQL Server Express." Because I've also covered many other concepts since those early chapters, let's spend time in this section reviewing the meaning of these two terms and exploring your SQL Server Express environment to provide some additional context.

## Understanding the SQL Server Express Service

A service is like other Windows programs because it is an executable file—that is, a file with an EXE file extension. However, rather than starting because a user opens the executable file, a service usually starts automatically when you start Windows on your computer, and it responds to requests from anyone logged onto your computer while Windows is running. Put another way, a service is available to everyone logging onto a computer and doesn't shut down when each person logs off, whereas a program usually starts at someone's specific request and ends when that person logs off the computer. A service uses a portion of the total memory available on the computer as long as it continues to run. SQL Server Express runs as a service that starts when your computer boots and executes the sqlserver.exe file.

You can alternatively configure the service to start manually, which allows you to prevent the allocation of server memory to SQL Server Express. However, in this situation, databases aren't available to anyone until you start the service. For this reason, I wouldn't recommend this option unless you're the only one that uses the database, and you want to run other applications requiring a lot of memory that aren't dependent on your SQL Server Express databases.

Whether a program runs as an application launched by a person or as a service launched when your computer boots, a Windows user account is associated with that program. One important reason for requiring a service to have a user account is to manage the security access of the service and thereby to ensure that it can access only the folders and files that you permit. The default installation procedure for SQL Server Express assigns a user account for you automatically, which may be the Local System account or the Network Service account. Either of these built-in service accounts is typically just fine. The Local System account is a built-in account that, when assigned as the service account, connects only to

the SQL Server Express instance running on your computer. This account will not usually be able to connect to other network servers running SQL Server. In contrast, the Network Service account can access both the local SQL Server Express instance and other network resources.

Let's take a look at the service configuration of SQL Server Express on your computer and the memory consumption of that service by following these steps:

**TIP**

**For more information about service accounts, take a look at *msdn2.microsoft.com/en-us/library/ms143170.aspx*.**

## TO REVIEW THE SQL SERVER EXPRESS SERVICE

**1** First, let's open SQL Server Configuration Manager to see how the SQL Server Express service is configured. Click **Start**, point to **All Programs**, point to **Microsoft SQL Server 2005**, point to **Configuration Tools**, and click **SQL Server Configuration Manager**. The SQL Server Configuration Manager console displays, as shown in Figure 6-1.

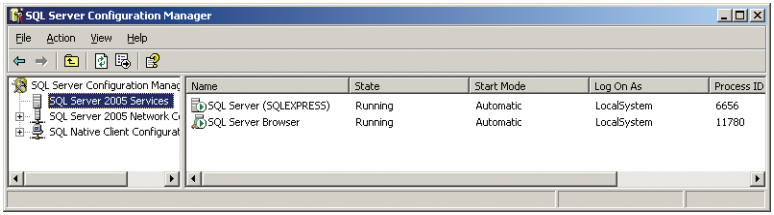



*Figure 6-1*
*SQL Server Configuration Manager*

Although many services are currently running on your computer, the SQL Server Configuration Manager enables you to focus on services related to SQL Server instances. The State column displays Running or Stopped, depending on whether the service is currently running.

Notice also the Start Mode column. When the value in this column is Automatic, the service starts when the computer reboots. It usually continues running until the computer shuts down. In some cases, a problem can force the service to stop running, at which point you'll have to perform some type of diagnosis and intervention.

Finally, notice the Log On As column, which identifies the account under which the service is running. Usually, the values configured during installation are exactly what you need, so as a general rule, I wouldn't recommend you make changes here. If you really must reconfigure the service, right-click the services, click Properties, and change the settings as you see fit in the Properties dialog box.

**TIP**

**If you ever have trouble connecting to SQL Server Express by using Management Studio or some other application, you should immediately check the SQL Server Configuration Manager console to see if the service is running. If it is not running, you can start it again by selecting the service and then clicking the Start Service button on the toolbar. If the service refuses to start, you've got a bigger problem that is more than I can help you with in this book. At the very least, in this situation, I can suggest you reboot your computer as a first attempt at resolving the problem.**

2. Now close the SQL Server Configuration Manager console so we can next take a look at how much memory SQL Server Express is using.

3. Press **Ctrl+Alt+Delete** and click **Task Manager** (or, if you're using Windows XP Home Edition, just press **Ctrl+Alt+Delete**) to open the Windows Task Manager dialog box, shown in Figure 6-2.

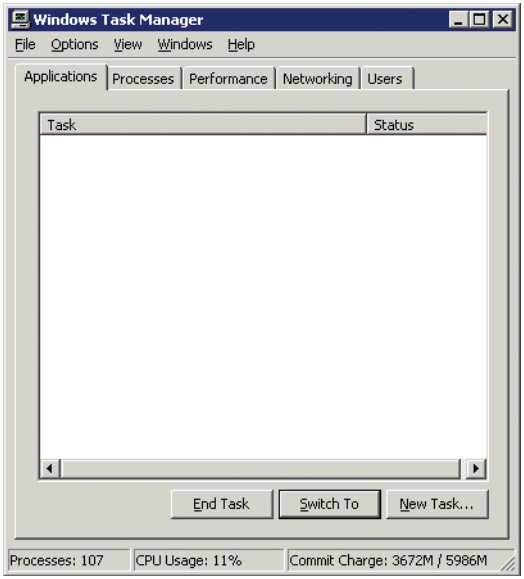



*Figure 6-2*
*Windows Task Manager dialog box*

Your screen may list one or more programs on the Applications page of this dialog box. In my case, at the time I opened the Windows Task Manager, I had no other applications running, so the page shown in Figure 6-2 is empty. If you recall, I had services running when I checked the SQL Server Configuration Manager (Figure 6-1), but notice none of those services is listed now.

4. Click the **Processes** tab and then click **Image Name** (the first column) to sort the services alphabetically. If you scroll through the services, can you find sqlserver.exe? Hmm, wasn't the service running?

5. Remember the service is not running under your user account, so you'll need to change your view on the Processes tab of the Windows Task Manager dialog box. At the bottom of the dialog box on the Processes tab, select the **Show Processes From All Users** check box. Now scroll through the services to locate sqlserver.exe, as shown in Figure 6-3.

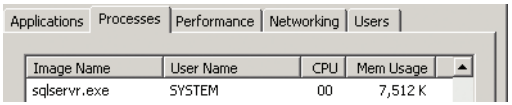



*Figure 6-3*
*Windows Task Manager showing sqlservr.exe*

Your Windows Task Manager might show some different values in the Mem Usage column, but you should be able to find the service running and view the amount of memory, expressed in kilobytes, the service is using.

6 Now that you have verified the SQL Server Express service is running and have determined how much memory the service requires to run, you have completed your tour of the service configuration for SQL Server Express. Close the Windows Task Manager.

## Using Named Instances

You can install SQL Server multiple times on the same computer as named instances, as I first discussed in Chapter 2. Why would you want to do this? Well, creating separate instances provides another way to insulate database activity. As a rather simple example, let's say you have expanded beyond a home inventory database and have built databases for you and your family to manage your music collection, recipes, or whatever suits your interests. You might find it easier to set up one SQL Server Express instance for your family to use and to create a separate SQL Server Express instance in which you develop your databases before making them available for use. That is, you have two instances installed—one instance for family use (or "production," as we'd say in the business world) and one instance for development. Each instance can host multiple databases. You can quite easily block everyone from using your development databases without going through all the security configuration steps I covered in the previous chapter.

