msdn® training

msdn®

Programming with ADO.NET (Prerelease)

Delivery Guide

Course Number: 2389A

Part Number: X08-58142
Released: 10/2001

Microsoft®

Course Number: 2389A
Part Number: X08-58142
Released: 10/2001

Contents

About This Course

This section provides you with a brief description of the course, audience, suggested prerequisites, and course objectives.

Description

Audience

Student Prerequisites

This course requires that students meet the following prerequisites:

-
-
-

Course Objectives

After completing this course, the student will be able to:

-
-
-

Course Timing

The following schedule is an estimate of the course timing. Your timing may vary.

Day 1

Start	End	Module
9:00	9:30	Introduction
		Module x: Title
		Break
		Lab x: Title
		Lunch
		Lab x: Title *(continued)*
		Break
	4:00	

Day 2

Start	End	Module
9:00	9:30	Day 1 review
		Break
		Lunch
		Break
	4:00	

Day 3

Start	End	Module
9:00	9:30	Day 2 review
		Break
		Lunch
		Break
	4:00	

Day 4

Start	End	Module
9:00	9:30	Day 3 review
		Break
		Lunch
		Break
	4:00	

Day 5

Start	End	Module
9:00	9:30	Day 4 review
		Break
		Lunch
		Break
	4:00	

Trainer Materials Compact Disc Contents

The Trainer Materials compact disc contains the following files and folders:

- *Autorun.exe.* When the compact disc is inserted into the compact disc drive, or when you double-click the **Autorun.exe** file, this file opens the compact disc and allows you to browse the Student Materials or Trainer Materials compact disc.
- *Autorun.inf.* When the compact disc is inserted into the compact disc drive, this file opens Autorun.exe.
- *Default.htm*. This file opens the Trainer Materials Web page.
- *Readme.txt*. This file explains how to install the software for viewing the Trainer Materials compact disc and its contents and how to open the Trainer Materials Web page.
- *2389a_ms.doc*. This file is the Manual Classroom Setup Guide. It contains the steps for manually installing the classroom computers.
- *2389a_sg.doc*. This file is the Automated Classroom Setup Guide. It contains a description of classroom requirements, classroom configuration, instructions for using the automated classroom setup scripts, and the Classroom Setup Checklist.
- *Errorlog.* This folder contains an error log.
- *Powerpnt*. This folder contains the PowerPoint slides that are used in this course.
- *Pptview*. This folder contains the PowerPoint Viewer, which is used to display the PowerPoint slides.
- *Setup*. This folder contains the files that install the course and related software to computers in a classroom setting.
- *StudentCD*. This folder contains the Web page that provides students with links to resources pertaining to this course, including additional reading, review and lab answers, lab files, multimedia presentations, and course-related Web sites.
- *Tools*. This folder contains files and utilities used to complete the setup of the instructor computer.
- *Webfiles*. This folder contains the files that are required to view the course Web page. To open the Web page, open Windows Explorer, and in the root directory of the compact disc, double-click **Default.htm** or **Autorun.exe**.

Student Materials Compact Disc Contents

The Student Materials compact disc contains the following files and folders:

- *Autorun.exe.* When the compact disc is inserted into the CD-ROM drive, or when you double-click the **Autorun.exe** file, this file opens the compact disc and allows you to browse the Student Materials compact disc.
- *Autorun.inf.* When the compact disc is inserted into the compact disc drive, this file opens Autorun.exe.
- *Default.htm*. This file opens the Student Materials Web page. It provides students with resources pertaining to this course, including additional reading, review and lab answers, lab files, multimedia presentations, and course-related Web sites.
- *Readme.txt*. This file explains how to install the software for viewing the Student Materials compact disc and its contents and how to open the Student Materials Web page.
- *2389a_ms.doc*. This file is the Manual Classroom Setup Guide. It contains a description of classroom requirements, classroom setup instructions, and the classroom configuration.
- *Addread*. This folder contains additional reading pertaining to this course.
- *Appendix*. This folder contains appendix files for this course.
- *Flash.* This folder contains the installer for the Macromedia Flash 5.0 browser plug-in.
- *Fonts*. This folder contains fonts that are required to view the Microsoft PowerPoint® presentation and Web-based materials.
- *Jobaids.* This folder contains the job aids pertaining to this course.
- *Labfiles*. This folder contains files that are used in the hands-on labs. These files may be used to prepare the student computers for the hands-on labs.
- *Media*. This folder contains files that are used in multimedia presentations for this course.
- *Mplayer*. This folder contains the setup file to install Microsoft Windows Media™ Player.
- *Sampapps*. This folder contains the sample applications associated with this course.
- *Sampcode*. This folder contains sample code that is accessible through the Web pages on the Student Materials compact disc.
- *Webfiles*. This folder contains the files that are required to view the course Web page. To open the Web page, open Windows Explorer, and in the root directory of the compact disc, double-click **Default.htm** or **Autorun.exe**.
- *Wordview*. This folder contains the Word Viewer that is used to view any Word document (.doc) files that are included on the compact disc.

Document Conventions

The following conventions are used in course materials to distinguish elements of the text.

Convention	Use
◆	Indicates an introductory page. This symbol appears next to a topic heading when additional information on the topic is covered on the page or pages that follow it.
bold	Represents commands, command options, and syntax that must be typed exactly as shown. It also indicates commands on menus and buttons, dialog box titles and options, and icon and menu names.
italic	In syntax statements or descriptive text, indicates argument names or placeholders for variable information. Italic is also used for introducing new terms, for book titles, and for emphasis in the text.
Title Capitals	Indicate domain names, user names, computer names, directory names, and folder and file names, except when specifically referring to case-sensitive names. Unless otherwise indicated, you can use lowercase letters when you type a directory name or file name in a dialog box or at a command prompt.
ALL CAPITALS	Indicate the names of keys, key sequences, and key combinations—for example, ALT+SPACEBAR.
`monospace`	Represents code samples or examples of screen text.
[ ]	In syntax statements, enclose optional items. For example, [*filename*] in command syntax indicates that you can choose to type a file name with the command. Type only the information within the brackets, not the brackets themselves.
{ }	In syntax statements, enclose required items. Type only the information within the braces, not the braces themselves.
|	In syntax statements, separates an either/or choice.
►	Indicates a procedure with sequential steps.
...	In syntax statements, specifies that the preceding item may be repeated.
. . .	Represents an omitted portion of a code sample.

msdn training

Introduction (Prerelease)

Contents

Microsoft

Instructor Notes

Presentation:
30 Minutes

The Introduction module provides students with an overview of the course content, materials, and logistics for this course.

Required Materials

To teach this course, you need the following materials:

- Delivery Guide
- Trainer Materials compact disc

Preparation Tasks

To prepare for this course, you must:

- Complete the Course Preparation Checklist that is included with the trainer course materials.

How to Teach This Module

This section contains information that will help you to teach this module.

Introduction

Welcome students to the course and introduce yourself. Provide a brief overview of your technical background and your interest in the subject of the course to establish credibility and rapport.

Ask students to introduce themselves and to describe their background, product experience, and expectations of the course.

Record the expectations of students on a whiteboard or flip chart so that you can refer to them later.

Course Materials

Tell students that everything they will need for this course is provided at their desk.

Tell student to write their names on both sides of the name card.

Describe the contents of the student workbook and the Student Materials compact disc.

Tell students where they can send comments and feedback on this course.

To demonstrate how to open the Web page for the course open **Default.htm** located on the Trainer Materials compact disc in the StudentCD folder.

Prerequisites

Describe prerequisites for this course. Try to determine whether your students either exceed or do not meet the prerequisites.

Course Outline

Briefly describe each module and what students will learn.

For each chapter relate the terminal objective to student expectations for the course.

Setup

Describe the classroom setup. Cover the following aspects of the classroom setup:

- Go over the location of course files. Prompt students to create a desktop shortcut to their course files. If you performed a default installation of the course files, they will be installed in C:\Program Files\Msdntrain\2500.
- Describe the lab file structure in terms of start files and solution files
- Tell students whether they have Internet access.
- Describe the computer naming convention used in the classroom
- Describe whatever passwords are in place for the operating system and for SQL Server.
- Read the required software list to students. Distinguish between software titles that are available through retail channels versus those freely downloadable from the Web. Make sure that students understand what is needed to reproduce the classroom environment.

Microsoft Certified Professional Program

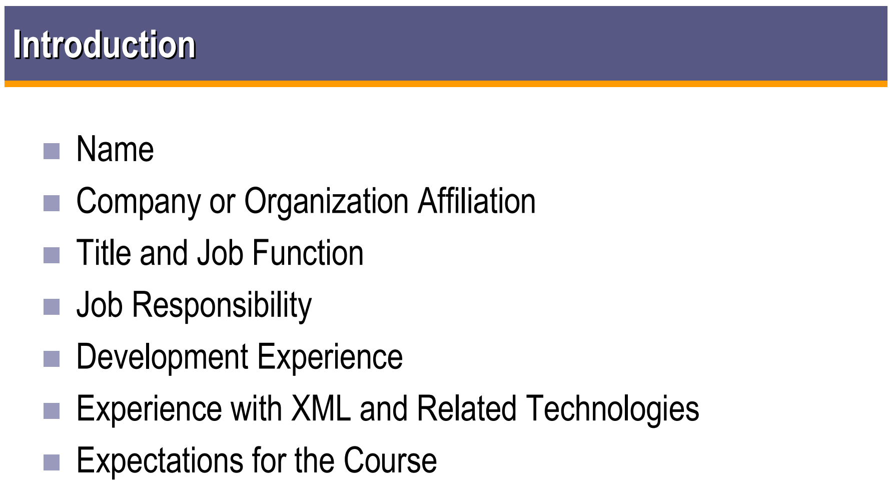
Briefly describe the Microsoft Certified Professional (MCP) program. Contrast the MCP against other Microsoft certification options.

Facilities

Explain the following:

- Start and end time for regular classroom activities
- Any extended hours for working on labs
- Estimated time and duration of meals and stretch breaks
- Restroom location
- Parking facilities and policies
- Location and availability of telephones and Internet messaging facilities
- Local policies regarding smoking
- Availability of recycling facilities

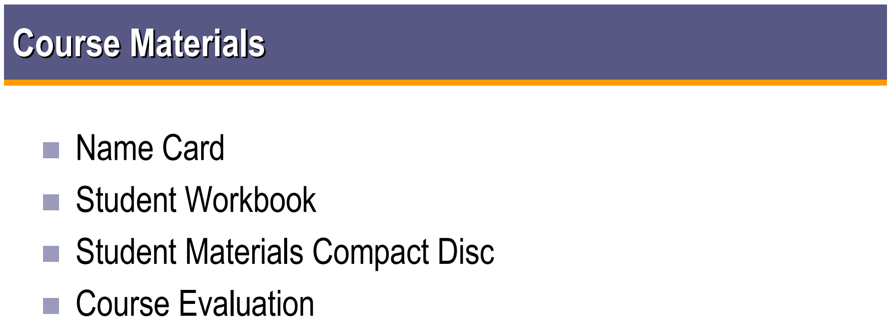
Introduction



Introduction

- Name
- Company or Organization Affiliation
- Title and Job Function
- Job Responsibility
- Development Experience
- Experience with XML and Related Technologies
- Expectations for the Course

Course Materials



Course Materials

- Name Card
- Student Workbook
- Student Materials Compact Disc
- Course Evaluation

The following materials are included with your kit:

- *Name card.* Write your name on both sides of the name card.
- *Student workbook.* The student workbook contains the material covered in class, in addition to the hands-on lab exercises.
- *Student Materials compact disc.* The Student Materials compact disc contains the Web page that provides you with links to resources pertaining to this course, including additional readings, review and lab answers, lab files, multimedia presentations, and course-related Web sites.

  Note To open the Web page, insert the Student Materials compact disc into the CD-ROM drive, and then in the root directory of the compact disc, double-click **Autorun.exe** or **Default.htm**.

- *Course evaluation.* To provide feedback on the instructor, course design or materials, or software product, send e-mail to mstrain@microsoft.com. Be sure to type **Course 2389A** in the subject line. Your comments will help us improve future courses.

  To provide additional comments or inquire about the Microsoft Certified Professional program, send e-mail to mcp@msprograms.com.

Prerequisites

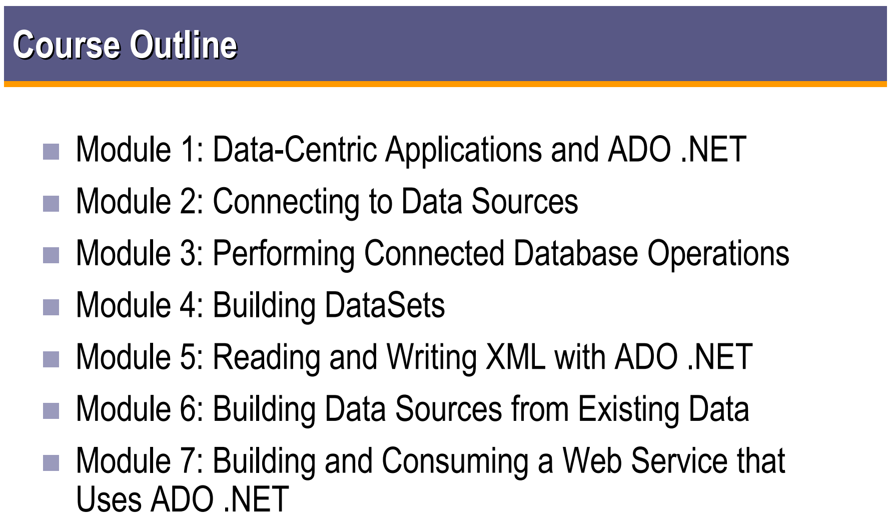
Prerequisites

- Database Basic Concepts
- XML Concepts and Implementation
- Visual Basic
- Distributed Application Architecture
- User Interface Design

This course requires that you meet the following prerequisites:

- Understanding of Database Basics: Table, Row, Column, Primary Keys, Foreign Keys, Constraints, Views)
- Concepts including SELECT, INS, UPD, DEL from tables.
- Exposure to XML documents, stylesheets, and schema
- Visual Basic.NET, Visual Basic for Applications, or previous version of Visual Basic
- Describe distributed application architecture
- Building UI – Web applications or Windows applications

Course Outline



Course Outline

- Module 1: Data-Centric Applications and ADO .NET
- Module 2: Connecting to Data Sources
- Module 3: Performing Connected Database Operations
- Module 4: Building DataSets
- Module 5: Reading and Writing XML with ADO .NET
- Module 6: Building Data Sources from Existing Data
- Module 7: Building and Consuming a Web Service that Uses ADO .NET

Module 1: Data-Centric Applications and ADO .NET - After completing this module, you will be able to diagram the architecture of data-centric applications, give examples of storage options, choose a connected or disconnected environment based on application requirements, diagram the ADO .NET object model, use the System.Data namespaces in applications, analyze typical business scenarios, and describe the use of XML in ADO .NET.

Module 2: Connecting to Data Sources - After completing this module, you will be able to choose a .NET data provider, connect to SQL Server, connect to OLE DB data sources. manage a connection, handle common connection exceptions, and implement and control connection pooling.

Module 3: Performing Connected Database Operations - After completing this module, you will be able to build a command object, execute a command that returns a single value, execute a command that returns a set of rows, and process the result, execute a command that returns multiple results, and process the results, execute a command that defines data by using the data definition language (DDL), execute a command that modifies data by using the data manipulation language (DML), and use transactions.

Module 4: Building DataSets - After completing this module, you will be able to build a **DataSet** and a **DataTable**, bind a **DataSet** to a **DataGrid**, create a custom **DataSet** by using inheritance, define a data relationship, modify data in a **DataTable**, find and select rows in a **DataTable**, and sort and filter a **DataTable** by using a **DataView**.

Module 5: Reading and Writing XML with ADO .NET - After completing this module, you will be able to generate an XSD Schema from a **DataSet** by using graphical tools, identify the purpose and uses of the **XmlDataDocument** object, save a **DataSet** structure to an XSD Schema file, create and populate a **DataSet** from an XSD Schema and XML data, load data and schema simultaneously into a **DataSet, s**ave **DataSet** data as XML, write and load changes by using a **DiffGram,** and manipulate data in an **XmlDataDocument** object.

Module 6: Building DataSets from Existing Data Sources - After completing this module, you will be able to configure a DataAdapter to retrieve information, populate a DataSet by Using a DataAdapter, configure a DataAdapter to modify information, persist data changes to a server, and manage data conflicts

Module 7: Building and Consuming a Web Service That Uses ADO .NET - After completing this module, you will be able to build and consume a Web service, and troubleshoot errors in an ADO .NET application.

Microsoft Certified Professional Program

Microsoft Certified Professional Program

- Microsoft Certified Systems Engineer (MCSE)
- Microsoft Certified Database Administrator (MCDBA)
- Microsoft Certified Solution Developer (MCSD)
- Microsoft Certified Professional + Site Building (MCP + Site Building)
- Microsoft Certified Professional (MCP)
- Microsoft Certified Trainer (MCT)

The Microsoft Certified Professional program includes the following certifications:

- Microsoft Certified Systems Engineer (MCSE)
- Microsoft Certified Database Administrator (MCDBA)
- Microsoft Certified Solution Developer (MCSD)
- Microsoft Certified Professional + Site Building (MCP + Site Building)
- Microsoft Certified Professional (MCP)
- Microsoft Certified Trainer (MCT)

For More Information See the Microsoft Training and Certification Web site at http://www.microsoft.com/trainingandservices/

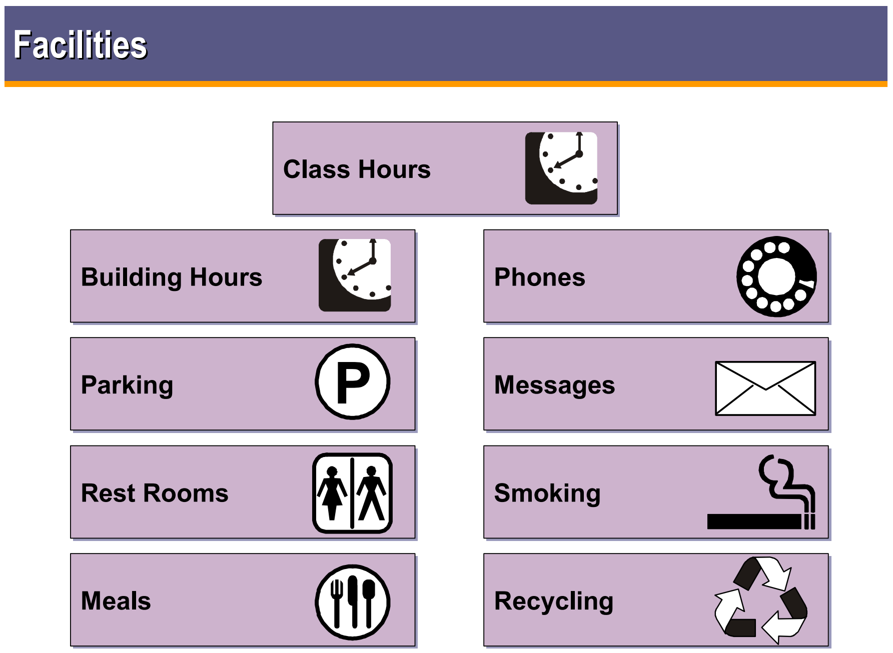
You can also send e-mail to mcp@msprograms.com if you have specific certification questions.

Preparing for an MCP Exam

MSDN Training curriculum helps you prepare for Microsoft Certified Professional (MCP) exams. However, no one-to-one correlation exists between MSDN Training courses and MCP exams. Microsoft does not expect or intend for MSDN Training to be the sole preparation tool for passing an MCP exam. Practical product knowledge and experience is also necessary to pass an MCP exam.

To help prepare for the MCP exams, you can use the preparation guides that are available for each exam. Each Exam Preparation Guide contains exam-specific information, such as a list of the topics on which you will be tested. These guides are available on the Microsoft Certified Professional Web site at http://www.microsoft.com/trainingandservices/

Facilities



Facilities

Class Hours

Building Hours

Phones

Parking

Messages

Rest Rooms

Smoking

Meals

Recycling

msdn® training

Module 1: Data-Centric Applications and ADO .NET (Prerelease)

Contents

Microsoft®

Instructor Notes

Presentation
45 Minutes
Lab:
30 Minutes

This course contains code samples in two languages, Microsoft Visual Basic and Microsoft Visual C#. This is to accommodate students who are currently Visual Basic programmers and who might be considering working with C#, and C# programmers who do not have Visual Basic experience. All of the Lab exercises have solutions in both languages. Most examples are also in two languages, except when the example differs in two languages in only the most minor ways.

You access the examples by clicking on the example links at the bottom of the PowerPoint slides for the topics containing the examples. The examples are displayed in Internet Explorer. The examples for each module are contained in a single .htm file for that module. The files have internal links for navigating within the example files.

This module teaches students about the architecture of the .NET Framework and of ADO .NET. In addition, this module teaches students the differences between connected and disconnected environments.

After completing this module, students will be able to:

- Diagram the architecture of data-centric applications.
- Give examples of storage options.
- Choose a connected or disconnected environment based on application requirements.
- Diagram the ADO .NET object model.
- Use the System.Data namespaces in applications.
- Analyze typical business scenarios.
- Describe the use of XML in ADO .NET.

Required materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2389A_01.ppt
- Module 1, Data-centric Applications and ADO .NET., “”
- Lab 1.1,
- Lab 1.2

Preparation tasks

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices and labs.
- Read the latest .NET Development news at http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000519

Classroom setup

The information in this section provides setup instructions that are required to prepare the instructor computer or classroom configuration for a lab.

► **To prepare for the lab**

1. Make sure that Internet Information Server (IIS) is set up properly.
2. You must have the solution project files for this lab.
3. You must have the Northwind Traders database, Northwind.mdb, installed.

How to Teach This Module

This section contains information that will help you to teach this module.

Lesson: Evolution of Data of Data-Centric Applications

This section describes the instructional methods for teaching each topic in this lesson.

Design of Data-Centric Applications

Technical Notes:

- This module focuses on defining how data-centric applications have evolved from simple one-tier applications to complex, distributed n-tier applications. In addition, this module also defines several common types of data storage.

Transition to Practice Exercise

Now that you have seen several examples of how logic can be divided in different types of data-centric applications, you can explain how the logic divided in an application that you know.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Practices

The first practice in this module asks students to draw a diagram that shows the separation and relationship of the logical layers of a database application that they have designed oar worked on.. This practice helps discern the experience levels of your students. If students have not designed or worked on a database application, you can provide a simple application example for them to work with.

Practice/Discussion Questions

Personalize the following question to the background of the students in your class.

- Which data storage models do you need to access?

Transition to Practice

Write the list on a whiteboard and refer back to it when discussing the .NET Data Providers.

Now that you have seen several examples of how logic can be divided in different types of data-centric applications, you can explain how the logic divided in an application that you know.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

After the practice

Questions for discussion after the practice:

- What lessons did you learn during the practice exercise?
- What did you discover as you created the DataSet, DataTable, and DataColumns?

Lesson: Choosing a Connected or Disconnected Application Environment

This section describes the instructional methods for teaching each topic in this lesson.

Connected versus disconnected environments

The lesson introduces connected and disconnected scenarios. ADO.NET is primarily a set of tools for creating applications for disconnected scenarios.

Discussion Questions: Personalize the following questions to the background of the students in your class.

Transition to Practice Exercise:

The point of this is to discuss the design and architecture of the solution, to start the students thinking about the problems that ADO.NET needs to solve, so they will understand why ADO.NET works the way it does.

Practice

Decide whether to use a connected or disconnected application architecture in each of the following scenarios. Also, describe some of the design issues that might arise in each case.

1. Northwind Traders needs to produce a report using data from two sources. Customer and order information is held in a SQL Server 2000 database. Financial and accounting information is held in an Access database.

   Not enough information is given to decide between a connected or a disconnected application.

   The application will have to combine data from two different sources though. One approach would be to import the data from one source into the other before running the reports. Another approach would be to use a higher level tool that can pull from the two sources and merge the data in a third format. This approach might also indicate a disconnected solution.

2. A Northwind Traders salesperson needs to update information about customers and orders while on the road.

   Create a disconnected application, which provides access to the data when the salesperson is on the road. Some temporary data format is required to hold the data, and to capture any changes the salesperson makes to the data.

   The application enables the salesperson to merge data changes when reconnected to the server, and handle any conflicts that might occur, such as concurrency errors, constraint errors, and so on.

3. Northwind Traders needs to have its SQL Server 2000 data persisted as XML, in order to transfer the data to a Web application. The XML data must be regenerated once a day, to ensure reasonably up-to-date information in the Web application.

   Create a connected application that runs on the Web server or application server. Open a connection to the database. Execute a SQL query to get the required data. Loop through the returned rows, generate an XML string that can be persisted to an XML document on the Web server.

   Note SQL Server 2000 enables you to retrieve a query result directly in XML format. Use the **FOR XML** clause in the **SELECT** statement.

4. Northwind Traders requires a Web application to display information about available products. The database might contain millions of product records. The application must enable the user to scroll through the results one page at a time.

   Create a disconnected Web application. When the user requests product information, open a connection to the database and retrieve the requested information into a cache on the web server. Then close the database connection.

   You can display the cached data on a web page. Provide buttons or some other Web control, to enable the user to page through the product records in the collection. The next time you request the same products, use the cache on the web server to generate a page for the user, without requiring a round-trip to the database server.

Lesson: ADO.NET Architecture

This section describes the instructional methods for teaching each topic in this lesson.

Business use case

As application development has evolved, new applications have become loosely coupled based on the Web application model. An increasing number of applications use XML to encode data to be passed over network connections. ADO.NET provides a programming model that incorporates features of both XML and ADO within the .NET Framework.

What is ADO.NET

Instructor Demonstration:

In some ways you can think of ADO.NET as a marketing term that covers the classes in the System.Data namespace.

Transition to Practice Exercise:

Practice Solution:

After the practice

Questions for discussion after the practice:

The ADO.NET object model

The ADO.NET object model consists of two major parts, the DataSet classes and the .NET data provider classes.

Discussion Questions: Personalize the following questions to the background of the students in your class.

The DataSet object modelThe **DataSet** class has a **Tables** property, which gets a collection of **DataTable** objects in the **DataSet**, and a **Relations** property, which gets a collection of the **DataRelation** objects in the **DataSet**.

Discussion Questions: Personalize the following question to the background of the students in your class.

The .NET Data Provider object model

Technical Notes:

Discussion Questions: Personalize the following questions to the background of the students in your class.

Practice

Describe which ADO.NET classes you would use in the following scenarios:

1. You want to execute an existing stored procedure in SQL Server 7.

   Create a SqlConnection object, to connect to the database. Use a SqlCommand object to execute the stored procedure.

2. You want to execute an existing stored procedure in an Oracle database.

   Create an OleDbConnection object, to connect to the database. Use an OleDbCommand object to execute the stored procedure.

3. You want to list all of the records in a SQL Server 2000 table.

   Create a SqlConnection object, to connect to the database. Use a SqlCommand object to execute the SQL query. Use a SqlDataReader to iterate through the results quickly and efficiently.

4. You want to list all of the records in a SQL Server 6.5 table.

   Create an OleDbConnection object, to connect to the database. Use an OleDbCommand object to execute the SQL query. Use an OleDbDataReader to iterate through the results.

5. You want to display data for an existing SQL Server 2000 table as a grid.

   Create a SqlConnection object, to connect to the database. Create a SqlDataAdapter object containing a SqlCommand object, to query the database.

   Use the SqlDataAdapter to fill a DataSet with the query result. Bind the DataSet to a DataGrid, to display the data to the user.

6. You want to update a SQL Server 2000 data source with changes in a grid.

   Create a SqlConnection object, to connect to the database. Create a SqlDataAdapter object with four SqlCommands objects. The SqlCommand objects specify SQL SELECT, INSERT, DELETE, and UPDATE statements.

   Use the SqlDataAdapter to fill a DataSet with the query result. Bind the DataSet to a DataGrid, to display the data to the user.

   To update the data source, use the SqlCommand objects in the SqlDataAdapter object.

7. You want to create an XML Web service to query and update a SQL Server 2000 database. You want client applications to use XML to represent data passed to and from the XML Web services.

   In the XML Web service, create a SqlConnection object to connect to the database. Create a SqlDataAdapter object with four SqlCommand objects. The SqlCommand objects specify SQL SELECT, INSERT, DELETE, and UPDATE statements.

   Use the SqlDataAdapter to fill a DataSet with the query result. Extract the data in XML format, and send the XML data to the client application.

   When the XML Web service receives updated XML data, load the data back into the DataSet. Use the SqlDataAdapter to reconcile the differences in the data, and to execute appropriate SQL statements to update the database.

What are the data-related namespaces?

The data-related namespaces are provided by the .NET Framework class library to make available the classes, interfaces, enumerations, and delegates for performing disconnected data management.

ADO.NET and XML integration

XML is a rich and portable way of representing data in an open and platform-independent way. An important characteristic of XML data is that it is text-based. This makes it easier to pass XML data between applications and services, rather than passing binary data such as ADO.NET recordsets.

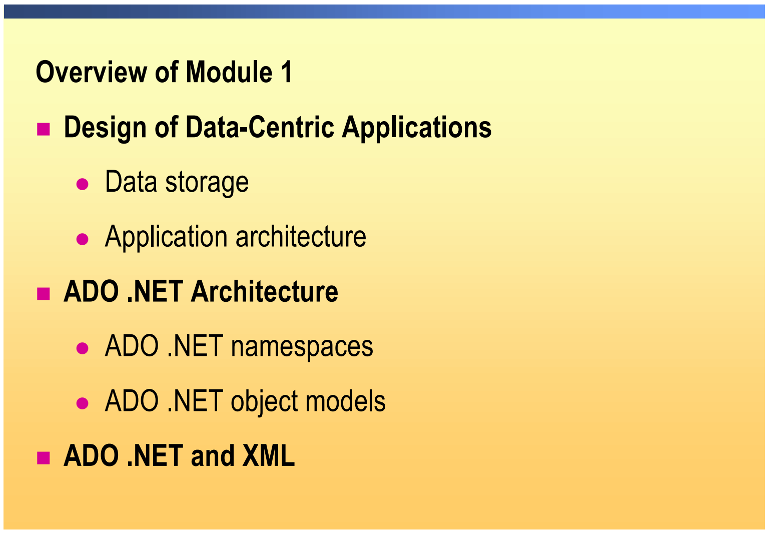
Discussion Questions: Personalize the following questions to the background of the students in your class.

- You can use either the DataRelation constructor or the **Add** method. Why would you use one or the other? What is the difference in results between the two?

Transition to Practice Exercise:

Practice Solution:

Overview


Overview of Module 1

- **Design of Data-Centric Applications**
  - Data storage
  - Application architecture
- **ADO .NET Architecture**
  - ADO .NET namespaces
  - ADO .NET object models
- **ADO .NET and XML**

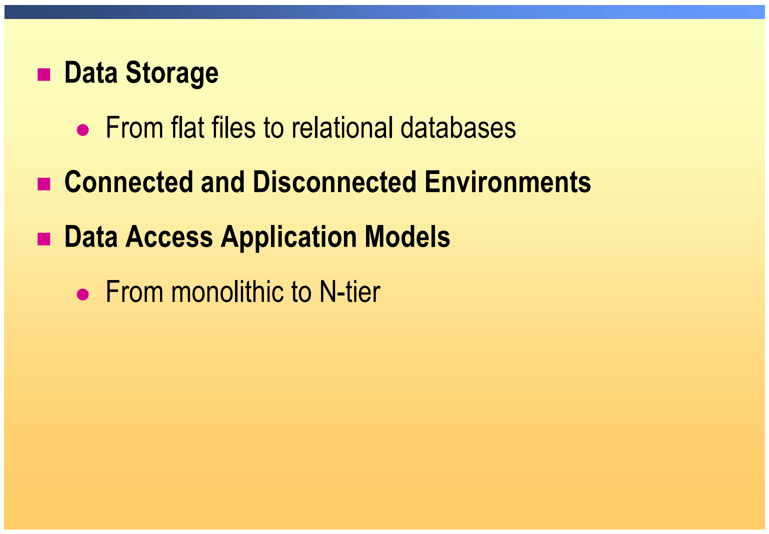
Objectives

Design data-centric applications, describe the ADO .NET architecture, and the integration between ADO .NET and XML.

After completing this module, you will be able to:

- Diagram the architecture of data-centric applications.
- Give examples of storage options.
- Choose a connected or disconnected environment based on application requirements.
- Diagram the ADO .NET object model.
- Use the System.Data namespaces in applications.
- Analyze typical business scenarios.
- Describe the use of XML in ADO .NET.

Lesson: Design of Data-Centric Applications



- **Data Storage**
  - From flat files to relational databases
- **Connected and Disconnected Environments**
- **Data Access Application Models**
  - From monolithic to N-tier

Introduction

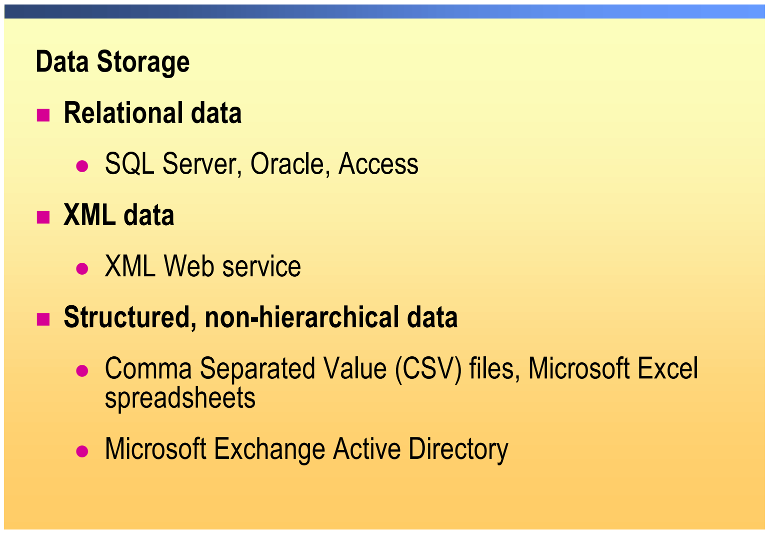
This lesson describes the design of data-centric application architecture and data storage options.

Lesson objectives

After completing this lesson, you will be able to:

- Give examples of common types of data storage.
- Choose between a connected and disconnected application environment.
- Diagram how data access application models have evolved.

Data Storage



Data Storage

- **Relational data**
  - SQL Server, Oracle, Access
- **XML data**
  - XML Web service
- **Structured, non-hierarchical data**
  - Comma Separated Value (CSV) files, Microsoft Excel spreadsheets
  - Microsoft Exchange Active Directory

Definition of data storage

Data storage is a method of storing specific items that together constitute a unit of information. Individual data items themselves are of little use; they become value resources only when put into context with other data items.

Types of data storage

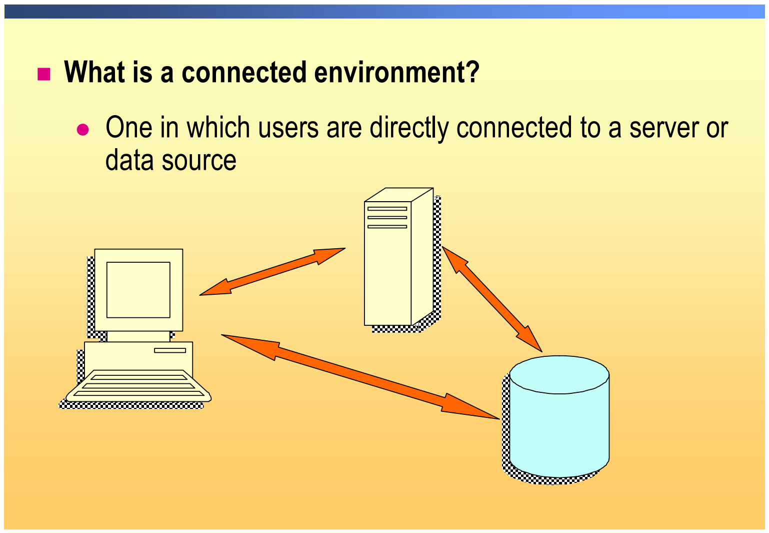
The following table describes different methods of data storage.

Type	Characteristics	Examples
Unstructured	Data has no logical order.	Simple memo
Structured, non-hierarchical	Data is separated into units, but the units are organized strictly by their order.	Comma Separated Value files or tab-separated files, Microsoft Excel spreadsheets, Microsoft Exchange Active Directory™, Indexed Sequential Access Method (ISAM) files
Hierarchical	Data is organized in a tree structure, with nodes that contain other nodes.	XML data document
Relational	Data is organized in tables, with columns containing a specific type of data and rows containing a single record. Tables can be related over columns with identical data.	Microsoft SQL Server™ and Microsoft Access databases, Oracle databases

ADO .NET Support

ADO .NET supports all of these data formats.

What Is a Connected Environment?



- **What is a connected environment?**
  - One in which users are directly connected to a server or data source

Introduction

For much of the history of computers, the only environment available was the connected environment.

Definition

A connected environment is one in which a user or an application is directly connected to a data source.

Advantages

A connected scenario offers the following advantages:

- An easier to secure environment.
- Easier control over locking.
- Data is more likely to be current than in other scenarios.