By working with security at this type of macro level, you have less work to do when setting up user access to databases. However, the trade-off is the requirement to have more hardware resources available on your server. Each named instance runs its very own sqlserver.exe as a service and increases the consumption of memory on your computer. In addition,

each named instance has its own security configuration and set of system databases. For all intents and purposes, these named instances are isolated from one another and function completely independently.

When installing other editions of SQL Server, it's quite common to create a default instance, which is essentially an unnamed instance. You connect to a default instance simply by providing the server name. In contrast, connecting to a named instance requires both the server and instance name, as you've been doing as you work through the steps in this book. The SQL Server Express installation defaults to a named instance; however, if you select the advanced configuration option on the Registration Information page of the setup wizard, you can choose to install a default instance instead.

**NOTE**

**Each SQL Server edition supports a limited number of named instances, ranging from 16 named instances in the Standard, Workgroup, and Express editions to 50 named instances in the Enterprise edition.**

Do you need to do anything special to create a named instance? Simply provide a name on the Instance Name page of the installation wizard. When you install the first instance of SQL Server Express, you won't see this page of the wizard unless you happen to have another edition of SQL Server on your computer or you choose to run the installation with the advanced configuration options. Assuming SQL Server Express is your first installation, the wizard never asks you for an instance name because the instance name is created for you, SQLEXPRESS.

## Working with User Instances

You should now have a better understanding of why you might use a named instance and how using named instances affects your computer's resources. If you plan to build single-user database applications using Microsoft Visual Studio, you also need to understand user instances. This special feature is available exclusively in SQL Server Express.

Not all database applications require data to be shared between users and are built to work with one and only one person. This type of scenario is exactly the reason SQL Server Express includes the user instance feature.

Returning to the home inventory database we've been discussing throughout this book, let's say that you've built an application using one of the Microsoft Visual Studio Express

editions to make it easier to input data into your home inventory database and to build simple queries. Now you want to share the application and database with a friend who is less adventurous than you when it comes to working with computers. In other words, your friend might be comfortable installing the application by itself but might reconsider the whole idea if also required to install both SQL Server Express and Management Studio, and then to attach the home inventory database. Fortunately, you can set up the application with a user instance to avoid all these administrative tasks. In this section, we'll start an application for an embedded database, building it up just enough to create a user instance that we can study.

## Enabling User Instances

Although user instances are a special feature of SQL Server Express, you have to explicitly enable this feature to use it. During installation, you have the option to enable user instances on the Configuration Options page of the setup wizard. The instructions I provided in Chapter 2 suggested you enable user instances to work through this book, specifically the remainder of this chapter. If you chose not to enable user instances at that time, use the *sp_configure* system stored procedure as described in the following steps to enable user instances in SQL Server Express.

### TO ENABLE USER INSTANCES

1. In Management Studio, click **New Query** to open the Query Editor window.
2. Type the following expression:

```
sp_configure 'user instances enabled', 1
```

3. Click **Execute**. You'll see the following message display: `Configuration option 'user instances enabled' changed from 0 to 1. Run the RECONFIGURE statement to install.`
4. Replace the statement in the Query Editor with the following statement:

```
RECONFIGURE
```

5. Click **Execute**. Now you're ready to work with user instances!

**NOTE**

**If your intention is to disable user instances, follow these procedures but use the following expression instead: *sp_configure 'user instances enabled', 0*. You simply use 1 or 0 to indicate whether the feature should be enabled or disabled, respectively. After executing this statement, you then need to execute the RECONFIGURE statement.**

## Creating a User Instance

Thus far, you know that a user instance is associated with a single user, and that you need to configure SQL Server Express explicitly to use this feature. However, you are no doubt wondering how a user instance differs from a named instance, and how you create a user instance. First and foremost, a user instance allows someone who's not an administrator to attach a database on the local computer to the SQL Server Express instance. In particular, they don't need to know how to do this or even that it happens. Furthermore, the database file is located in the user profile directory, also known as the Documents and Settings folder (or the Users folder if you're using Windows Vista), which is off-limits to anyone except the owner of the folder. By creating a user instance, you're creating an exception to the rules that normally prevent administrators from attaching a database to a SQL Server Express instance and from accessing files in the user profile directory.

As I mentioned earlier in this chapter, you create a user instance as a way to associate a database with an application and insulate the user of that application from all the administrative details necessary to set up and maintain the database. When you build an application with an embedded database, you include a *connection string* that creates a new user instance and connects to that instance. A connection string is simply a string of text that tells an application, such as Management Studio or some other application that connects to SQL Server, where to find the database you want to use, what credentials to use to authorize access, and any other bits of information describing how you want to connect to the server hosting that database. Each time you connect to Management Studio and select the SQL Server Express instance in the Connect To Server dialog box, for example, Management Studio generates a connection string for you so you don't really need to know the details to successfully connect, although in that case the connection is to a named instance, not a user instance.

If you attempt to connect to a user instance that doesn't exist, you trigger several events. First, a connection opens to your existing SQL Server Express named instance, which is also known as the *parent instance*. Because the connection string for a user instance includes some important text—`"User Instance=True"`—SQL Server Express looks for the user instance and then, if no user instance is found, creates the user instance by copying the master and msdb databases from the Template Data directory of the parent instance to the user profile directory. When the user instance starts, SQL Server Express also copies the

**NOTE**

**For more in-depth information about user instances, the problems that user instances solve for application developers, and connection strings used with user instances, see *msdn2.microsoft.com/en-us/library/bb264564.aspx*.**

tempdb database from the parent instance's Template Data directory to the user profile directory, along with log and trace files. Voila! You now have a user instance that in many ways resembles its parent instance because it is associated with a set of system databases and supporting files. However, the user instance is associated with a specific user, requires Windows Authentication, and denies remote connections. Further, the system databases and files for the user instance are created from template files; therefore, they don't contain the same data found in the parent instance's system databases and files.

So how do you get started? The most common technique is to create a project in Visual Studio Express and add a data source that opens a connection and creates the user instance. That's it!

**NOTE**

**You don't need to build an entire application to create a user instance, but if you're curious about using Visual Studio Express, you should download it from *msdn.microsoft.com/vstudio/express/vb/download* and investigate the Movie Collection Starter Kit that you'll see as one of the options for a new project. This starter kit is an embedded database application that could give you many ideas for building an application for your home inventory database. A tutorial is included with the starter kit to guide you through the development process.**

If you've tried building database applications in Visual Studio Express, you may already have created user instances, even if you didn't realize it. Or if you did know you were creating them, you might be perplexed about how to use Management Studio to work with the user instance. In your situation, you already have a user instance you can use to follow the procedures later in this chapter, and you can ignore the procedure in this section.

If, on the other hand, you've never used Visual Studio Express to develop a database application, you'll need a way to create a user instance without going to the trouble of starting application development. The purpose of this book, after all, is to teach you about SQL Server Express, not Visual Studio Express. Instead, *purely for the purposes of following this chapter's procedures*, you can use SQL Server Express Utility to create a database for your user instance. You can download this utility from *go.microsoft.com/fwlink/?LinkId=44178*. Follow the instructions at that Web site to download and install the utility. During installation, you must provide a location for the extracted files, the SSEUtil.exe file (which is the utility), and the ReadMe.htm (which explains how to use the utility). I recommend you select a location that's easy to remember, such as C:\SSEUtil. If the folder you specify doesn't exist, the installation procedure will create it for you. To get started with a user instance, follow these steps.

## TO CREATE A USER INSTANCE MANUALLY

**NOTE**

**Skip this entire procedure if you already have a user instance created by developing an application. I included this procedure for those readers who are curious about working with user instances but aren't ready to develop a database application.**

1. Open the command prompt window and type **cd C:\SSEUtil**, replacing SSEUtil with the folder to which you extracted SQL Server Express Utility (if you specified a different location).

2. Open a console window to make it easier to work with commands and scripts. Type the following statement and press **Enter**:

```
sseutil -consolewnd
```

SSEUtil Interactive Command Window displays, as shown in Figure 6-4.

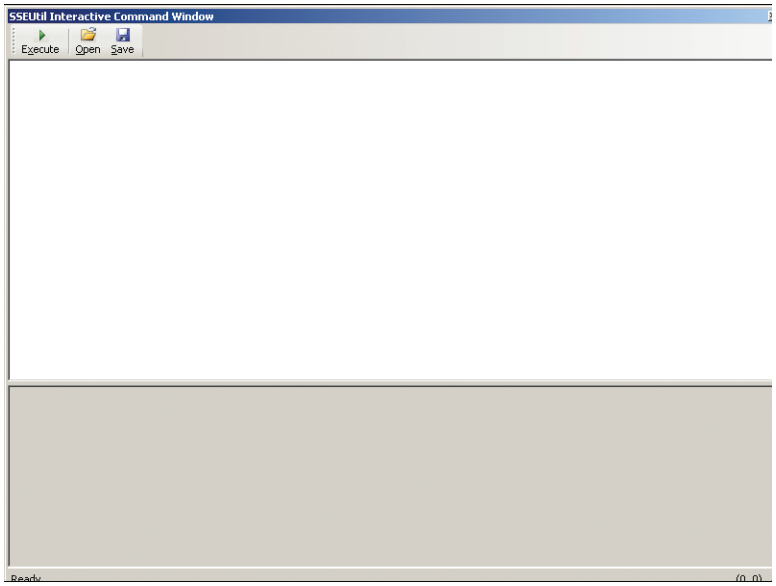



*Figure 6-4*
*SSEUtil Interactive Command Window*

3. To create a user instance, you can use the database script you created in Chapter 4, "Working with Transact-SQL." To use that script, open the CreateHomeInventory.sql file you created in Chapter 4. Click **Open** and navigate to the folder containing your script file. In the Files Of Type drop-down list, select

**All Files (*.*)** to find your file, which has an SQL extension. Select **CreateHomeInventory.sql** and click **Open**. The script now displays in the command window, as shown in Figure 6-5.

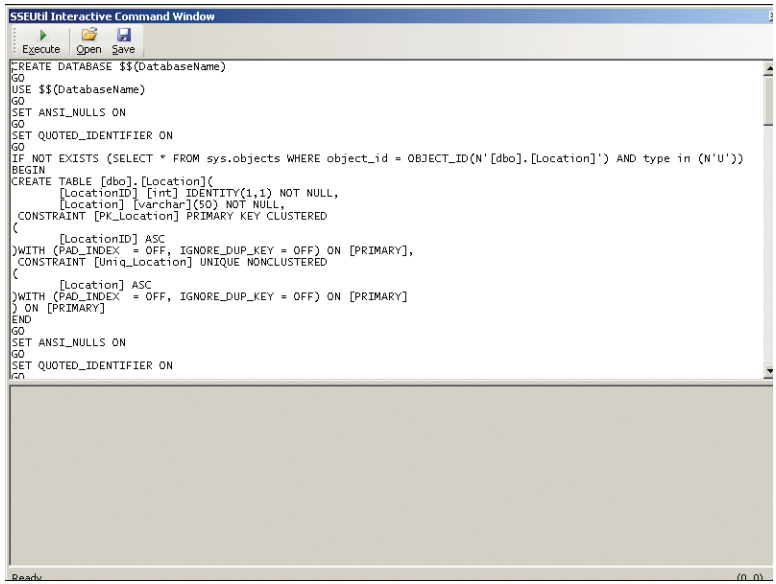



*Figure 6-5*
*SSEUtil Interactive Command Window displaying the CreateHomeInventory.sql script file*

4 Replace each occurrence of $(DatabaseName) with a unique database name. In my example, I used MisnerHomeInventory, as shown in Figure 6-6.

```
CREATE DATABASE MisnerHomeInventory
GO
USE MisnerHomeInventory
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[Location]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Location](
        [LocationID] [int] IDENTITY(1,1) NOT NULL,
        [Location] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Location] PRIMARY KEY CLUSTERED
(
        [LocationID] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [Uniq_Location] UNIQUE NONCLUSTERED
(
        [Location] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

*Figure 6-6*
*Updated script using unique database name*

5 On the toolbar, click **Execute**. The results of execution display in the bottom section of the command window as shown in Figure 6-7, which shows only the bottom of the command window. You need to scroll to review the entire set of messages resulting from the script execution. Close the command window when you're finished reviewing the results.

```
1> CREATE DATABASE MisnerHomeInventory
2> GO

Command completed successfully.

1> USE MisnerHomeInventory
2> GO

Command completed successfully.