Disadvantages

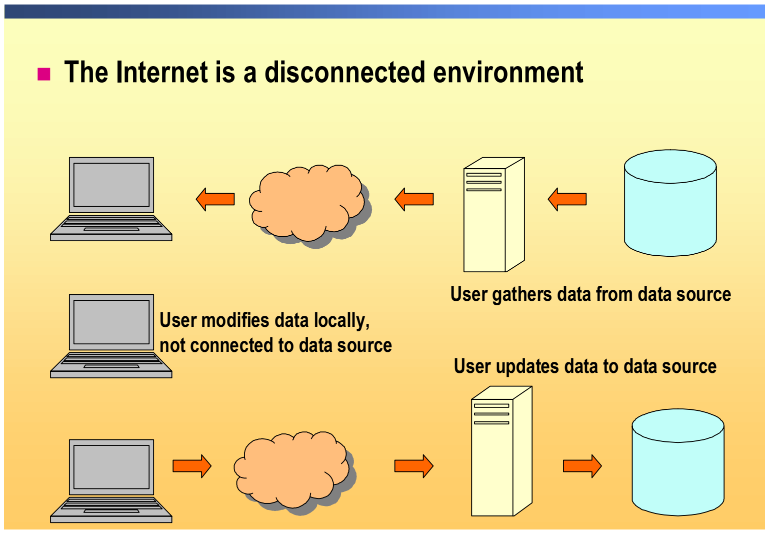
A connected scenario has the following disadvantages:

- Must have a constant network connection.

Example

An airline passenger check-in system requires a constant, always up-to-date connection to a central database.

What Is a Disconnected Environment?


- The Internet is a disconnected environment

User gathers data from data source
User modifies data locally, not connected to data source
User updates data to data source

Introduction

With the advent of the Internet, disconnected work scenarios have become commonplace, and with the increasing use of handheld devices, disconnected scenarios are becoming universal. Laptop, notebook, and other portable computers allow you to use applications when you are disconnected from servers or databases.

A web application is primarily a disconnected application that makes use of freshly designed, ad-hoc tools for managing data.

Definition

A disconnected environment is one in which a user or an application is not directly connected to a server for a source of data or for a middle-tier service. Mobile users who work with laptop computers are the primary users in disconnected environments. Users can take part of the set of application objects with them on a disconnected computer, and then merge changed versions back into the central store. Alternatively, users can simply download the data they need and take it with them on a disconnected computer wherever they go.

Advantages

A disconnected environment provides the following advantages:

- You can work at any time that is convenient for you, and can connect to a server or a database at any time to process requests.
- A connection is freed for other users.
- Improves the scalability of Web applications.

Disadvantages

A disconnected environment has the following disadvantages:

- Data is not always up-to-date.
- Change conflicts can occur and must be resolved.

Practice

Decide whether to use a connected or disconnected architecture in each of the following scenarios and describe some of the design issues that might arise.

1. Northwind Traders needs to produce a report using data from two sources. Customer and order information is held in a SQL Server 2000 database. Financial and accounting information is held in an Access database.

   Not enough information is given to decide between a connected or a disconnected application.

   The application will have to combine data from two different sources though. One approach would be to import the data from one source into the other before running the reports. Another approach would be to use a higher level tool that can pull from the two sources and merge the data in a third format. This approach might also indicate a disconnected solution.

2. A Northwind Traders salesperson needs to update information about customers and orders while on the road.

   Create a disconnected application, which provides access to the data when the salesperson is on the road. Some temporary data format is required to hold the data, and to capture any changes the salesperson makes to the data.

   The application enables the salesperson to merge data changes when reconnected to the server, and handle any conflicts that might occur, such as concurrency errors, constraint errors, and so on.

3. Northwind Traders needs to have its SQL Server 2000 data persisted as XML, in order to transfer the data to a Web application. The XML data must be regenerated once a day, to ensure reasonably up-to-date information in the Web application.

   Create a connected application that runs on the Web server or application server. Open a connection to the database. Execute a SQL query to get the required data. Loop through the returned rows, generate an XML string that can be persisted to an XML document on the Web server.

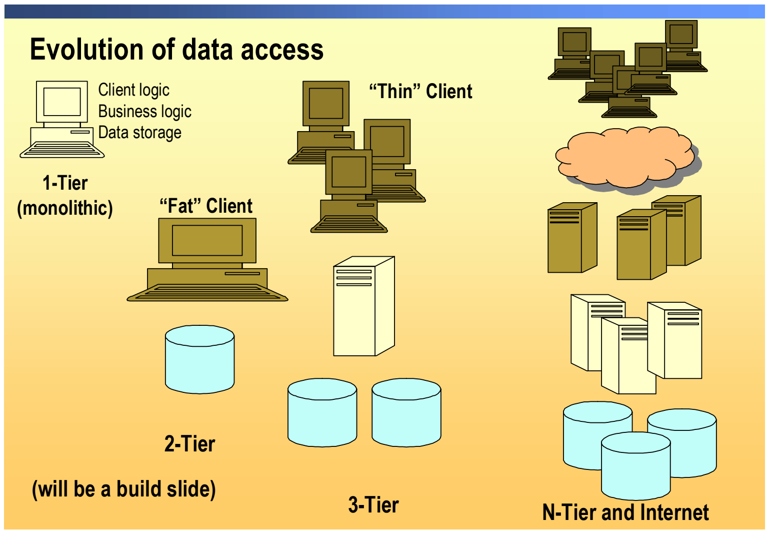
   Note SQL Server 2000 enables you to retrieve a query result directly in XML format. Use the **FOR XML** clause in the **SELECT** statement.

4. Northwind Traders requires a Web application to display information about available products. The database might contain millions of product records. The application must enable the user to scroll through the results one page at a time.

   Create a disconnected Web application. When the user requests product information, open a connection to the database and retrieve the requested information into a cache on the web server. Then close the database connection.

   You can display the cached data on a web page. Provide buttons or some other Web control, to enable the user to page through the product records in the collection. The next time you request the same products, use the cache on the web server to generate a page for the user, without requiring a round-trip to the database server.

Data Access Application Models



Introduction

Data access models have evolved with the evolution of computers, from highly localized to highly distributed. As the number of users and the amount of data increased, data access models evolved from a single-user on a single computer to multiple users on the Internet. The latest development in this evolution is the XML Web service model.

Definition of tier

Within a data access model, a tier is logical level or layer at which the logical components of an application reside, not a physical tier. Tiers can reside on one or more computers. The number of tiers refers to the number of levels, not the number of physical computers, into which services are divided. These levels typically include the following:

- Client tier, also known as the presentation or user services layer. This tier contains the user interface.
- Business logic tier, which contains the logic that interacts with the source of data. This "middle" tier contains the part of the application that interacts with the data, for example, creating a connection to the source of the data.
- Data services tier, which contains the data that the business logic uses in the applications.
- Interoperability tier, which contains the logic that allows interaction between applications on different operating systems, or different types of data.

Benefits of tiers

The major advantage of adding tiers is the ability to scale applications. Each additional tier allows you to add more users and isolate a level of application logic. Isolating the logic enables you to make changes to a specific area of an application without requiring changes to the other tiers.

For example, in a 1-tier application, a change to any level of logic requires that the entire application be recompiled and redistributed.

Evolution of access models

The following table compares the different types of data access models.

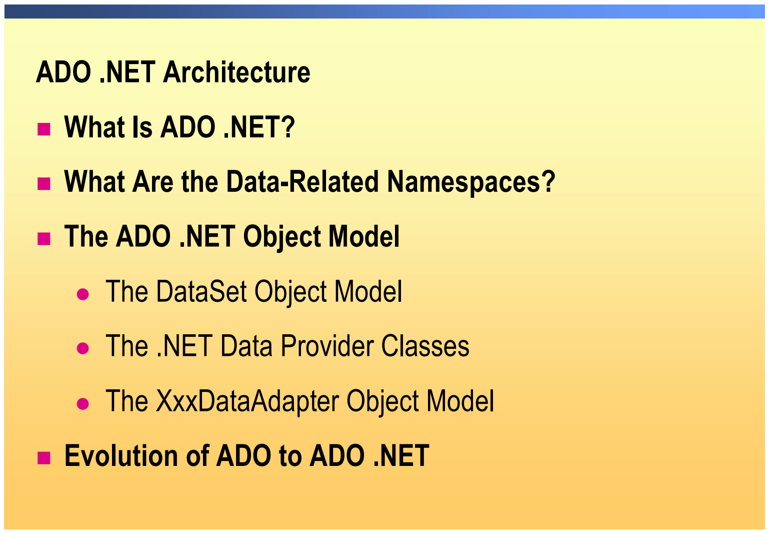
Model	Description	Advantages	Disadvantages
1-tier, or Monolithic	This model typically involves a single user and all three layers on a single computer. For example, an old-style Microsoft Access database with a single user.	Because everything is in one place, all components are easily accessible.	Program update requires source code to be modified, recompiled, and redistributed for every user. This model provides no real ability to scale.
2-tier – client/server	The user layer and business logic layer reside in one tier, data services on another. Typically involves two or more machines. For example, a business personnel database. Often the business logic is split between the two tiers: some logic in the client application, and some as stored procedures in the data tier.	Provides some separation of functions.	Difficult to scale. The client is a “Fat Client” that contains both the presentation and business logic layers.
3-tier	Each service is in a separate layer. Business logic moves into a new “Middle tier.”	Good separation of functions. The client layer is a “thin client” that contains only the client logic, or presentation layer.	More complex to manage. Security, is not as scalable/flexible as n-tier
N-tier within an organization	An enterprise-level personnel database where several clients access a single application server. New tiers can be added as new logical needs occur.	Allows different applications on different operating systems to interact with both the user and the data.	Security
N-tier with Web interface	Services are distributed among Internet and intranet, with additional tier and additional servers dedicated to the network.	Zero client deployment costs. Only updates are to web/application servers.	Security

Practice

1. Think of a database application that you have developed in the past.
2. Draw a diagram showing the data access application model it used.
3. Draw the data storage formats you needed to access.

   Did the architecture use a connected or disconnected environment?

Lesson: ADO .NET Architecture



ADO .NET Architecture

- What Is ADO .NET?
- What Are the Data-Related Namespaces?
- The ADO .NET Object Model
  - The DataSet Object Model
  - The .NET Data Provider Classes
  - The XxxDataAdapter Object Model
- Evolution of ADO to ADO .NET

Introduction

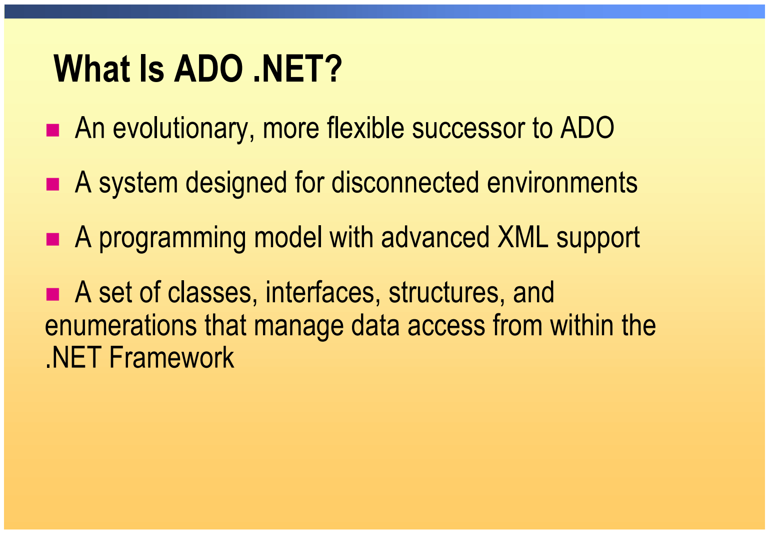
This lesson introduces ADO .NET and establishes its place in the .NET Framework.

Lesson objectives

After completing this lesson, you will be able to:

- Explain how to use ADO .NET.
- Describe how ADO .NET is divided into namespaces.
- Diagram the ADO .NET object model.

What Is ADO .NET?



What Is ADO .NET?

- An evolutionary, more flexible successor to ADO
- A system designed for disconnected environments
- A programming model with advanced XML support
- A set of classes, interfaces, structures, and enumerations that manage data access from within the .NET Framework

Introduction

ADO .NET is the next step in the evolution of Microsoft ActiveX Data Objects (ADO). It does not share the same programming model, but shares much of the functionality.

Definition

ADO .NET is a set of classes for working with data.

Business use case

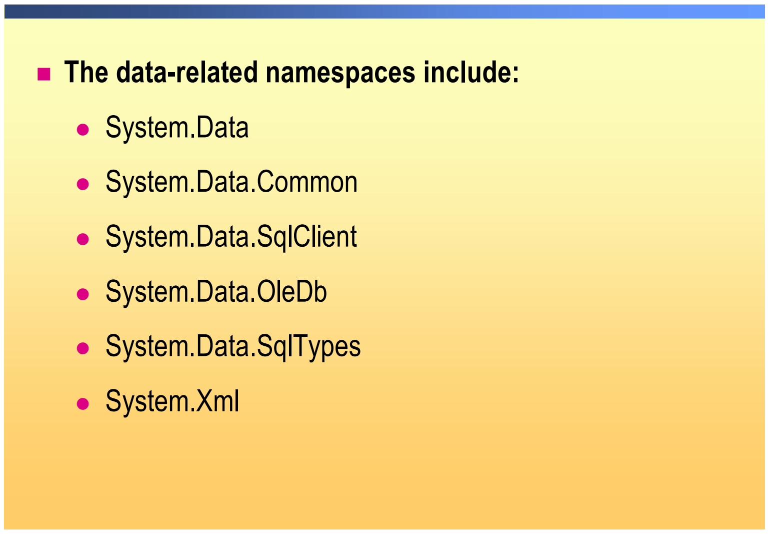
As application development has evolved, new applications have become loosely coupled based on the Web application model. An increasing number of applications use XML to encode data to be passed over network connections. ADO .NET provides a programming model that incorporates features of both XML and ADO .NET within the .NET Framework.

Benefits

ADO .NET provides these advantages over other data access models and components:

- Interoperability. ADO .NET uses XML as the format for transmitting data from a data source to a local in-memory copy of the data.
- Maintainability. When an increasing number of users work with an application, the increased use can strain resources. By using n-tier applications, you can spread application logic across additional tiers. ADO .NET architecture uses local in-memory caches to hold copies of data, making it easy for additional tiers to trade information.
- Programmability. The ADO .NET programming model uses strongly-typed data. Strongly typed data makes code more concise and easier to write because Visual Studio .NET provides statement completion.
- Performance. ADO .NET helps you to avoid costly data type conversions because of its use of strongly typed data.
- Scalability. The ADO .NET programming model encourages programmers to conserve system resources for applications that run over the Web. Because data is held locally in in-memory caches, there is no need to retain database locks or maintain active database connections for extended periods.

What Are the Data-Related Namespaces?



- **The data-related namespaces include:**
  - System.Data
  - System.Data.Common
  - System.Data.SqlClient
  - System.Data.OleDb
  - System.Data.SqlTypes
  - System.Xml

Introduction

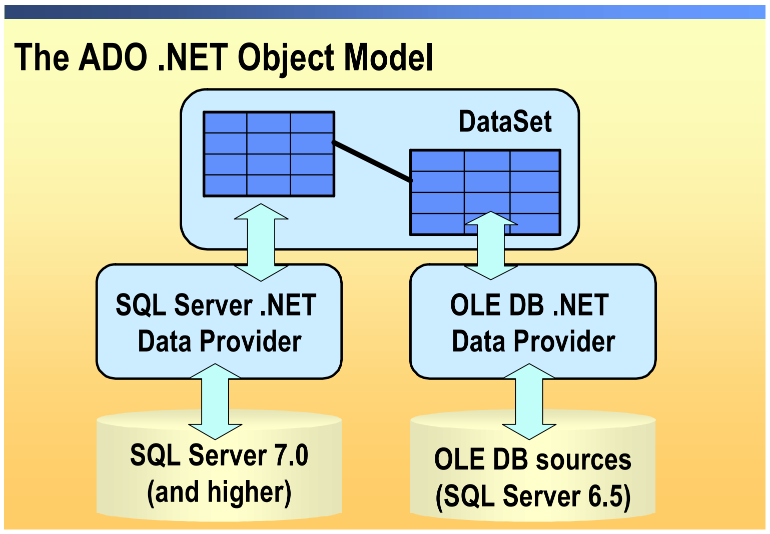
The .NET Framework divides functionality into logical namespaces, and ADO .NET is no exception. ADO .NET is implemented primarily in the **System.Data** namespace hierarchy, which physically resides in the **System.Data.dll** assembly. Some parts of ADO .NET are part of the **System.Xml** namespace hierarchy, for example the **XmlDataDocument** class.

The data-related namespaces

The following table describes the data-related namespaces.

Namespace	**Description**
System.Data	The core of ADO .NET. Includes classes that make up the disconnected part of the ADO .NET architecture, for example, the DataSet classes
System.Data.Common	Utility classes and interfaces that are inherited and implemented by .NET data providers
System.Data.SqlClient	The SQL Server .NET Data Provider
System.Data.OleDb	The OLE DB .NET Data Provider
System.Data.SqlTypes	Classes and structures for native SQL Server data types; a safer, faster alternative to other data types
System.Xml	Classes, interfaces, and enumerations that provide standards-based support for processing XML. For example, the XmlDataDocument class.

The ADO .NET Object Model


The ADO .NET Object Model
DataSet
SQL Server .NET Data Provider
OLE DB .NET Data Provider
SQL Server 7.0 (and higher)
OLE DB sources (SQL Server 6.5)

Introduction

The ADO .NET object model consists of two major parts, the DataSet classes and the .NET data provider classes.

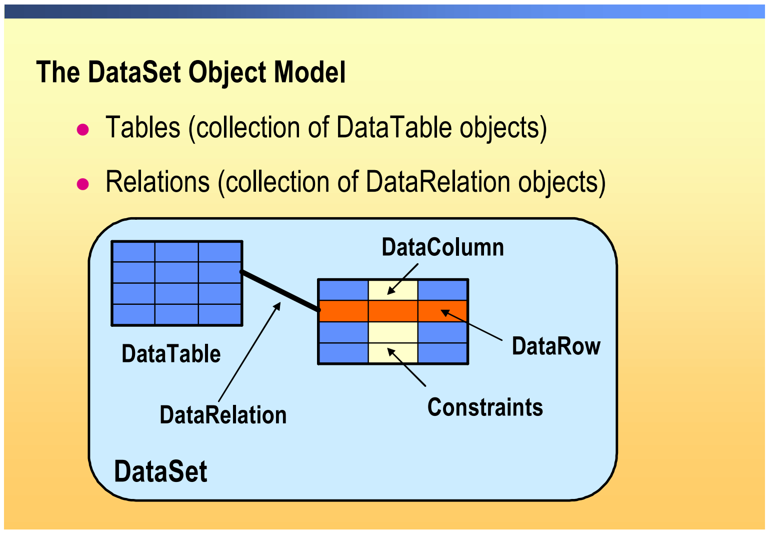
The DataSet classes

The DataSet classes allow the storage and management of data in a disconnected cache. The DataSet is independent of any underlying data source, so its features are available to all applications, regardless of where the applications data originated.

The .NET data provider classes

The .NET data provider classes are specific to a data source. The .NET data providers must therefore be written specifically for a data source, and will work only with that data source. The .NET data provider classes provide the ability to connect to a data source, retrieve data from the data source, and perform updates on the data source.

The DataSet Object Model


The DataSet Object Model

- Tables (collection of DataTable objects)
- Relations (collection of DataRelation objects)

Tables in a DataSet

The **DataSet** class has a **Tables** property, which gets a collection of **DataTable** objects in the **DataSet**, and a **Relations** property, which gets a collection of the **DataRelation** objects in the **DataSet**.

A **DataTable** object contains several collections that describe the data in the table and to cache the data in memory. The following table describes the most important collections.

Collection name	Type of object in collection	Description of object in collection
Columns	DataColumn	Contains metadata about a column in the table, such as the column name, data type, and whether rows can contain a NULL value in this column.
Rows	DataRow	Contains a row of data in the table. A **DataRow** object also maintains the original data in the row, before any changes were made by the application.
Constraints	Constraint	Represents a constraint on one or more **DataColumn** objects. Constraint is an abstract class. There are two concrete subclasses: **UniqueConstraint** and **ForeignKeyConstraint**.
ChildRelations	DataRelation	Represents a relationship to a column in another table in the dataset. Use **DataRelation** objects to create links between primary keys and foreign keys in your tables.

Relations between tables in a DataSet

If a **DataSet** has multiple tables, some of the tables might contain related data. You create **DataRelation** objects to describe these relationships to the **DataSet**. A **DataSet** can contain a collection of **DataRelation** objects.

You can use **DataRelation** objects to programmatically fetch related child records for a parent record, or a parent record from a child record.

Example of using the DataSet object model

The Northwind database in SQL Server contains a Products table. This table contains information for each product, such as its product ID, product name, and supplier ID. The following table shows some data in the Products table.

ProductID	Product Name	Supplier ID
1	Chai	1
2	Chang	1
3	Aniseed Syrup	1
4	Chef Anton's Cajun Seasoning	2

The Northwind database also contains a Suppliers table. This table contains information about companies that supply products to Northwind Traders. The following table shows some data in the Suppliers table.

SupplierID	CompanyName	City
1	Exotic Liquids	London
2	New Orleans Cajun Delights	New Orleans
3	Grandma Kelly's Homestead	Ann Arbor

You can create a **DataSet** object that contains copies of both these tables, including both structure and data. You can also create a **DataRelation** object, to describe the relationship between the tables through the SupplierID column. This enables you to get the supplier details for a particular product. You can also get a list of all products supplied by a particular supplier.

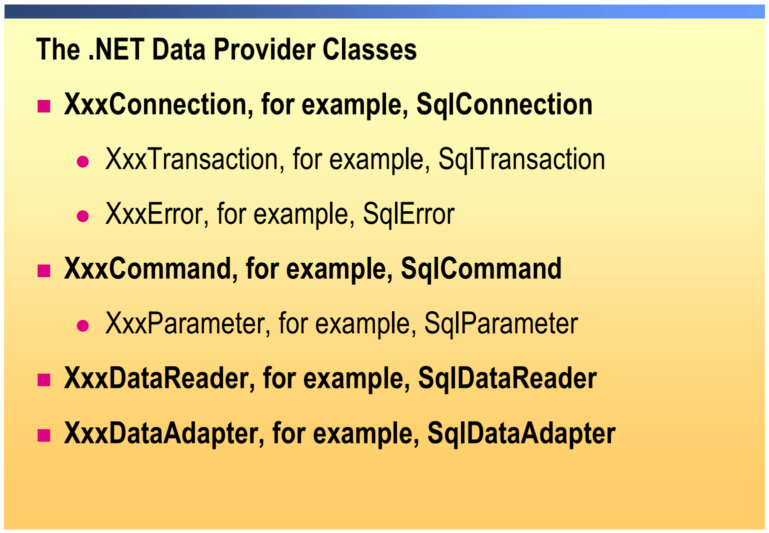
Data Binding

DataSets can be bound to most controls in Windows Forms or Web Forms so they make building data-related user interfaces easy.

DataSet Schema

The schema (structure) of a DataSet can be defined programmatically using the DataSet object model, for example, by writing code to create DataTable and DataRelation objects, or by using an XSD (XML Schema Definition) file. XSD files are covered in Module 5: *Using XML With ADO .NET*.

The .NET Data Provider Classes



The .NET Data Provider Classes

- XxxConnection, for example, SqlConnection
  - XxxTransaction, for example, SqlTransaction
  - XxxError, for example, SqlError
- XxxCommand, for example, SqlCommand
  - XxxParameter, for example, SqlParameter
- XxxDataReader, for example, SqlDataReader
- XxxDataAdapter, for example, SqlDataAdapter

Introduction

ADO .NET uses the .NET data providers to connect to a data source, retrieve data, manipulate data, and update the data source. The .NET data providers are designed to be lightweight. They create a minimal layer between your code and the data source, to increase performance without sacrificing functionality.

Definition

The .NET Framework includes the following two data providers.

Data provider	Description
SQL Server .NET	Provides optimized access to SQL Server 2000 and SQL Server 7.0 databases.
OLE DB .NET	Provides access to SQL Server versions 6.5 and earlier. Also provides access to other databases, such as Oracle, Sybase, DB2/400, and Microsoft Access.

In addition, Microsoft will provide an ODBC .NET Data Provider for access to other data sources. This data provider will be available as a publicly accessible Web release download.

SQL Server .NET Data Provider

To use the SQL Server .NET Data Provider, you will need to include the **System.Data.SqlClient** namespace in your applications. This provider is more efficient that using the OLE DB .NET Data Provider because it does not pass through an OLE DB or ODBC layer.

OLE DB .NET Data Provider

To use the OLE DB .NET Data Provider, you will need to include the **System.Data.OleDb** namespace in your applications.

Data provider classes

ADO .NET exposes a common object model for .NET data providers. The following table describes the four core classes that make up a .NET data provider.

In the SQL Server .NET Data Provider, the class names begin with the prefix **Sql**. For example, the connection class is called **SqlConnection**.

In the OLE DB .NET Data Provider, the class names begin with the prefix **OleDb**. For example, the connection class is called **OleDbConnection**.

In the future, more .NET data providers will be written with other prefixes. In the following table, these different prefixes are indicated with Xxx.

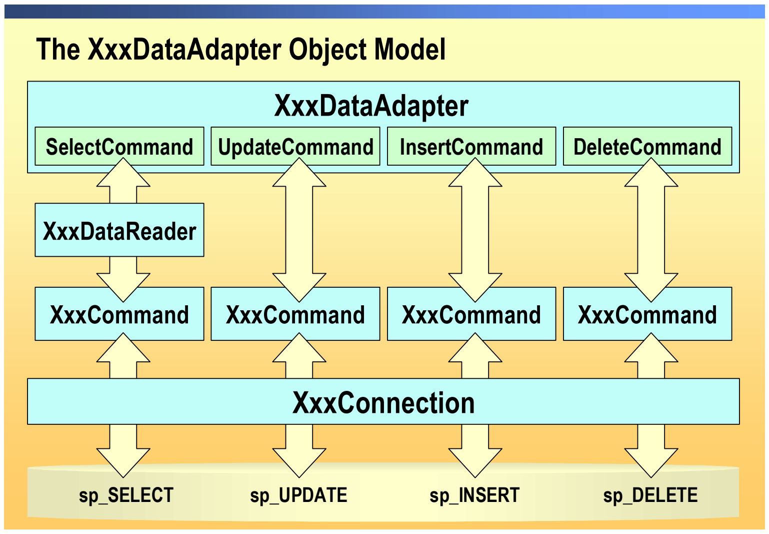
Class	Description
XxxConnection	Establishes a connection to a specific data source. For example, the **SqlConnection** class connects to SQL Server data sources.
XxxCommand	Executes a command from a data source. For example, the **SqlCommand** class can execute stored procedures in a SQL Server data source.
XxxDataReader	Reads a forward-only, read-only stream of data from a data source. For example, the **SqlDataReader** class can read rows from tables in a SQL Server data source. It is returned by the **ExecuteReader** method of the **XxxCommand** class, typically as a result of a SELECT SQL statement.
XxxDataAdapter	Uses **XxxCommand** objects to populate a **DataSet**, and resolves updates with the data source. For example, the **SqlDataAdapter** class can manage the interaction between a DataSet and the underlying data in a SQL Server data source.

Example of using an XxxDataReader

The **XxxDataReader** class provides forward-only, read-only access to data in a data source. For example, to use a **SqlDataReader** to read data from a SQL Server database:

1. Create a **SqlConnection** object, to connect to the SQL Server database.
2. Create a **SqlCommand** object, containing a SQL SELECT statement to query the database.
3. Create a **SqlDataReader** object.
4. Execute the **SqlCommand** object using the **ExecuteReader** method, and assign the results to the **SqlDataReader** object.
5. Use the **Read** method of the **SqlDataReader** to iterate forward through the data, and process the rows.

The XxxDataAdapter Object Model



Introduction

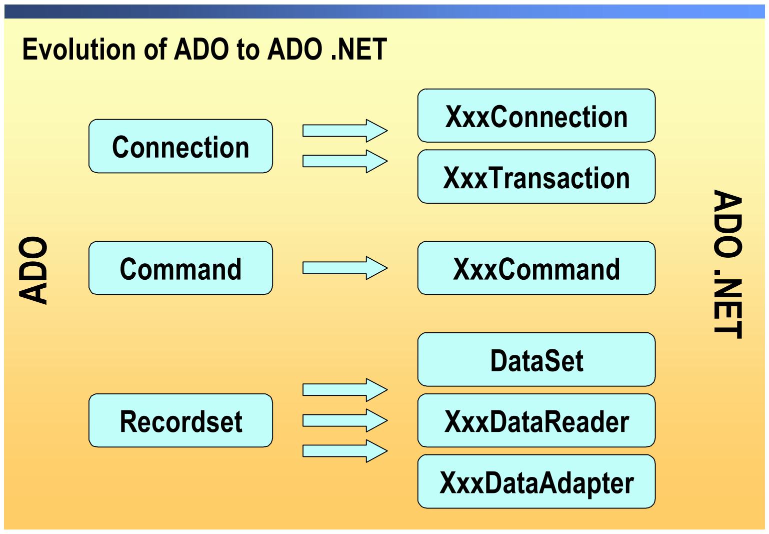
The XxxDataAdapter class provides easy to manage disconnected functionality. It is used to populate datasets, and then update the underlying data source with any changes made to the dataset.

Example of using a data adapter

You can use a data adapter to populate a **DataSet**, and to send data updates back to the data source. For example, to use a **DataSet** with a SQL Server .NET Data Provider:

1. Create a **SqlConnection** object, to connect to a SQL Server database.
2. Create a **SqlDataAdapter** object. The object contains properties that can point to four **SqlCommand** objects. These objects specify SQL statements to SELECT, INSERT, DELETE, and UPDATE data in the database.
3. Create a **DataSet** object that contains one or more tables.
4. Use the **SqlDataAdapter** object to fill a **DataSet** table by calling the **Fill** method. The **SqlDataAdapter** implicitly executes the **SqlCommand** object that contains a SELECT statement.
5. Modify the data in the **DataSet**. You can do this programmatically, or by binding the **DataSet** to a user interface control such as a **DataGrid** and then changing the data in the grid.
6. When you are ready to send the data updates to the database, use the **SqlDataAdapter** to call the **Update** method. The **SqlDataAdapter** object implicitly uses its **SqlCommand** objects to execute INSERT, DELETE, and UPDATE statements on the database.

Evolution of ADO to ADO .NET



Introduction

There have been many changes between ADO and ADO .NET. Most of the changes are as a response to studies of how developers use (and misuse) ADO. The other changes are to make ADO .NET more flexible, more powerful, more scalable, and better performing than ADO.

Divide and Conquer

Not all .NET data providers will provide transactional functionality, so ADO .NET moves that functionality into a separate class. This also means that the new ADO .NET connection object is lighter-weight than the old ADO connection object.

The ADO Recordset was a huge object in ADO. It provided the ability to support multiple types of cursor, from a fast, light-weight “firehose” cursor, to a disconnected client-side cursor that supported change tracking, optimistic locking, and automatic batch updates of a central database. However, all this functionality was hard (or impossible) to customize.

ADO .NET breaks the functionality of the old ADO Recordset into multiple classes, thereby allowing a focused approach to developing code. The data reader is the equivalent of a “firehose” cursor. The dataset provides disconnected data cache with tracking and control binding functionality. The data adapter provides the ability to completely customize how the central data store is updated with the changes to a dataset.

Reference

See Appendix B for more on the reasons for the changes between ADO and ADO .NET.

Practice

Describe which ADO .NET classes you would use in the following scenarios:

1. You want to execute an existing stored procedure in SQL Server 7.

   Create a SqlConnection object, to connect to the database. Use a SqlCommand object to execute the stored procedure.

2. You want to execute an existing stored procedure in an Oracle database.

   Create an OleDbConnection object, to connect to the database. Use an OleDbCommand object to execute the stored procedure.

3. You want to list all of the records in a SQL Server 2000 table.

   Create a SqlConnection object, to connect to the database. Use a SqlCommand object to execute the SQL query. Use a SqlDataReader to iterate through the results quickly and efficiently.

4. You want to list all of the records in a SQL Server 6.5 table.

   Create an OleDbConnection object, to connect to the database. Use an OleDbCommand object to execute the SQL query. Use an OleDbDataReader to iterate through the results.

5. You want to display data for an existing SQL Server 2000 table as a grid.

   Create a SqlConnection object, to connect to the database. Create a SqlDataAdapter object containing a SqlCommand object, to query the database.

   Use the SqlDataAdapter to fill a DataSet with the query result. Bind the DataSet to a DataGrid, to display the data to the user.

6. You want to update a SQL Server 2000 data source with changes in a grid.

   Create a SqlConnection object, to connect to the database. Create a SqlDataAdapter object with four SqlCommands objects. The SqlCommand objects specify SQL SELECT, INSERT, DELETE, and UPDATE statements.

   Use the SqlDataAdapter to fill a DataSet with the query result. Bind the DataSet to a DataGrid, to display the data to the user.

   To update the data source, use the SqlCommand objects in the SqlDataAdapter object.

7. You want to create an XML Web service to query and update a SQL Server 2000 database. You want client applications to use XML to represent data passed to and from the XML Web services.

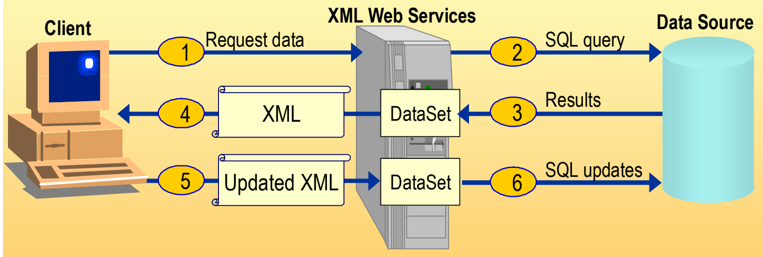
   In the XML Web service, create a SqlConnection object to connect to the database. Create a SqlDataAdapter object with four SqlCommand objects. The SqlCommand objects specify SQL SELECT, INSERT, DELETE, and UPDATE statements.

   Use the SqlDataAdapter to fill a DataSet with the query result. Extract the data in XML format, and send the XML data to the client application.

   When the XML Web service receives updated XML data, load the data back into the DataSet. Use the SqlDataAdapter to reconcile the differences in the data, and to execute appropriate SQL statements to update the database.

Lesson: ADO .NET and XML

- ADO .NET is tightly integrated with XML
- Using XML in a disconnected ADO .NET application



- Using XML in a connected ADO .NET application
  - Load XML into a DOM tree, or use a forward-only reader

Introduction

ADO .NET is tightly integrated with XML. The ADO .NET object model has been designed with XML at the core, rather than as an added extra, as in ADO 2.x. ADO .NET makes it easy for you to convert relational data into XML format. You can also convert data from XML into a collection of tables and relations.

Importance

XML is a rich and portable way of representing data in an open and platform-independent way. An important characteristic of XML data is that it is text-based. This makes it easier to pass XML data between applications and services, rather than passing binary data such as ADO Recordsets.

Scenario

You need to write an application that processes XML data. The XML data may come from an external business via an XML Web service, email, BizTalk Server, or many other sources.

Definition

The ADO .NET object model includes extensive support for XML. Consider the following facts and guidelines when using the XML support in ADO .NET:

- You can read data from a dataset in XML format. This is useful if you want to pass data between applications or services in a distributed environment.
- You can fill a dataset with XML data. This is useful if you receive XML data from another application or service, and want to update a database using this data.
- You can create an XML Schema for the XML representation of the data in a dataset. You can use the XML Schema to perform tasks such as serializing the XML data to a stream or file.

- You can load XML data into a Document Object Model (DOM) tree, from a stream or file. You can then manipulate the data as XML or as a dataset. To do this, you must have an XML Schema to describe the structure of the data to the dataset.
- You can create typed datasets. A typed dataset is a subclass of **DataSet**, with added properties and methods to expose the structure of the dataset. Visual Studio generates an equivalent XML Schema definition for the typed dataset, to describe the XML representation of the dataset.

Example of using XML in a disconnected ADO .NET application

This example describes how to use XML in a disconnected ADO .NET application. You can XML to pass data between the different parts of the system as follows:

1. The client application invokes an XML Web service, to request data from a database.
2. The XML Web service queries a data source, to obtain the requested data.
3. The XML Web service loads the results into a dataset.
4. The XML Web service translates the data into XML format, and returns the XML data to the client application.
5. The client application processes the XML data in some way. For example, the client can load the XML data into a dataset, and bind it to user-interface controls such as a **DataGrid**. When the client application is ready, it invokes an XML Web service to update the data source with the data changes.
6. The XML Web service loads the new XML data into a dataset, and uses the new data to update the data source.

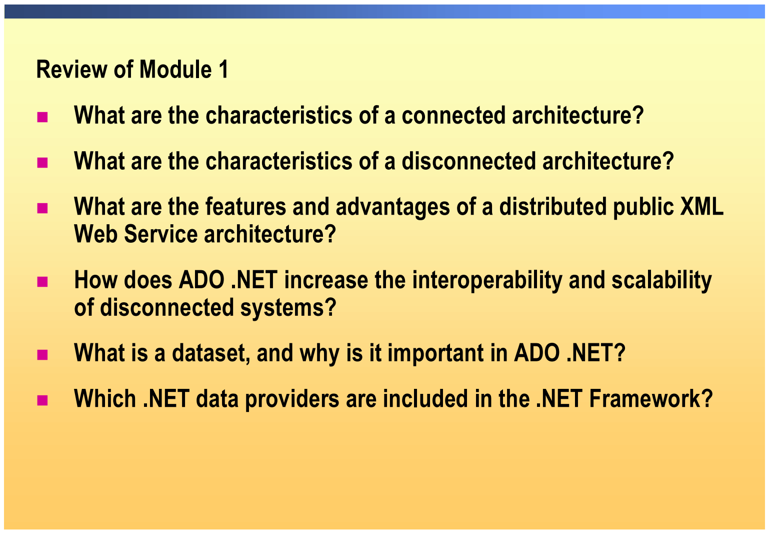
Reference

Module 5: *ADO .NET and XML*, includes more detail about the integration between ADO .NET and XML.

There are also some other MOC courses if you are interested in XML.

- 1905B: Building XML-Based Applications.
- 1913A: Exchanging and Transforming Data Using XML and XSLT.
- 2500A: Introduction to XML and the .NET Platform.
- 2503A: XML within the .NET Framework.
- 2091A: Building XML-Enabled Applications using Microsoft SQL Server 2000.

Review



Review of Module 1

- What are the characteristics of a connected architecture?
- What are the characteristics of a disconnected architecture?
- What are the features and advantages of a distributed public XML Web Service architecture?
- How does ADO .NET increase the interoperability and scalability of disconnected systems?
- What is a dataset, and why is it important in ADO .NET?
- Which .NET data providers are included in the .NET Framework?

1. What are the characteristics of a connected architecture? Describe some scenarios where a connected architecture is appropriate.

   A connected architecture is one where an application can connect directly to a data source, to query and modify the stored data.

   A connected architecture is appropriate for in-house applications that access a data source over a Local Area Network. Another example is an ASP .NET Web Application that queries a database to generate a read-only report of the data.

2. What are the characteristics of a disconnected architecture? Describe a scenario where a disconnected architecture is appropriate.

   A disconnected architecture is one where an application does not connect directly to a data source. ADO .NET provides extensive support for building disconnected applications, to meet the needs of modern-day distributed systems.

   A disconnected architecture is appropriate for mobile workers. At the start of the day, a private copy of customers and products data can be downloaded from the database. During the day, the worker uses and modifies this copy of the data. At the end of the day, the worker posts the data changes back to the database. XML Web services have an important role to play in this style of application.

3. What are the features and advantages of the XML Web service architecture?

   Business services are made available to disconnected users. XML Web services can retrieve private copies of data from a data source to enable users to work with the data remotely. XML Web services also allow the user to post data updates back to the data source when required.

4. How does ADO .NET increase the interoperability and scalability of disconnected systems?

   XML makes interoperability possible. Data from a data source can be expressed in XML format, which makes it easier to exchange the data between application tiers and across organizational boundaries. XML is tightly integrated into the design and philosophy of ADO .NET.

   Scalability is achieved by enabling data to be cached locally in in-memory caches. This reduces the number of active connections and database locks required, which means that more users can be supported at the same time.

5. What is a dataset, and why is it important in ADO .NET?

   The DataSet class is the most fundamental concept in the design of the ADO .NET disconnected architecture. A dataset is an in-memory cache of data tables, relations, and constraints. You can populate a dataset from a SQL query, an XML document, or by creating tables, relations, and constraints programmatically.

   You can manipulate a dataset in a disconnected application. You can also write out the contents of a dataset in XML format, and pass it to other applications and services.

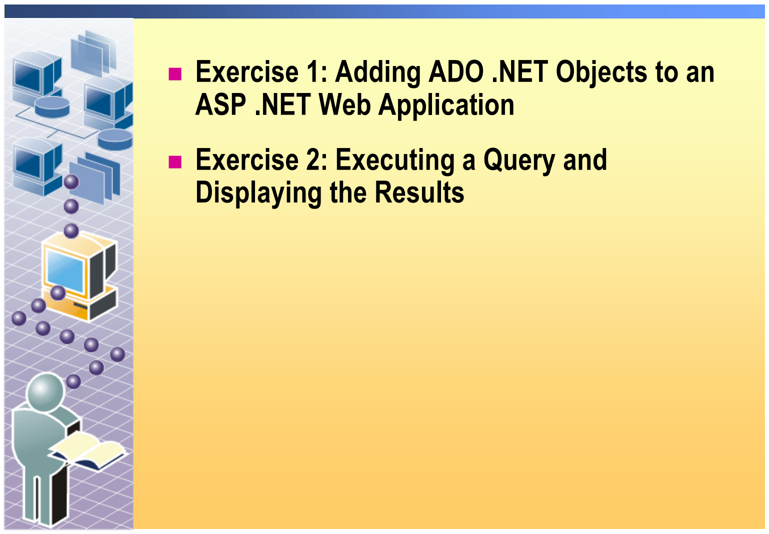
6. Which .NET data providers are included in the .NET Framework?

   The .NET Framework includes two data providers: the SQL Server .NET Data Provider, and the OLE DB .NET Data Provider.

   The SQL Server .NET Data Provider gives optimized access to SQL Server 2000 and SQL Server 7.0 databases.

   The OLE DB .NET Data Provider gives access to SQL Server 6.5 (and earlier), Oracle, Sybase, and other databases.

Lab 1: Data-Centric Applications and ADO .NET



- Exercise 1: Adding ADO .NET Objects to an ASP .NET Web Application
- Exercise 2: Executing a Query and Displaying the Results

Objectives

After completing this lab, you will be able to:

- Identify the architecture of data-centric applications and storage options.
- Decide when to use different classes in the ADO .NET object model.
- Identify and use the System.Data namespaces and assemblies.
- Choose a connected or disconnected environment, based on application requirements.
- Propose an ADO .NET application architecture for typical business scenarios.

Prerequisites

Before working on this lab, you must have:

- The solution project files for this lab.
- The Northwind database installed.

Scenario

Northwind Traders needs to provide its customers and suppliers with an up-to-date product list. The company decides to develop a Web application to make this information available to as many users as possible.

The product data is held in a SQL Server 2000 database. The Web application will use the SQL Server .NET Data Provider because it provides optimized access to SQL Server 2000.

The Web application will use a disconnected architecture. Each time a user requests product data, the application performs the following tasks:

1. Open a connection to the database by using a **SqlConnection** object.
2. Query the database by using a **SqlCommand** object.
3. Create a **SqlDataReader** object, to read the results as efficiently as possible. The **SqlDataReader** object provides forward-only, read-only access to the data, and is the fastest way to process data in ADO .NET. Speed is an important design issue in Web applications, where demand can peak sharply.
4. Bind a **DataGrid** to the **SqlDataReader** object. This will cause the **SqlDataReader** object to loop through the data, and populate the **DataGrid** with that data.
5. Close the **SqlDataReader** object.
6. Close the **SqlConnection** object.

This design ensures that the database connection is kept open for the shortest possible time. Releasing database connections quickly is an important requirement in many applications.

Starter and solution files

You will create a new ASP .NET Web application in the folder <*install folder*>\Labs\Lab01\Starter.

A Visual Basic solution is provided in the folder <*install folder*>\Labs\Lab01\Solution\VB.

A Visual C# solution is provided in the folder <*install folder*>\Labs\Lab01\Solution\CS.

Estimated time to complete this lab: 30 minutes

Exercise 1
Adding ADO .NET Objects to an ASP .NET Web Application

In this exercise, you will select specific data from tables in the Northwind database.

► To create an ASP.NET Web application

In this procedure, you will create a new ASP .NET Web application in Visual Studio .NET. You will examine the data-related assemblies that are referenced by default in the project.

1. Start Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, select the following options, and then click **OK**.

Option	Selection
Project Types	Visual Basic Projects (or Visual C# Projects if you prefer)
Templates	ASP .NET Web Application
Name	MyWebApplication
Location	http://localhost/2389/Labs/Lab01/Starter

4. In the **Solution Explorer**, expand the **References** folder. Notice that the **System.Data** and **System.XML** assemblies are already referenced in the project, in addition to other core assemblies.

► To add a SqlConnection object to the application

In this procedure, you will add a **SqlConnection** object to your application. You will set its connection string to the Northwind database in SQL Server 2000.

1. In the **Toolbox,** select the **Data** tab.

   What data-related controls are available in this tab?

2. Drop a **SqlConnection** object onto the design surface of **WebForm1.aspx**. This action causes a **SqlConnection** object named **SqlConnection1** to appear beneath the form.
3. Right-click **SqlConnection1**, and then select **Properties**.
4. In the **Properties** window, select the **ConnectionString** property. In the drop-down list for this property, select **<New Connection…>**.

5. The **Data Link Properties** dialog box appears. Enter the following information in this dialog box, and then click **OK**:

Field in dialog box	Enter this information
Server name	(local)
Information to log on to server	Use Windows NT Integrated Security
Database on the server	Northwind

6. In the **Properties** window, examine the new value of the **ConnectionString** property.

► To add a SqlCommand object to the application

In this procedure, you will add a **SqlCommand** object to your application. You will configure the object to execute a SQL query, by using the database connection held in **SqlConnection1**.

1. In the **Toolbox**, select the **Data** tab, and then drop a **SqlCommand** object onto your web form. This action creates a new object named **SqlCommand1** in your application.
2. In the **Properties** window, set the following properties for **SqlCommand1.** (Set the properties in the order shown; you can use the Query Builder or type the SQL statement manually)

Property	Value
Connection	SqlConnection1
CommandText	SELECT ProductName, UnitPrice FROM Products

3. When prompted, choose **Yes** to regenerate the parameters collection.

► To examine the code generated by the Web Form Designer

When you add objects to your Web form, the Web Form Designer generates code to create and configure these objects in your application. In this procedure, you will examine the code that is generated for the **SqlConnection1** and **SqlCommand1** objects.

1. Right-click **WebForm1.aspx**, and then select **View Code**.
2. In the code editor, expand the gray box labeled **Web Form Designer Generated Code**.
3. Examine the code that the Web Form Designer has generated for the **SqlConnection1** and **SqlCommand1** objects. In the next few modules you will learn in detail how this code works.
4. Verify that this code corresponds to the properties that you have set in the **Properties** window.

Caution Do not modify the generated code in any way. You will lose these modifications the next time you view the form in the Web Form Designer. You must use the Web Form Designer to make any changes to this code.

Exercise 2
Executing a Query and Displaying the Results

In this exercise, you will add a button and a **DataGrid** to your form.

When the user clicks the button, you will query the Northwind database by using the **SqlConnection** and **SqlCommand** objects that you created in Exercise 1. You will use a **SqlDataReader** object to copy the retrieved data into the **DataGrid**.

► To add a button and a DataGrid to the form

1. In the **Web Form Designer**, view your form.
2. In the **Toolbox**, select the **Web Forms** tab, and then drop a **Button** onto your form.
3. Set the following properties for the **Button** object.

Property	Value
(ID)	btnQuery
Text	Query

4. Drop a **DataGrid** object onto your form.
5. Set the following property for the **DataGrid** object.

Property	Value
(ID)	dgResult

► To execute a query and display the results in the DataGrid

1. In the **Web Form Designer**, double-click the button. This adds a method named **btnQuery_Click,** to handle the button-click event.
2. In the **btnQuery_Click** method, add the following Visual Basic or Visual C# code to open a connection to the Northwind database:

```
' Visual Basic
SqlConnection1.Open()

// Visual C#
sqlConnection1.Open();
```

3. Add the following code to query the database, and then create a **SqlDataReader** object to read the results:

   ```
   ' Visual Basic
   Dim Reader As System.Data.SqlClient.SqlDataReader
   Reader = SqlCommand1.ExecuteReader()

   // Visual C#
   System.Data.SqlClient.SqlDataReader Reader;
   Reader = sqlCommand1.ExecuteReader();
   ```

4. Add the following code to bind the **DataGrid** to the **SqlDataReader** object. The **SqlDataReader** object will iterate through the data, and the **DataGrid** will display the data on the screen.

   ```
   ' Visual Basic
   dgResult.DataSource = Reader
   dgResult.DataBind()

   // Visual C#
   dgResult.DataSource = Reader;
   dgResult.DataBind();
   ```

5. Add the following code to close the **SqlDataReader** and **SqlConnection** objects:

   ```
   ' Visual Basic
   Reader.Close()
   SqlConnection1.Close()

   // Visual C#
   Reader.Close();
   sqlConnection1.Close();
   ```

6. Build the project, and fix any compiler errors that might occur.
7. Run the project. When your Web form appears in Internet Explorer, click the **Query** button. A **DataGrid** appears in the Web form, showing the values of the ProductName and UnitPrice of every product in the Northwind database.

msdn® training

Module 2: Connecting to Data Sources (Prerelease)

Contents

Microsoft®

Instructor Notes

Presentation:
60 Minutes

Lab:
60 Minutes

This module explains the concepts and procedures necessary to create and manage a Microsoft® ADO .NET connection to Microsoft SQL Server™ or other data sources.

After completing this module, students will be able to:

- Build a DataSet and a DataTable.
- Bind a DataSet to a DataGrid.
- Create a custom DataSet by using inheritance.
- Define a data relationship.
- Modify data in a DataTable.
- Sort and filter a DataTable by using a DataView.

Required materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2389A_02.ppt
- Module 2, "Connecting to Data Sources"
- Lab 2, Connecting to Data Sources

Preparation tasks

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices and labs.
- Read the latest .NET Development news at http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000519

How to Teach This Module

This section contains information that will help you to teach this module.

Lesson: Choosing a .NET Data Provider

This section describes the instructional methods for teaching each topic in this lesson.

What Are .NET Data Providers?

Discussion Questions: Personalize discussion questions to the background of the students in your class.

Which .NET Data Provider Should You Use?

Discussion Questions: Personalize discussion questions to the background of the students in your class.

Lesson: Defining a Connection

This section describes the instructional methods for teaching each topic in this lesson.

Database Security

Discussion Questions: Personalize the following questions to the background of the students in your class.

- When would you use Windows authentication and when would you use Mixed Mode?

Setting a Connection String

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Practice Solution:

Here are the solutions to the practice above. Changes have been highlighted in bold.

Exercise 1

The name of the provider was wrong, and the database path is specified using the Data Source parameter, not the Initial Catalog:

```
Use OLE DB .NET Data Provider
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=\MyDB\MyDB.mdb;
```

Exercise 2

We should be using Windows Authentication, not SQL standard security:

```
Use SQL Server .NET Data Provider
Data Source=ProdServ01;Initial Catalog=Pubs;Integrated
Security=SSPI;
```

Exercise 3

The names of the SQL Server and database should be swapped:

```
Use SQL Server .NET Data Provider
Data Source=ProdServ01;Initial Catalog=Pubs;User
ID=JohnK;Password=JohnK;
```

Exercise 4

The SQL Server .NET Data Provider will not work with SQL Server 6.5; use the OLE DB .NET Data Provider instead:

```
Use OLE DB .NET Data Provider
Provider=SQLOLEDB;Data Source=ProdServ01;Initial
Catalog=Pubs;Integrated Security=True;
```

Exercise 5

This one was correct, no changes required:

```
Use SQL Server .NET Data Provider
Data Source=ProdServ02;Initial Catalog=Northwind;Integrated
Security=SSPI;
```

Exercise 6

We should be using Windows Authentication, not SQL standard security:

```
Use SQL Server .NET Data Provider
DataSource=ProdServ02;Initial Catalog=Pubs;Integrated
Security=SSPI;
```

Exercise 7

The connection timeout is measured in seconds, not minutes:

```
Use SQL Server .NET Data Provider
Data Source=ProdServ01;Initial Catalog=Pubs;Integrated
Security=True;Connection Timeout=60;
```

Exercise 8

This one was correct, because 15 seconds is the default timeout, so no changes required:

```
Use SQL Server .NET Data Provider
Data Source=ProdServ01;Initial Catalog=Pubs;Integrated
Security=True;
```

Exercise 9

Add an additional option to save the password in the connection string:

```
Use SQL Server .NET Data Provider
Data Source=ProdServ02;Initial Catalog=Pubs;User
ID=JohnK;Password=JohnK;Persist Security Info=True;
```

Lesson: Managing a Connection

This section describes the instructional methods for teaching each topic in this lesson.

Opening and Closing a Connection

Discussion Questions: Personalize discussion questions to the background of the students in your class.

Handling Connection Events

Discussion Questions: Personalize discussion questions to the background of the students in your class.

Instructor Demonstration: Programmatically demonstrate how to handle the StateChange event.

Lesson: Handling Exceptions

This section describes the instructional methods for teaching each topic in this lesson.

What is Structured Exception Handling?

Discussion Questions: Personalize discussion questions to the background of the students in your class.

How to Handle Multiple Types of Exceptions

Discussion Questions: Personalize discussion questions to the background of the students in your class.

Communicating Errors to Users

Discussion Questions: Personalize discussion questions to the background of the students in your class.

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Lesson: Connection Pooling

This section describes the instructional methods for teaching each topic in this lesson.

What is Connection Pooling?

Discussion Questions: Personalize discussion questions to the background of the students in your class.

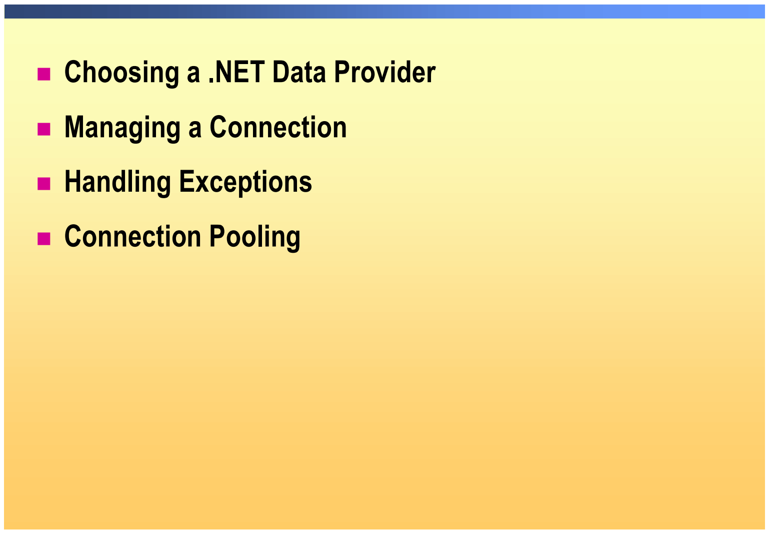
Controlling OLE DB Connection Pooling

Discussion Questions: Personalize discussion questions to the background of the students in your class.

Controlling SQL Server Connection Pooling

Discussion Questions: Personalize discussion questions to the background of the students in your class.

Overview



- Choosing a .NET Data Provider
- Managing a Connection
- Handling Exceptions
- Connection Pooling

Introduction

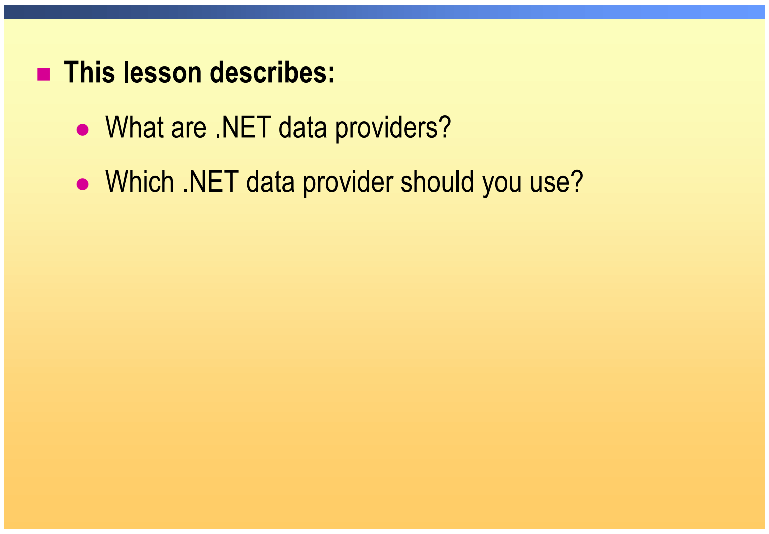
This module explains the concepts and procedures necessary to create and manage a Microsoft® ADO .NET connection to Microsoft SQL Server™ or other data sources.

Objectives

After completing this module, you will be able to:

- Choose a .NET data provider.
- Connect to SQL Server.
- Connect to OLE DB data sources.
- Manage a connection.
- Handle common connection exceptions.
- Implement and control connection pooling.

Lesson: Choosing a .NET Data Provider



- **This lesson describes:**
  - What are .NET data providers?
  - Which .NET data provider should you use?

Introduction

When connecting to a data source, you must first choose a .NET data provider. The data provider includes classes that enable you to connect to the data source, read data efficiently, modify and manipulate data, and update the data source.

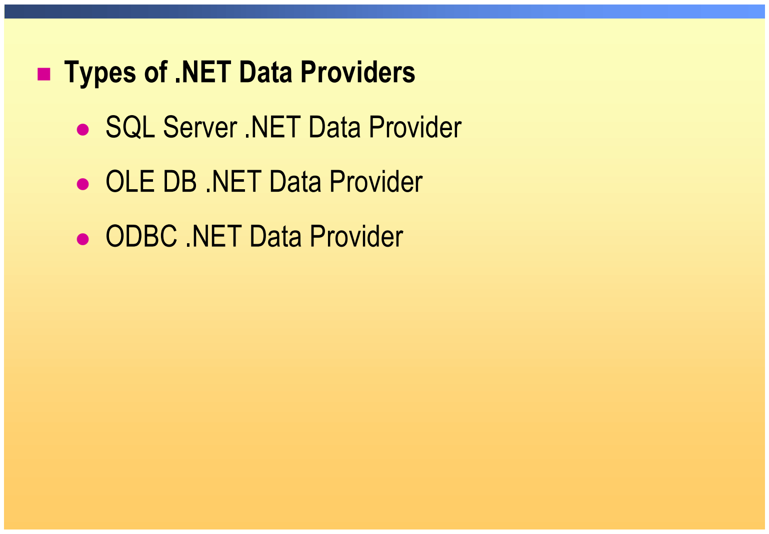
This lesson explains the various types of data providers and enables you to choose the appropriate provider for your application.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the different .NET data providers.
- Choose a data provider.

What Are .NET Data Providers?



- Types of .NET Data Providers
  - SQL Server .NET Data Provider
  - OLE DB .NET Data Provider
  - ODBC .NET Data Provider

Definition

The .NET data providers are a core component within the ADO .NET architecture that enable communication between a data source and a component, an XML Web service, or an application. A data provider allows you to connect to a data source, retrieve and manipulate data, and update the data source.

Types of .NET Data Providers

The following .NET data providers are included with the release of the .NET Framework:

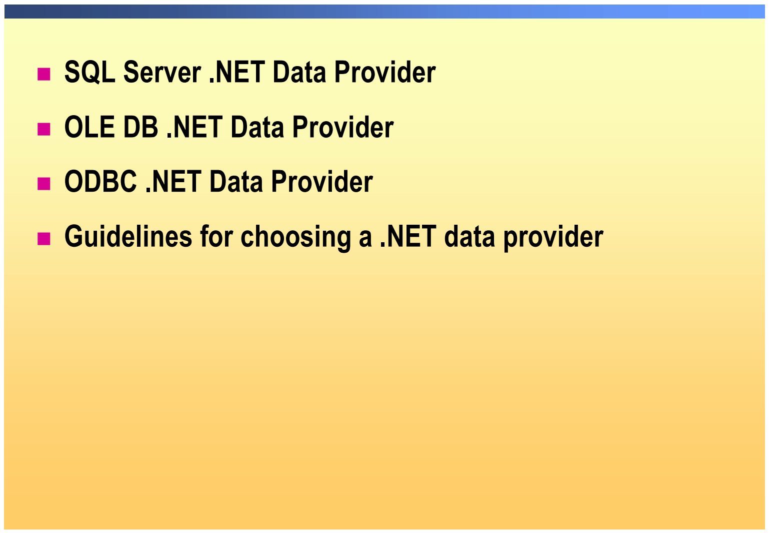
- SQL Server .NET Data Provider
- OLE DB .NET Data Provider

Other .NET data providers will be made available for other data sources. Microsoft will make the following provider available as a World Wide Web release download:

- Open Database Connectivity (ODBC) .NET Data Provider

Each of these data providers includes implementations of the generic ADO .NET classes so that you can programmatically communicate with different data sources in a similar way.

Which .NET Data Provider Should You Use?



- SQL Server .NET Data Provider
- OLE DB .NET Data Provider
- ODBC .NET Data Provider
- Guidelines for choosing a .NET data provider

Introduction

Choosing the appropriate .NET data provider for your application depends on the type of data source that is being accessed.

How To Reference a .NET Data Provider

Use the Visual Studio .NET Solution Explorer to manage references to assemblies that implement .NET data providers.

The **System.Data.dll** assembly (physically a single DLL file) implements the SQL Server .NET Data Provider and the OLE DB .NET Data Provider in the System.Data.SqlClient and System.Data.OleDb namespaces.

The **System.Data.Odbc.dll** assembly implements the ODBC .NET Data Provider. This assembly is not part of the Visual Studio .NET installation. You can download the assembly from the Microsoft web site. You can then manually reference it in your project to use the ODBC .NET Data Provider.

SQL Server .NET Data Provider

The SQL Server .NET Data Provider establishes a thin layer of communication between an application and Microsoft SQL Server. Because the SQL Server .NET Data Provider uses its own protocol, Tabular Data Stream (TDS), to communicate with SQL Server, it is lightweight and accesses SQL Server directly without any additional layers. This results in improved performance and scalability.

It is also recommended that you use the SQL Server .NET Data Provider for single-tier applications that use the Microsoft Data Engine (MSDE), because MSDE is based on the SQL Server engine.

OLE DB .NET Data Provider

The OLE DB .NET Data Provider uses native OLE DB and COM interoperability to connect and communicate with a data source. Because of this, you must use an OLE DB provider to use the OLE DB .NET Data Provider.

To use the OLE DB .NET Data Provider you must indicate the provider type in the connection string. The Provider keyword in a connection string indicates the type of OLE DB data source you will connect to; for example, “Provider=MSDAORA” to connect to an Oracle database. You do not need to include a Provider keyword in a connection string when using the SQL Server .NET Data Provider because the data source is assumed to be Microsoft SQL Server version 7.0 or later.

Data source	Example connection string parameters
SQL Server 6.5	Provider=SQLOLEDB;Data Source=London;Initial Catalog=pubs;User ID=sa;Password=2389;
Oracle Server	Provider=MSDAORA;Data Source=ORACLE8I7;User ID=OLEDB;Password=OLEDB;
Microsoft Access database	Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\bin\LocalAccess40.mdb;

Warning!

The OLE DB .NET Data Provider does not work with the OLE DB Provider for ODBC (MSDASQL). To access data sources by using ODBC, use the ODBC .NET Data Provider.

ODBC .NET Data Provider

The ODBC .NET Data Provider uses native ODBC application programming interface (API) calls to connect and communicate with a data source.

The ODBC .NET Data Provider has been implemented as a separate assembly called System.Data.Odbc.dll. It is not selected by default in project templates, and must be manually referenced.

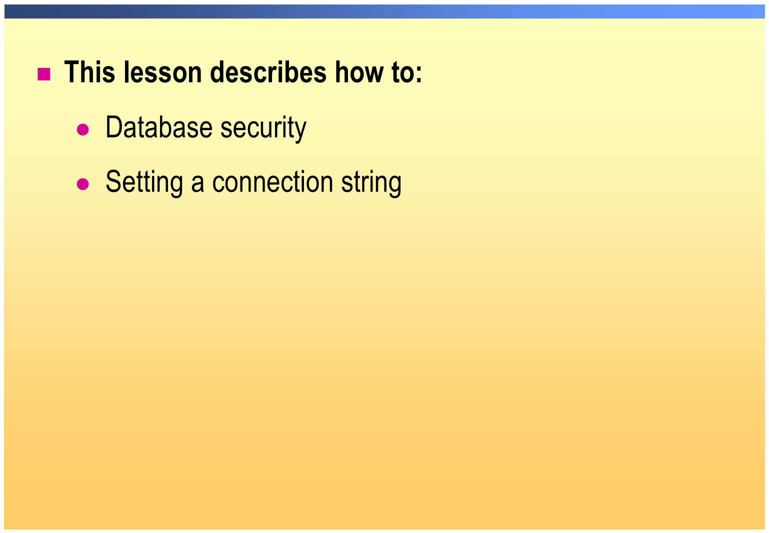
Data source	Provider/driver	Example connection string parameters
Oracle Server	ORA ODBC	Driver={Microsoft ODBC for Oracle}; Server=ORACLE8I7; UID=OLEDB; PWD=OLEDB;
Microsoft Access database	Jolt OLEDB	Driver={Microsoft Access Driver (*.mdb)}; DBQ=c:\bin\localaccess40.mdb;

Guidelines for Choosing a .NET Data Provider

The following table lists general guidelines for choosing a .NET data provider.

If your data source is	Then choose
Microsoft SQL Server 7.0 or Microsoft SQL Server 2000	SQL Server .NET Data Provider
Microsoft SQL Server version 6.5 or earlier	OLE DB .NET Data Provider
Any heterogeneous data source that can be accessed by using an OLE DB provider	OLE DB .NET Data Provider
Any heterogeneous data source that can be accessed by using an ODBC driver	ODBC .NET Data Provider

Lesson: Defining a Connection



- **This lesson describes how to:**
  - Database security
  - Setting a connection string

Introduction

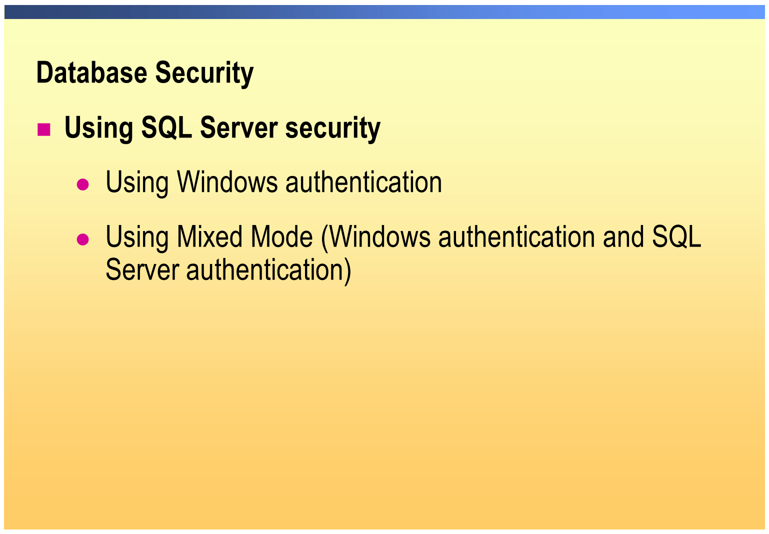
A *connection string* is an essential part of connecting to a data source. The **ConnectionString** property of a connection object provides information for that connection object. This lesson describes what a connection string is and how to use one.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe SQL Server database security options.
- Set a connection string property.

Database Security


Database Security

- **Using SQL Server security**
  - Using Windows authentication
  - Using Mixed Mode (Windows authentication and SQL Server authentication)

Introduction

When you build an application that accesses data by using ADO .NET, you will usually have to connect to secure databases. To do so, security information such as username and password will need to be passed to the database before a connection can be made. The database security that is available depends on the database that is accessed.

Using SQL Server Security

SQL Server can operate in one of two authentication modes: Windows authentication and Mixed Mode (Windows authentication and SQL Server authentication).

Using Windows Authentication

Windows authentication allows a user to connect through a Windows user account. Network security attributes for the user are established at network login time, and are validated by a Windows domain controller.

When a network user tries to connect, SQL Server verifies that the user is who they say they are, and then permits or denies login access based on that network user name alone, without requiring a separate login name and password.

Benefits of Using Windows Authentication

Windows authentication provides:

- Secure validation and encryption of passwords.
- Auditing.
- Password expiration.
- Minimum password length.
- Account lockout after multiple invalid login requests.

Warning!

Because Windows users and groups are maintained only by Windows, SQL Server reads information about a user's group membership when the user connects. If changes are made to the accessibility rights of a connected user, the changes become effective the next time the user connects to an instance of SQL Server or logs on to Windows (depending on the type of change).

Using Mixed Mode (Windows Authentication and SQL Server Authentication)

Mixed Mode authentication allows users to connect to an instance of SQL Server by using either Windows authentication or SQL Server authentication. Users who connect through a Windows NT 4.0 or Windows 2000 user account can use trusted connections in either Windows Authentication Mode or Mixed Mode.

When a user connects by using a specified login name and password from a non-trusted connection, SQL Server performs the authentication itself by checking to see if a SQL Server login account has been set up and if the specified password matches the one that was previously recorded. If SQL Server does not have a login account set, authentication fails and the user receives an error message.

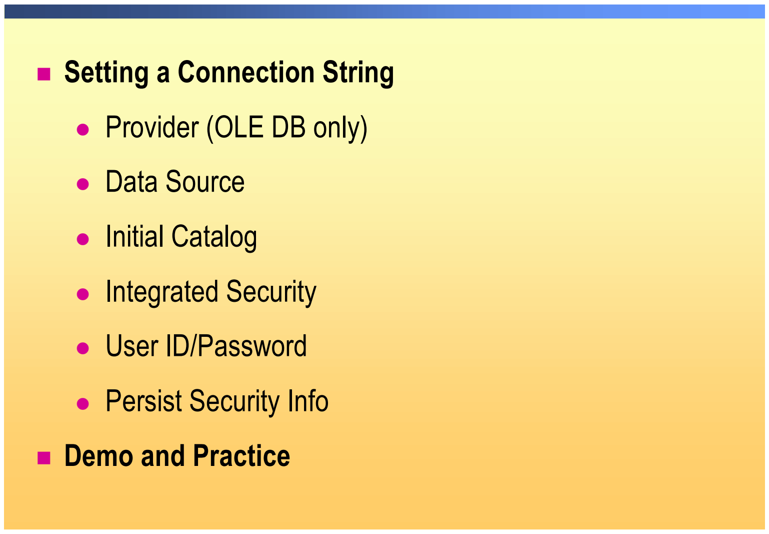
Warning!

If a user attempts to connect to an instance of SQL Server by providing a blank login name, SQL Server uses Windows authentication. Additionally, if a user attempts to connect to an instance of SQL Server configured for Windows authentication Mode by using a specific login, the login is ignored and Windows authentication is used.

Mixed Mode is primarily for backwards compatibility

SQL Server authentication is provided primarily for backward compatibility because applications written for SQL Server version 7.0 or earlier may require the use of SQL Server logins and passwords. Additionally, SQL Server authentication is required when an instance of SQL Server is running on Windows 98 because Windows Authentication Mode is not supported on Windows 98. Therefore, SQL Server uses Mixed Mode when running on Windows 98 (but supports only SQL Server authentication).

Setting a Connection String



- **Setting a Connection String**
  - Provider (OLE DB only)
  - Data Source
  - Initial Catalog
  - Integrated Security
  - User ID/Password
  - Persist Security Info
- **Demo and Practice**

Visual Basic Example C# Example

Introduction

To move data between a data store and your application, you must first have a connection to the data store. You can create and manage a connection by using one of the connection objects that ADO .NET makes available, including the **SqlConnection** object and the **OleDbConnection** object.

Note

You can set the **ConnectionString** property only when the connection is closed. To reset a connection string, you must close and reopen the connection.

Definition

The **ConnectionString** property provides the information that defines a connection to a data store.

Syntax

The values that you include in a connection string depend on which of the connection objects you use. The following table describes several common parameters of connection strings. The table contains only a partial list of the values, and not all of these are needed to establish a connection.

Parameter	Description
Provider	Use this property to set or return the name of the provider for the connection, used only for **OleDbConnection** objects.
Connection Timeout or Connect Timeout	Length of time in seconds to wait for a connection to the server before terminating the attempt and generating an error. **15** is the default.
Initial Catalog	The name of the database.
Data Source	The name of the SQL Server to be used when a connection is open.
Password	Logon password for the SQL Server account.
User ID	The SQL Server login account.
Integrated Security or Trusted Connection	Determines whether or not the connection is to be a secure connection. **True**, **False**, and **SSPI** are the possible values. (**SSPI** is the equivalent of **True**)
Persist Security Info	When set to **False**, security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Setting this property to **True** can be a security risk. **False** is the default.

Examples

The following examples show connection strings that contain commonly used properties. Note that not all connection strings contain the same properties.

The following is an example of connecting to a SQL Server 2000 database using a **SqlConnection** and Visual Basic.

Product	Microsoft SQL Server 2000
Server name	London
Database name	Northwind
Security	Mixed mode
Username	sa
Password	2389
Timeout	1 minute

```
Dim cnNorthwind as New _
  System.Data.SqlClient.SqlConnection()

cnNorthwind.ConnectionString = _
  "User ID=sa;" & _
  "Password=2389;" & _
  "Initial Catalog=Northwind;" & _
  "Data Source=London;" & _
  "Connection TimeOut=60;"
```

The following is an example of connecting to a Microsoft Access database using an **OleDbConnection** and Visual Basic.

Product	Microsoft Access 2000
Database location	\MyDB\MyDB.mdb

```
Dim cnNorthwind as New _
  System.Data.OleDb.OleDbConnection()

cnNorthwind.ConnectionString = _
  “Provider=Microsoft.Jet.OLEDB.4.0;” & _
  “Data Source=\MyDB\MyDB.mdb;”
```

The following is an example of connecting to a SQL Server 6.5 database using an **OleDbConnection** and Visual C#.

Product	Microsoft SQL Server 6.5
Server name	ProdServ01
Database name	Pubs
Security	Windows authentication

```
System.Data.OleDb.OleDbConnection cnNorthwind = new
  System.Data.OleDb.OleDbConnection();

cnNorthwind.ConnectionString =
  "Provider=SQLOLEDB;” +
  “Data Source=ProdServ01;” +
  “Initial Catalog=Pubs;” +
  “Integrated Security=SSPI;";
```

Demo

The easiest method of setting a connection string is to use the Visual Studio .NET development environment. Drag and drop a connection object from the Toolbox and use the Property Window to set the connection string.