1> SET ANSI_NULLS ON
2> GO

Command completed successfully.
```

*Figure 6-7*
*Results of executing the script*

6 Let's see what happened on your computer now that a user instance exists. Press **Ctrl+Alt+Delete** and click **Task Manager** (or, for Windows XP Home users, simply press **Ctrl+Alt+Delete**) to open the Windows Task Manager dialog box. Click the **Processes** tab and click **Image Name** (the first column) to sort the services alphabetically. Now look for sqlserver.exe with your login name listed in the User Name column. That's the user instance! Remember that SQL Server Express as a named instance usually runs under a service account such as Local System or Network Service.

The user instance continues running even if the application that started the instance is no longer running. Eventually, after 60 minutes by default, it shuts itself down. Unlike a named instance service, the user instance doesn't restart when your computer restarts. It only starts when you connect to it.

7 In the next section, I'm going to show you how to start an instance without using Visual Studio Express. To prepare, let's go ahead and shut down sqlservr.exe while we're here in the Windows Task Manager. Be sure to select the one with your login associated with it and then click **End Process**. Now close Windows Task Manager by opening the **File** menu and clicking **Exit Task Manager**. The specific steps you use to close Windows Task Manager depend on your operating system.

**NOTE**

**If you're running Windows Vista, you'll need an account with administrator privileges to access the Local Settings folder, which is found in the folder C:\Users*<your login account>*.**

8 Finally, let's have a look at the file system. Open Windows Explorer and navigate to the Local Settings folder in the C:\Documents and Settings*<your login account>* folder. You might have to reconfigure the file system to show hidden files and folders by opening the **Tools** menu, selecting **Folder Options**, clicking the **View** page, and selecting **Show Hidden Files And Folders**.

9 Once you've located the Local Settings folder, navigate to the Application Data\Microsoft\Microsoft SQL Server Data\SQLEXPRESS folder. In this folder, you should see several files, similar to those in the Windows Explorer window shown in Figure 6-8.

| Name | Size | Type | Date Modified | Attribi |
|---|---|---|---|---|
| Entropy.bin | 1 KB | BIN File | 2/28/2007 11:12 PM | HA |
| error1.log | 18 KB | Text Document | 3/8/2007 3:03 PM | A |
| error2.log | 13 KB | Text Document | 3/1/2007 1:12 AM | A |
| error3.log | 17 KB | Text Document | 2/28/2007 11:21 PM | A |
| error4.log | 0 KB | Text Document | 2/28/2007 11:12 PM | A |
| error.log | 14 KB | Text Document | 3/8/2007 6:08 PM | A |
| log.trc | 8 KB | TRC File | 2/28/2007 11:33 PM | A |
| log_1.trc | 128 KB | TRC File | 3/1/2007 1:12 AM | A |
| log_2.trc | 128 KB | TRC File | 3/8/2007 3:03 PM | A |
| log_3.trc | 22 KB | TRC File | 3/8/2007 6:19 PM | A |
| master.mdf | 4,096 KB | Database Primary File | 3/8/2007 6:19 PM | A |
| mastlog.ldf | 768 KB | Database Log File | 3/8/2007 6:19 PM | A |
| MisnerHomeInventory.mdf | 1,216 KB | Database Primary File | 3/8/2007 6:08 PM | A |
| MisnerHomeInventory_log.LDF | 504 KB | Database Log File | 3/8/2007 6:08 PM | A |
| msdbdata.mdf | 4,992 KB | Database Primary File | 3/8/2007 6:19 PM | A |
| msdblog.ldf | 2,048 KB | Database Log File | 3/8/2007 6:19 PM | A |
| tempdb.mdf | 2,240 KB | Database Primary File | 3/8/2007 5:25 PM | A |
| templog.ldf | 504 KB | Database Log File | 3/8/2007 6:19 PM | A |

*Figure 6-8*
*User instance database files*

The system databases are copied only the first time the user instance is created. When you use Visual Studio Express to create the user instance, your database file can be found in the bin\Debug folder in the location specified when you saved the project.

## Starting User Instances with SQL Server Express Utility

Once you've created a user instance, it will eventually stop running due to its built-in 60-minute time limit. It's not necessary to launch the application or use Visual Studio Express to start it again. Microsoft has released SQL Server Express Utility to perform a number of functions, including connecting to a user instance. It's perfect for our needs in this chapter. If you haven't done so already, open your browser, navigate to *go.microsoft.com/fwlink/?LinkId=44178*, and follow the instructions there to download and install the utility in a location that's easy to remember, such as C:\SSEUtil.

**TIP**

**Don't let its name fool you. SQL Server Express Utility isn't limited to use solely with SQL Server Express. If you later need to work with other editions of SQL Server, you'll find this utility quite handy for many tasks. Of course, you can still perform these tasks using Management Studio or other means, but it's always nice to have options. For example, without opening Management Studio, you can use this utility to get a list of all the databases attached to a particular SQL Server instance. You can even get a list of all the instances installed on a computer. Take note, however, that this utility is not officially supported by Microsoft and may not work with future versions of SQL Server Express. Meanwhile, use it to your advantage while you can!**

To use SQL Server Express Utility to start and connect to a user instance, follow these steps.

### TO START AND CONNECT TO A USER INSTANCE

1. Open the command prompt window and type **cd C:\SSEUtil**, replacing SSEUtil with the folder to which you extracted the SQL Server Express Utility (if you specified a different location).

2. To start the user instance, type the following statement and press **Enter**:

```
sseutil -l
```

The result of this command is shown in Figure 6-9.

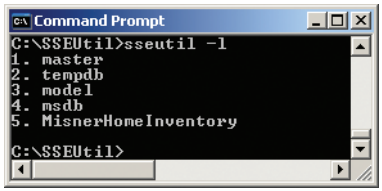



*Figure 6-9*
*Results of executing the command to start a user instance*

## Finding Information About User Instances

A dynamic management view is a special type of view that enables you to discover server information through queries. We're going to take advantage of this view to find the named pipe that SQL Server Express assigned to the user instance. To learn more about the named pipe assigned to your user instance, follow the steps below.

### TO DISCOVER NAMED PIPE DETAILS

1. Click **Start**, point to **All Programs**, point to **Microsoft SQL Server 2005**, and click **SQL Server Management Studio Express**. In the Connect To Server dialog box, click **Connect**.

2. Click **New Query** and then, in the Query Editor, type the following statement and click **Execute** (Figure 6-10):

```
select owning_principal_name, instance_pipe_name, heart_beat
from sys.dm_os_child_instances
```

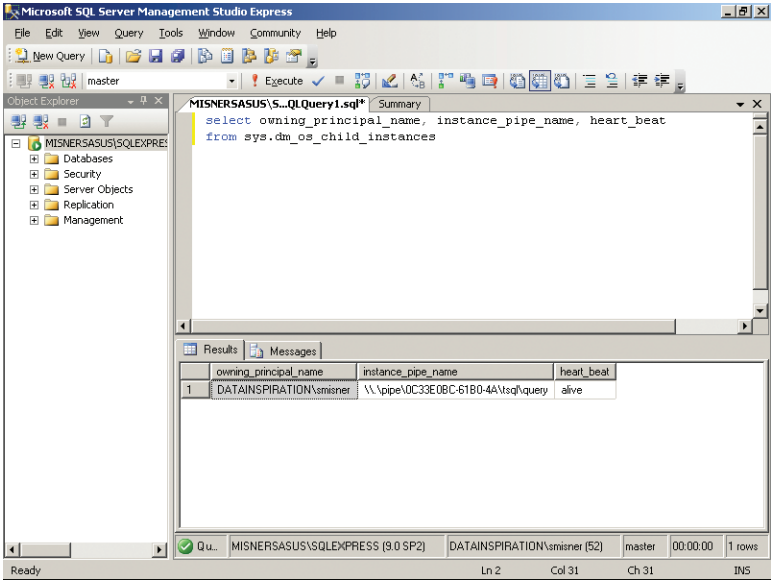



*Figure 6-10*
*Querying Sys.dm_os_child_instances*

There are actually quite a few columns in this dynamic management view, but the three columns we care about are described in the following list:

**owning_principal_name** The value in this column is a Windows account in the form *domain\user* or *computername\user*.

**instance_pipe_name** The value in this column is a named pipe that is random at the time the user instance is created, but it persists so that it can be reused in the future to connect to the database. The benefit of using named pipes is the complete insulation of the server running SQL Server, including protection from threats across the network.

**heart_beat** This column's value provides a current status of the user instance—dead or alive.

3 In the Query Editor's Results pane, right-click the value in the instance_pipe_name column and then click **Copy**. We'll save the pipe name in the clipboard to use in the next step.

## Connecting to a User Instance in Management Studio

If you're an application developer relying on SQL Server Express to store data for your application, you might already be familiar with user instances as a concept. You might even have tried to use Management Studio to connect to your database and been puzzled about how to do that. Because it's a separate instance, you cannot connect to your SQL Server Express instance as described throughout this book up to this point. A database can only be attached to one instance at a time, so your application database that's attached to the user instance can't be attached to the parent instance of SQL Server Express.

Even though most of your interaction with the database will be through your application, or even through Visual Studio Express, you do still have other options. You could use Management Studio, or a third-party query tool, to connect to your database; but, as you've discovered, the steps to perform the connection aren't particularly obvious. It's actually quite easy once you know how. Just follow these steps to connect to a user instance.

### TO CONNECT TO A USER INSTANCE

1. At this point (after completing the final step of the previous procedure), you should have the pipe name copied to the clipboard. If not, return to the previous procedure and complete all the steps before continuing on with this procedure.

2. On the **File** menu, click **Connect Object Explorer** and paste the clipboard contents into the Server box, as shown in Figure 6-11.

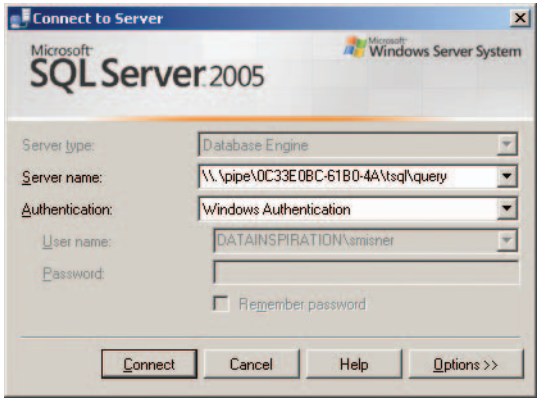



*Figure 6-11*
*Connecting to a user instance*

**TIP**

**If you have problems connecting to the user instance, use the dynamic management view described in the previous section to verify the user instance is alive. If the user instance is not alive, use SSEUtil.exe to start the user instance as described earlier in this chapter.**

3. Click **OK**. Notice the Object Explorer window (Figure 6-12) now displays both the SQL Server Express instance and the new user instance.

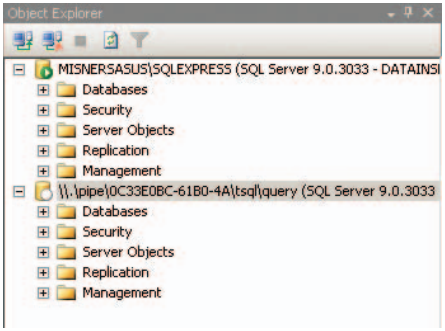



*Figure 6-12*
*Viewing multiple instances in Object Explorer*

4. Now let's investigate the database in this user instance. Expand **Databases**, expand the database (which in my case is MisnerHomeInventory), expand **Tables**, right-click **Location**, and click **Open Table**. The Location table displays, as shown in Figure 6-13.

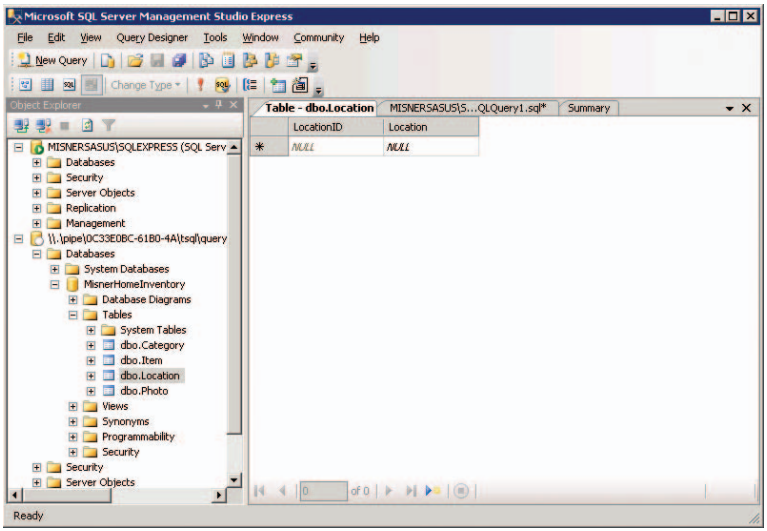



*Figure 6-13*
*Viewing the Location table from a user instance*

As you can see, there is no data currently in the database. You can either enter data here directly into the tables by using the Management Studio interface, or you can use Transact-SQL (T-SQL) to insert rows, as you learned in Chapter 4. The key here is that this database is yours and yours alone. Usually, you use the database application you build using Visual Studio Express to help manage inserts, updates, and deletes, but now you have another option for interacting with the application database.

## In Summary...

The purpose of this chapter is to give you a broad perspective of how SQL Server can be set up to accommodate multiple users, whether all access the same named instance or separate name instances, or whether each user should have his or her own copy of the database. It's quite possible you'll never need to use these features, but just in case a need arises in the future, you'll be better prepared to understand the consequences of implementing one approach instead of the other. I hope this fresh perspective also helps you better understand the relationship between SQL Server Express and your computer's resources. This chapter also showed you how to create a user instance and access that user instance by using Management Studio. Don't forget about SQL Server Express Utility, which enables you to perform many administrative tasks without launching Management Studio.

You've learned a lot in this chapter, not to mention throughout this entire book. Think back to how unfamiliar the tools, concepts, and terminology were when you started this book, and now you've got a working database and ideas for expanding on it to share with others. Be creative! A database is meant to evolve over time. As your skills grow, add finishing touches here and there to personalize it exactly to your needs. Your next step is to look around and think about how else you might organize the information around you into a brand new database that you build from the ground up all on your own. Have fun with that next project!

# Glossary

## A

**Application database** A database that stores application data in a central location for access by multiple users on a network.

**Attach** Associate a database with a specific SQL Server instance.

**Authentication** Confirmation that a principal is using a valid login with the correct password.

**Authorization** Validation of the tasks a principal can perform and the objects with which the principal can interact.

## B

**Bulk load** A fast way to import data from a file and insert it as new rows into a table.

## C

**CHECK constraint** A logical test of a data value before it's added to a new row or before it replaces a value in an existing row to make sure it's within the range of valid values.

**Child table** A term used to describe a table containing a foreign key column that references a primary key column in a parent table.

**Client** A user application that makes a request, often across a network, to a server application such as a database management system.

**Column** A definition of the type of information to store for all rows in a table, such as text, numeric, or date information, to name a few types.

**Composite index** An index based on multiple table columns.

**Composite key** A primary key created by using multiple columns to uniquely identify each row in a table.

**Connection string** A string of text that tells an application, such as Management Studio or some other application that connects to SQL Server, where to find the database you want to use, what credentials to use to authorize access, and any other bits of information describing how you want to connect to the server hosting that database.

## D

**Data manipulation language** A programming language used with relational databases to insert, update, delete, or retrieve data.

**Data service** A database management system that responds to requests from multiple users across a network simultaneously.

**Data type** A column property used to specify the type of data that can be stored in the column, such as numeric data or text characters, and how much storage space the column requires.

**Database** A container for information.

**Database management system** Software used to build, maintain, and query a database.