Practice

For each of the following examples, determine if the .NET data provider and connection string that follow each example, are valid; if not, correct them.

Exercise 1

Product	Microsoft Access 2000
Database location	\MyDB\MyDB.mdb

```
Use OLE DB .NET Data Provider
Provider=Microsoft.Access;Initial Catalog=\MyDB\MyDB.mdb;
```

Exercise 2

Product	Microsoft SQL Server 2000
Server name	ProdServ01
Database name	Pubs
Security	Windows authentication

```
Use SQL Server .NET Data Provider
Data Source=ProdServ01;Initial Catalog=Pubs;User
ID=JohnK;Password=JohnK;
```

Exercise 3

Product	Microsoft SQL Server 2000
Server name	ProdServ01
Database name	Pubs
Security	Mixed Mode
Username	JohnK
Password	JohnK

```
Use SQL Server .NET Data Provider
Data Source=Pubs;Initial Catalog=ProdServ01;User
ID=JohnK;Password=JohnK;
```

Exercise 4

Product	Microsoft SQL Server 6.5
Server name	ProdServ01
Database name	Pubs
Security	Windows authentication

```
Use SQL Server .NET Data Provider
Data Source=ProdServ01;Initial Catalog=Pubs;Integrated
Security=True;
```

Exercise 5

Product	Microsoft SQL Server 7.0
Server name	ProdServ02
Database name	Northwind
Security	Windows authentication

```
Use SQL Server .NET Data Provider
Data Source=ProdServ02;Initial Catalog=Northwind;Integrated
Security=SSPI;
```

Exercise 6

Product	Microsoft SQL Server 7.0
Server name	ProdServ02
Database Name	Pubs
Security	Windows authentication

```
Use SQL Server .NET Data Provider
DataSource=ProdServ02;Initial Catalog=Pubs;User
ID=AmyJ;Password=AmyJ;
```

Exercise 7

Product	Microsoft SQL Server 2000
Server name	ProdServ01
Database name	Pubs
Security	Windows authentication
Timeout	1 minute

```
Use SQL Server .NET Data Provider
Data Source=ProdServ01;Initial Catalog=Pubs;Integrated
Security=True;Connection Timeout=1;
```

Exercise 8

Product	Microsoft SQL Server 2000
Server name	ProdServ01
Database name	Pubs
Security	Windows authentication
Timeout	15 seconds

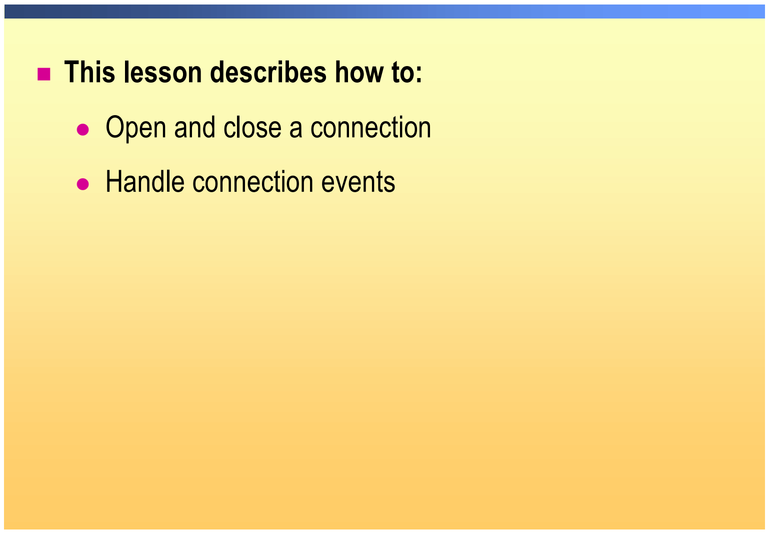
```
Use SQL Server .NET Data Provider
Data Source=ProdServ01;Initial Catalog=Pubs;Integrated
Security=True;
```

Exercise 9

Product	Microsoft SQL Server 2000
Server name	ProdServ02
Database name	Pubs
Security	Mixed Mode
Username	JohnK
Password	JohnK (visible if connection string is read)

```
Use SQL Server .NET Data Provider
Data Source=ProdServ02;Initial Catalog=Pubs;User
ID=JohnK;Password=JohnK;
```

Lesson: Managing a Connection



- **This lesson describes how to:**
  - Open and close a connection
  - Handle connection events

Introduction

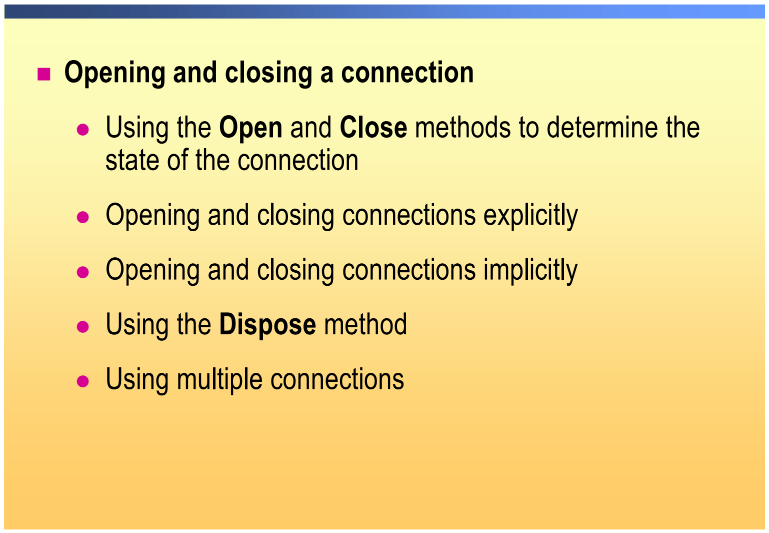
Once you have defined the **ConnectionString** property of a connection object you use the Open and Close methods to manage the connections current state. This lesson describes how to use these methods, and how to respond to connection events.

Lesson Objectives

After completing this lesson, you will be able to:

- Open and close a connection.
- Handle connection events (such as **StateChange**, **InfoMessage**).

Opening and Closing a Connection



- **Opening and closing a connection**
  - Using the **Open** and **Close** methods to determine the state of the connection
  - Opening and closing connections explicitly
  - Opening and closing connections implicitly
  - Using the **Dispose** method
  - Using multiple connections

C# Example

Introduction

You can open and close connections either implicitly, by calling methods on an object that uses the connection, or explicitly, by calling the **Open** and **Close** methods.

The two primary methods for connections are **Open** and **Close**.

- The **Open** method uses the information in the **ConnectionString** property to contact the data source and establish an open connection.
- The **Close** method shuts down the connection.

Closing connections is essential, because most data sources support only a limited number of open connections, and open connections take up valuable system resources.

Opening and Closing Connections Explicitly

Opening and closing connections explicitly is the recommended approach, because it:

- Makes for cleaner, more readable code.
- Helps you debug.
- Is more efficient.

You can open a database connection by using the **Open** method, and with the values specified by the **ConnectionString**. Conversely, you can close a database connection by using the **Close** method.

Opening and Closing Connections Implicitly

If you work with data adapters, you do not have to explicitly open and close a connection. When you call a method of these objects (for example, the DataAdapter's **Fill** or **Update** method), the method checks whether the connection is already open. If not, the DataAdapter opens the connection, performs its logic, and then closes the connection.

Best Practice

If you are filling multiple tables in a DataSet from the same database you will have multiple data adapters, one for each table, but only one connection. When filling the connection will open and close multiple times if you use connections implicitly. It is better to explicitly open the connection, call the Fill methods of the multiple data adapters, and then explicitly close the connection.

Opening a Connection

To open a connection, you use the **Open** method on the connection object you are working with, **OleDbConnection** or **SqlConnection**. The **Open** method opens a database connection with the property settings that you have specified in the **ConnectionString**.

Closing a Connection

You must always close the connection when you have finished using it. To do this, you can use either the **Close** or **Dispose** methods of the connection object. Connections are not released implicitly when the connection object falls out of scope or is reclaimed by garbage collection.

The **Close** method rolls back any pending transaction. It then closes the connection, or releases the connection to the connection pool if pooling is enabled. An application can call the **Close** method more than one time.

Using the Dispose Method

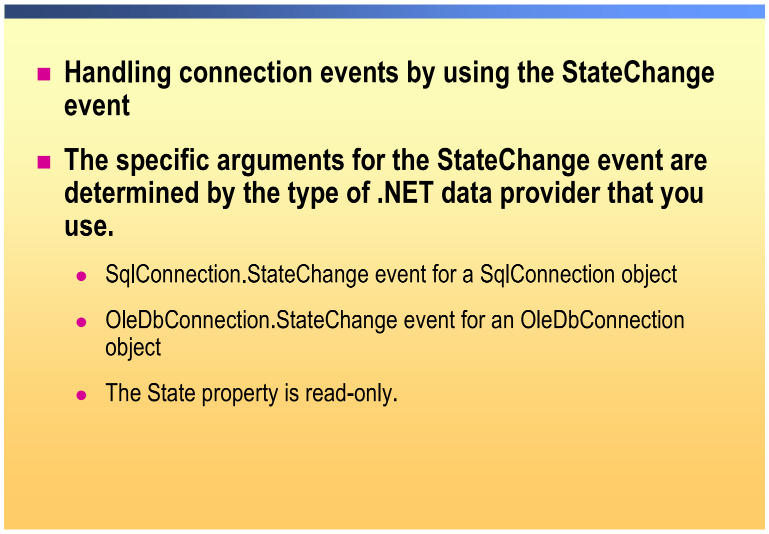
When you close a connection, the flow to and from the data source closes, but unmanaged resources used by the connection object have not been released. Both the **SqlConnection** object and the **OleDbConnection** object have a **Dispose** method to release the unmanaged resources.

Example of Using the Dispose Method

The following Visual Basic example shows how to create a **SqlConnection** object, open the connection with the **Open** method, and then release the resources used by the connection by calling the **Dispose** method and then setting the object to **Nothing**.

```
Dim cnNorthwind As New _
  System.Data.SqlClient.SqlConnection()

cnNorthwind.ConnectionString = _
  “Data Source=(local);” & _
  ”Initial Catalog=Northwind;” & _
  “Integrated Security=SSPI;”

cnNorthwind.Open()

' perform some database task

cnNorthwind.Close()

cnNorthwind.Dispose()

cnNorthwind = Nothing
```

Handling Connection Events



- **Handling connection events by using the StateChange event**
- **The specific arguments for the StateChange event are determined by the type of .NET data provider that you use.**
  - SqlConnection.StateChange event for a SqlConnection object
  - OleDbConnection.StateChange event for an OleDbConnection object
  - The State property is read-only.

Visual Basic Example

Introduction

The **StateChange** event occurs whenever the connection state changes; from closed to open or from open to closed. To handle any event, you must have an *event handler*, which is a method with a signature defined by the class that defines the event you want to handle. Different events have slightly different event handlers. The event handler for the **StateChange** event is a method that must have an argument of the type **StateChangeEventArgs**. This argument contains data related to this event.

The StateChange Event

The specific arguments for the **StateChange** event are determined by the type of .NET data provider that you use.

- **SqlConnection.StateChange** event for a **SqlConnection** object
- **OleDbConnection.StateChange** event for an **OleDbConnection** object

Remember that all events in the .NET Framework have two parameters.

- **sender** (of type Object)
- **e** (of type XxxEventArgs)

For the StateChange event, e is of type **StateChangeEventArgs**.

Definition

The event handlers for the two events receive the same type of argument, **StateChangeEventArgs**, that contains data related to this event. The following table describes the properties of the **StateChangeEventArgs** class.

Property	Description
CurrentState	Gets the new state of the connection. The connection object will already be in the new state when the event is fired.
OriginalState	Gets the original state of the connection.

Example of a StateChange Event Handler

The following example shows code for creating a **StateChangeEventHandler** delegate using Visual Basic and Visual C#. Note the different ways of handling events in Visual Basic and Visual C#.

```
' Visual Basic

Private Sub cnNorthwind_StateChange( _
  ByVal sender As Object, _
  ByVal e As System.Data.StateChangeEventArgs _
  ) Handles cnNorthwind.StateChange

  ' Display current and original state
  ' in a message box whenever
  ' the connection state changes

  MessageBox.Show( _
    "CurrentState: " & e.CurrentState.ToString & vbCrLf & _
    "OriginalState: " & e.OriginalState.ToString, _
    "cnNorthwind.StateChange", _
    MessageBoxButtons.OK, _
    MessageBoxIcon.Information)

End Sub

// Visual C#

// the following code is usually added to the constructor
// for the class so that the function is linked to the
// appropriate event

this.cnNorthwind.StateChange += new
  System.Data.StateChangeEventHandler(
  this.cnNorthwind_StateChange);


private void cnNorthwind_StateChange(
  object sender,
  System.Data.StateChangeEventArgs e)
{
  MessageBox.Show(
    "CurrentState: " & e.CurrentState.ToString +
    "OriginalState: " & e.OriginalState.ToString,
    "cnNorthwind.StateChange",
    MessageBoxButtons.OK,
    MessageBoxIcon.Information);
}
```

Demo

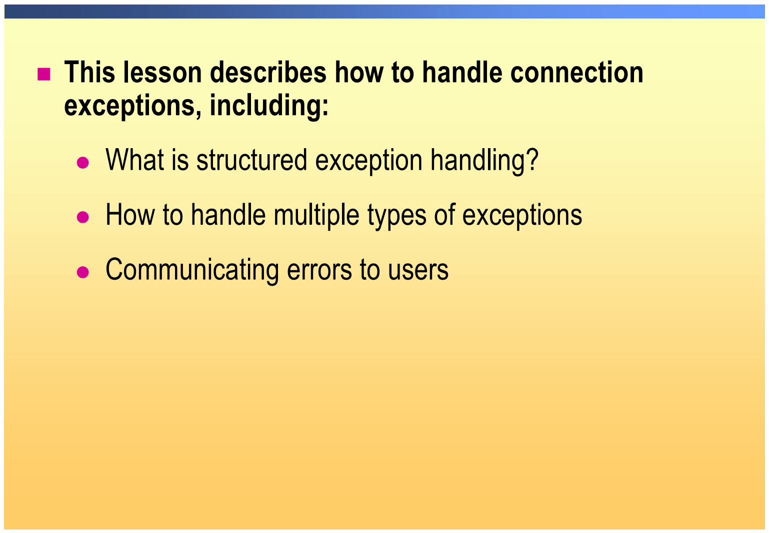
Create a new Windows application using Visual Basic that has two buttons that open and close a **SqlConnection** to the Northwind database. Disable the close button initially.

Write code to handle the **StateChange** event by checking the current state, and enabling or disabling the two buttons appropriately.

Notice the environment allows the event to be picked from the drop down lists in the Code Editor.

Repeat the demonstration using Visual C#. Note the event handling code must be written manually.

Lesson: Handling Exceptions



- **This lesson describes how to handle connection exceptions, including:**
  - What is structured exception handling?
  - How to handle multiple types of exceptions
  - Communicating errors to users

Introduction

This lesson describes how to handle connection exceptions in a .NET environment. Visual Basic programmers will find that exception handling in the .NET environment is different from previous versions of Visual Basic. The familiar **On Error** style error handling is supported but not recommended for the .NET Framework.

In the past, each language had a different method of handling errors. All .NET languages use the Common Language Runtime, and must be able to interact closely with each other. This means that the languages that use the .NET Framework need to support the new, standardized style of error handling, called *structured exception handling*.

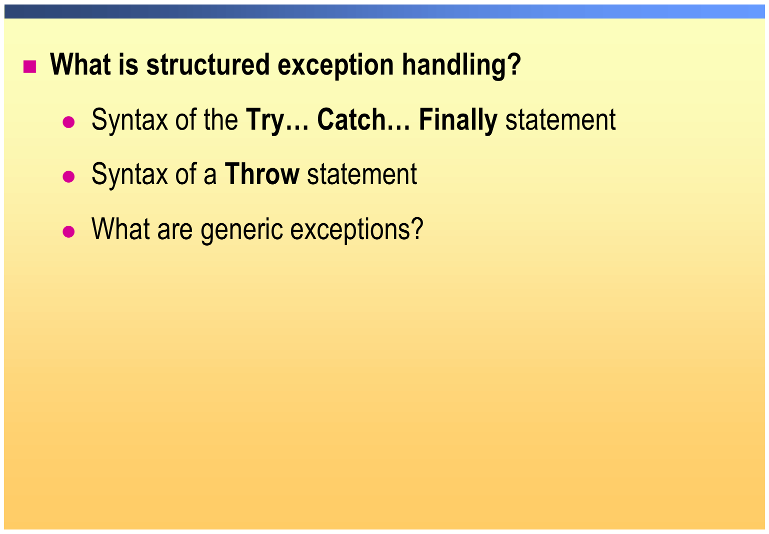
Structured exception handling provides more specific information when an error occurs. A class can define its own custom exceptions with additional information that is specific to the task the class was written to perform, such as the name of a stored procedure, a line number or an error, or a Server name. You can create your own exception classes by deriving classes from the appropriate base exception.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe what structured exception handling is.
- Handle multiple types of exceptions.
- Handle connection exceptions.

What Is Structured Exception Handling?



- **What is structured exception handling?**
  - Syntax of the **Try… Catch… Finally** statement
  - Syntax of a **Throw** statement
  - What are generic exceptions?

Visual Basic Syntax Visual Basic Example

Introduction

An exception is an error condition or unexpected behavior that is encountered by an executing program from the program itself or from the run-time environment.

Earlier versions of Visual Basic had only one error, with a limited amount of information about the error; therefore you had to check the number property on the **Err** object to determine what problem had occurred. You would typically write a **Select Case** statement to branch to different chunks of code in order to deal with those specific errors. Today, instead of having one error object, you have the option of handling more specific errors, now known as *exceptions*.

Exception handling in ADO .NET is similar to a game of catch: code in the application throws an exception, and exception handling in your code catches it.

Definition of Structured Exception Handling

Structured exception handling is code designed to detect and respond to errors during program execution by combining a control structure with exceptions, protected blocks of code, and filters. In the .NET Framework, the control structure is the **Try…Catch…Finally** statement. If an error occurs in the **Try** block, code in the **Catch** block handles the error.

Syntax for the Try…Catch…Finally Statement

The following example shows the syntax for the **Try…Catch…Finally** statement in Visual Basic. One **Try** block can have many **Catch** blocks.

```
Try
   [ tryStatements ]
[Catch1 [ exception1 [ As type1 ] ] [ When expression1 ]
   catchStatements1
[Exit Try]
Catch2 [ exception2 [ As type2 ] ] [When expression2 ]
   catchStatements2
[ Exit Try ]
...
Catchn [ exceptionn [ As typen ] ] [ When expressionn ]
   catchStatementsn ]
[ Exit Try ]
[ Finally
   [ finallyStatements ] ]
End Try
```

Visual Basic Example

```
Dim cnNorthwind As System.Data.SqlClient.SqlConnection

Try

  cnNorthwind = New System.Data.SqlClient.SqlConnection()

  cnNorthwind.ConnectionString = _
      "Data Source=(local);" & _
      "Initial Catalog=Northwind;" & _
      "Integrated Security=SSPI;"

  cnNorthwind.Open()

  ' perform some database task

Catch XcpNullRef As System.NullReferenceException

  MessageBox.Show("Failed to create connection object")

Catch Xcp As System.Exception

  MessageBox.Show(Xcp.ToString())

Finally

  cnNorthwind.Close()

  cnNorthwind.Dispose()

  cnNorthwind = Nothing

End Try
```

Definition of a Throw Condition

A program indicates that an exception condition has occurred by executing a **Throw** statement. Exceptions thrown by the **Throw** statement can be caught in a **Try** statement. A **Throw** statement creates an exception within a procedure.

Syntax of a Throw Statement

```
Throw New Exception
```

Example of a Throw Statement

The following Visual Basic example shows a **Throw** statement:

```
Throw New FileNotFoundException( _
  "data.txt not in c:\dev directory", e)
```

What Are Generic Exceptions?

The **System.Exception** class applies to all exceptions in the .NET environment. The properties that all exceptions share include the following:

- **Message**. This is a read-only property that you can use to display a description of the cause of the exception in a message box.
- **InnerException**. This read-only property refers to the cause of the current exception. When **InnerException** is a non-null value (non-Nothing value in Visual Basic), this property refers to the exception that ultimately caused the current exception.
- **HelpLink**. This property gets or sets a link to a Help file that you associate with the current exception. The return value is a Uniform Resource Name (URN) or Uniform Resource Locator (URL) that you associate with the Help file.

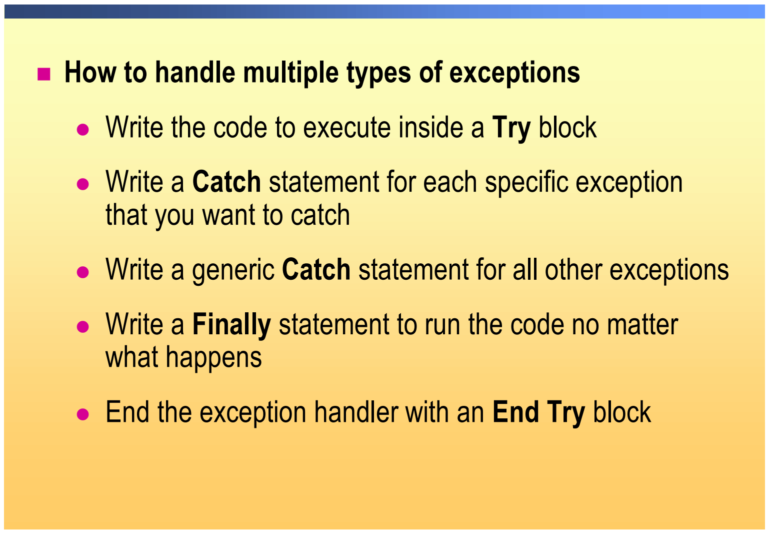
Demo

Create a new Windows application using Visual Basic that has two buttons that open and close a SqlConnection to the Northwind database. Add exception handling code.

Practice

The Visual Studio .NET documentation lists the specific exceptions that can occur for a specific method call. Look up the exceptions that can occur when calling the **Open** and **ChangeDatabase** methods of the **System.Data.SqlClient.SqlConnection** class.

How to Handle Multiple Types of Exceptions



- **How to handle multiple types of exceptions**
  - Write the code to execute inside a **Try** block
  - Write a **Catch** statement for each specific exception that you want to catch
  - Write a generic **Catch** statement for all other exceptions
  - Write a **Finally** statement to run the code no matter what happens
  - End the exception handler with an **End Try** block

Visual Basic Example

Introduction

When you write an exception handler that uses a **Try…Catch…Finally** block, you can use as many **Catch** blocks as you feel are necessary. Write a **Finally** block to run the code unconditionally.

For multiple exceptions, you must start with the most specific exceptions and then proceed to the least specific, which is the **System.Exception** class, the generic exception.

Scenario

You want to call a method on one of your objects, such as opening a file. Although opening a file is one method call, multiple problems can occur when the call is made. You cannot assume that there will be only one problem. You must be able to handle multiple exceptions.

Procedure

The exception handler in the following procedure handles a series of exceptions that might occur when code in an application tries to open a text file. These are low-level errors that a user can fix and then try again to execute the file-opening code. The exceptions are in the **Catch** blocks that are provided by the .NET Framework.

► **To handle multiple types of exceptions**

1. Write the code to execute inside a **Try** block.

```
' This code simply opens a text file and gets its size

Try
  s = File.Open(txtFileName.Text, FileMode.Open)
  lngSize = s.Length
  s.Close()
```

2. Write a **Catch** statement for each exception that you want to catch.

```
' These statements specify different exceptions that might
' need to be caught and provides an action for each
' exception. Each statement creates an exception object
that ' can provide specific information.

Catch e As ArgumentException
  MessageBox.Show( _
    “You specified an invalid file name.”)
Catch e As FileNotFoundException
  MessageBox.Show( _
    “The file you specified can’t be found.”)
Catch e As ArgumentNullException
  MessageBox.Show( _
    “You passed in a null argument.”)
Catch e As AccessException
  MessageBox.Show( _
    “You specified a folder name.”)
Catch e As DirectoryNotFoundException
  MessageBox.Show( _
    “You specified a folder that does not exist.”)
Catch e As SecurityException
  MessageBox.Show( _
    “You don’t have sufficient access.”)
Catch e As IOException
  MessageBox.Show( _
    “The drive you selected is not ready.”)
```

3. Write a generic **Catch** statement for all other exceptions. If in a class that will be called, then throw the exception up to the calling code. Otherwise, if your code is part of the presentation tier, display a warning message to the end user.

```
' This statement handles all non-specific exceptions
Catch e As System.Exception

' For a Windows application you might use a MessageBox...
MessageBox.Show("An unknown error occurred.")

' ...for a Web application you might write to the page...
Response.Write("An unknown error occurred.")

' ...for an exception within a custom class you would Throw
' the exception up to the calling code to handle.
Throw e
```

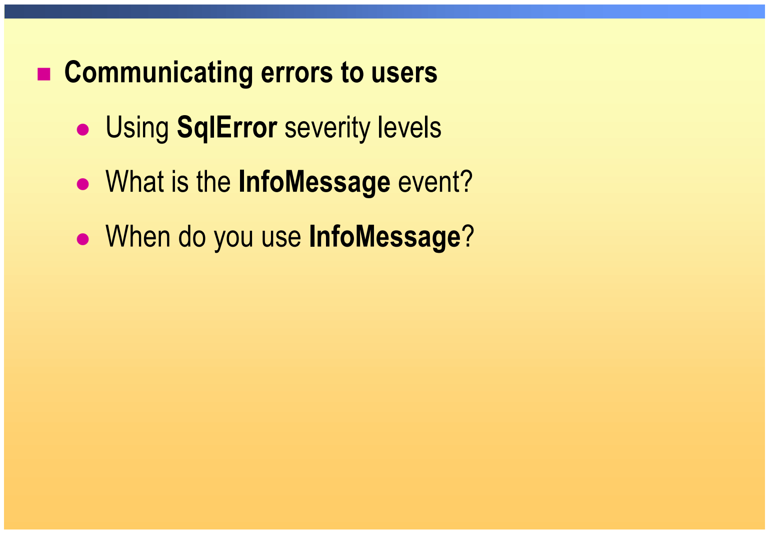
4. Write a **Finally** block after the **Catch** blocks.

```
Finally
  ' Run this code no matter what else happens
  s = Nothing
```

5. End the **Try** block.

```
' This statement ends the Try block
End Try
```

Communicating Errors to Users



- **Communicating errors to users**
  - Using **SqlError** severity levels
  - What is the **InfoMessage** event?
  - When do you use **InfoMessage**?

Introduction

Many different problems can occur when you work with databases. Because some problems are more serious than others, you can apply severity levels to them. For less serious problems, exceptions might not even be thrown. Instead, an event might be triggered. For example, the **InfoMessage** event of the **SqlConnection** object is triggered when the database server needs to communicate potentially important information to the user.

The SqlException Class

The **SqlException** class contains the exception that is thrown when SQL Server returns a warning or error. This class is created whenever the SQL Server .NET Data Provider encounters a situation that it cannot handle. The class always contains at least one instance of **SqlError**. You can use the severity level to help determine the content of a message that an exception displays.

The following table describes severity levels of the **SqlException** class.

Severity	Description	Action
1-10	Informational, indicating problems caused by mistakes in information that a user has entered	SqlConnection remains open, so you can continue working.
11-16	Generated by user	Can be corrected by user.
17-19	Software or hardware errors	You can continue working, but might not be able to execute a particular statement. SqlConnection remains open.
20-25	Software or hardware errors	Server closes SqlConnection. User can reopen connection.

What Is The InfoMessage Event?

There are **InfoMessage** events for both the **SqlConnection** and the **OleDbConnection** classes. These events occur when the provider sends a warning or an information message.

When Do You Use InfoMessage?

Use the **InfoMessage** event when you add an information message to an error-handling routine. When an **InfoMessage** event is triggered, an argument containing a **SqlErrorCollection** is passed to the event handler. The handler can then loop through all of the **SqlError** objects and retrieve detailed information about the messages.

Practice

The solution for this practice is located here
<*install folder*>\Practices\Mod02\Lesson4\HandlingInfoMessage\

Add code to handle the **InfoMessage** event of the connection, and use a message box to display a description and the class (severity level) of each **SqlError** contained in the InfoMessage.

The Northwind Traders IT Director would like all applications developed by his team to provide better feedback to enable the Help Desk staff to track issues. In a previous lesson you learned how to write code to handle the **StateChange** event. In this practice you will write code to handle the **InfoMessage** event.

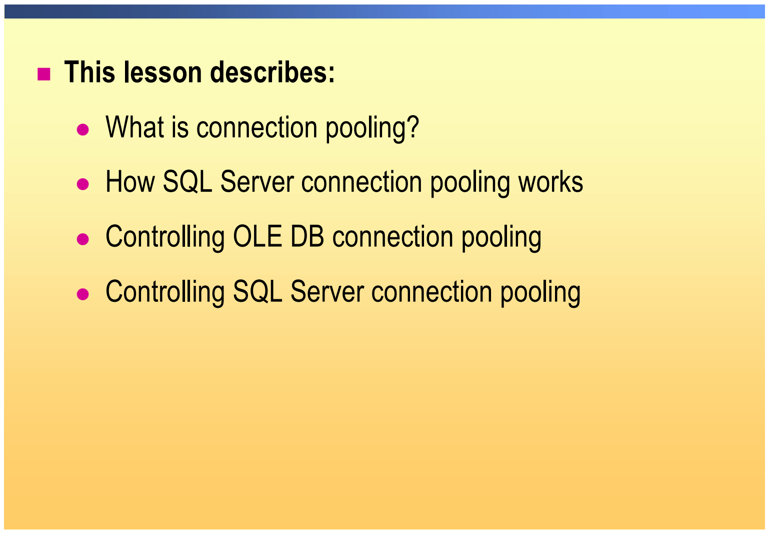
1. Create a new Windows Application solution using Visual Studio .NET named **HandlingInfoMessage**.
2. Add two button controls from the Toolbox, with captions **Open** and **Close**, and name them **btnOpen** and **btnClose**.
3. Add a **SqlConnection** control from the Toolbox and name it **cnNorthwind**.
4. Set the **ConnectionString** property of **cnNorthwind** to connect to the Northwind database in your local SQL Server using integrated security.
5. Add code to the **Open** button that opens the connection.
6. Add code to the **Close** button that closes and disposes of the connection.
7. Add code to handle the **InfoMessage** event by looping through all the **SqlError** objects in the **Errors** collection of the **SqlInfoMessageEventArgs** object and showing the message.

```
' Visual Basic
Dim se As SqlClient.SqlError
For Each se In e.Errors
   MessageBox.Show(se.Message, "InfoMessage", _
       MessageBoxButtons.OK, MessageBoxIcon.Information)
Next

// Visual C#
foreach (SqlClient.SqlError se in e.Errors)
{
   MessageBox.Show(se.Message, "InfoMessage", _
       MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

8. Run and test the application.

How many SqlErrors are generated when a connection is made to a SQL Server database?

Lesson: Connection Pooling



- **This lesson describes:**
  - What is connection pooling?
  - How SQL Server connection pooling works
  - Controlling OLE DB connection pooling
  - Controlling SQL Server connection pooling

Introduction

Each time you establish a connection to a data source, cycles and memory are used. Because applications often require multiple connections with multiple users, connecting to a data source can be resource-intensive. By pooling connections, you can keep connections available for reuse, which enhances application performance and scalability.

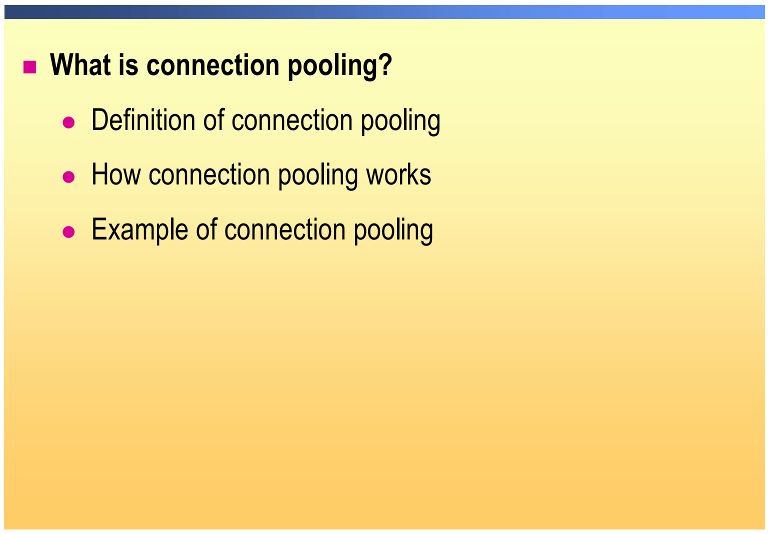
This lesson describes how to implement connection pooling with SQL Server and OLE DB data sources in ADO .NET.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe what connection pooling is and how it works.
- Control OLE DB connection pooling.
- Control SQL Server connection pooling.

What Is Connection Pooling?



- **What is connection pooling?**
  - Definition of connection pooling
  - How connection pooling works
  - Example of connection pooling

Visual Basic Example

Definition

Connection pooling is the process of keeping connections active and pooled so that they can be efficiently reused. Connections with identical connection strings are pooled together and can be reused without reestablishing the connection.

A connection pool is created when one or more connections share a unique connection string, and connection-pooling functionality has been requested. If any one parameter of the new connection string is not identical to the first string, the new connection will be placed in its own pool.

Applications often require many of the same connections for different purposes. By pooling these connections and reusing them:

- Application performance is improved.
- Scalability is enhanced.

How Connection Pooling Works

Connection pooling is the ability of SQL Server or an OLE DB data source to reuse connections for a particular user or security context.

Connection pooling occurs on the computer where the database is installed. When the database is installed, memory is allocated for its processes, including connections. The ADO .NET connection string specifies different parameters for connection pooling.

When the connection to the data source is attempted, the security context and the Pooling parameter are examined. If the Pooling parameter is set to ‘false’, then connection pooling will not occur. However, if pooling is enabled (it is set to ‘true’ by default), connection pooling will occur.

A security context is a unique combination of parameter values in the connection string. The security context is checked to see if it is valid and if it is identical to other connection strings. If two connections have the same connection string parameters, then those connections have the same security context. Connections that have the same security context are pooled together. If any part of a connection string is not identical to another connection string, a new pool is created.

When a connection is closed, the connection is returned to the pool and ready for reuse. When a connection is released, the connection is deleted from memory. It is not returned to the pool and is not available for reuse. When the last connection in a connection pool is deleted, the pool is also deleted from memory.

Example of Connection Pooling

Out of the three connection strings below, the first two connection strings match exactly, and because connection pooling is enabled by default, they would be pooled together. For the third connection string, the initial catalog (the default database that is accessed upon connection) is different than the initial catalog in the first two connection strings. A separate pool would be created for the third connection, and any connections identical to it would be added to that pool.

Connection 1

```
Dim myConnection as New SqlClient.SqlConnection()
myConnection.ConnectionString = "User ID=sa;” & _
  “Password=me2I81sour2;” & _
  ”Initial Catalog=Northwind;” & _
  “Data Source=mySQLServer;” & _
  “Connection TimeOut=30;"
```

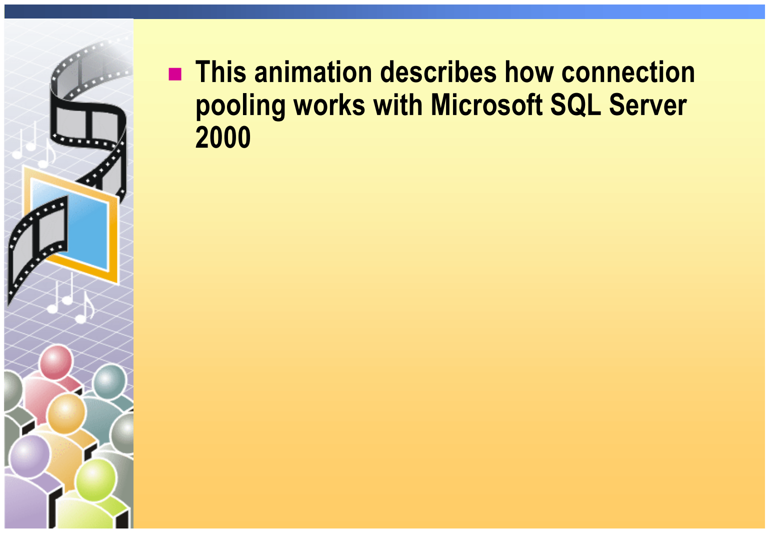
Connection 2

```
Dim myConnection as New SqlClient.SqlConnection()
myConnection.ConnectionString = "User ID=sa;” & _
  “Password=me2I81sour2;” & _
  ”Initial Catalog=Northwind;” & _
  “Data Source=mySQLServer;” & _
  “Connection TimeOut=30;"
```

Connection 3