**DEFAULT definition** A valid value automatically entered into a column if no other value is provided when a new row is added.

**Denormalization** The process of combining columns from multiple tables to improve query performance.

**Detach** Remove a database from a SQL Server instance and keep database files intact.

**Distributed client database** A database on a client computer that synchronizes periodically with a central database so the user can have access to information when disconnected from the network.

## E

**Embedded database** A database integrated into an application intended for a single user, which stores its data in a database rather than on the file system.

**Entity** A generic term used to represent either an object or an occurrence as the subject of information in a table.

## F

**Foreign key** A data value stored in one table that is a primary key value in a lookup table.

**FOREIGN KEY constraint** A test to ensure a value in a foreign key column in a child table exists in a parent table.

**Foreign key relationship** A definition added to a child table to link a foreign key column to a column in the parent table.

## I

**Identifier** The name of any database or database object, such as a table, column, or stored procedure, to name a few.

**Identity column** A column for which you define a starting number and which is automatically incremented each time you add a new row to the table.

**Index key** A table column used to create an index.

## L

**Length** The amount of storage space to allocate to a column.

**Login** A combination of user name and password assigned to a principal.

**Lookup table** A table storing values referenced in another table in a normalized database for efficient storage and data integrity enforcement.

## M

**Metadata** Data about data, such as column names and data types defined within a table or view.

## N

**Named instance** A set of SQL Server components defined by a unique name running independently of and in parallel with other instances that may be installed on the same

computer. It has its own memory requirements and system databases.

**Normal form** A table structure conforming to a set of rules used to optimize performance and manageability.

**Normalization** The process of decomposing data into the smallest possible units and distributing data across multiple tables to improve data performance and minimize redundancy.

**NOT NULL constraint** A test to prevent a column from containing a NULL value.

## O

**Orphan row** A row in a child table that has no corresponding row in the parent table.

## P

**Parent instance** A SQL Server Express instance required to create a user instance.

**Parent table** Another term for a lookup table, a table whose primary key is used as a foreign key in a child table.

**Permissions** Access privileges assigned to a principal that determine whether the principal can only read data, read and change data, or modify the structure of database objects.

**Precision** The number of digits that can be used to the left and to the right of the decimal point in a noninteger number.

**Primary key** A table column used to uniquely identify each row in the table.

**PRIMARY KEY constraint** A test to ensure every row in the table is unique by preventing the addition of new rows containing a primary key value already present in the table or containing a NULL value.

**Principal** A person, a Windows service, or a group that includes people, services, or both, requesting access to a SQL Server database.

## Q

**Query** A request for information from a database.

**Query result set** One or more rows retrieved from a database in response to a SELECT statement.

## R

**Relational model** A data format using a set of related tables to structure data in rows and columns.

**Role** A group of principals that are assigned the same access privileges to an object.

**Row** Data values separated into multiple columns for a single unit of information contained in a table.

## S

**Scale** A limit on the number of digits to the right of the decimal point in a noninteger number.

**Script** One or more Transact-SQL (T-SQL) statements stored in a file.

**Service** A special type of program that runs on a computer without requiring a user to log in.

**Simple index** An index based on a single table column.

**Stored procedure** A collection of code built to perform certain functions and stored in the database to process when requested by a user or an application.

**Structured Query Language (SQL)** A language used to manipulate data and define database objects in relational databases.

**Subquery** A query that evaluates separately from the main query to, for example, select a single value from a table for use by the main query or select multiple rows that you use for comparison in the WHERE clause of the main query.

**Surrogate key** A derived primary key, often hidden from the user because it has no meaning, used to guarantee each row in the table has a unique primary key.

**Syntax** Rules defining the keywords in a programming language and the way these keywords are combined to form a valid statement.

## T

**Table** A two-dimensional structure using rows and columns to store information.

**Transaction log file** A record of database activity in case of database failure to reproduce transactions after restoring a database backup.

**Transact-SQL** The language used to create and modify a database and the objects it contains as well as to query the database or manage the server.

**Trigger** A special type of stored procedure processed when attempts to add new rows or to update or delete existing rows is detected.

## U

**Unicode** A data type used to support international characters in a database.

**UNIQUE constraint** A test on a column or group of columns to ensure each distinct value, including NULL, is used only once in all rows.

## V

**View** A virtual representation of a table that does not physically contain data.

# Index


## A

Add Objects dialog box, 160
Add User To The SQL Server Administrator Role, 22, 25
Administrative role, and Windows Vista, 20–23, 25
Administrative tasks
  attaching database, 175–179
  backing up database, 170–172
  restoring database, 173–174
  selecting recovery mode, 168–170
Advance Services, 16
AdventureWorks
  attaching database, 52–56
  download sample database for, 31–33
  *See also* specific tasks
AdventureWorksDB - InstallShield Wizard, 32
Aggregation
  defined, 108
  T-SQL, 108–114
  *See also* Filtering
Alias, 100
ALTER, 124
ALTER PROCEDURE statement, 135
ALTER PROCEDURE template, 134
ALTER statement, 140
ALTER TABLE statement, 128–129
American National Standards Institute (ANSI), 89
AND, 114
ANSI. *See* American National Standards Institute (ANSI)
Application database, 15
Approximate numerics data types, 63–64
Ascending sort, 105
Attach database, 52–56, 175–179
Attach Database dialog box, 53, 55, 178–179
Authentication, 146–148. *See also* Security
Authentication Mode, 24
Authorization, 146–148. *See also* Security
Autogrowth property, 59
AVG function, 108, 112

## B

Backing up, database, 170–172
Back Up Database dialog box, 171–172
BETWEEN clause, 116
Bigint data type, 63
Binary data type, 65
Binary settings data types, 65
Biometrics, 147
Bit data type, 63
Brackets, 92
Browse For Objects dialog box, 162
BUILTINAdministrator, 20
BUILTINUsers, 146
Bulk load, 60
Bulk-logged recovery, 60, 168
Bulk operations, 168
Buyens, Jim, 15

## C

Cascade rule, 72
Category Table, 127, 163
Change Autogrowth For HomeInventory dialog box, 59
Character strings data types, 64
Char data type, 64
CHECK constraint, 11, 13, 73–75, 87
Check Constraint Expression dialog box, 74
Check Constraints dialog box, 73–75
Check Existing Data On Creation Or Re-Enabling property, 72, 75
Choose Name dialog box, 69
Client, 2–3
CLR. *See* Common Language Runtime (CLR) objects
Clustered index, 79
Column name, 102
Column Permissions, 163
Column Properties pane, 68
Columns, defined, 4–5
Columns folder, 49
Common Language Runtime (CLR) objects, 17
Community Technology Preview, 21
Completing Microsoft SQL Server 2005 Setup, 27
Completing setup, 30
Completing The Microsoft SQL Server 2005 Books Online Setup, 37
Composite index, 78
Composite key, 6
Computer Management Console149
Configuration Options, 22, 25
Confirm Password box, 150
Connection string, 188
Connect To Database dialog box, 154
Connect To Database Engine dialog box, 123, 140
Connect To Server dialog box, 44–45, 165, 188
Constraints, add, 73–77. *See also entries for individual constraints*

COUNT function, 108
COUNT (*) function, 112–113
COUNT(ITEMID) function, 112
CREATE DATABASE, 140–141
CREATE PROCEDURE template data, 132
CREATE statement, 124–127, 140
CREATE TABLE statement, 126–128, 140
CREATE VIEW statement, 129–131
Current Position, 56

## D

Database
  add constraints, 73–77
  add relationships, 70–72
  add tables to, 62–69
  attaching, 175–179
  backing up, 170–172
  create, 58
  defined, 2
  restoring, 173–174
  structure of, 4–13
  *See also* Table
Database Engine Query, 46
Database file, 59
Database folders, 48–49
Database management system
  defined, 2–4
  security of, 146–167
Database management system software, 2
Database Name box, 58
Database objects, 124–142
Database Properties dialog box, 169–170
Database security. *See* Security
Database Tuning Advisor, 16
Database User dialog box, 159–160
Data definition language (DDL), 124
Data integrity, 10–13
Data manipulation language, 90
Data redundancy, 5–7
Data service, 2–3
Data Space Specification property, 77
Data type, 62
  defined, 5
  length of, 57
  table for, 63–65
  *See also* individual data types
Data type drop-down list, 66
DATEADD function, 114
Date and time data type, 64
DATEDIFF function, 114
DATEPART function, 114
Datetime column, 113
Datetime data type, 64
DDL. *See* Data definition language (DDL)
Decimal or numeric data type, 63
DEFAULT definition, 11, 13
Default instance, 186
DELETE statement, 90, 121–123
Denormalization, 10
DESC keyword, 105
Description property, 72, 75, 77
Description Property dialog box, 77
Destination Folders page, 32
Detach database, 52, 175–179
Detach dialog box, 175–176
Differential backup, 171
DINSTINCT keyword, 112–113
Distributed client database, 15
Documents and Settings folder, 188
Document window, of SQL Server Management Studio Express, 45–46
Domain integrity, 10–12
Download, copy of SQL Server Books Online, 34–39
Download Books Online From The Download Center, 34
Download sample database, 31–33
DROP, 124

## E

Embedded database, 15
Empty string, 12
Enforced For Replication property, 72
Enforce Foreign Key Constraint property, 72
Enforce For INSERTS and UPDATES property, 75
Enforce For Replication property, 75
Entity, defined, 4
Entity integrity, 12
Error And Usage Report Settings page, 25
Error message, 85–86
Exact numerics data types, 63
Executable file, 182
EXECUTE command, 98
Execute Procedures dialog box, 166
EXE file extension, 185
Expand Data Space Specification property, 77

## F

Feature Selection page, 24, 29, 36
Filegroup, 61
Filegroup Or Partition Scheme Name property, 77
Fill Specification property, 77
Filtering, 115–118
Float data type, 63
FOREIGN KEY constraint, 11, 13, 86, 122
Foreign key relationships, 12–13, 70–72, 74, 84
Foreign Key Relationships dialog box, 70–72
Full Control permission, 177
Full join, 100
Full outer join, 101
Full recovery mode, 168
Full recovery model, 60

## G

General SQL Server Scripts Wizard, 137–140
Generate Change Script, 67
GO command, 141
GROUP BY clause, 108, 111, 114

## H

HAVING clause, 118
Heart_beat, 195

## I

Identifier, 58
Identity column, 7
Identity Specification property, 68
Idf extension, 59
Image data type, 66
Image files, storing, 9–10, 97. *See also* Photo table
Index Columns dialog box, 76
Indexes/Keys dialog box, 75, 77
Index keys, 78
Index Name box, 79
Initial Size MB property, 59
Inner join, 100, 104
INSERT, 90–97
INSERT and UPDATE Specification property, 72
INSERT statement
  for all location rows, 93
  copy and paste, 93
  create, to add rows to Photo table, 97
  method of naming table, 92
  successful query execution of, 93
  viewing rows in Location table, 94–95
INSERT template, 91
Installing Prerequisites page, 22
InstallShield Wizard Completed page, 33
Instance
  default, 186
  how to start, without using Visual Studio Express, 192–193
  named, 185–186
  parent, 188–189
  server, 182–185
  unnamed, 186
  user, 186–197
  viewing multiple, in Object Explorer, 197
Instance_pipe_name, 195
Int data type, 57, 63
Integers, 66
Integrity, data, 10–13
International characters, 65
ItemID column, 84
Item table, viewing rows in, 95

## J

JOIN clause, 99–104

## L

Left outer join, 104
License agreement
  for AdventureWorks sample database, 32
  for Microsoft SQL Server 2005 Books Online, 35
  for Microsoft SQL Server 2005 Express Edition, 21
  for SQL Server Management Studio Express Edition, 28
Local copy of SQL Server Books Online, install, 34–39
Local database, 3
Local System account, 182–185
Locate Database Files dialog box, 54, 178
LocationID, defined, 85
LocationID column, 84–86
Location table
  foreign key relationship between item and, 84
  viewing, from user instances, 197
  viewing rows in, 94–95
Login
  create, 148–154
  defined, 148
Login dialog box, 151, 156–158
Login Name dialog box, 152
Log On As column, 183
Lookup table, 8

## M

Manage Check Constraints, 67
Manage Fulltext Index, 67
Manage Indexes and Keys, 67
Management Studio
  connecting to user instances in, 196–197
  trouble connecting to SQL Server Express using, 183
  use INSERT statement, 90–97
Manage XML Indexes, 67
Master database, 47–48
MAX function, 108
Mdf extension, 59
Memory, server, 182–185
Menu bar, of SQL Server Management Studio Express, 45
Metadata, 49
Microsoft, royalties to, 14
Microsoft SQL Server 2005 Books Online Setup Wizard, 35
Microsoft SQL Server Desktop Engine 2000 (MSDE), 14
Microsoft SQL Server 2005 Express Edition
  administrative role and Windows Vista, 20–23, 25
  install, 20–27
  key features, 15–17
  kinds of database built in, 14–15
  online resources, 18
Microsoft SQL Server 2005 Express Edition Books Online, 18
Microsoft SQL Server TechCenter, 18
Microsoft Update, 20
Microsoft Visual Basic 2005 Express Edition, 15
*Microsoft Visual Basic 2005 Express Edition: Build a Program Now!*, 15
Microsoft Visual Basic .NET, 90
Microsoft Visual C# .NET, 90
Microsoft Visual Studio, 186. *See also* User instance
Microsoft Visual Web Developer 2005 Express Edition, 15
*Microsoft Visual Web Developer 2005 Express Edition: Build a Web Site Now!*, 15
Microsoft Windows Installer 3.1, 20
MIN function, 108
Minimal redundancy, 4
Mixed Mode Authentication, 148
Model database, 47–48

Money data type, 63, 66
Move First, 56
Move Last, 56
Move Next, 56
Move Previous, 56
Move to New Row, 56
Msdb database, 47
MSDE. *See* Microsoft SQL Server Desktop Engine 2000 (MSDE)
MSDN Forums for SQL Express, 18
Multiple joins, 103–104

## N

Name
  of CHECK constraint, 74
  column, 102
  convention for indexes, 79
  database, 58
Named instance, 24, 185–186, 188
Named pipe details, 194–195
Nchar data type, 64
Ndf file extension, 61
.NET 2.0 Framework, 20
Network Service account, 182–183
New Database dialog box, 58
New Index dialog box
  add index to table in, 78–83
  choose index type in, 79
  with column added, 80
  Included Columns page of, 82
  Options page, 81
  Storage page of, 83
New Query, 46
New Query Editor window, 50–52
New User dialog box, 149–150
Nonclustered index, 79, 82
Non-null value, 108
Non-numeric data type, 62
Normal form, 6
Normalization, 5–10
Notification Services, 16
NOT NULL, 12–13, 73, 127
Ntext data type, 66
NULL value, 12–13, 73, 108, 112
Numeric data type, 62–63
Nvarchar or ncarchar(max) data type, 64

## O

Object Explorer, 47–49
  as appears to authenticated user with no authorization, 154