```
Dim myConnection as New SqlClient.SqlConnection()
myConnection.ConnectionString = "User ID=sa;” & _
  “Password=me2I81sour2;” & _
  ”Initial Catalog=Pubs;” & _
  “Data Source=mySQLServer;” & _
  “Connection TimeOut=30;"
```

Multimedia: How SQL Server Connection Pooling Works

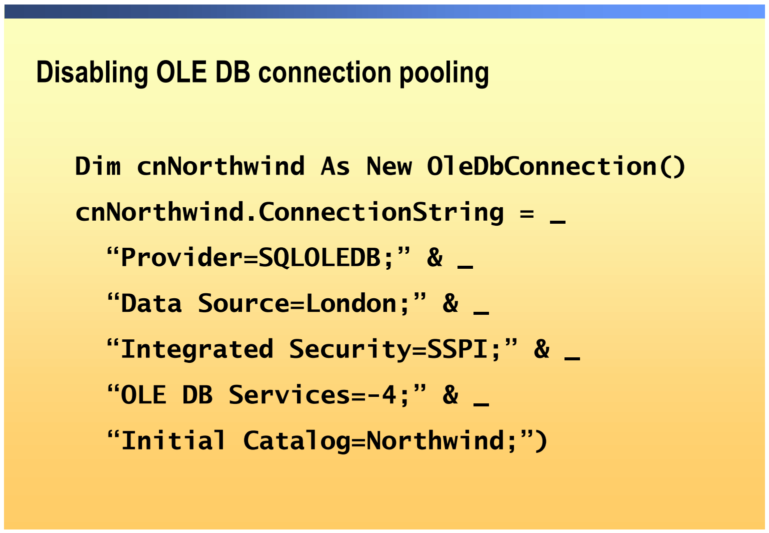


- This animation describes how connection pooling works with Microsoft SQL Server 2000

Introduction

This animation describes connection pooling and shows how connection pooling works with Microsoft SQL Server 2000. Understanding connection pooling will help you to plan, design, and deploy your ADO .NET applications for enhanced performance, security, and scalability.

Controlling OLE DB Connection Pooling



Disabling OLE DB connection pooling

```
Dim cnNorthwind As New OleDbConnection()
cnNorthwind.ConnectionString = _
  “Provider=SQLOLEDB;” & _
  “Data Source=London;” & _
  “Integrated Security=SSPI;” & _
  “OLE DB Services=-4;” & _
  “Initial Catalog=Northwind;”)
```

Introduction

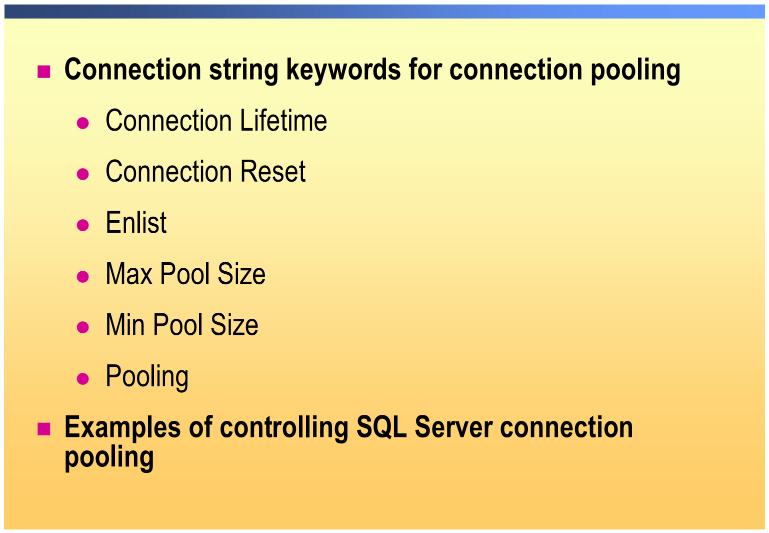
When you connect to an OLE DB data source, connection pooling is automatic. The OLE DB .NET Data Provider uses OLE DB session pooling by default.

You can control OLE DB connection pooling from your application by:

- Disabling pooling in individual connection strings.
- If you write directly to the OLE DB API, controlling pooling through the properties you set when connecting to the database.

To disable pooling in a connection string that uses the OLE DB .NET Data Provider, specify “OLE DB Services=-4” in your connection string.

Controlling SQL Server Connection Pooling



- **Connection string keywords for connection pooling**
  - Connection Lifetime
  - Connection Reset
  - Enlist
  - Max Pool Size
  - Min Pool Size
  - Pooling
- **Examples of controlling SQL Server connection pooling**

Visual Basic Example

Introduction

You can control several elements of connection pooling when you connect to SQL Server by using the SQL Server .NET Data Provider. You can control how long the connection exists, whether the database connection is reset when the connection is removed from the pool, how many connections are allowed and maintained in the pool, and whether pooling is enabled.

Connection String Keywords

The following keywords are specified in the connection string in order to control connection pooling.

Connection String Keyword	Default	Description
Connection Lifetime	0	When a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by connection lifetime. This is useful in clustered configurations to force load balancing between a running server and a server that was just brought online.

(*continued*)

Connection String Keyword	Default	Description
Connection Reset	True	Determines whether the database connection is reset when the connection is removed from the pool. Setting this to False prevents an additional server round-trip when obtaining a connection, but you must be aware that the connection state, such as database context, is not being reset. This default option makes the connection change back to its original database context automatically when it is reused. This costs an extra (potentially unnecessary) call to the server. If database contexts will not be changed, set this parameter to **False**. Use the **ChangeDatabase** method rather than the SQL USE command to enable ADO .NET to automatically reset connections when they are returned to the pool.
Enlist	True	When set to True, the pooler automatically enlists the connection in the current transaction context of the creation thread if a transaction context exists.
Max Pool Size	100	The maximum number of connections allowed in the pool.
Min Pool Size	0	The minimum number of connections maintained in the pool.
Pooling	True	When set to **True**, the SqlConnection object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool.

Examples of Controlling SQL Server Connection Pooling

The following code samples show how you can control connection pooling by using the parameters of a SQL Server connection string.

To disable connection pooling:

```
Dim cnNorthwind as New SqlClient.SqlConnection()

cnNorthwind.ConnectionString = "User ID=sa;” & _
  “Password=me2I81sour2;” & _
  “Initial Catalog=Northwind;” & _
  “Data Source=mySQLServer;” & _
  “Connection TimeOut=30;” & _
  “Pooling=False;”
```

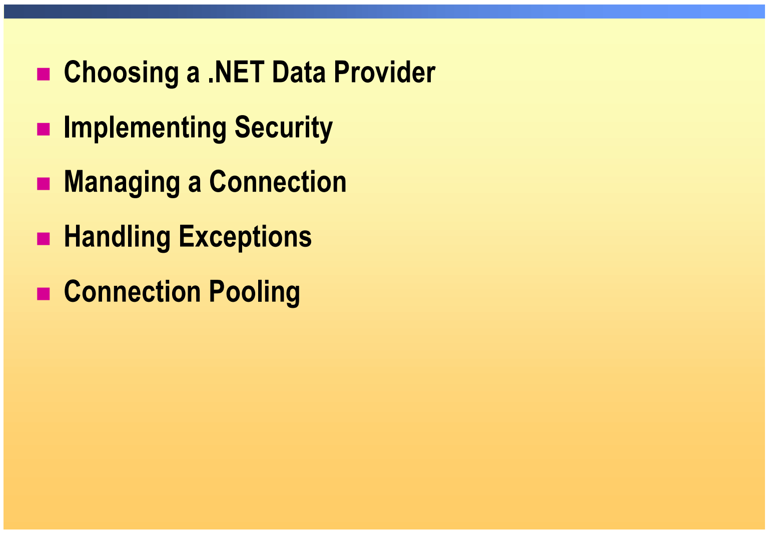
To specify the minimum pool size:

```
cnNorthwind.ConnectionString = "User ID=sa;” & _
  “Password=me2I81sour2;” & _
  “Initial Catalog=Northwind;” & _
  “Data Source=mySQLServer;” & _
  “Connection TimeOut=30;” & _
  “Min Pool Size=5;"
```

To specify the connection lifetime:

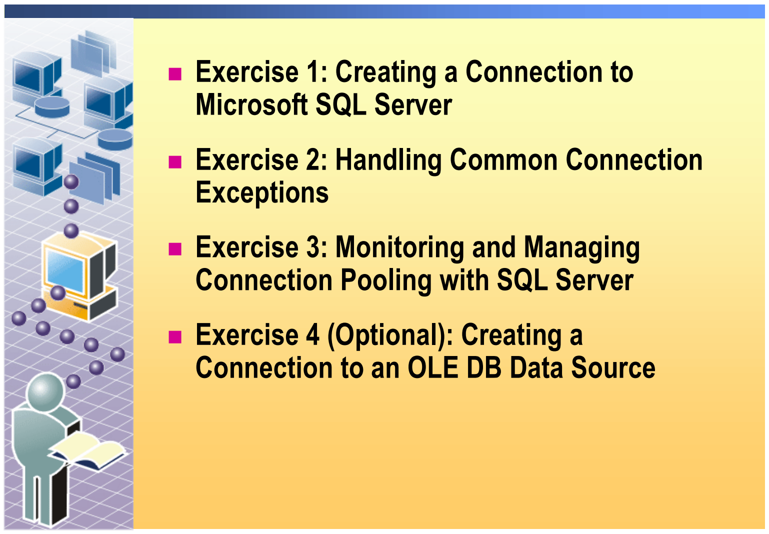
```
cnNorthwind.ConnectionString = "User ID=sa;” & _
  “Password=me2I81sour2;” & _
  “Initial Catalog=Northwind;” & _
  “Data Source=mySQLServer;” & _
  “Connection TimeOut=30;” & _
  “Connection Lifetime=120;"
```

Review: Connecting to Data Sources



- Choosing a .NET Data Provider
- Implementing Security
- Managing a Connection
- Handling Exceptions
- Connection Pooling

Lab2 : Connecting to Data Sources



- Exercise 1: Creating a Connection to Microsoft SQL Server
- Exercise 2: Handling Common Connection Exceptions
- Exercise 3: Monitoring and Managing Connection Pooling with SQL Server
- Exercise 4 (Optional): Creating a Connection to an OLE DB Data Source

Objectives

After completing this lab, you will be able to:

- Create a connection to an OLE DB data source.
- Create a connection to SQL Server.
- Handle common connection exceptions.
- Monitor and manage connection pooling.

Scenario

Northwind Traders, an international company that imports and exports specialty food, stores data in many different locations and different data sources. In order to build needed applications, the company needs to establish connections to the different data sources.

This lab allows you to practice connecting to multiple types of data sources and practice handling common problems, by using the Northwind Traders sample databases that are included with Microsoft SQL Server 2000.

Estimated time to complete this lab: 60 minutes

Exercise 0
Lab Setup

To complete this lab, you must:

► **To create SQL Server login accounts**

1. Open and execute <install folder>\Labs\Lab02\createusers.sql by using SQL Server Query Analyzer. The instructor will demonstrate if you do not know how.Use **SQL Server Enterprise Manager** to confirm that the following two SQL Server login accounts were created.
3. Look in the **Security – Logins** folder to find the accounts. You may need to refresh the folder.

Account name	**Password**	**Default database**
JohnK	JohnK	Northwind
AmyJ	AmyJ	Northwind

Exercise 1
Creating a Connection to Microsoft SQL Server

Microsoft Visual Studio .NET includes a native .NET data provider for Microsoft SQL Server versions 7.0 and later. This native provider gives better performance and functionality when connecting to SQL Server than using the OLE DB .NET Data Provider to connect to SQL Server.

Scenario

Northwind Traders needs to build an application that accesses customer and sales data. The company uses a variety of different data sources, including SQL Server 2000. When building a data-related application, you begin by connecting to the data source.

In this exercise, you will create a connection to read from a SQL Server 2000 database by using the SQL Server .NET Data Provider within a Windows application. The connections that you make will be used in future exercises.

► **To open an existing Visual Studio .NET solution**

The path is <install folder>\Labs\Lab02\Starter\VB\ConnectingToDataSources.sln or
<install folder>\Labs\Lab02\Starter\CS\ConnectingToDataSources.sln

► **To declare a new connection object**

1. Declare a new connection object that can respond to events by using the SQL Server .NET Data Provider.

Variable attributes	Values
Name	cnSQLNorthwind

► **To create a connection string**

In the **Exercises 1 and 2** group box, create a connection string on the **Open** button. You will be adding exception handling in Exercise 3. It is not necessary to write any exception handling during this exercise.

1. Build the connection string by using the information provided in the controls in the **Exercises 1 and 2** group box.

Connection parameter	Control and property
Data Source	txtServer.Text
Initial Catalog	txtSQLDatabase.Text
Connection Timeout	txtTimeout.Text
Integrated Security	chkIntegratedSecurity.Checked
User ID	txtUsername.Text
Password	txtPassword.Text

2. Write code to open the connection.

► To close the connection

- Write code to dispose of the connection by adding code to the **Close** button.

► To trace the application events

1. Start Microsoft SQL Profiler.
2. Create a new trace by using the following information.

Connect to SQL Server	Values
SQL Server	. *(Local)*
Authentication	Windows

Events	Values
Security Audit	Audit Login
	Audit Logout
	Audit Login Failed
Sessions	Existing Connection

Data columns	Values
Columns	EventClass
	TextData
	ApplicationName
	LoginName
	LoginSID
	SPID
	StartTime

3. Run the trace.
4. Run the **ConnectingToDataSources** solution.
5. When the solution starts, enter **2389** as the password for the **sa** login account, and then click the **Open** button in the **Exercise 2** group box.
6. Switch to SQL Profiler and verify that a successful Audit Login occurred.
7. Switch to the running solution and click the **Close** button.
8. Switch to SQL Profiler and verify that a successful Audit Logout occurred.
9. Switch to the running solution. In the **Exercises 1 and 2** group box, enter an invalid database name in the **Database** text box, and then click the **Open** button.
10. Switch to SQL Profiler and verify that an Audit Login Failed occurred.
11. Stop the SQL Profiler.

► **To explore connection timeouts**

1. Run the **ConnectingToDataSources** solution within the Visual Studio .NET environment.
2. When the **ConnectingToDataSources** solution starts, change the server name to Alpha, and then click the **Open** button in the **Exercises 1 and 2** group box.

   How long do you have to wait before an error is displayed?

   15 seconds.

3. Click the **Exit** button to stop the application.
4. Modify the connection string to time out after 5 seconds.
5. Repeat steps 1 through 4.

   How long do you have to wait before an error is displayed?

   Optional: test the application's ability to log on to a SQL Server using Integrated Security.

   5 seconds.

Exercise 2
Handling Common Connection Exceptions

There are many situations in which errors can occur, including:

- An invalid connection string; for example:
  - Wrong or missing database name or location
  - Wrong or missing security information
- Network problems (for example, slow or down)
- Server problems (for example, overloaded, over license limit, or unavailable)

Scenario

Although Northwind Traders uses a robust Windows networking environment, there could be situations where unexpected errors occur. Northwind Traders is a rapidly growing company, and increased network usage may impact network latency. To handle these and other situations, you must implement exception handling in the Northwind Traders data application.

In this exercise, you will handle common errors that can occur when connecting to a data source.

► **To open an existing Visual Studio .NET solution**

- The path is <install folder>\Labs\Lab02\Solution\Ex1\VB\ConnectingToDataSources.sln or
  <install folder>\Labs\Lab02\Solution\Ex1\CS\ConnectingToDataSources.sln

► **To code the Open and Close buttons**

- In the **Exercises 1 and 2** group box, add code to the **Open** and **Close** buttons (for the SQL Server connection) to handle the following common exceptions.

Action	Exception	Result
Attempt to close connection before the connection has been opened	System. NullReferenceException	Show a message box with a message that tells the user that he or she must open the connection first.
Attempt to open connection with an invalid connection string	System. Data. SqlClient. SqlException	Check the exception number and display a message as follows: 17: invalid server name 18452: invalid user name 18456: invalid password 4060: invalid database
Unexpected problem when attempting to open or close the connection	System. Exception	Show a message box with a message that includes a description of the exception.

► **To handle the InfoMessage event**

Handle the **InfoMessage** event for the **SqlConnection** by displaying a message box for each error that occurs.

1. Add an event handler for the **InfoMessage** event on **SqlConnection**.
2. Declare a variable of the type **SqlError** to act as an enumerator for the **Errors** collection of the **SqlInfoMessageEventArgs** object.
3. Write a For Each loop to enumerate through the members of the **Errors** collection.
4. Inside the loop, use a message box to show the information contained in each **SqlError** object.

► **To test the exception and event handling code**

1. Run the **ConnectingToDataSources** solution within the Visual Studio .NET environment.
2. When the Lab02 solution starts, attempt to close connection before the connection has been opened.

   Does your code handle the error?
3. Enter valid connection information and attempt to open the connection.

   Do you receive InfoMessage events?
4. Change the server name to Alpha, and then click the **Open** button in the **Exercises 1 and 2** group box.

   Does your code handle the error?
5. Enter other invalid connection information like an invalid username or password.

   Does your exception handling code differentiate between error types?

Exercise 3
Monitoring and Managing Connection Pooling with SQL Server

Because applications often require multiple connections with multiple users, connecting to a data source can be resource-intensive. By pooling connections, you can keep connections available for reuse, which will enhance application performance and scalability.

Scenario

The number of employees at Northwind Traders is rapidly growing, and as a result more applications are connecting to data sources. The company needs to optimize server resources by pooling connections.

In this exercise, you will monitor SQL Server connection pooling and modify connections based on the connection string settings and security context.

► **To examine the Pooling Monitor application**

1. Open <install folder>\Labs\Lab02\Pooling Monitor\PoolingMonitor.vbproj with Visual Studio .NET.
2. Examine **LaunchConnection.vb**. Note that the **btnNew_Click** procedure creates a new instance of the **OpenNewConnection** class, and sets the title bar of the new form with a supplied string.
3. Examine **OpenNewConnection.vb**. Note the following points:
4. The **btnOpenSQL_Click** procedure builds a connection string and attempts to connect to SQL Server.
5. The **btnCloseSQL_Click** procedure closes the current connection but does not release resources.
6. The **btnRelease_Click** procedure releases connection resources and closes the current window.
7. Compile the application. Note the path of the executable e.g. <install folder>\Labs\Lab02\PoolingMonitor\bin\PoolingMonitor.exe.
8. Create a shortcut to the executable on the desktop.

► To create a SQL Profiler trace to monitor connection activity

1. Start SQL Profiler.
2. Create a new trace by using the following information.

Connect to SQL Server	Values
SQL Server	. *(Local)*
Authentication	Windows

Events	Values
Security Audit	Audit Login
	Audit Logout
Sessions	Existing Connection

Data columns	Values
Columns	EventClass
	TextData
	ApplicationName
	LoginName
	ClientProcessID
	SPID
	Start Time

3. Run the trace.

► To examine non-pooled connections

In this procedure, you will examine connection activity by using SQL Profiler.

Important You must run the PoolingMonitor.exe program executable rather than running the program from within Visual Studio.

1. Run **PoolingMonitor.exe** by using the desktop shortcut you created previously.
2. Create a connection by using the following information. Use the default value for any unspecified information.

Parameter	Value
Connection Name	JohnK1
Database	Northwind
User Name	JohnK
Password	JohnK
Enable Pooling	Cleared

3. Examine the activity in SQL Profiler.

   What is the SPID (server process identifier) of this connection?

   Answers will vary.

4. Leave the **JohnK1** connection form running and switch back to the **Launch Connection** form.
5. Create another connection by using the following information. Use the default value for any unspecified information.

Parameter	Value
Connection Name	JohnK2
User Name	JohnK
Password	JohnK

6. Examine the activity in SQL Profiler.

   What is the SPID of this connection?

   Answers will vary.
7. Leave the **JohnK2** connection form running and switch back to the **Launch Connection** form.
8. Create another connection by using the following information. Use the default value for any unspecified information.

Parameter	Value
Connection Name	AmyJ1
User Name	AmyJ
Password	AmyJ

9. Examine the activity in SQL Profiler.

   What is the SPID of this connection?

   Answers will vary.

   How is SQL Server responding to requests for new connections? Why?

   SQL Server creates a new connection for each request. Pooling is disabled through the connection string.
10. Close and release the **JohnK1**, **JohnK2**, and **AmyJ1** connections.
11. Examine the trace.
12. Close PoolingMonitor.exe and stop the trace.

► To examine pooling by setting the Pooling Parameter

In this procedure, you will examine pooling behavior when specifying **Pooling** in the connection string. Examine the trace after each step.

1. Start the trace.
2. Run **Pooling monitor.exe** by using the desktop shortcut you created previously.
3. Create a connection by using the following information. Use the default value for any unspecified information.

Parameter	Value
Connection Name	JohnK1
User Name	JohnK
Password	JohnK
Enable Pooling	Checked

   How many connections are created for this user? Why?

   Two connections are created. When pooling is enabled and the min pool size is 0, SQL Server creates a connection in the pool in order to optimize performance for the next request.
4. Close and release the **JohnK1** connection.
5. Create a new connection named **JohnK2** by using the same set of information.

   How many connections are created for this user? Why?

   No new connections are created. A connection is retrieved from the pool.
6. Close and release the **JohnK2** connection.

   What activity does the trace show? Why?

   No activity on SQL Server. A connection is released to the pool.
7. Close the Pooling Monitor application.

   What activity does the trace show? Why?

   All connections are released when the client application is closed.
8. Stop the trace.

► To manage pooling by using security context

In this procedure, you will examine pooling behavior when specifying various security contexts in the connection string. Examine the trace after each step.

1. Start the trace.
2. Run **Pooling monitor.exe** by using the desktop shortcut you created previously.
3. Create a connection by using the following information. Use the default value for any unspecified information.

Parameter	Value
Connection Name	JohnK1
Database	Pubs
User Name	JohnK
Password	JohnK
Enable Pooling	Checked

   How many connections are created for this user? Why?

   Two connections are created because pooling is enabled.

4. Create a new connection named **JohnK2** by using the following information.

Parameter	Value
Connection Name	JohnK2
Database	Northwind
User Name	JohnK
Password	JohnK
Enable Pooling	Checked

   How many connections are created for this user? Why?

   Two additional connections in a new pool are created. Because the database parameter is different, JohnK cannot use a connection from the first pool.

5. Close and release the **JohnK1** and **JohnK2** connections.
6. Close the Pooling Monitor application.
7. Stop the trace.

► To manage the pool size

In this procedure, you will examine pooling behavior when specifying **Pool Size** in the connection string. Examine the trace after each step.

1. Start the trace.
2. Run **Pooling monitor.exe** by using the desktop shortcut you created previously.
3. Create a connection by using the following information. Use the default value for any unspecified information.

Parameter	Value
Connection Name	JohnK1
User Name	JohnK
Password	JohnK
Min Pool Size	5
Enable Pooling	Checked

   How many connections are created for this user? Why?

   SQL Server creates five connections. Minimum pool size is 5.
4. Create a new connection named **JohnK2** by using the same set of information.

   How many connections are created for this user? Why?

   SQL Server does not create any additional connections. A connection is retrieved from the pool.
5. Close and release the **JohnK1** and **JohnK2** connections.

   What activity does the trace show? Why?

   No activity. Connections are released to the pool.
6. Close the Pooling Monitor application.

   What activity does the trace show? Why?

   All connections are released.
7. Restart the Pooling Monitor application.

8. Create a connection by using the following information. Use the default value for any unspecified information.

Parameter	**Value**
Connection Name	JohnK1
User Name	JohnK
Password	JohnK
Max Pool Size	3
Enable Pooling	Checked

9. Create three more connections named **JohnK2**, **JohnK3**, and **JohnK4** by using the same set of information.

   What happens when the fourth connection is attempted? Why?

10. Close the Pooling Monitor application.
11. Stop the trace.

Exercise 4 (optional)
Creating a Connection to an OLE DB Data Source

Scenario

Northwind Traders uses a variety of different data sources, including Microsoft SQL Server 6.5, Microsoft Access, and Microsoft Excel.

In this exercise, you will create an OLE DB connection to read from an Access database by using the OLE DB .NET Data Provider within a Microsoft Windows® application.

► To open an existing Microsoft Visual Studio .NET solution

The path is <install folder>\Labs\Lab02\Starter\VB\
ConnectingToDataSources.sln or
<install folder>\Labs\Lab02\Starter\CS\
ConnectingToDataSources.sln

- Open the class module **Form1** and review the code that is provided.

  Note that the **Integrated Security** check box and the **Exit** button have existing code that handles their click events. You will write code to complete the functionality for this application.

► To declare a new connection object

- Use the OLE DB .NET Data Provider to declare a new connection object that can respond to events.

Variable attributes	Values
Name	cnOleDbNorthwind

► To create the new connection

Create a connection on the **Open** button in the **Exercise 4** group box.

1. Add code to handle the **Click** event of the **Open** button.
2. Build the connection string using the values in the following text boxes.

Connection parameter	Text box
Provider	txtProvider
Data Source	txtOleDbDatabase

3. Write code to open the connection.

► To close the connection

1. Close the connection by adding code to the **Close** button in the **Exercise 4** group box.
2. Write code to dispose the connection.

► **To handle the StateChange event for the connection**

1. Add an event handler for the **StateChange** event for the **cnOleDbNorthwind** connection object.
2. Use a message box to show the current and original state of the connection.

► **To run and test the connection**

1. Run the **ConnectingToDataSources** solution.
2. Click the **Open** button.

   Is a successful connection made?

   Yes.
3. Click the **Close** button.

   Is the connection closed?

   Yes.
4. Click the **Close** button again.

   What happens?

   An exception occurs.

   Why?

   The OleDbConnection object has been released from memory and therefore cannot respond to the Close method call.
5. Click the **Open** button twice.

   What happens?

   No exceptions occur.

   Why?

   The first connection goes out of scope and is removed from memory automatically when the cnOleDbConnection variable is pointed to the second object instance.

► **To test for exceptions**

1. In Windows Explorer, move the database file to a different location (the initial database path is \Program Files\Microsoft Office\Office10\Samples\Northwind.mdb). Or, to save time, change the path specified in the text box on the form of the solution.
2. Run the **ConnectingToDataSources** solution.
3. Click the **Open** button.

   Does an exception occur?

   Yes.

   Why?

   The Data Source parameter of the connection string is referencing a missing file.
4. In Windows Explorer, move the database file back to its original location.

5. Use Microsoft Access to open the database file exclusively.

   Hint: The **Open** dialog box in Microsoft Access has an **Open** button with multiple options if you click the drop-down arrow part of the button.

6. Run the **ConnectingToDataSources** solution.
7. Click the **Open** button.

   Does an exception occur?

   Yes.

   Why?

   Although the Data Source parameter of the connection string references an existing file, the file has already been opened exclusively by another application.

8. Close Microsoft Access.

► **To add exception handling**

- In the Exercise 4 group box, add exception handling to the **Open** and **Close** buttons (for example, for the OLE DB .NET Data Provider).

  Use the .NET Framework documentation to find the list of exceptions that occur for the OLE DB .NET Data Provider.

MICROSOFT
TRAINING
AND CERTIFICATION

Microsoft Official Curriculum

Module 3: Performing Connected Database Operations (Prerelease)

Contents

Microsoft®

Instructor Notes

Presentation:
60 Minutes

Lab:
60 Minutes

In many situations, you will design your data-access strategy around using a DataSet. In other situations, however, you might find it useful or necessary to bypass DataSets and use data commands to communicate directly with the data source (typically, a database). This module teaches student how to use ADO .NET command objects to access a database.

After completing this module, students will be able to:

- Build a command object.
- Execute a command that returns a single value.
- Execute a command that returns a set of rows, and process the result.
- Execute a command that returns multiple results, and process the results.
- Execute a command that defines data by using the data definition language (DDL).
- Execute a command that modifies data by using the data manipulation language (DML).

Required materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2389A_03.ppt
- Module 3, "Performing Connected Database Operations"
- Lab 3, Performing Connected Database Operations

Preparation tasks

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices and labs.
- Read the latest .NET Development news at http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000519

How to Teach This Module

This section contains information that will help you to teach this module.

Lesson: Building Command Objects

This section describes the instructional methods for teaching each topic in this lesson.

What is a Command Object?

Discussion Questions: Personalize questions to the background of the students in your class.

How to Create a Stored Procedure

Discussion Questions: Personalize questions to the background of the students in your class.

Instructor Demonstration

How to Create a Command Object

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

What are Command Parameters?

Discussion Questions: Personalize questions to the background of the students in your class.

How to Create Parameters for a Command Object

Discussion Questions: Personalize questions to the background of the students in your class.

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Lesson: Executing Command Objects That Return a Single Value

This section describes the instructional methods for teaching each topic in this lesson.

Why Return a Single Value in a Command?

Discussion Questions: Personalize questions to the background of the students in your class.

How to Execute a Command That Returns a Single Value

Discussion Questions: Personalize questions to the background of the students in your class.

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

How to Retrieve Output and Return Values

Discussion Questions: Personalize questions to the background of the students in your class.

Lesson: Executing Commands That Return Rows

This section describes the instructional methods for teaching each topic in this lesson.

Returning Rows

Discussion Questions: Personalize questions to the background of the students in your class.

DataReader Properties and Methods

Discussion Questions: Personalize questions to the background of the students in your class.

How to Use the DataReader to Process Rows

Discussion Questions: Personalize questions to the background of the students in your class.

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

■

Lesson: Executing Multiple SQL Statements

This section describes the instructional methods for teaching each topic in this lesson.

Why Use Multiple SQL Statements?

Discussion Questions: Personalize questions to the background of the students in your class.

How to Process Multiple SQL Statements

Discussion Questions: Personalize questions to the background of the students in your class.

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Lesson: Using Data Definition Language

This section describes the instructional methods for teaching each topic in this lesson.

What is Data Definition Language?

Discussion Questions: Personalize questions to the background of the students in your class.

How to Execute a Data Definition Language Command

Discussion Questions: Personalize questions to the background of the students in your class.

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Lesson: Manipulating Data Using Data Manipulation Language

This section describes the instructional methods for teaching each topic in this lesson.

What is Data Manipulation Language?

Discussion Questions: Personalize questions to the background of the students in your class.

How to Execute a Data Manipulation Language Command

Discussion Questions: Personalize questions to the background of the students in your class.

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

■

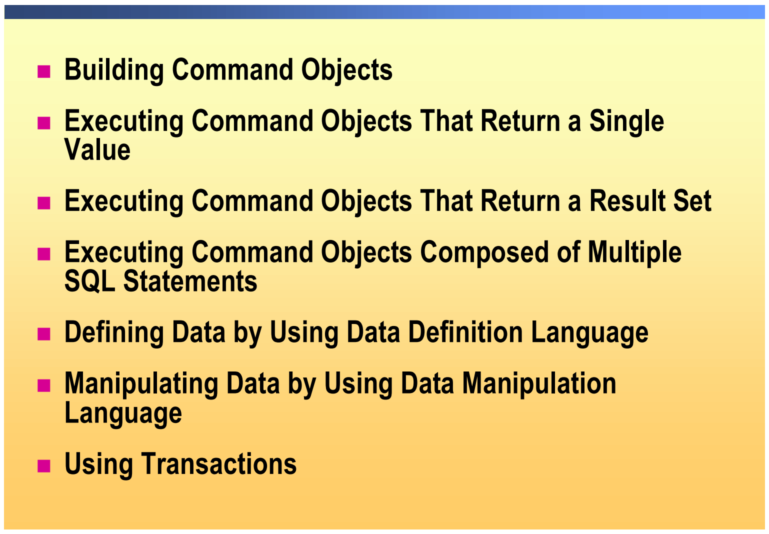
Troubleshooting Data Modification

Discussion Questions: Personalize questions to the background of the students in your class.

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Overview



- Building Command Objects
- Executing Command Objects That Return a Single Value
- Executing Command Objects That Return a Result Set
- Executing Command Objects Composed of Multiple SQL Statements
- Defining Data by Using Data Definition Language
- Manipulating Data by Using Data Manipulation Language
- Using Transactions

Introduction

In many situations, you will design your data-access strategy around using a DataSet. In other situations, however, you might find it useful or necessary to bypass DataSets and use data commands to communicate directly with the data source (typically, a database). These situations include:

- Performing queries on data intended to be read-only in your application. This might include executing a command that performs a database lookup.
- Designing data access in an Active Server Page (ASP) .NET Web application that only requires a single pass through data, such as displaying the results of a search.
- Executing a query that returns a single value, such as a calculation or the result of an aggregate function.
- Creating and modifying database structures, such as tables and stored procedures.

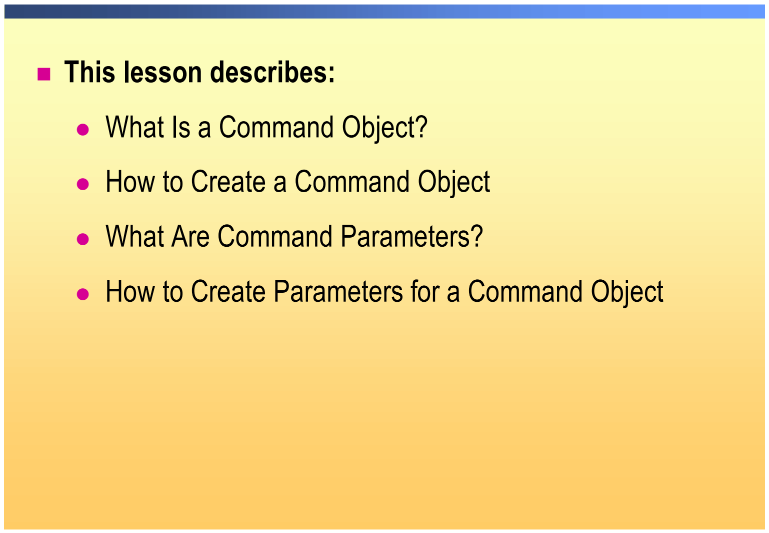
When you create tables and stored procedures, or otherwise execute logic that does not return a result set, you cannot use a DataSet and must use data commands.

Objectives

After completing this module, you will be able to:

- Build a command object.
- Execute a command that returns a single value.
- Execute a command that returns a set of rows, and process the result.
- Execute a command that returns multiple results, and process the results.
- Execute a command that defines data by using the data definition language (DDL).
- Execute a command that modifies data by using the data manipulation language (DML).
- Use transactions.

Lesson: Building Command Objects



- **This lesson describes:**
  - What Is a Command Object?
  - How to Create a Command Object
  - What Are Command Parameters?
  - How to Create Parameters for a Command Object

Introduction

Command objects allow you to access data directly in the database in a connected environment.

You can use a command object to perform the following tasks:

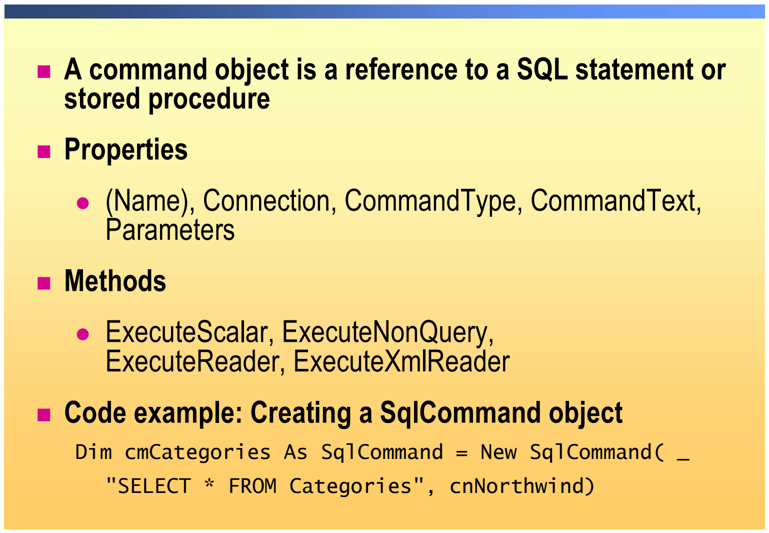
- Execute SELECT statements that return a result directly, rather than loading it into a DataSet. This reduces memory usage, and is an efficient way of loading read-only data into a control such as a Web Forms DataList or DataGrid.
- Execute DDL statements to create, edit, and remove tables, stored procedures, and other database structures. You need the required permissions to perform these actions.
- Execute statements to get database catalog information.
- Execute DML statements to update, insert, or delete records, rather than updating DataSet records and then copying changes to the database.
- Execute commands that return a scalar value (that is, a single value), such as the results of a credit-card authentication lookup or a calculated value.
- Execute commands that return data from a Microsoft® SQL Server™ database (version 7.0 or later) in XML format. A typical use is to execute a query and get back data in XML format, apply an XSLT transformation to convert the data to HTML format, and then send the results to a browser such as Microsoft Internet Explorer.

Lesson objectives

After completing this lesson, you will be able to:

- Create a command object.
- Configure the properties in a command object.
- Set parameters in a command object.

What Is a Command Object?



- A command object is a reference to a SQL statement or stored procedure
- Properties
  - (Name), Connection, CommandType, CommandText, Parameters
- Methods
  - ExecuteScalar, ExecuteNonQuery, ExecuteReader, ExecuteXmlReader
- Code example: Creating a SqlCommand object

```
Dim cmCategories As SqlCommand = New SqlCommand( _
  "SELECT * FROM Categories", cnNorthwind)
```

Introduction

A command object contains a reference to a SQL statement or stored procedure that you can execute directly. The two command classes are described in the following table.

Command class	Description
System.Data.SqlClient.SqlCommand	SQL Server .NET Data Provider command
System.Data.OleDb.OleDbCommand	OLE DB .NET Data Provider command

Properties of a command object

The properties of a command object contain all of the information necessary to execute a statement against a database. This information includes:

- Connection. The command object references a connection object, which it uses to communicate with the database.
- CommandType. One of: Text, StoredProcedure, TableDirect.
- CommandText. The command object includes the text of a SQL statement, or the name of a stored procedure to execute.
- Parameters. The command object may include zero or more parameters.

Examples

The following example creates a **SqlCommand** object. The **SqlCommand** object specifies a query that returns a list of categories from the Northwind database assuming an existing connection object named cnNorthwind.