  with database show, 179
  with detached database no longer visible in window, 176
  item table's columns displayed, 69
  of SQL Server Management Studio Express, 45–46
  updated, 61
  viewing multiple instances in, 197
Object-oriented language, 90
Objects. *See* Database objects
Open File, 46
Operators, 74, 101
OR, 114
ORDER BY clause, 105–107, 111, 114
Orphan row, 11
Outer join, 100–101, 104
Owning_principal_name, 195

## P

Parameter, 134–136, 166
Parent-child relationship, 8
Parent instance, 188–189
Password, 147
Password box, 150
Pattern matching, filtering with, 117
Pelland, Patrice, 15
Permissions
  defined, 155
  granting, 158–163
  testing, 164–167
  *See also* Roles; individual types of permissions
Photos. *See* Image files
Photo table, create INSERT statement to add rows to, 97
Precision, 65
PRIMARY filegroup, 61
Primary key, 6, 8, 11, 68
PRIMARY KEY constraint, 12–13, 73
Primary XML index, 79
Principals, 146
Profile directory, 188. *See also* Documents and Settings folder
Programming languages, 90. *See also* individual languages
Properties dialog box, 183
Properties Window, 46
Protocols, for SQL Server (SQLEXPRESS) service, 41

## Q

Query
  defined, 3
  limiting rows retrieved by, 114–118
  *See also* New Query Editor window; Subquery
Query Editor's Results pane, 195
Query Result Pane navigation bar, 56
Query result set, 98, 105–107

## R

Ready To Install page, 26
Ready To Install The Program page, 30, 33, 37
Real data type, 64
RECONFIGUR statement, 187
Recovery mode, 168–170
Recovery Model selection, 60
Redundancy, 5, 7
Referential integrity, 12–13, 101
Registered Servers, 46
Registration Information page, 23, 29, 36
Relational model, defined, 4–5
Relationships, 67
Remote access, configure SQL Server 2005 Express for, 40–42
Replicated database, 3
Replication, 24
Reporting Services, 16
Restore Database dialog box, 173–174
Restoring database, 173–174
Restricted File Growth (MB) option, 59

Reverse sort order, 105–106
Roles, assigning, 155–157
Row(s)
  add new, to table, 90–97
  create INSERT statement to add, 97
  defined, 3
  delete, from table, 86
  delete from table, 121–123
  filtering, 115–116
  limiting, retrieved by query, 114–118
  use join to filter, 121
  viewing, in Location table, 94–95
  viewing in Item table, 95
Run As dialog box, 164

## S

Save All, 46
Save Selected Items, 46
Scale, 65
Scripts, 136–142
Script Wizard Summary, 139–140
Securables page, of Database User dialog box, 160, 162–163
"Secure by default," 17
Security
  assigning roles, 155–157
  creating logins, 148–154
  granting permissions, 158–163
  purposes of, 146–148
  required in service, 182
  testing permissions, 164–167
Select Columns From 'Dbo.item' dialog box, 80, 82
Select Object Types dialog box, 161
SELECT statement, 90, 98–99
  adding ORDER BY clause to, 105–107
  add WHERE clause to, 114–118
  executing, on view for which login is not authorized, 167
  including aggregate function, 108–109
  multiple aggregate functions in, 112
  rearranging columns in, 103
  reversing sort order in, 106
  using, with view, 131
  using column list in, 102
  using multiple sorts in, 107
  viewing results of multiple, 99
  YEAR function in, 114
  *See also* Views
Select User Or Group dialog box, 151–152
Self-join, 100
Serial number, 6
Server instances, 182–185
Server memory, 182–185
Server Roles page, of Login dialog box, 156
Service
  availability of, 182
  memory consumption, 182–185
  starting, 182
  *See also* individual services
Set Default rule, 72
Set Null rule, 72
Set Primary Key, 67
Setup Progress page, 26
Simple index, 78
Simple recovery mode, 69
Smalldatetime column, 113
Smalldatetime data type, 64, 66
Smallint data type, 63
Smallmoney data type, 63, 66
Smartcard system, 147
"Soft delete," 123
Sorting, query result set, 105–107
SQL. *See* Structured query language (SQL)
SQL Administrator Role, 20–23, 25
SQLCMD utility, 27, 137, 141–142
SQL Editor toolbar, 50–52
SQLEXPRESS. *See* SQL Server (SQLEXPRESS) service
SQL Server authentication, 148
SQL Server Books Online
  apply SQL Server Express filter to, 38–39
  install local copy of, 34–39
SQL Server Browser, 42
SQL Server Configuration Manager, 16, 40–41, 183
SQL Server 2005 Express Edition
  configure, for remote access, 40–42
  installing, 185
  trouble connecting to, using Management Studio, 183
SQL Server 2005 Express Edition Developer Center, 18
SQL Server Express filter, 38–39
SQL Server Express for replication, 24
SQL Server Express Service, 182–185
SQL Server Express Utility, 189–190, 193–194
SQL Server Express Weblog, 18
SQL Server Management Studio Express
  Connect To Server dialog box, 44–45
  main window, 45
  standard toolbar, 45–46
SQL Server Management Studio Express Edition, 16, 27–30
SQL Server 2005 Service Pack 2 Express Edition, 21
SQL Server Setup, 22–23
SQL Server (SQLEXPRESS) service, 24, 41
SQL Server Surface Area Configuration, 16, 23
Sql_variant data type, 65
SSEUtil Interactive Command Window, 190–191
Standard toolbar, of SQL Server Management Studio Express, 45–46
Stop Retrieving Data, 56
Stored procedure, 13, 131–136
Structured query language (SQL), 89
Subquery, 120–121
SUM function, 108–111
Summary, 46, 48
Summary Log, 27
Surface Area Configuration Tool, 27
Surrogate key, 7
Syntax, 90

SysAdmin role, 22
SysAdmin server role, 20
Sys.database columns, 49
System configuration check, 23
System database, 3, 47
System-defined data type, 62
System table, 52
System Views folder, 48–49

## T

Table
  add, to database, 62–69
  add constraints to columns, 73–77
  add data to, 84–87
  add foreign key relationship to, 70–72
  add new row to, 90–97
  changing data in, 119–121
  combining contents of multiple, 99–104
  create INSERT statement to add rows to, 97
  defined, 4
  delete row from, 86
  deleting row from, 121–123
  index, 78–83
  join views with, 130
  method of naming, 92
  viewing contents of, 98–99
  *See also* Database; individual tables; individual table types
Table Designer toolbar, 67
Table Designer window, 68
Table name, 92, 102
Tables and Columns dialog box, 71
Tables and Columns Specifications box, 71
Table size, total, 62
Table window, 56, 84
Tempdb database, 47
Template Data directory, 188–189
Template Explorer, 46
Text data type, 66
Third normal form, 6
Time data type, 64
Timestamp data type, 65
Tinyint data type, 63
Total Rows, 56
Transaction log file, 59
Transact-SQL language, 16
Trigger, 13
TRUNCATE statement, 124
TRUNCATE TABLE item, 123
T-SQL Aggregate Functions, 108–114
T-SQL statement
  add new rows to table, 90–97
  capitalizing words in, 90
  *See also* Data definition language (DDL); individual statements

## U

UNICODE, 65, 77
Unicode character string data types, 64
Uniq_Location, 77
UNIQUE constraint, 12–13, 73, 76–77
Uniqueidentifier data type, 65
Unnamed instance, 186
UPDATE statement, 90, 119–121
User account, 182–183
User Account Control, 21
User-defined data type, 62
User-defined integrity, 13
User instance
  connecting to, in Management Studio, 196–197
  create, manually, 190–193
  creating, 188–193
  defined, 186
  disable, 187
  enabling, 187
  finding information about, 194–195
  how, differs from named instance, 188
  starting, with SQL Server Express Utility, 193–194
  viewing Location table from, 197
User instance database files, 193
User Mapping page, of Login dialog box, 157–158
User profile directory, 188

## V

Varbinary or varbinary(max) data type, 65
Varchar data type, 57, 66, 68
Varchar or varchar(max) data type, 64
Variable-length data type, 68
Views, 129–131
Views folder, 48
Visual Studio 2005, 17

## W

Welcome page, of SQL Server Setup, 22–23
WHERE clause, 114–118, 121
Wildcard character, for pattern matching, 117
Windows authentication, 147–148, 189
Windows Firewall dialog box, 42
Windows service, 3
Windows Task Manager dialog box, 184–185
Windows Vista, 25, 188
Windows XP Home, 150
Windows XP Professional, 150

## X

Xml data type, 65
XML format, 17
X symbol, 23

## Y

YEAR function, 113–114