```
Dim cmCategories As SqlCommand = New SqlCommand( _
  "SELECT * FROM Categories", cnNorthwind)
```

The following example creates an **OleDbCommand** object. The **OleDbCommand** object specifies a stored procedure that returns a list of all categories from the Northwind database.

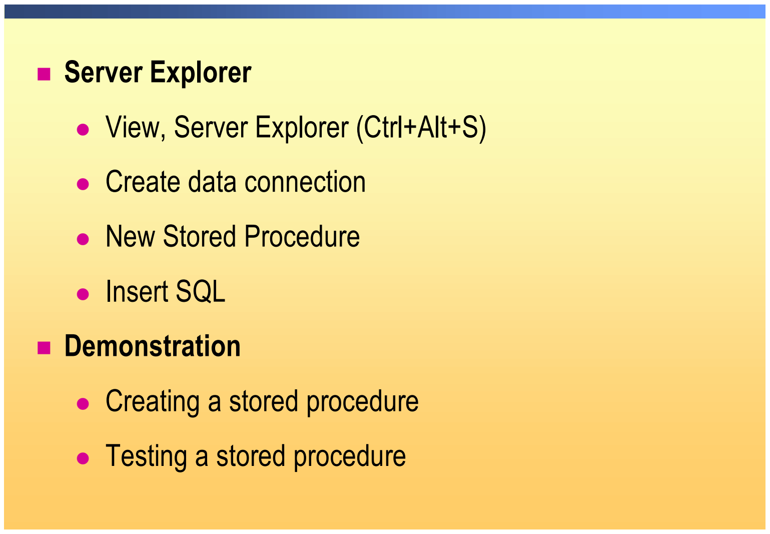
```
Dim cmCategories As OleDbCommand = New OleDbCommand( _
  "dbo.AllCategories", cnNorthwind)

cmCategories.CommandType = CommandType.StoredProcedure
```

Methods of a command object

After configuring the properties for a command object, you call one of the following methods to execute the command. The method that you call depends on the statement or procedure being executed, and the results that you expect to be returned.

Method in XxxCommand class	Description
ExecuteScalar	Executes a command that returns a single value.
ExecuteReader	Executes a command that returns a set of rows.
ExecuteXmlReader	Executes a command that returns an XML result. This capability is supported by SQL Server version 7.0 or later.
ExecuteNonQuery	Executes a command that updates the database or changes the database structure. This method returns the number of rows affected.

How to Create a Stored Procedure



- **Server Explorer**
  - View, Server Explorer (Ctrl+Alt+S)
  - Create data connection
  - New Stored Procedure
  - Insert SQL
- **Demonstration**
  - Creating a stored procedure
  - Testing a stored procedure

Introduction

Microsoft Visual Studio® .NET includes tools to help you create a stored procedure.

Creating a stored procedure

► **To use the Server Explorer create a stored procedure**

1. Show the Server Explorer by choosing **Server Explorer** from the **View** menu or pressing **Ctrl+Alt+S**.
2. Create a connection to the database you in which you wish to create the stored procedure by right-clicking the **Data Connections** folder and choosing **Add Data Connection**.
3. Expand the new connection and the **Stored Procedures** folder.
4. Right-click the **Stored Procedures** folder and choose **New Stored Procedure**.
5. Enter the SQL script. You may right-click and choose **Insert SQL** to use the graphical Query Editor to help you.
6. Close and save the stored procedure.

Demonstration

► **To use the Server Explorer to create a stored procedure**

In this practice, you will create a project and a stored procedure in the SQL Server Northwind database. The stored procedure will return the number of orders made by a specified customer when passed a customer ID value as a parameter.

1. Start the Visual Studio .NET development environment.
2. View the Server Explorer.
3. Expand your data connections, or create a new connection to the local SQL Server and the Northwind sample database.
4. Expand the Northwind database.
5. Right-click the **Stored Procedures** folder, and then choose **New Stored Procedure**.
6. Change the name of the stored procedure to **dbo.CountOrders**.
7. Delete the comment characters and edit the parameter area to declare the following two parameters using this code:

   ```
   @CustomerID nchar(5),
   @CompanyName nvarchar(40) OUTPUT
   ```

8. Delete the /* **SET NOCOUNT ON** */ comment.
9. Declare a local variable using this code:

   ```
   DECLARE @OrdersCount int
   ```

10. On a new line, right-click and choose **Insert SQL**.
11. Add the **Customers** and **Orders** tables.
12. In the **Customers** table, select the CustomerID and CompanyName fields.
13. In the **Orders** table, select the OrderID field.
14. Disable the output of the CustomerID field.
15. Set the **CustomerID Criteria** column to **=@CustomerID**.
16. Right-click the Query Editor and choose **Group By**.
17. Change the **OrderID** Group By column to **Count**.
18. Close the Query Builder and click **Yes** to save changes.
19. Edit the **SELECT** line to look like the following:

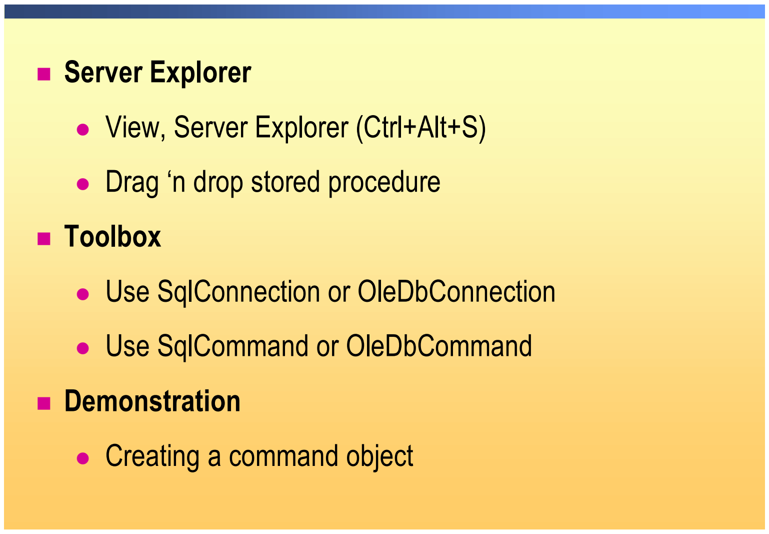
   ```
   SELECT @CompanyName = Customers.CompanyName, @OrdersCount =
   COUNT(Orders.OrderID)
   ```

20. Return the count of orders from the stored procedure as follows:
   `RETURN @OrdersCount`
21. Save the stored procedure.
22. To test, in the editor, right-click the stored procedure and choose **Step Into Stored Procedure**.

23. For the **CustomerID**, type **ALFKI**. For the **CompanyName**, type **<NULL>** (change it from <**DEFAULT**>).
24. The yellow active line marker should point to the SELECT statement. Hover over the parameters and the local variable to see their values. The @CompanyName and @OrdersCount should both be NULL.
25. Step into the line to execute it, and hover over the parameters and the local variable to see their values. The @CompanyName and @OrdersCount should be set to Alfreds Futterkiste and 6, respectively.
26. Stop debugging the stored procedure.

Hint: N' means Unicode.

How to Create a Command Object



- **Server Explorer**
  - View, Server Explorer (Ctrl+Alt+S)
  - Drag 'n drop stored procedure
- **Toolbox**
  - Use SqlConnection or OleDbConnection
  - Use SqlCommand or OleDbCommand
- **Demonstration**
  - Creating a command object

Introduction

Microsoft Visual Studio® .NET includes tools to help you create a command object in your form or component. You can create a **SqlCommand** object or an **OleDbCommand** object, depending on the type of your data source.

Creating a command object

► **To add a command object to a form or component using Toolbox**

1. If you do not already have a connection object available on the form or component, add one.
2. From the **Data** tab of the Toolbox, drag a **SqlCommand** or **OleDbCommand** onto your form or component.
3. Set the following properties for the command object.

Property	Description
(Name)	The name by which you want to refer to the command object in your code.
Connection	A reference to a connection object that the command will use to communicate with the database. You can select an existing connection from the drop-down list, or create a new connection.
CommandType	A value specified by the **CommandType** enumeration, indicating what type of command you want to execute: • **Text**. A SQL statement. • **StoredProcedure**. A stored procedure. • **TableDirect**. A way of fetching the entire contents of a table. (This option is available only for **OleDbCommand** objects.)

(***continued***)

Property	Description
CommandText	The command to execute. The command text you specify depends on the value of the **CommandType** property: • **Text**. Enter the SQL statement to execute. • **StoredProcedure**. Enter the name of the stored procedure. • **TableDirect**. Enter the name of the table to fetch.
Parameters	A collection of objects of the type **SqlParameter** or **OleDbParameter**. You use this collection to pass parameters into the command, and to retrieve output parameters from the command. You will learn more about parameters in the next lesson.

Note The SQL Server .NET Data Provider does not support the question mark (?) placeholder for parameters in a SQL statement. Instead, you must use a named parameter. For example, **SELECT * FROM Products WHERE ProductID = @ProdID**

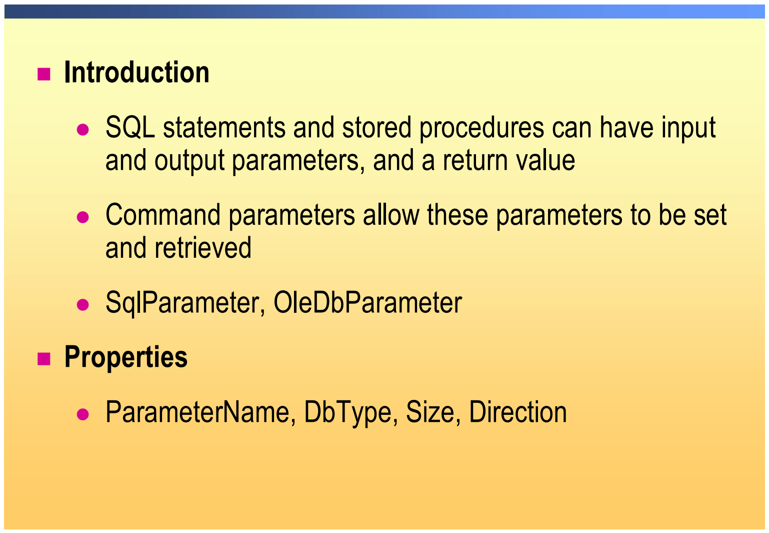
Demonstration

► To use the Server Explorer to create a command object

In this practice, you will use a stored procedure that will return the number of orders made by a specified customer when passed a customer ID value as a parameter.

1. Start the Visual Studio .NET development environment.
2. View the Server Explorer.
3. Expand your data connections.
4. Expand the Northwind database and the Stored Procedures folder.
5. Drag the stored procedure named **CountOrders** onto the form.
6. View the code written by the designer.

What Are Command Parameters?



- Introduction
  - SQL statements and stored procedures can have input and output parameters, and a return value
  - Command parameters allow these parameters to be set and retrieved
  - SqlParameter, OleDbParameter
- Properties
  - ParameterName, DbType, Size, Direction

Visual Basic Example

Introduction

SQL statements and stored procedure can specify both input parameters and output parameters. A stored procedure can also specify a separate return value.

You must configure your command object so that it deals correctly with these input parameters, output parameters, and the return value.

Definition

SqlCommand and **OleDbCommand** have a **Parameters** collection. This collection specifies a set of **SqlParameter** or **OleDbParameter** objects, which represent the input parameters, output parameters, and return value for the command.

Before you execute a command, you must set a value for every input parameter in the command. After execution, you can retrieve the output parameters and the return value from the command.

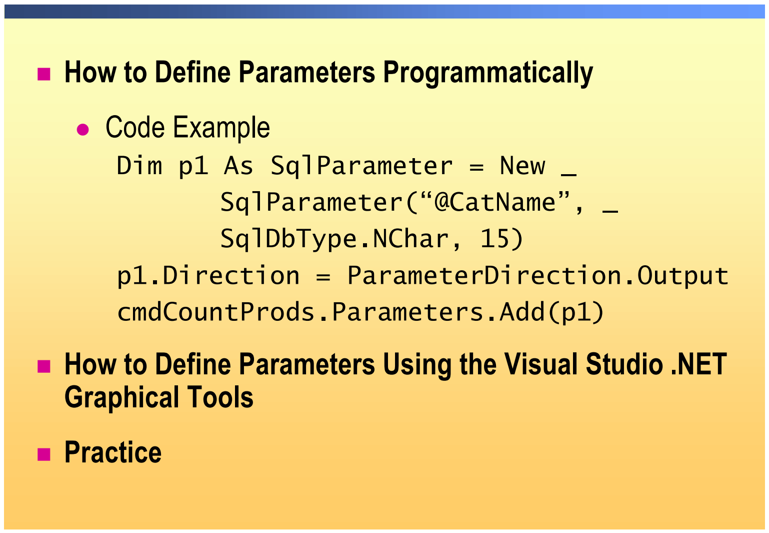
Example

The following stored procedure returns information about a particular category of products in the Northwind database.

The @CatID input parameter specifies the required category. The stored procedure assigns the name of the category to the @CatName output parameter, and returns the number of products in the category.

```
/* Stored procedure with an input parameter named @CatID,
   an output parameter named @CatName, and a return value */

CREATE PROCEDURE dbo.CountProductsInCategory
  (
    @CatID int,
    @CatName nvarchar(15) OUTPUT
  )
AS
  DECLARE @ProdCount int

  SELECT @CatName = Categories.CategoryName,
         @ProdCount = COUNT(Products.ProductID)
  FROM Categories INNER JOIN Products
    ON Categories.CategoryID = Products.CategoryID
  WHERE (Categories.CategoryID = @CatID)
  GROUP BY Categories.CategoryName

  RETURN @ProdCount
```

How to Create Parameters for a Command Object



- How to Define Parameters Programmatically
  - Code Example

```
Dim p1 As SqlParameter = New _
       SqlParameter("@CatName", _
       SqlDbType.NChar, 15)
p1.Direction = ParameterDirection.Output
cmdCountProds.Parameters.Add(p1)
```

- How to Define Parameters Using the Visual Studio .NET Graphical Tools
- Practice

Visual Basic Example

Introduction

There are two ways to create parameters for a command object:

- Programmatically create XxxParameter objects, and then add these objects to the Parameters collection in the command object.
- Set the **Parameters** property automatically, by using the Properties window.

How to define parameters programmatically

Follow these steps to define a parameter programmatically:

1. Create a new **SqlParameter** object or **OleDbParameter** object.
2. Set the properties for the parameter object. The following table describes the most commonly used properties.

Property	Description
ParameterName	The name of the parameter, such as "@CatID".
DbType, SqlDbType, or OleDbType	The data type of the parameter. The **DbType** property is linked to the **SqlDbType** or **OleDbType** property, depending on which data provider you are using.
Size	The maximum size, in bytes, of the data in the parameter.
Direction	A value specified by the ParameterDirection enumeration. Use one of the following values: • **ParameterDirection.Input** (default value) • **ParameterDirection.InputOutput** • **ParameterDirection.Output** • **ParameterDirection.ReturnValue**

3. Call the **Add** method on the **Parameters** collection for the command object. If the command calls a stored procedure that returns a result, you must add the **ParameterDirection.ReturnValue** parameter before any other parameters. The order of the other parameters is insignificant.

Example

The following example creates three parameters for the **CountProductsInCategory** stored procedure, which was introduced on the previous page. The parameters are added to a **SqlCommand** object named **cmdCountProductsInCategory**.

```
Dim p1, p2, p3 As SqlParameter

p1 = New SqlParameter("@RETURN_VALUE", SqlDbType.Int, 4)
p1.Direction = ParameterDirection.ReturnValue

p2 = New SqlParameter("@CatID", SqlDbType.Int, 4)
p2.Direction = ParameterDirection.Input

p3 = New SqlParameter("@CatName", SqlDbType.NChar, 15)
p3.Direction = ParameterDirection.Output

cmdCountProductsInCategory.Parameters.Add(p1)
cmdCountProductsInCategory.Parameters.Add(p2)
cmdCountProductsInCategory.Parameters.Add(p3)
```

How to define parameters using the Visual Studio .NET graphical tools

To define parameters automatically, use the Visual Studio .NET developer environment as follows:

1. Drag a **SqlCommand** or **OleDbCommand** object from the Toolbox and drop it onto your form or component.
2. In the Properties window, set the **Connection**, **CommandType**, and **CommandText** properties for the command object.
3. When you set the **CommandText** property, you are asked if you want to regenerate the parameters for the command. Choose **Yes**.
4. The Visual Studio .NET developer environment generates the code to create the parameters for your command object.

Practice

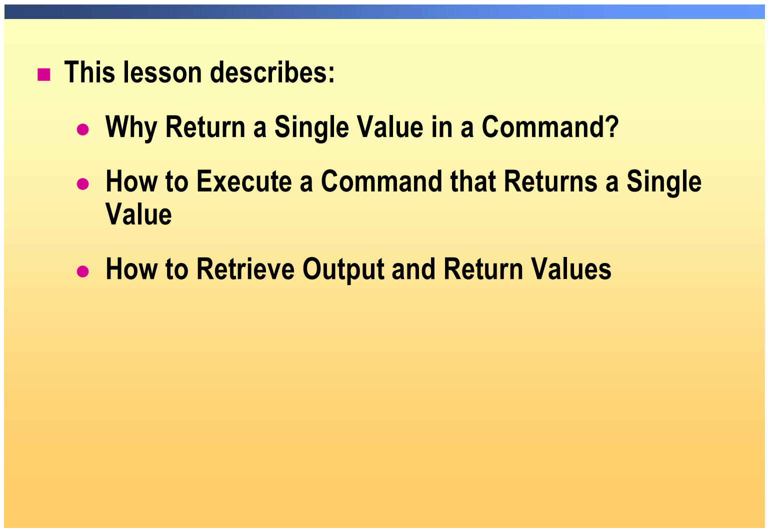
► **To build command objects and parameters**

1. Create a new Windows Application project named **BuildingCommandObjects**.
2. Create a stored procedure in the Northwind database named **CountOrders**.

```
CREATE PROCEDURE dbo.CountOrders
(@CustomerID nchar(5), @CompanyName nvarchar(40) OUTPUT) AS
DECLARE @OrdersCount int
SELECT @CompanyName = Customers.CompanyName,
   @OrdersCount = COUNT(Orders.OrderID)
FROM Customers INNER JOIN Orders
   ON Customers.CustomerID = Orders.CustomerID
WHERE (Customers.CustomerID = @CustomerID)
   GROUP BY Customers.CompanyName
RETURN @OrdersCount
```

3. Drag **CountOrders** from the Server Explorer onto **Form1**.
4. Right-click the form and choose **View Code**. Expand and examine the Windows Form Designer generated code that creates the connection and command objects, and initializes the command parameters.

Lesson: Executing Command Objects That Return a Single Value



- This lesson describes:
  - Why Return a Single Value in a Command?
  - How to Execute a Command that Returns a Single Value
  - How to Retrieve Output and Return Values

Introduction

After you have built a command object, you are ready to execute the command against the database. The **SqlCommand** and **OleDbCommand** classes provide four different ways to execute a command, depending on the nature of the SQL statement or stored procedure.

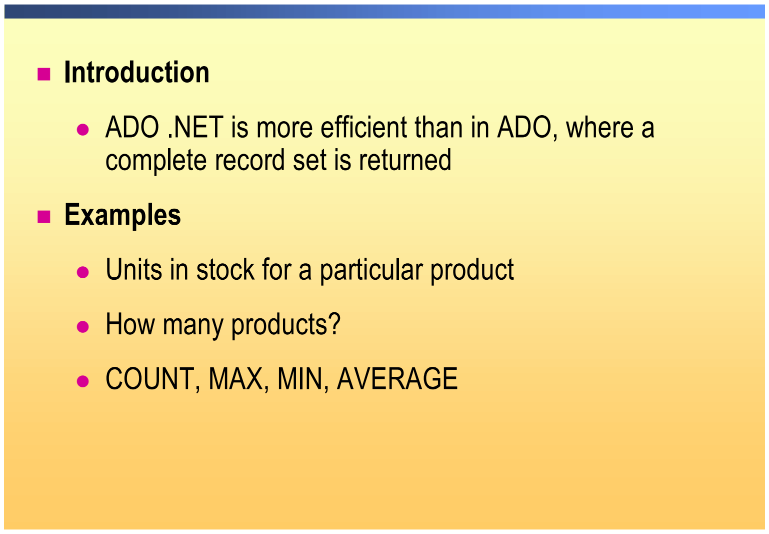
In this lesson, you will learn how to execute a command that returns a single value. You will also learn how to set input parameters before you execute the command, and how to retrieve output parameters and the return value after execution.

Lesson objectives

After completing this lesson, you will be able to:

- Execute a command that returns a single value.
- Pass input parameters into a command.
- Retrieve output parameters and a return value from the command.

Why Return a Single Value in a Command?



- **Introduction**
  - ADO .NET is more efficient than in ADO, where a complete record set is returned
- **Examples**
  - Units in stock for a particular product
  - How many products?
  - COUNT, MAX, MIN, AVERAGE

Introduction

Occasionally, you may want to execute a database command or function that returns a single value — that is, a *scalar value*. Because you are returning only one value, this type of command is typically not performed with DataSets. Instead, you execute the statement by using a command object.

Examples

The following are example scenarios of situations where you might want to return a single value in a command:

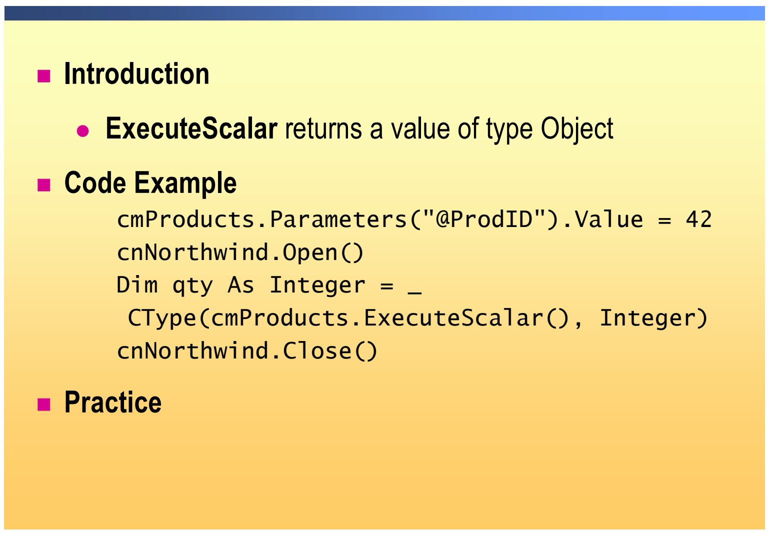
- You want to find the units in stock for a particular product. To do this, write a SQL statement that returns the UnitsInStock field for the product.
- You want to find out how many products are in the Northwind database. To do this, you write a SQL statement that uses the **COUNT()** function to count the products.
- You want to find out how many products there are in a particular category, and also obtain the name of that category. To do this, you can write a stored procedure that uses the category ID as an input parameter, and sets the category name as an output parameter. The stored procedure can also return the product count.

Definition

The **SqlCommand** and **OleDbCommand** classes provide the **ExecuteScalar** method, to execute a command and obtain a scalar result. The method returns the value of the first column of the first row in the record set.

If the SQL statement or stored procedure returns a complete record set, the extra columns or rows are ignored. This behavior in ADO .NET is more efficient than in ADO, where the complete record set is returned.

How to Execute a Command that Returns a Single Value



- **Introduction**
  - **ExecuteScalar** returns a value of type Object
- **Code Example**

```
cmProducts.Parameters("@ProdID").Value = 42
cnNorthwind.Open()
Dim qty As Integer = _
 CType(cmProducts.ExecuteScalar(), Integer)
cnNorthwind.Close()
```

- **Practice**

Visual Basic Example

Introduction

The **ExecuteScalar** method allows you to execute a SQL statement or stored procedure that returns a scalar result. The **ExecuteScalar** method returns a value using the **Object** data type so you will usually want to convert the value to a more efficient data type.

Guidelines for using the ExecuteScalar method

► **To use the ExecuteScalar method**

1. Add a command object to your form or component and set the properties and parameters if necessary.
2. Write code to open the database connection.
3. Write code to call the **ExecuteScalar** method of the command. Assign the return value to a variable of the appropriate data type.
4. Write code to close the database connection.

Example

The following example uses a command object to execute a SQL statement that returns a scalar value. The statement queries the Products table. It takes the product ID as a parameter, and returns an integer value indicating the quantity in stock for that product. In this example, there is no aggregate function (for example, SUM), because the quantity in stock is stored as a column value in the Products table.

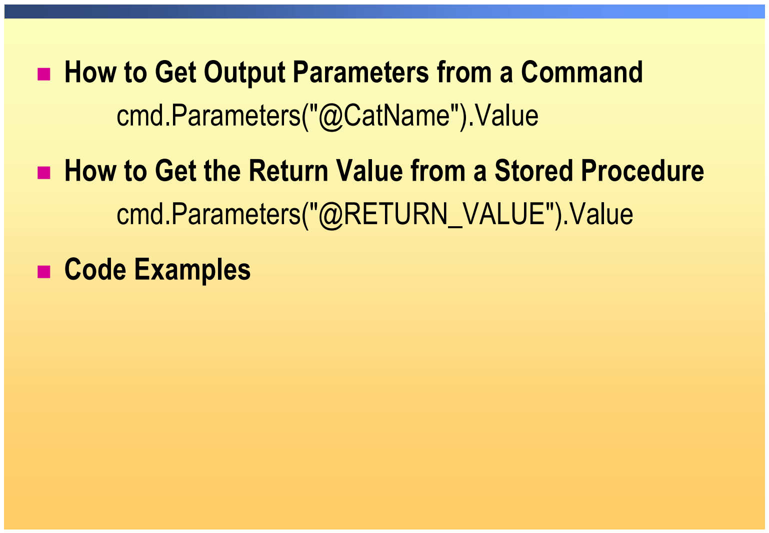
```
Dim sql As String = "SELECT UnitsInStock FROM Products " & _
                    "WHERE ProductID = @ProdID"

Dim cmProducts As SqlCommand = _
  New SqlCommand(sql, cnNorthwind)

Dim param As SqlParameter = cmProducts.Parameters.Add( _
      New SqlParameter("@ProdID", SqlDbType.Int, 4))

cmProducts.Parameters("@ProdID").Value = 42

cnNorthwind.Open()

Dim qty As Integer = _
  CType(cmProducts.ExecuteScalar(), Integer)

cnNorthwind.Close()

MessageBox.Show("Quantity in stock: " & qty.ToString())
```

Practice

► **To use the ExecuteScalar method**

1. Start the Visual Studio .NET development environment.
2. Create a new Windows Application project named **ScalarValues**.
3. Use the Server Explorer to create a new stored procedure in the Northwind database named **CountCustomers** with the following statement:
   `SELECT COUNT(*) from Customers`
4. Drag the **CountCustomers** stored procedure onto **Form1**.
5. Add a label and a button to the form.
6. Define a click event handler for the button, and write some code to perform the following tasks:
   a. Open the connection.
   ```
   Me.SqlConnection1.Open()
   ```
   b. Execute the command, and display the result in the label.
   ```
   Me.Label1.Text = _
       Me.SqlCommand1.ExecuteScalar() & _
       " customers"
   ```
   c. Close the connection.
   ```
   Me.SqlConnection1.Close()
   ```
7. Run and test the program.

How to Retrieve Output and Return Values



- How to Get Output Parameters from a Command
  cmd.Parameters("@CatName").Value
- How to Get the Return Value from a Stored Procedure
  cmd.Parameters("@RETURN_VALUE").Value
- Code Examples

Stored Procedure and Visual Basic Example

Introduction

SQL statements and stored procedures often pass values back to the application that called them. They can do so by assigning a value to an output parameter. In addition, stored procedures can also specify a distinct return value.

How to get output parameters from a command

► **To get values returned by procedures**

To get an output parameter from a command object, follow these steps:

1. Configure the Parameters collection in the command object. For each output parameter, define a **Parameter** object with the **Direction** property set to **ParameterDirection.Output**. If a parameter is used to both receive and send values, set the **Direction** to **ParameterDirection.InputOutput**.
2. Ensure that the data type of each parameter matches the expected data type in the stored procedure.
3. After executing the procedure, read the **Value** property of the parameter that is being passed back.

Note If you use the Designer to create the connection and command objects, the parameters are configured automatically. You only need to write code to retrieve the parameter values after executing the command.

How to get the return value from a stored procedure

► **To get the return value of a procedure**

1. Configure the Parameters collection for the stored procedure. The first parameter in the collection must have a **Direction** property set to **ParameterDirection.ReturnValue**.
2. Ensure that the data type of this parameter matches the data type that is returned from the stored procedure. Note that **INSERT**, **UPDATE**, and DELETE statements return an integer value, which indicates the number of records affected by the statement.
3. After executing the procedure, read the **Value** property of the parameter that is being passed back.

Note If you use the Designer to create the connection and command objects, the parameters are configured automatically. The default name of the return parameter is **@RETURN_VALUE**.

Example

The following example uses the **CountProductsInCategory** stored procedure.

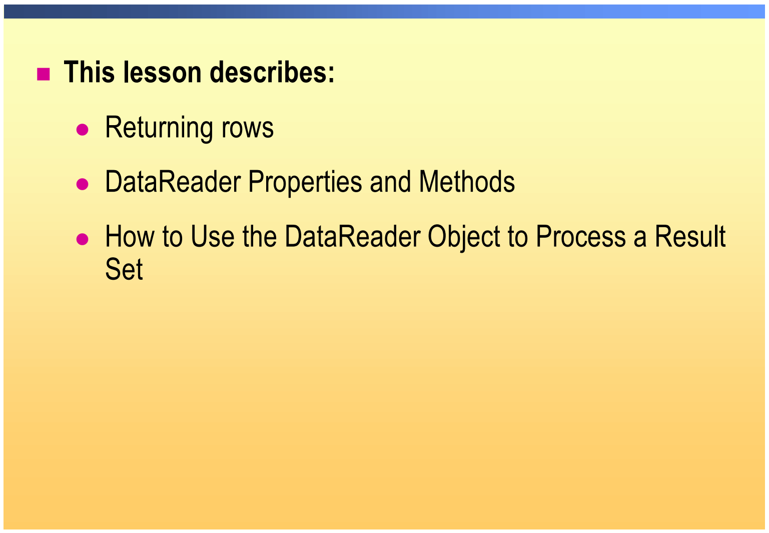
```
/* Stored procedure with an input parameter named @CatID,
   an output parameter named @CatName, and a return value */

CREATE PROCEDURE dbo.CountProductsInCategory
  (
    @CatID int,
    @CatName nvarchar(15) OUTPUT
  )
AS
  DECLARE @ProdCount int

  SELECT @CatName = Categories.CategoryName,
         @ProdCount = COUNT(Products.ProductID)
  FROM Categories INNER JOIN Products
    ON Categories.CategoryID = Products.CategoryID
  WHERE (Categories.CategoryID = @CatID)
  GROUP BY Categories.CategoryName

  RETURN @ProdCount
```

The stored procedure receives an input parameter named @CatID, and assigns an output parameter named @CatName. The stored procedure also returns a count of the products in this category.

This example assumes that the connection, command, and parameter objects have already been configured. The following code shows how to set and retrieve parameter values when you execute the stored procedure.

```
' Set input parameters, execute the stored procedure, then
' retrieve the output parameter and the return value
' Note: we could use any ExecuteX method, but
'       ExecuteNonQuery is the most efficient

cmd.Parameters("@CatID").Value = 1
cnNorthwind.Open()
cmd.ExecuteNonQuery()
cnNorthwind.Close()

MessageBox.Show("Category name: " & _
                cmd.Parameters("@CatName").Value & _
                "Number of products in category: " & _
                cmd.Parameters("@RETURN_VALUE").Value)
```

Lesson: Executing Commands That Return Rows



- **This lesson describes:**
  - Returning rows
  - DataReader Properties and Methods
  - How to Use the DataReader Object to Process a Result Set

Introduction

In this lesson, you will learn how to execute a command that returns a result set. This is a common requirement for applications that need to query a database, to obtain data that matches specific criteria.

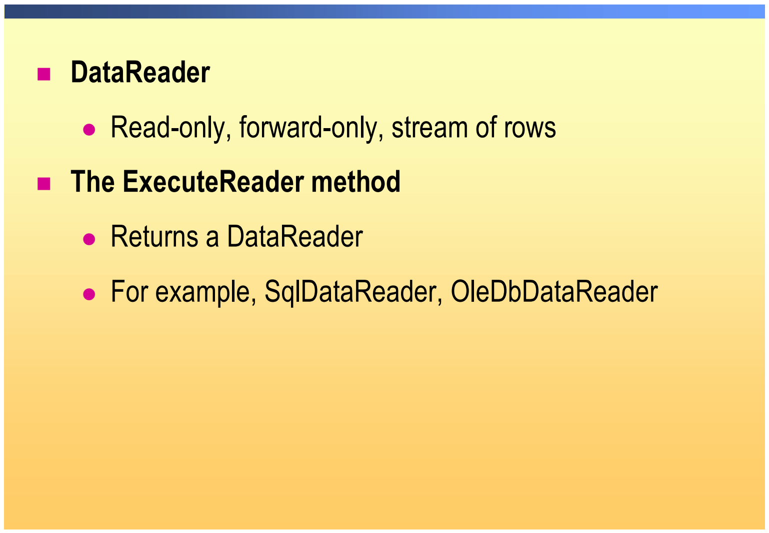
You will also learn how to iterate efficiently through the result set, by using a DataReader object.

Lesson objectives

After completing this lesson, you will be able to:

- Execute a command that returns rows.
- Use a DataReader object to iterate through the rows.
- Access the fields in a row by using strongly typed methods in the DataReader object.
- Describe scenarios where it is appropriate to use a DataReader object.

Returning Rows



- **DataReader**
  - Read-only, forward-only, stream of rows
- **The ExecuteReader method**
  - Returns a DataReader
  - For example, SqlDataReader, OleDbDataReader

Introduction

The DataReader is a fast, forward-only cursor through a stream of rows. The DataReader only keeps one row in memory at a time. This increases application performance, and reduces system overhead. A DataReader is a good choice when you need to retrieve large amounts of data, because the data is not cached in memory.

The ExecuteReader method

You can use a command object and the **ExecuteReader** method to return a DataReader. You can execute any SELECT statement or a stored procedure that contains a SELECT statement.

The DataReader provides strongly typed methods, to get the value of a specific column in the current row. You can also obtain metadata about the rows, such as the column name and the column data type.

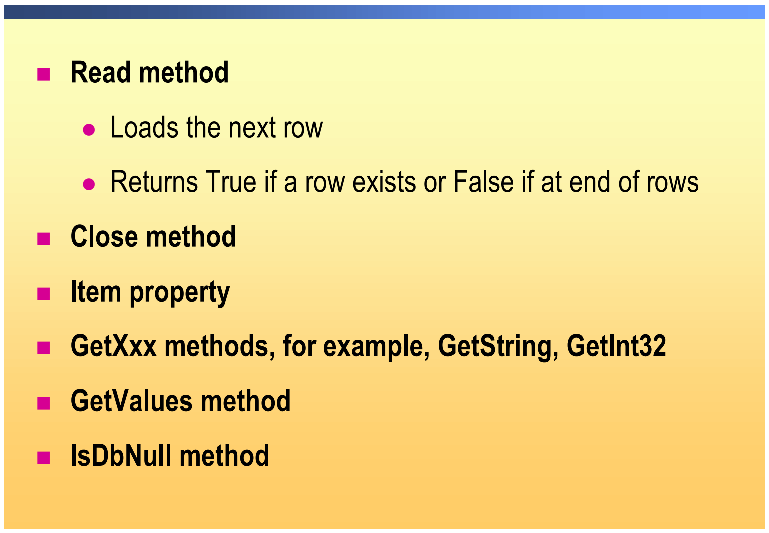
When you process a result set with a DataReader, the associated connection is kept busy until you close the DataReader. For this reason, you should close the DataReader as soon as you finish processing the result set.

Example

The following are examples of situations where you might want to use a command object to return a DataReader.

- You want to obtain a single record from a table, such as the details for a particular customer. To do this, you specify the customer ID, and get back a single record containing the details for that customer.
- You want to obtain a set of records that you insert into a control on a form. This is especially useful in Web Forms, which often display read-only information such as search results or inventory lists.

DataReader Properties and Methods



- Read method
  - Loads the next row
  - Returns True if a row exists or False if at end of rows
- Close method
- Item property
- GetXxx methods, for example, GetString, GetInt32
- GetValues method
- IsDbNull method

Introduction

SqlDataReader and **OleDbDataReader** contain properties and methods for processing a result set retrieved by a command object. These properties and methods enable you to:

- Iterate through the result set, one row at a time.
- Get the value of a specific column, or all columns, in the current row.
- Check whether a column contains a missing or non-existent value.
- Get metadata for a column, such as its name, ordinal position, and data type.

Guidelines for iterating through a result set

To iterate through a result set, call the **Read** method on the DataReader object. The **Read** method reads the next row in the result set, by using the associated connection object.

The **Read** method returns **false** when there are no more records to read. At this point, you should call the **Close** method to close the DataReader and release the connection object.

Guidelines for getting column values

The following are various ways to get values for columns in the current row:

- The **Item** property gets the value of a column with a specified name or ordinal position. The value is returned in its native format, so you might need to cast the value before you can use it in your code.

> **Note** In Microsoft Visual C#™, **Item** is the indexer for the DataReader object. Use the syntax **aReader["aColumnName"]** or **aReader[columnPosition]** to access the required column value.

- The DataReader has strongly typed accessor methods, such as **GetDateTime**, **GetDouble**, **GetGuid**, and **GetInt32**. These methods return .NET Framework data types, such as **DateTime**, **Guid**, and **Int32**. Use these methods when you know the data types in the record set, to minimize the amount of type conversion required in your code.

Note The SQL Server .NET Data Provider also has methods such as **GetSqlDateTime**, **GetSqlDouble**, and so on. These methods return SQL Server data types such as **SqlDateTime** and **SqlDouble**. These types are located in the **System.Data.SqlTypes** namespace.

- The **GetValues** method returns an array of objects containing all of the column values for the current row. This can be more efficient than retrieving each column individually.

Guidelines for checking for missing column values

When you design a database, you can specify whether a column is allowed to contain a null value. You can also specify a default value for a column, if appropriate.

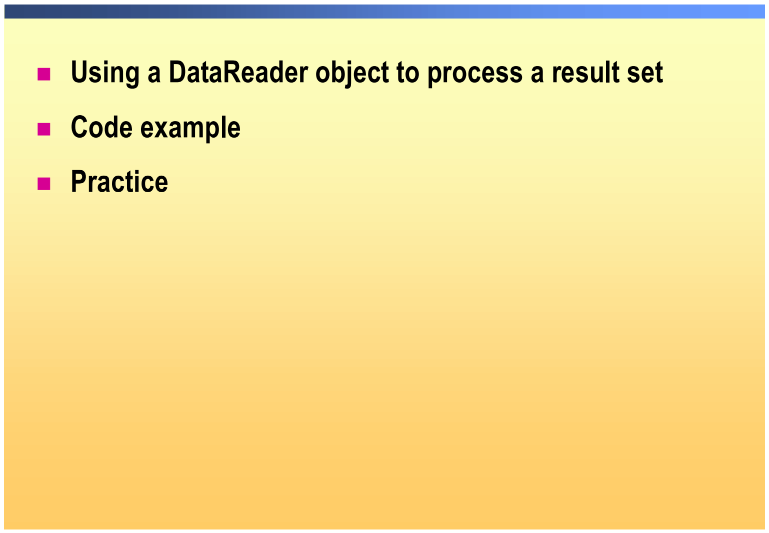
To test whether a column value is null, use the **IsDbNull** method in the DataReader object. **IsDbNull** returns **true** if the column value is null and there is no default value for the column.

Guidelines for getting result set metadata

The following are various ways to get metadata for the result set:

- The **GetName** method returns the name of the column with a specified ordinal position.
- The **GetOrdinal** method returns the ordinal position of the column with a specified name.
- The **GetSchemaTable** method returns detailed schema information about the current result set. **GetSchemaTable** returns a DataTable object, which contains one row for each column in the result set. Each column of the DataTable maps to a property of the column returned in the result set.

How to Use a DataReader to Process Rows



- Using a DataReader object to process a result set
- Code example
- Practice

Visual Basic Example

Introduction

You use a DataReader object to process the result set returned by the **ExecuteReader** method in a command object.

Using a DataReader object to process rows

► **To use a DataReader object to process rows**

1. Add a command object to your form or component and set the properties and parameters if necessary.
2. Declare a **SqlDataReader** or **OleDbDataReader** variable, depending on which data provider you are using.
3. Write code to open the database connection.
4. Call the **ExecuteReader** method on the command object, including the option to close the connection immediately after the DataReader is closed. Assign the return value to the DataReader variable.
5. Loop through the DataReader by using its **Read** method, until the method returns **False**.
6. Close the DataReader.

Example

The following example executes a SELECT statement to get product details from the Northwind database. The example iterates through the rows by using a **SqlDataReader**, and gets the ProductName and UnitsInStock for each product.

```
Dim cmdProducts As SqlCommand = New SqlCommand( _
  "SELECT ProductName, UnitsInStock " & _
  "FROM Products", cnNorthwind)

cnNorthwind.Open()

Dim rdrProducts As SqlDataReader

rdrProducts = cmdProducts.ExecuteReader( _
  CommandBehavior.CloseConnection)

Do While rdrProducts.Read()

  ListBox1.Items.Add(rdrProducts.GetString(0) & _
      vbTab & rdrProducts.GetInt16(1))

Loop
rdrProducts.Close()
```

Practice

► **To call the ExecuteReader method**

1. Start the Visual Studio .NET development environment.
2. Create a new Windows Application project named **ProcessingMultipleRows**.
3. In the Northwind database, create a stored procedure named **AllCustomers** that returns all of the data in the **Customers** table, sorted by company name.
4. Add a list box and button to **Form1**.
5. Drag the **AllCustomers** stored procedure onto **Form1**.
6. Define a click event handler for the button, and write some code to perform the following tasks:
   a. Declare a **SqlDataReader** variable.
   ```
   Dim rdrCustomers As SqlClient.SqlDataReader
   ```
   b. Open the connection.
   ```
   Me.SqlConnection1.Open()
   ```
   c. Call the **ExecuteReader** method of the command object.
   ```
   rdrCustomers = Me.SqlCommand1.ExecuteReader( _
       CommandBehavior.CloseConnection)
   ```

d. Loop through the rows in the result set.

```
Do While rdrCustomers.Read()
```

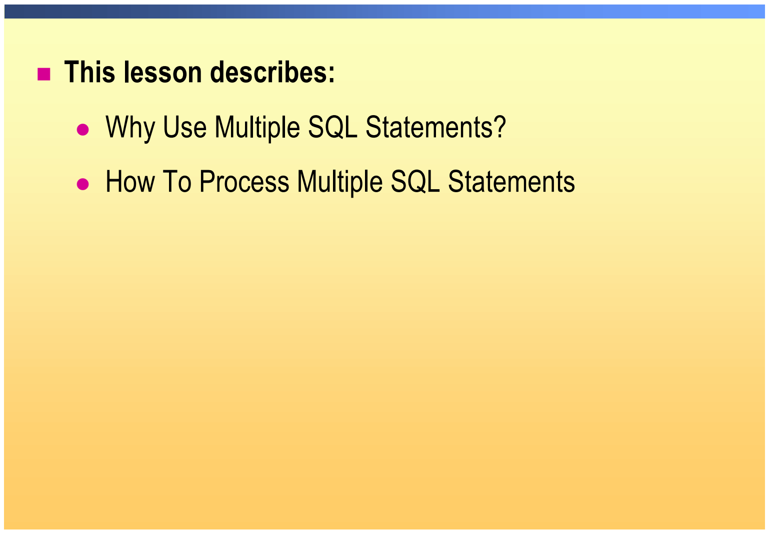
e. Inside the loop, add items to the list box by using the **GetString** method.

```
Me.ListBox1.Items.Add(rdrCustomers.GetString(1))
```

f. Outside the loop, close the DataReader.

```
Loop
rdrCustomers.Close()
```

Lesson: Executing Multiple SQL Statements



- **This lesson describes:**
  - Why Use Multiple SQL Statements?
  - How To Process Multiple SQL Statements

Introduction

A command object can specify more than one SQL statement. For example, a stored procedure can execute multiple SELECT statements, and return multiple result sets.

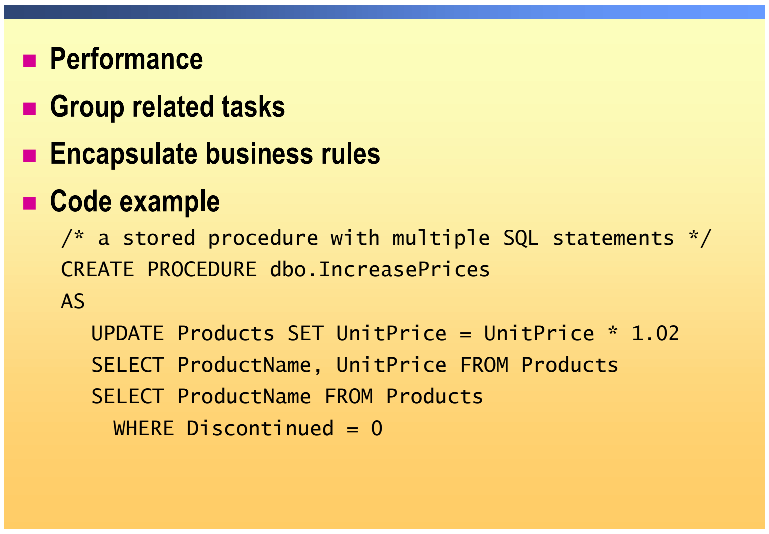
In this lesson, you will learn how to use a DataReader to process a command object that executes multiple SQL statements.

Lesson objectives

After completing this lesson, you will be able to:

- Describe scenarios where you can use multiple SQL statements.
- Use a DataReader to process combined Data Manipulation Language (DML) and SELECT statements.
- Use a DataReader to process multiple result sets.

Why Use Multiple SQL Statements?



- Performance
- Group related tasks
- Encapsulate business rules
- Code example

```
/* a stored procedure with multiple SQL statements */
CREATE PROCEDURE dbo.IncreasePrices
AS
   UPDATE Products SET UnitPrice = UnitPrice * 1.02
   SELECT ProductName, UnitPrice FROM Products
   SELECT ProductName FROM Products
     WHERE Discontinued = 0
```

Introduction

A stored procedure can contain any number of DML and SELECT statements. This enables you to group related tasks into the same stored procedure, to encapsulate business rules and improve run-time performance.

Examples

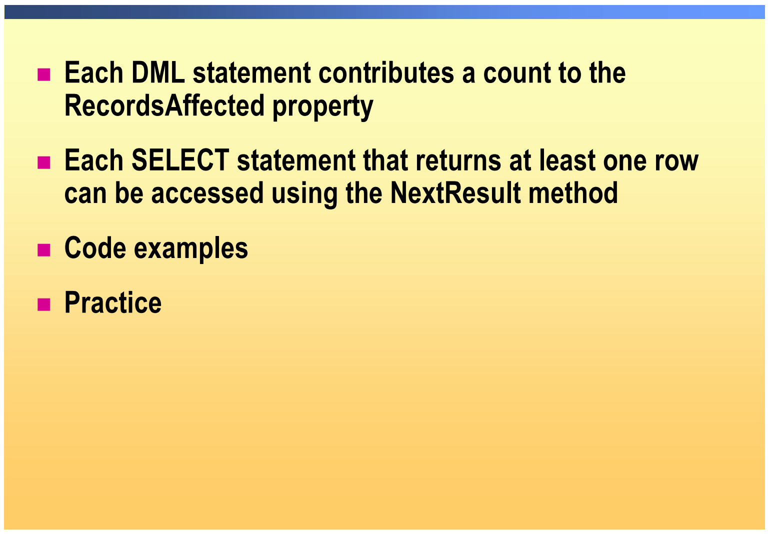
A retailer decides to increase the price of its products by 2 percent and produce a report showing the new price for all products. To do this, the retailer uses a stored procedure that uses an UPDATE statement to increase the price of all products. The stored procedure can then use a SELECT statement to return the new details for each product. The following example shows a stored procedure that performs these tasks:

```
/* Create a stored procedure that executes an UPDATE statement
   and a SELECT statement */
CREATE PROCEDURE dbo.IncreasePrices
AS
  UPDATE Products SET UnitPrice = UnitPrice * 1.02
  SELECT ProductName, UnitPrice FROM Products
```

In the next example, a retailer needs a report showing which products have been discontinued and which are still current. To do this, the retailer uses a stored procedure that contains two SELECT statements. The first SELECT statement returns details for discontinued products. The second SELECT statement returns details for current products. The following example shows a stored procedure that performs these tasks:

```
/* Create a stored procedure that executes two SELECT
   statements */
CREATE PROCEDURE dbo.ProductStatusReport
AS
  SELECT ProductName FROM Products WHERE Discontinued = 1
  SELECT ProductName FROM Products WHERE Discontinued = 0
```

How to Process Multiple SQL Statements



- Each DML statement contributes a count to the RecordsAffected property
- Each SELECT statement that returns at least one row can be accessed using the NextResult method
- Code examples
- Practice

Visual Basic Example

Introduction

If a command contains a mixture of DML and SELECT statements, you can use a DataReader to retrieve the following two pieces of information:

- The result set obtained by the first SELECT statement
- The total number of records affected by all of the DML statements

If a command contains multiple SELECT statements, you can use a DataReader to iterate through each result set in turn. The DataReader has a **NextResult** method, which allows you to move on to the next result set returned by the command.

To process combined DML and SELECT statements

► **To process combined DML and SELECT statements**

1. Create a connection object and a command object, and configure these objects for the command that you wish to execute.
2. Open the database connection.
3. Call **ExecuteReader** on the command, and assign the return value to a DataReader variable.
4. Use the DataReader to loop through the rows in the first result set.
5. If there are multiple SELECT statements, call the **NextResult** method on the DataReader, to advance to the next result set, then repeat step 4.
6. Close the DataReader and the database connection.
7. Use the **RecordsAffected** property in the DataReader, to find the total number of records changed, inserted, or deleted during execution of the command object.

Note **RecordsAffected** is not set until you have read all records in the result set, and you have closed the DataReader object. Until then it returns the value -1.

Example

The following example executes the **IncreasePrices** stored procedure.

```
CREATE PROCEDURE dbo.IncreasePrices
AS
  UPDATE Products SET UnitPrice = UnitPrice * 1.02
  SELECT ProductName, UnitPrice FROM Products
      WHERE Discontinued=0
  SELECT ProductName, UnitPrice FROM Products
      WHERE Discontinued=1
```

The following Visual Basic example code uses a **SqlDataReader** to display the rows returned by the two SELECT statements in two list boxes. The example then uses the **RecordsAffected** property to find out how many records were affected by the UPDATE statement.

```
Dim cmdProducts As SqlCommand = New SqlCommand( _
  "dbo.IncreasePrices", cnNorthwind)

cmdProducts.CommandType = CommandType.StoredProcedure

Dim rdrProducts As SqlDataReader

cnNorthwind.Open()

rdrProducts = cmdProducts.ExecuteReader( _
  CommandBehavior.CloseConnection)

Do While rdrProducts.Read()

  lstProducts.Items.Add(rdrProducts.GetString(0) & _
      vbTab & rdrProducts.GetSqlMoney(1).ToDouble())

Loop

rdrProducts.NextResult()

Do While rdrProducts.Read()

  lstDiscontinued.Items.Add(rdrProducts.GetString(0) & _
      vbTab & rdrProducts.GetSqlMoney(1).ToDouble())

Loop

MessageBox.Show("Products affected: " & _
  rdrProducts.RecordsAffected)

rdrProducts.Close()
```

Practice

► **To retrieve multiple result sets**

1. Start the Visual Studio .NET development environment.
2. Create a new Windows Application project named **ExecutingMultipleStatements**.
3. Create a stored procedure named **CategoriesAndProducts** in the Northwind database. Define a SELECT statement to return all of the Categories, sorted by name. Define another SELECT statement to return all of the Products, sorted by name. Define an UPDATE statement to increase the unit price of the discontinued products by 10%.

```
CREATE PROCEDURE dbo.CategoriesAndProducts
AS
   SELECT * FROM Categories ORDER BY CategoryName
   SELECT * FROM Products ORDER BY ProductName
   UPDATE Products
       SET UnitPrice = UnitPrice * 1.1
       WHERE (Discontinued = 1)
```

4. Add two list boxes and a button to **Form1**.
5. Drag the **CategoriesAndProducts** stored procedure onto **Form1**.
6. Define a click event handler for the button, and add the following code:
   a. Declare a data reader variable.

   ```
   Dim rdr As SqlClient.SqlDataReader
   ```

   b. Open the database connection.

   ```
   Me.SqlConnection1.Open()
   ```

   c. Call the **ExecuteReader** method, and assign the return value to a SqlDataReader variable.

   ```
   rdr = Me.SqlCommand1.ExecuteReader( _
       CommandBehavior.CloseConnection)
   ```

   d. Use a loop to read the rows. Inside the loop, add items to the first list box by using the **GetString** method of the **SqlDataReader**.

   ```
   Do While rdr.Read()
       Me.ListBox1.Items.Add(rdr.GetString(1))
   Loop
   ```

   e. Outside the loop, call the **NextResult** method of the **SqlDataReader**.

   ```
   rdr.NextResult()
   ```

   f. Use another loop to read the rows in the second result set. Add items to the second list box, by using the **GetString** method of the **SqlDataReader**.

   ```
   Do While rdr.Read()
       Me.ListBox2.Items.Add(rdr.GetString(1))
   Loop
   ```

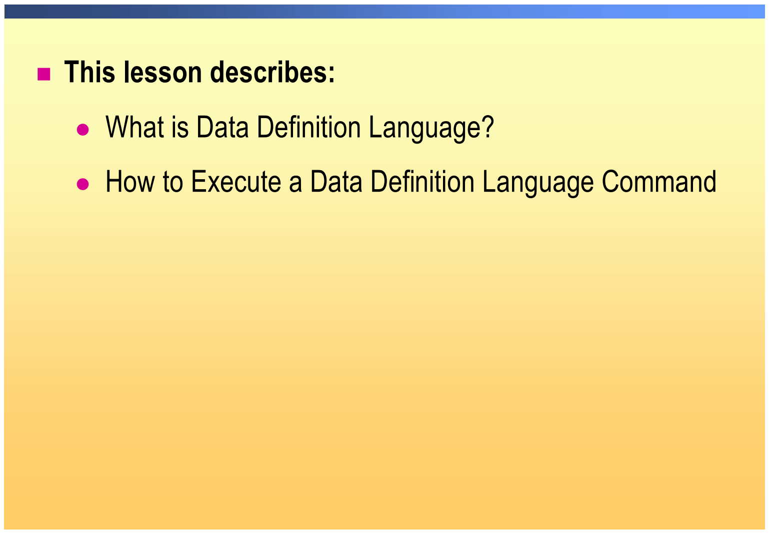
g. Outside the loop, display the number of products updated.

```
MessageBox.Show(“Products updated: ” & _
    rdr.RecordsAffected)
```

h. close the **SqlDataReader**.

```
rdr.Close()
```

Lesson: Using Data Definition Language



- **This lesson describes:**
  - What is Data Definition Language?
  - How to Execute a Data Definition Language Command

Introduction

Data Definition Language (DDL) enables you to create and manage database structures, such as tables, views, and triggers. You can also use DDL to specify security settings for a database.

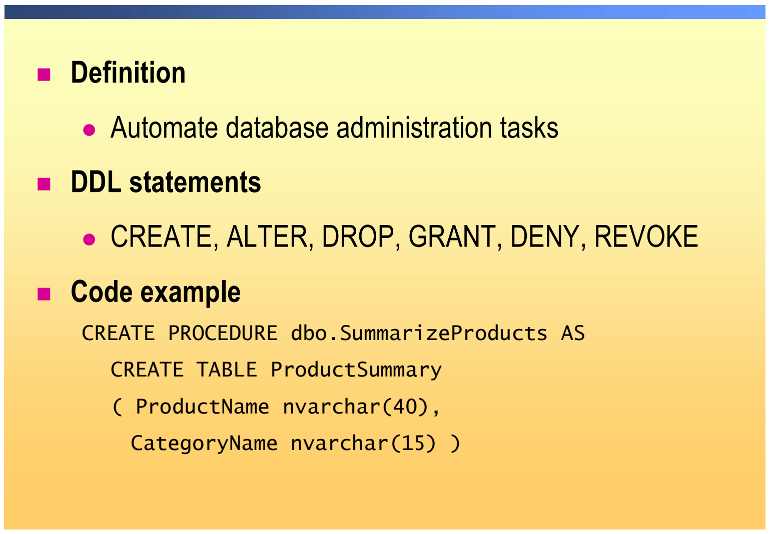
In this lesson, you will learn how to execute DDL statements by using a **SqlCommand** or **OleDbCommand** object.

Lesson objectives

After completing this lesson, you will be able to:

- Describe scenarios where DDL statements are used.
- Use a command object to execute a DDL statement.

What Is Data Definition Language?



- **Definition**
  - Automate database administration tasks
- **DDL statements**
  - CREATE, ALTER, DROP, GRANT, DENY, REVOKE
- **Code example**

```
CREATE PROCEDURE dbo.SummarizeProducts AS
   CREATE TABLE ProductSummary
   ( ProductName nvarchar(40),
     CategoryName nvarchar(15) )
```

Introduction

DDL enables you to automate database administration tasks in your application. You can programmatically execute DDL statements to manage the structure of the database. You can also grant or deny permissions for user accounts, to control who can do what in the database.

Definition

The following table describes the DDL statements.

DDL statement	Description
CREATE	Create a new database object such as a table, view, index, stored procedure, or trigger.
ALTER	Alter an existing database object.
DROP	Drop an existing database object.
GRANT	Grant permissions to a user account, to allow the user to perform specific actions on the current database.
DENY	Deny permissions to a user account, to prevent the user from performing specific actions on the current database.
REVOKE	Revoke a previously granted or denied permission.

Examples

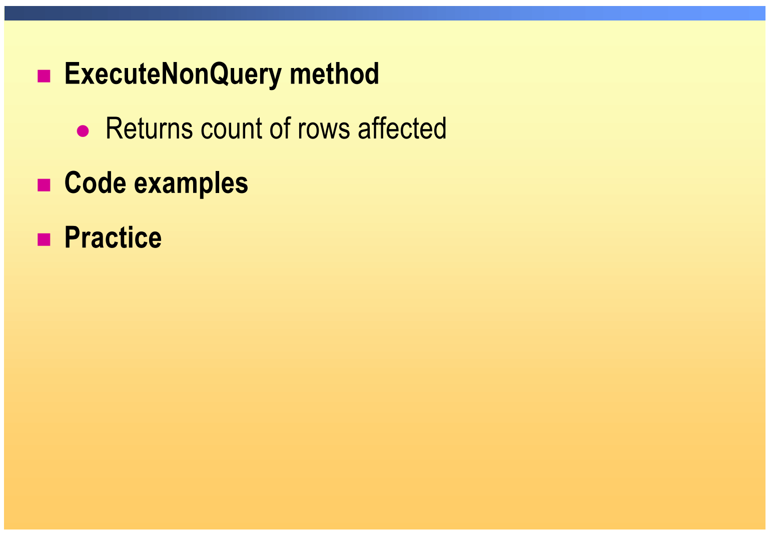
The following example shows a stored procedure that creates a new table named ProductSummary. The table contains product and category names:

```
/* Stored procedure to create a new table */
CREATE PROCEDURE dbo.SummarizeProducts
AS
  CREATE TABLE ProductSummary
  (
    ProductName nvarchar(40),
    CategoryName nvarchar(15)
  )
```

The following example shows a stored procedure that grants or denies permissions for all users to query the ProductSummary table:

```
/* Stored procedure to grant or deny permission to query
   the ProductSummary table */
CREATE PROCEDURE dbo.ManagePermission
  (
    @Allow int
  )
AS
  IF @Allow = 1
    GRANT SELECT ON ProductSummary TO PUBLIC
  ELSE
    DENY SELECT ON ProductSummary TO PUBLIC
```

How to Execute a Data Definition Language Command



- **ExecuteNonQuery method**
  - Returns count of rows affected
- **Code examples**
- **Practice**

Visual Basic Example

Introduction

To execute a DDL statement in ADO .NET, you call the **ExecuteNonQuery** method on a **SqlCommand** or **OleDbCommand** object.

When you use the **ExecuteNonQuery** method to execute DDL, the method returns the number of rows affected.

To execute a DDL command

► **To execute a DDL command**

1. Create a connection object and a command object, and configure these objects for the DDL statement you wish to execute.
2. Open the database connection.
3. Call **ExecuteNonQuery** on the command.
4. Close the database connection.

Example

The following example calls the **SummarizeProducts** stored procedure.

```
CREATE PROCEDURE dbo.SummarizeProducts
AS
  CREATE TABLE ProductSummary
  (
    ProductName nvarchar(40),
    CategoryName nvarchar(15)
  )
```

The stored procedure uses DDL to create a new table named **ProductSummary**.

```
Dim cmSummarizeProducts As SqlCommand = New SqlCommand( _
  "dbo.SummarizeProducts", cnNorthwind)

cmSummarizeProducts.CommandType = CommandType.StoredProcedure

cnNorthwind.Open()

Dim affected As Integer = _
  cmSummarizeProducts.ExecuteNonQuery()

cnNorthwind.Close()

MessageBox.Show("Rows affected: " & affected)
```

Practice

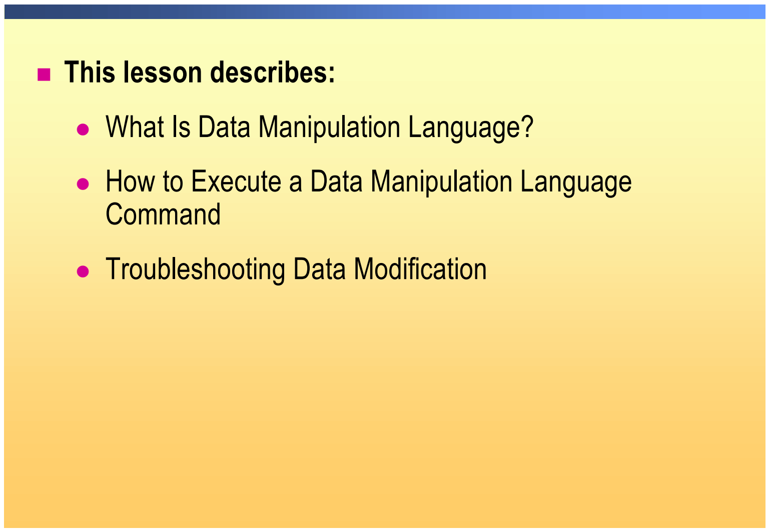
► **To execute DDL statements**

1. Start the Visual Studio .NET development environment.
2. Create a new Windows Application project named **ExecutingDDL**.
3. Create a stored procedure named **CreateContactsTable** in the Northwind database. In the stored procedure, create a new table named **Contacts** as follows:

   ```
   CREATE PROCEDURE dbo.CreateContactsTable
   AS
     CREATE TABLE Contacts
     (
       CustomerID nvarchar(5),
       EmployeeID int,
       Started datetime
     )
   ```

4. Add a button to **Form1**.
5. Drag the **CreateContactsTable** stored procedure onto **Form1**.
6. Define a click event handler for the button, and add the following code:
   a. Open the database connection.
   b. Call the **ExecuteNonQuery** method on the command object.
   c. Close the database connection.
7. Run the application, click the button on the form, and then close the application.
8. Use the Server Explorer to verify that the Contacts table has been created in the database.

Lesson: Manipulating Data Using Data Manipulation Language



- **This lesson describes:**
  - What Is Data Manipulation Language?
  - How to Execute a Data Manipulation Language Command
  - Troubleshooting Data Modification

Introduction

Data Manipulation Language (DML) enables you to modify data in a database. ADO .NET provides the following two ways to modify data by using DML statements:

- In a disconnected application, create a data adapter object that specifies a SQL query. Use the data adapter to fill a DataSet object with data. Modify the data in the DataSet, and use the data adapter to update the database with the changes.
- In a connected application, use a data command object to execute DML statements directly.

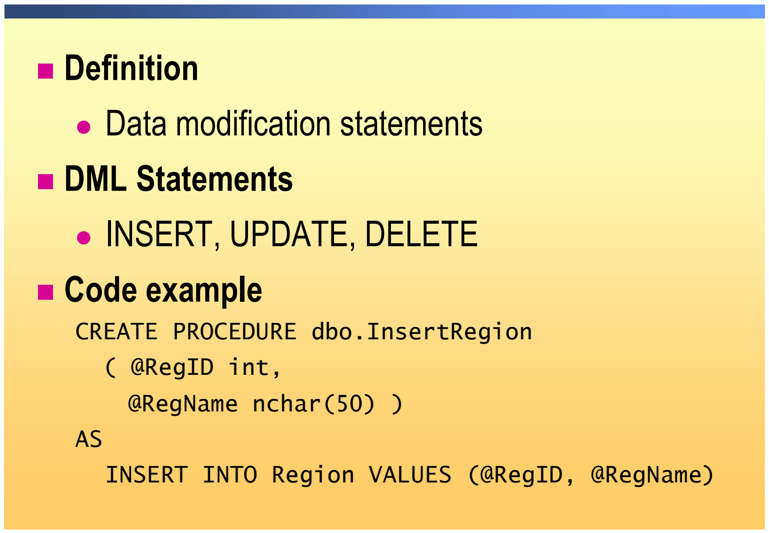
In this lesson, you will learn how to execute DML statements to modify data by using a **SqlCommand** or **OleDbCommand** object.

Lesson objectives

After completing this lesson, you will be able to:

- Describe scenarios where DML modification statements are used.
- Use a command to execute a DML modification statement.
- Test whether the DML statement has executed successfully.

What Is Data Manipulation Language?



- **Definition**
  - Data modification statements
- **DML Statements**
  - INSERT, UPDATE, DELETE
- **Code example**

```
CREATE PROCEDURE dbo.InsertRegion
  ( @RegID int,
    @RegName nchar(50) )
AS
  INSERT INTO Region VALUES (@RegID, @RegName)
```

Introduction

DML includes statements that insert new rows, update existing rows, or delete rows in a database.

Definition

The following table describes the DML statements that modify data in a database.

DDL statement	Description
INSERT	Insert a new row into a table or view. To insert multiple rows from another table or view, use the INSERT…SELECT statement.
UPDATE	Update existing rows in a table or view. Use this statement to set specific columns or parameter values.
DELETE	Delete existing rows from a table or view.

Note For more information about DML modification statements, see Appendix A, “Best Practices for Writing SQL Statements and Stored Procedures,” in Course 2389A, *Programming with ADO .NET*.

Examples

The following stored procedure inserts a new row into the Region table in the Northwind database. The stored procedure takes two input parameters, to specify the values for the new row:

```
/* Insert a row into the Region table */
CREATE PROCEDURE dbo.InsertRegion
  (
    @RegID int,
    @RegName nchar(50)
  )
AS
  INSERT INTO Region VALUES (@RegID, @RegName)
```

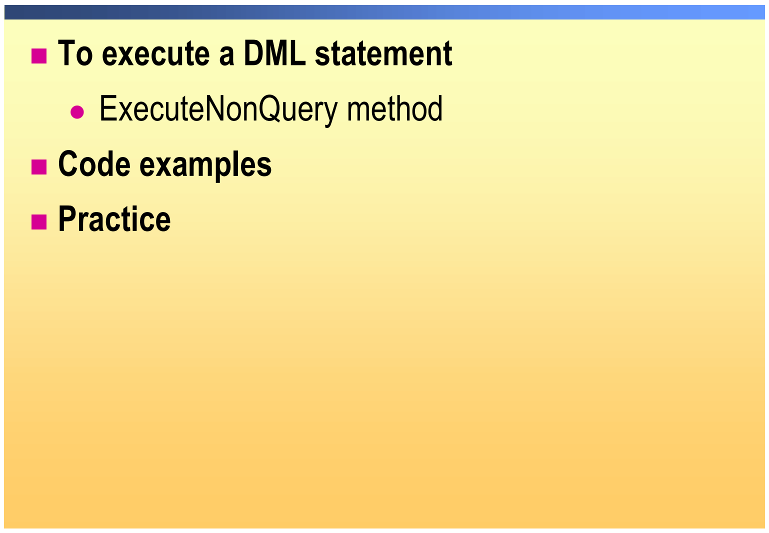
The following stored procedure increases the unit price for all products in the Products table. Each unit price is increased by 2 percent:

```
/* Update the UnitPrice for all products */
CREATE PROCEDURE dbo.IncreaseProductPrices
AS
  UPDATE Products SET UnitPrice = UnitPrice * 1.02
```

The following stored procedure deletes discontinued items from the Products table:

```
/* Delete discontinued items from the Products table */
CREATE PROCEDURE dbo.DeleteDiscontinuedProducts
AS
  DELETE FROM Products WHERE Discontinued = 1
```

How to Execute a Data Manipulation Language Command



- To execute a DML statement
  - ExecuteNonQuery method
- Code examples
- Practice

Visual Basic Example

Introduction

To execute a DML statement to modify data call the **ExecuteNonQuery** method on a command object. The method returns an integer, to indicate the number of rows affected.

To execute a DML statement

► **To execute a DML statement to modify data**

1. Create a connection object and a command object, and configure these objects for the DML statement that you wish to call.
2. Open the database connection.
3. Call **ExecuteNonQuery** on the command. Assign the return value to an integer variable, to indicate the number of rows affected by the DML statement.
4. Close the database connection.

Example

The following example calls the **IncreaseProductPrices** stored procedure, which was introduced earlier in this lesson. The stored procedure increases the unit price of all products by 2 percent:

```
Dim cmd As SqlCommand = New SqlCommand( _
  "dbo.IncreaseProductPrices", cnNorthwind)

cmd.CommandType = CommandType.StoredProcedure

cnNorthwind.Open()

Dim affected As Integer = cmd.ExecuteNonQuery()

cnNorthwind.Close()

MessageBox.Show("Records affected: " & affected)
```

Practice

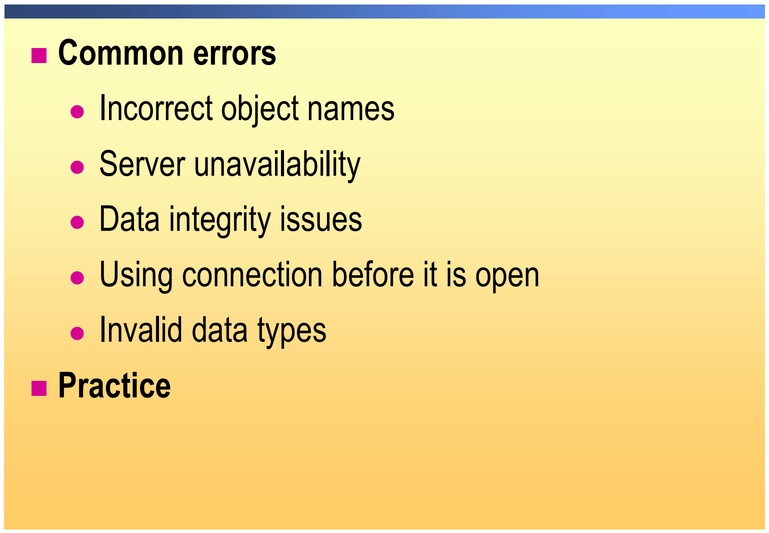
► **To execute a DML statement to insert a record into a table**

1. Start the Visual Studio .NET development environment.
2. Create a new Windows Application project named **ExecutingDML**.
3. Create a stored procedure named **InsertProduct** in the Northwind database. In the stored procedure, insert a new row into the Products table as follows:

```
/* Insert a product, and return the generated ProductID */
CREATE PROCEDURE dbo.InsertProduct
  (
    @ProductName nvarchar(40),
    @CategoryID int,
    @SupplierID int
  )
AS
  INSERT INTO Products(ProductName, CategoryID, SupplierID)
         VALUES(@ProductName, @CategoryID, @SupplierID)
  RETURN @@IDENTITY
```

4. Add a text field and a button to **Form1**.
5. Drag the **InsertProduct** stored procedure onto **Form1**.
6. Define a click event handler for the button, and add the following code:
   a. Open the database connection.
   b. Set the @ProductName parameter in the command object, by using the value in the text box.
   c. Set the @CategoryID and @SupplierID parameters to the value **1**.
   d. Call the **ExecuteNonQuery** method on the command object. Assign the return value to an integer variable.
   e. Close the database connection.
   f. Display the return value from the **ExecuteNonQuery** method.
   g. Also display the @RETURN_VALUE parameter of the data command object. The stored procedure assigns the generated ProductID to this parameter.
7. Run and test the application.
8. Enter any product name in the text field, and then click the button on the form. A message box should appear, indicating that one row has been affected. The message box should also display the generated ProductID for the inserted product.
9. Close the application.
10. Using the Server Explorer, verify that a new record has been inserted into the Products table.

Troubleshooting Data Modification



- Common errors
  - Incorrect object names
  - Server unavailability
  - Data integrity issues
  - Using connection before it is open
  - Invalid data types
- Practice

Introduction

When you execute a SQL statement against a database, many different types of errors can occur, including the following:

- Errors due to programmer fault, such as a spelling mistake
- Errors due to run-time conditions, such as server unavailability
- Errors due to data integrity issues, such as inserting a record with a duplicate primary key

Common programming errors

The following are some of the common programming errors that can occur. To resolve these errors, fix the code and rebuild the application.

- Forgetting to open a database connection before you execute the data command.
- Specifying invalid SQL syntax in the command text
- Specifying an invalid name for a stored procedure
- Forgetting to set an input parameter in a data command
- Setting an inappropriate value or data type for an input parameter
- Forgetting to set a required column when you insert a new record into a table

Common run-time errors

The following are some errors that can occur due to run-time conditions. To deal with these errors, catch a **SqlException** in your code and handle the error as appropriate:

- Unable to open a database connection
- Database connection broken during execution of the statement

Common data integrity errors

The following are some data integrity errors that can occur. To deal with these errors, catch a **SqlException** in your code:

- Inserting duplicate records into a table
- Inserting a record into a secondary table, but specifying a non-existent record in the primary table
- Deleting a record from a primary table, where the record is still referenced in a secondary table
- Attempting to create a table that already exists
- Attempting to drop a table that is referenced by a secondary table
- Attempting to execute a statement without sufficient user privileges

Note Even if a DML statement executes without any exceptions, this does not necessarily mean that the DML statement had the desired effect. For example, if you try to delete a non-existent record, the statement succeeds but returns 0 to indicate that no records were deleted.

Practice

► **To troubleshoot data modification errors**

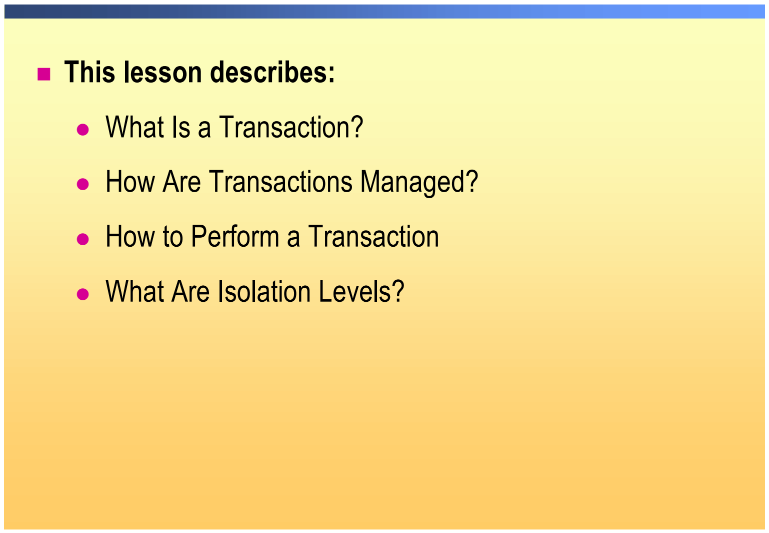
1. Open the project named **ExecutingDML**, which you created earlier in this lesson, or open the solution provided in the Practices folder.
2. Create a stored procedure named **DeleteProduct** in the Northwind database. In the stored procedure, delete a row from the Products table as follows:

```
/* Delete the product with the specified ProductID */
CREATE PROCEDURE dbo.DeleteProduct
  (
    @ProductID int
  )
AS
  DELETE FROM Products WHERE ProductID = @ProductID
```

3. Add another text box and a button to **Form1**.
4. Drag the **DeleteProduct** stored procedure onto **Form1**.
5. Define a click event handler for the new button, and add the following code:
   a. Open the database connection.
   b. Set the @ProductID parameter in the new command object, by using the value in the new text box.
   c. Call the **ExecuteNonQuery** method on the new command object. Assign the return value to an integer variable.
   d. Close the database connection.
   e. Display the return value from the **ExecuteNonQuery** method.

6. Run and test the application.
7. Try to delete one of the products you added in the previous lesson (ProductID > 77). You will be able to delete this product.
8. Try to delete the same product ID again. You will not get an error, but no records will be deleted.
9. Try to delete an original product, such as ProductID = 1. You will get an exception, because the record is referenced elsewhere in the database.
10. If time permits, modify the code in your application so that it catches any exceptions that might occur.

Lesson: Using Transactions



- **This lesson describes:**
  - What Is a Transaction?
  - How Are Transactions Managed?
  - How to Perform a Transaction
  - What Are Isolation Levels?

Introduction

A *transaction* is a single unit of work. If a transaction is successful, all of the data modifications made during the transaction are committed and become a permanent part of the database. If a transaction encounters errors and must be canceled or rolled back, then all of the data modifications are erased.

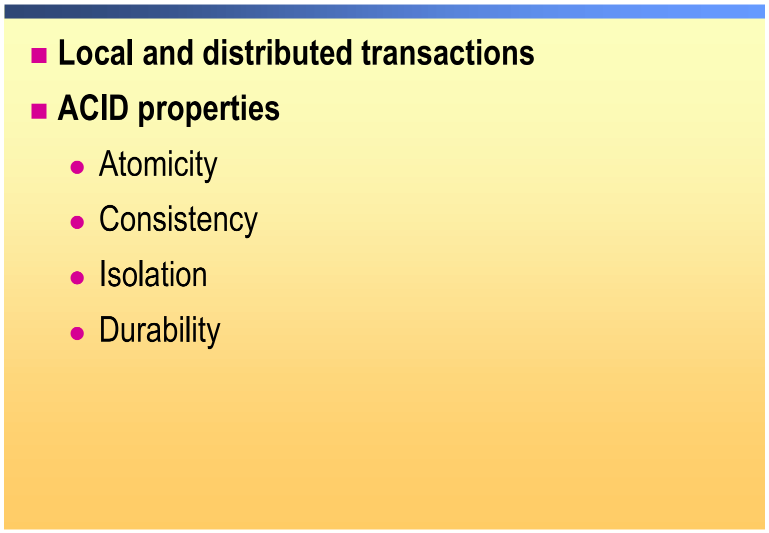
You can use transactions in ADO .NET, to ensure the consistency and integrity of the database.

Lesson objectives

After completing this lesson, you will be able to:

- Describe why transactions are important.
- Begin a transaction.
- Specify an appropriate isolation level for a transaction.
- Commit or rollback a transaction.

What Is a Transaction?



- Local and distributed transactions
- ACID properties
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Definition

A transaction is a set of related tasks that either succeed or fail as a unit. In transaction processing terminology, the transaction either *commits* or *aborts*. For a transaction to commit, all participants must guarantee that any change to data will be permanent. Changes must persist despite system crashes or other unforeseen events.

If even a single participant fails to make this guarantee, the entire transaction fails. All changes to data within the scope of the transaction are rolled back to a specific set point.

For an example of using transactions, consider the following scenario: An ASP .NET page performs two tasks. First, it creates a new table in a database. Next, it calls a specialized object to collect, format, and insert data into the new table. These two tasks are related and even interdependent, such that you want to avoid creating a new table unless you can fill it with data. Executing both tasks within the scope of a single transaction enforces the connection between them. If the second task fails, the first task is rolled back to a point before the new table was created.

Local and distributed transactions

You can create local or distributed transactions:

- Local transactions

  A local transaction is confined to a single data resource, such as a database or message queue. It is common for these data resources to provide local transaction capabilities. Controlled by the data resource, these transactions are efficient and easy to manage.

- Distributed transactions

  Transactions can also span multiple data resources. Distributed transactions enable you to incorporate several distinct operations occurring on different systems into a single pass or fail action.

ACID properties

The term ACID refers to the role transactions play in mission-critical applications. Coined by transaction-processing pioneers, ACID stands for atomicity, consistency, isolation, and durability.

These properties ensure predictable behavior, reinforcing the role of transactions as all-or-none propositions designed to reduce the management load when there are many variables.

- Atomicity

  A transaction is a unit of work in which a series of operations occur between the BEGIN TRANSACTION and END TRANSACTION statements of an application. A transaction executes exactly once and is atomic; that is, all of the work is done or none of it is.

  Operations associated with a transaction usually share a common intent and are interdependent. By performing only a subset of these operations, the system could compromise the overall intent of the transaction. Atomicity eliminates the chance of processing a subset of operations.

- Consistency

  A transaction is a unit of integrity because it preserves the consistency of data, transforming one consistent state of data into another consistent state of data.

  Consistency requires that data bound by a transaction be semantically preserved. Some of the responsibility for maintaining consistency falls to the application developer, who must make sure that all known integrity constraints are enforced by the application. For example, when developing an application that transfers money, you should avoid arbitrarily moving decimal points during the transfer.

- Isolation

  A transaction is a unit of isolation; that is, it allows concurrent transactions to behave as though each were the only transaction running in the system.

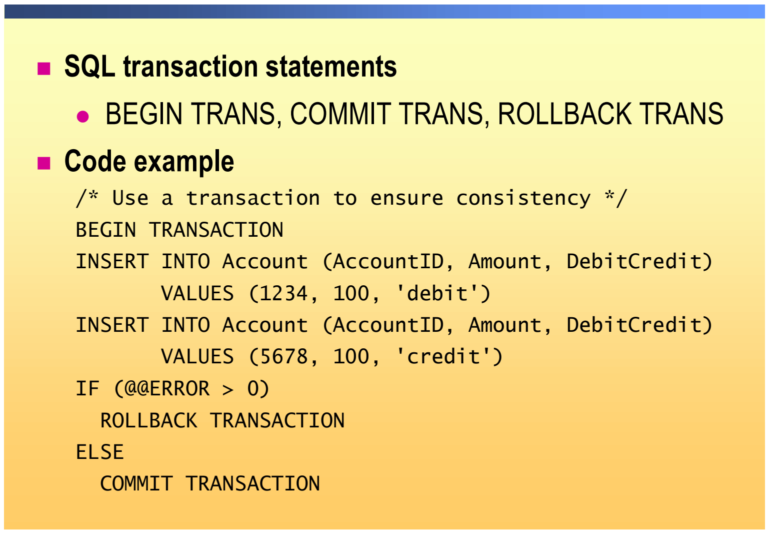
  Isolation requires that each transaction appear to be the only transaction manipulating the data store, even though other transactions may be running at the same time. A transaction should never see the intermediate stages of another transaction.

  Transactions attain the highest level of isolation when they have the ability to be serialized. At this level, the results obtained from a set of concurrent transactions are identical to the results obtained by running each transaction serially. Because a high degree of isolation can limit the number of concurrent transactions, some applications reduce the isolation level in exchange for better throughput.

- Durability

  A transaction is also a unit of recovery. If a transaction succeeds, the system guarantees that its updates will persist, even if the computer crashes immediately after the commit. Specialized logging allows the system restart procedure to complete unfinished operations, making the transaction durable.

How to Manage Transactions Using SQL



- **SQL transaction statements**
  - BEGIN TRANS, COMMIT TRANS, ROLLBACK TRANS
- **Code example**

```
/* Use a transaction to ensure consistency */
BEGIN TRANSACTION
INSERT INTO Account (AccountID, Amount, DebitCredit)
       VALUES (1234, 100, 'debit')
INSERT INTO Account (AccountID, Amount, DebitCredit)
       VALUES (5678, 100, 'credit')
IF (@@ERROR > 0)
  ROLLBACK TRANSACTION
ELSE
  COMMIT TRANSACTION
```

Introduction

Transactions can be managed at the database tier using SQL statements.

SQL transaction statements

The following table describes the SQL statements for managing transactions. You can use these statements in stored procedures, to control transactional behavior in the data tier.

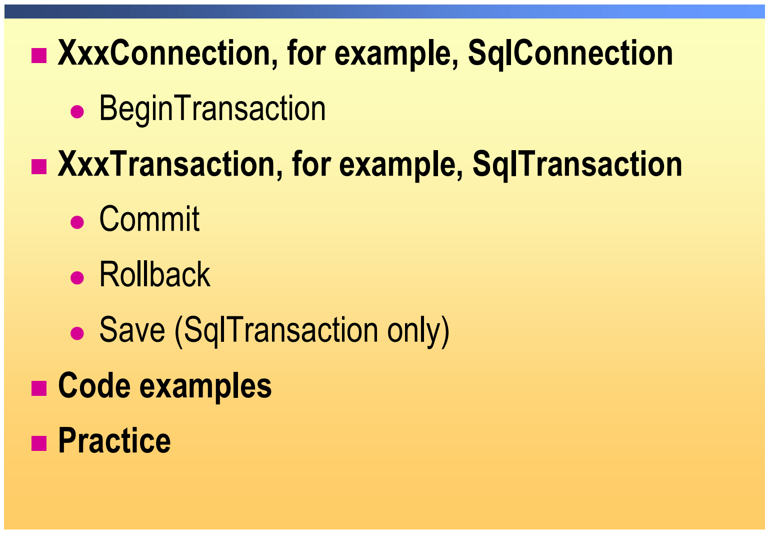
Transaction statement	Description
BEGIN TRANSACTION	Marks the beginning of the transaction. All statements executed after the BEGIN TRANSACTION statement are considered to be part of the transaction.
COMMIT TRANSACTION	Marks the end of a successful transaction, and commits all changes made since the BEGIN TRANSACTION statement.
SAVE TRANSACTION	Sets a savepoint in a transaction. The savepoint defines a location to which a transaction can return if part of the transaction is conditionally canceled.
ROLLBACK TRANSACTION	Rolls back a transaction to the beginning of the transaction, or to a savepoint in the transaction.

Example

The following example shows how to manage transactions by using Transact-SQL. The example tries to debit money from one account, and credit the money to another account. If any errors occur, the entire transaction is rolled back to ensure consistency.

```
BEGIN TRANSACTION
INSERT INTO Account (AccountID, Amount, DebitCredit)
       VALUES (1234, 100, 'debit')
INSERT INTO Account (AccountID, Amount, DebitCredit)
       VALUES (5678, 100, 'credit')
IF (@@ERROR > 0) ROLLBACK TRANSACTION ELSE COMMIT TRANSACTION
```

How to Manage Transactions Using ADO .NET


- XxxConnection, for example, SqlConnection
  - BeginTransaction
- XxxTransaction, for example, SqlTransaction
  - Commit
  - Rollback
  - Save (SqlTransaction only)
- Code examples
- Practice

Visual Basic Example

Introduction

ADO .NET allows you to manage transactions in a .NET Framework application at the middle tier. This is an alternative to performing transactions in the data tier.

The **SqlConnection** and **OleDbConnection** objects have a **BeginTransaction** method, which returns a **SqlTransaction** or **OleDbTransaction** object. The transaction object has methods named **Commit**, **Save**, and **Rollback** to manage the transaction in your application.

To perform a transaction

► **To perform a transaction using ADO .NET**

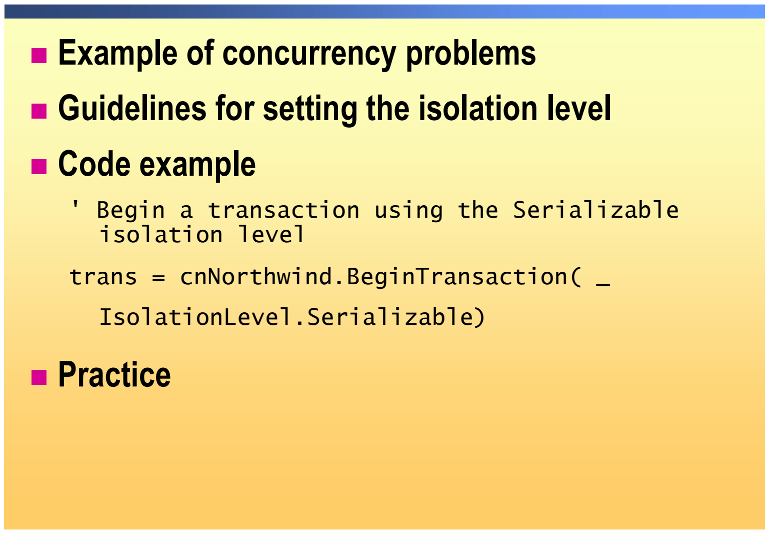
1. Call the **BeginTransaction** method of the connection object. Assign the return value to a **SqlTransaction** or **OleDbTransaction** variable.
2. For all commands that you want to execute within this transaction, set the **Transaction** property to refer to the transaction object.
3. Execute the required command objects.
4. The SQL Server .NET Data Provider allows you to specify a savepoint for the transaction. To set a savepoint, call the **Save** method at any time on the **SqlTransaction** object.
5. If the commands complete satisfactorily, call the **Commit** method on the transaction object. If any problems occur, call the **Rollback** method to roll back to the original conditions or to a savepoint.

Example

The following example uses a transaction to coordinate multiple DELETE statements in the Northwind database. The first DELETE statement deletes all items in the Order Details table, for the ProductID 42. The second DELETE statement deletes the product with ProductID 42 in the Products table. If any errors occur, the transaction is rolled back and all deletions are canceled.

```
' Open the database connection, and begin a transaction.
' Execute two DELETE statements within the transaction.
' Commit or rollback the transaction, as appropriate

cnNorthwind.Open()

Dim trans As SqlTransaction = cnNorthwind.BeginTransaction()

Dim cmd As New SqlCommand()
cmd.Connection = cnNorthwind
cmd.Transaction = trans

Try

  cmd.CommandText = _
      "DELETE [Order Details] WHERE ProductID = 42"

  cmd.ExecuteNonQuery()

  cmd.CommandText = "DELETE Products WHERE ProductID = 42"

  cmd.ExecuteNonQuery()

  trans.Commit()

Catch e As Exception

  trans.Rollback()

Finally

  cn.Close()

End Try
```

What Are Isolation Levels?



- Example of concurrency problems
- Guidelines for setting the isolation level
- Code example

```
' Begin a transaction using the Serializable
  isolation level
trans = cnNorthwind.BeginTransaction( _
  IsolationLevel.Serializable)
```

- Practice

Introduction

Isolation levels specify the transaction locking behavior for a connection. Choose an appropriate isolation level as follows to prevent concurrency problems when multiple transactions access the same data:

- At one extreme, you can allow transactions to have unimpeded access to the database. This minimizes the wait time for statements in the transactions, but increases the risk of data corruption due to concurrent access.
- At the other extreme, you can specify that transactions are completely isolated from each other. The transactions are executed serially, one after the other.

Examples of concurrency problems

If several transactions access the same data at the same time, the following concurrency errors may occur:

- Dirty reads

  A dirty read occurs when a transaction selects a row that is currently being updated by another transaction. The original transaction is reading data that has not yet been committed, and the data may be changed by the other transaction.

- Non-repeatable reads

  A non-repeatable read occurs when a transaction reads committed data once, then reads it again later and gets a different value. This happens if another transaction has updated the data between the two read operations.

- Phantom reads

  A phantom read occurs when a transaction reads data that is currently being deleted by another transaction. If the original transaction reads the data again, it will not see the deleted rows.

Guidelines for setting the isolation level

The **SqlTransaction** and **OleDbTransaction** objects have a property named **IsolationLevel**. You set this property when you call **BeginTransaction** on the connection object.

The following table describes the allowable isolation levels, in order of increasing isolation. These values are defined in the **IsolationLevel** enumeration.

Isolation level	Description
Chaos	The pending changes from more highly isolated transactions cannot be overwritten.
ReadUncommitted	Transaction isolation is only sufficient to prevent corrupt data from being read. Dirty reads, non-repeatable reads, and phantom reads can occur.
ReadCommitted	Shared locks are held while the data is being read, to prevent dirty reads. However, the data can be changed before the end of the transaction, causing non-repeatable reads or phantom reads. This is the default isolation level.
RepeatableRead	All data used in a query is locked. This prevents other users from updating the data, and therefore prevents non-repeatable reads. However, phantom reads can still occur.
Serializable	Transactions are completely isolated from each other. This prevents dirty reads, non-repeatable reads, and phantom reads.
Unspecified	A different isolation level than the one specified is being used, but the level cannot be determined

Example

The following example shows how to begin a transaction by using the **Serializable** isolation level. This ensures maximum protection against concurrency errors, at the expense of run-time performance.

```
trans = cnNorthwind.BeginTransaction( _
  IsolationLevel.Serializable)
```

Practice

► **To perform a transaction**

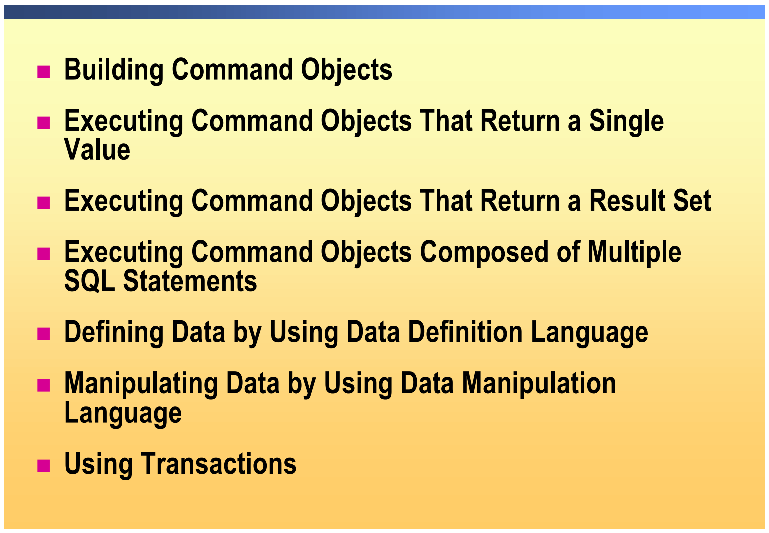
1. Start the Visual Studio .NET development environment.
2. Create a new Windows Application project named **ExecutingTransactions**.
3. Add two text boxes and a button to **Form1**
4. Drag the **InsertProduct** stored procedure in the Northwind database onto **Form1**.
5. Define a click event handler for the new button, and add the following code:
   a. Open the database connection.
   b. Create and assign a transaction to the command.
   c. Set the @ProductName parameter of the command, to the value in the first text box. Set the @CategoryID and @SupplierID parameters to **1**.
   d. Execute the command.
   e. Set the @ProductName parameter to the value of the second text box.
   f. Execute the command again.

   g. Display a message box, asking if the user wants to save the changes. If the user selects **Yes**, commit the transaction. Otherwise, roll back the transaction.

   h. Close the database connection.

6. Run and test the application.
7. Enter two product names in the text boxes, and then click the button. When you are asked if you want to commit the changes, select **Yes**. Use Server Explorer to verify that two new records have been added to the Products table.
8. Enter two different product names in the text boxes, and then click the button. When you are asked if you want to commit the changes, select **No**. Verify that neither record has been added to the Products table.

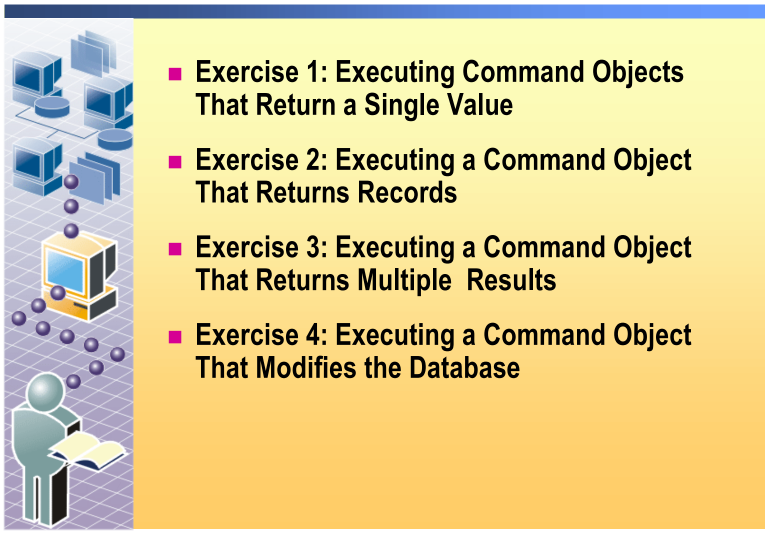
Review



- Building Command Objects
- Executing Command Objects That Return a Single Value
- Executing Command Objects That Return a Result Set
- Executing Command Objects Composed of Multiple SQL Statements
- Defining Data by Using Data Definition Language
- Manipulating Data by Using Data Manipulation Language
- Using Transactions

1.
2.
3.
4.
5.
6.

Lab 3: Performing Connected Database Operations



- Exercise 1: Executing Command Objects That Return a Single Value
- Exercise 2: Executing a Command Object That Returns Records
- Exercise 3: Executing a Command Object That Returns Multiple Results
- Exercise 4: Executing a Command Object That Modifies the Database

Objectives

After completing this lab, you will be able to:

- Build a command object.
- Execute a command object that returns a single value.
- Execute a command object that processes multiple rows.
- Execute a command object that consists of multiple SQL statements.
- Define data by using Data Definition Language.
- Manipulate data by using Data Manipulation Language.
- Use transactions.

Prerequisites

Before working on this lab, you must have:

- Visual Basic or Visual C# programming skills.
- Familiarity with the Visual Studio .NET development environment.

For more information

Search online help for the topic "Databases in Server Explorer". Hint: use quotes around the topic titles when searching the Visual Studio .NET online documentation.

Scenario

Northwind Traders has a corporate Local Area Network, which provides employees with easy access to the Northwind database. Employees need to access the information in this database, to make business decisions about which products to stock and the pricing policy for these products.

These tasks are only performed by office workers. Mobile workers do not perform these tasks. A connected Windows Application satisfies these requirements.

Estimated time to complete this lab: 60 minutes

Exercise 0
Lab Setup

To complete this lab, you must …

►

2.

3.

Exercise 1
Executing a Command Object that Returns a Single Value

In this exercise, you will create a new stored procedure in the Northwind database. The stored procedure will return the number of products in the database.

You will then open an existing Windows Application, and add a **SqlConnection** object to connect to the Northwind database. You will also add a **SqlCommand** object, to represent the new stored procedure. You will execute the stored procedure by using the **ExecuteScalar** method, and display the result in a message box.

Scenario

The Northwind database contains information about all the products stocked by Northwind Traders. Employees at Northwind Traders need to know how many products are in a specified price range. Employees can access the database over the corporate Local Area Network.

► **To add a stored procedure to get the number of products**

1. Start the Microsoft Visual Studio .NET development environment.
2. In the Server Explorer, select the Northwind database on your local machine.
3. Add a new stored procedure to the Northwind database.
4. The required code for this stored procedure is provided in the file \Program Files\MSDNTrain\2389\Labs\Lab03\Starter\CountProducts.sql. Copy this code into the stored procedure in the Visual Studio .NET code editor.
5. Save the stored procedure.

► **To add data objects to a Windows Application**

1. In Visual Studio .NET, open one of the following starter solutions:
   - If you wish to use Visual Basic, open the starter solution provided in \Program Files\MSDNTrain\2389\Labs\Lab03\Starter\VB.
   - If you wish to use Visual C#, open the starter solution provided in \Program Files\MSDNTrain\2389\Labs\Lab03\Starter\CS.
2. Open the form named **FormConnectedApp**.
3. Add a **SqlConnection** control to the form, and name it **cnNorthwind**.
4. In the Properties window for **cnNorthwind**, click **ConnectionString**. Create a New Connection with the following settings:

Property	Value
Server name	(local)
Information to log on to the server	Use Windows NT Integrated security
Database on the server	Northwind

5. Add a **SqlCommand** to the form, and set the following properties:

Property	Value
(Name)	cmCountProducts
Connection	cnNorthwind
CommandType	StoredProcedure
CommandText	dbo.CountProducts

   When you set the **CommandText** property, Visual Studio .NET asks if you want to regenerate the parameters collection for this command. Click **Yes**.

6. Review the code that was generated by Visual Studio .NET. Notice the code that creates the parameters for the command.

► **To execute the stored procedure**

1. Add an event handler method for the Click event on the Count Products button.
2. In the event handler method, get the text in the **txtMinimumPrice** and **txtMaximumPrice** text boxes. Use the **double.Parse** method to convert these strings to the **double** data type. Assign the **double** values to two local **double** variables.
3. Use these values to set the **@Min** and **@Max** parameters in the **cmCountProducts** stored procedure command.
4. Open a connection to the database.
5. Use the **ExecuteScalar** method to execute the **cmCountProducts** stored procedure command. Assign the return value of the method to a local integer variable.
6. Close the database connection.
7. Display a message box, to show the return value from the **cmCountProducts** stored procedure command.

► **To build and test the Windows Application**

1. Build the application, and correct any build errors.
2. Run the application.
3. Enter values such as 10 and 100 for the minimum and maximum prices.
4. Click Count Products, and observe the result displayed in the message box.

Exercise 2
Executing a Command Object that Returns Records

In this exercise, you will create another stored procedure in the Northwind database. The stored procedure will execute a SQL query, to obtain all the products in stock within a specified price range.

You will extend the Windows Application from Exercise 1, to call the stored procedure by using the **ExecuteReader** method. You will loop through the records using a **SqlDataReader**, and display the product details in a list box.

Note: you will extend the stored procedure in Exercise 3, to return the out-of-stock records as well.

Scenario

Employees at Northwind Traders need to obtain information about all the products currently in stock, within a specified price range.

► To add a stored procedure to return products in stock

1. In the Server Explorer, select the Northwind database on your local machine.
2. Add a new stored procedure to the Northwind database.
3. The required code for this stored procedure is provided in the file \Program Files\MSDNTrain\2389\Labs\Lab03\Starter\GetProductsInRange.sql. Copy this code into the new stored procedure in the Visual Studio .NET code editor.
4. Save the stored procedure.

► To add a SqlCommand object to represent the new stored procedure

1. Open the solution you completed in the previous exercise.
2. Add a **SqlCommand** to the form, and set the following properties:

Property	Value
(Name)	cmGetProductsInRange
Connection	cnNorthwind
CommandType	StoredProcedure
CommandText	dbo.GetProductsInRange

When you set the **CommandText** property, Visual Studio .NET asks if you want to regenerate the parameters collection for this command. Click **Yes**.

► **To execute the stored procedure**

3. Add an event handler method for the Click event on the Display Products button.
4. Clear the contents of the **lstInStock** list box.
5. Using the values in **txtMinimumPrice** and **txtMaximumPrice**, set the **@Min** and **@Max** parameters in the **cmGetProductsInRange** stored procedure command.
6. Open a connection to the database.
7. Declare a local variable named **reader**, of type **System.Data.SqlClient.SqlDataReader**.
8. Call the **ExecuteReader** method on the **cmGetProductsInRange** command. Assign the result to the **reader** variable.
9. Use **reader** to loop through the product records. For each product record, get the following column values:

Column	Code to get this column value
ProductID	reader.GetInt32(0)
ProductName	reader.GetString(1)
UnitPrice	reader.GetSqlMoney(2).ToDouble()

   For each product, add an item containing this information to the **lstInStock** list box.
10. Close **reader**.
11. Close the database connection.

► **To build and test the Windows Application**

1. Build the application, and correct any build errors.
2. Run the application.
3. Enter values such as 10 and 100 for the minimum and maximum prices.
4. Click Display Products.
5. Observe the information displayed in the in-stock list box. Note that out-of-stock list box is still empty at this stage.

Exercise 3
Executing a Command Object that Returns Multiple Results

In this exercise, you will extend the stored procedure from Exercise 2. The stored procedure will now return two results: the products in stock, and the products out of stock.

You will also extend the Windows Application from Exercise 2, to process the multiple results. You will use the **SqlDataReader** to display the in-stock products first. You will then call the **NextResult** method in **SqlDataReader**, to advance the data reader to the second result. You will loop through this result, to display the out-of-stock products.

Scenario

Employees at Northwind Traders need to know which products are in stock, and which products are out-of-stock. This enables employees to make business decisions based on current stock levels.

► **To return multiple results from a stored procedure**

1. In the Server Explorer, select the Northwind database on your local machine.
2. Open the **dbo.GetProductsInRange** stored procedure in the code editor.
3. Modify the stored procedure, so that it returns two results:
   - The in-stock products (within the specified price range). Note: the SELECT statement to do this already exists in the stored procedure.
   - The out-of-stock products (within the specified price range).

   The complete code for the stored procedure is provided in the file \Program Files\MSDNTrain\2389\Labs\Lab03\Starter\GetMultipleResults.sql
4. Save the stored procedure.

► **To process multiple results**

1. Open the solution you completed in the previous exercise.
2. Find the event handler method for the Click event on the Display Products button.
3. After the **lstInStock** list box has been populated with the first result, call the **NextResult** method on the **reader** object.
4. Clear the contents of the **lstOutOfStock** list box.
5. Use **reader** to loop through the out-of-stock products. For each record, get the following column values:

Column	Code to get this column value
ProductID	reader.GetInt32(0)
ProductName	reader.GetString(1)
UnitPrice	reader.GetSqlMoney(2).ToDouble()

   For each product, add an item containing this information to the **lstOutOfStock** list box.

► **To build and test the Windows Application**

6. Build the application, and correct any build errors.
7. Run the application.
8. Enter values such as 10 and 100 for the minimum and maximum prices.
9. Click Display Products.
10. Observe which products are in stock, and which products are out-of-stock.

Exercise 4
Executing a Command Object that Modifies the Database

In this exercise, you will write a stored procedure to create an **OrderSummary** table in the Northwind database. The stored procedure will populate the table with the total number of orders for each product.

You will also write a stored procedure to query the data in the **OrderSummary** table.

In your Windows Application, you will use the **ExecuteNonQuery** method to execute the first stored procedure. You will use the **ExecuteQuery** method to execute the second stored procedure, and create a **SqlDataReader** to loop through the result.

Scenario

The Orders table in the Northwind database contains information for each customer order. The details of each order are held in the Order Details table. The Order Details table indicates the quantity required for each product in the order.

Employees at Northwind Traders need a summary of the total number of orders for each product. This information will help Northwind Traders identify its most popular products, so that the company can offer the best possible service to its customers.

► **To add a stored procedure to create and fill the OrderSummary table**

1. In the Server Explorer, select the Northwind database on your local machine.
2. Add a new stored procedure to the Northwind database.
3. The required code for this stored procedure is provided in the file \Program Files\MSDNTrain\2389\Labs\Lab03\Starter\SummarizeOrders.sql. Copy this code into the new stored procedure in the Visual Studio .NET code editor.
4. Save the stored procedure.

► **To add a stored procedure to query the OrderSummary table**

1. Add another new stored procedure to the Northwind database.
2. The required code for this stored procedure is provided in the file \Program Files\MSDNTrain\2389\Labs\Lab03\Starter\GetOrderSummary.sql. Copy this code into the new stored procedure in the Visual Studio .NET code editor.
3. Save the stored procedure.

► To add SqlCommand objects to represent the new stored procedures

1. Open the solution you completed in the previous exercise.
2. Add a **SqlCommand** to the form, and set the following properties:

Property	Value
(Name)	cmSummarizeOrders
Connection	cnNorthwind
CommandType	StoredProcedure
CommandText	dbo.SummarizeOrders

   When you set the **CommandText** property, Visual Studio .NET asks if you want to regenerate the parameters collection for this command. Click **Yes**.

3. Add another **SqlCommand** to the form, and set the following properties:

Property	Value
(Name)	cmGetOrderSummary
Connection	cnNorthwind
CommandType	StoredProcedure
CommandText	dbo.GetOrderSummary

   When you set the **CommandText** property, Visual Studio .NET asks if you want to regenerate the parameters collection for this command. Click **Yes**.

► To execute the stored procedures

1. In the Windows Form Designer, click the Product Orders tab on your form.
2. Add an event handler method for the click event on the Summarize Orders button.
3. In the event handler method, clear the contents of the **lstOrderSummary** list box.
4. Open a connection to the database.
5. Call the **ExecuteNonQuery** method on the **cmSummarizeOrders** command.
6. Declare a local variable named **reader**, of type **System.Data.SqlClient.SqlDataReader**.
7. Call the **ExecuteReader** method on the **cmGetOrderSummary** command. Assign the result to the **reader** variable.
8. Use **reader** to loop through the records. For each record, get the following column values:

Column	Code to get this column value
Orders	reader.GetInt32(0)
ProductName	reader.GetString(1)

   For each record, add an item containing this information to the **lstOrderSummary** list box.

9. Close **reader**.
10. Close the database connection.

► **To build and test the Windows Application**

1. Build the application, and correct any build errors.
2. Run the application.
3. Click the Product Orders tab on the form.
4. Click Summarize Orders.
5. Observe the total number of orders for each product.

msdn® training

Module 4: Buidling DataSets (Prerelease)

Contents

Microsoft®

Instructor Notes

Presentation:
60 Minutes

Lab:
60 Minutes

This module teaches students how to build and manage DataSets, define data relationships, modify data, and use DataViews. Because practices in this module build on files built during Lesson 1, the starter file for Lesson 2 is the solution file for Lesson 1. Lesson 3 and practices in other lessons also build upon each other.

After completing this module, students will be able to:

- Build a DataSet and a DataTable.
- Bind a DataSet to a DataGrid.
- Create a custom DataSet by using inheritance.
- Define a data relationship.
- Modify data in a DataTable.
- Sort and filter a DataTable by using a DataView.

Required materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2389A_04.ppt
- Module 4, "Building DataSets"
- Lab 4.1, Building, Binding, Opening, and Saving DataSets
- Lab 4.2, Manipulating DataSets and Modifying Data

Preparation tasks

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices and labs.
- Read the latest .NET Development news at http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000519

Classroom setup

The information in this section provides setup instructions that are required to prepare the instructor computer or classroom configuration for a lab.

► To prepare for the lab

1.
2.

How to Teach This Module

This section contains information that will help you to teach this module.

Lesson: Building DataSets and DataTables

This section describes the instructional methods for teaching each topic in this lesson.

What Are DataSets, DataTables, and DataColumns?

Technical Notes:

- This module focuses on defining a DataSet programmatically by using the object model. In the real world, developers are likely to spend more time using XML Schema Definitions (XSD) to define the initial schema, and use code for applications that need to be more dynamic. Using XSD to define the initial schema is covered in Module 6, "NAME," in Course 2389A, *Programming with ADO.NET*.

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What type of application do you plan to create in which you can use DataSets?
- Why are DataSets and the disconnected environment suited to that type of application?

How to Create a DataSet, a DataTable, and a DataColumn

Technical Notes:

- Show students how to create a DataSet, a DataTable, and a DataColumn in Microsoft Visual Basic® and Microsoft Visual C# by clicking on the Visual Basic Example or C# Example buttons on the PowerPoint slide for this topic.
- In the sample for creating DataColumns, point out the use of the **GetType** statement in Visual Basic and the use of the **typeof** statement in C#. It is possible to use System.Type.GetType("System.Int32") to do the same thing in both languages if users want to be consistent, but the Microsoft Visual Studio .NET tools will use the shorthand syntax.
- The System.Data.DbType enumeration can be used to list all of the data types available for data columns and parameters.

Discussion Questions: Personalize the following questions to the background of the students in your class.

- When creating ADO.NET objects, why might you want to separate the declaration statement from the instantiation statement? When would you want to combine these statements?
- What other exceptions might occur that you will have to control when creating DataTables and DataColumns?

Transition to Practice Exercise: Now that you have seen several examples of building DataSets and DataTables programmatically, you can now practice creating a DataSet and a DataTable.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Practice Solution:

1. In Visual Studio .NET, start a new Microsoft Windows® Application project.
2. Drag a DataSet from the **Data** section of the **Toolbox** onto the form.
3. Use the Property Window to rename the DataSet. For the **(Name)** property, use **dsNorthwind**. For the **DataSetName** property, use **Northwind**.
4. Use the collection builder to add a DataTable to the **Tables** property of the DataSet.
5. For the **(Name)** property, use **dtProducts.** For the **TableName** property, use **Products**.
6. Use the collection builder to add three DataColumns to the **Columns** property of the DataTable.
7. For the **(Name)** property, use **dcProductID.** For the **ColumnName** property, use **ProductID**.
8. For the **DataType** property, choose **System.Int32**.
9. Repeat steps 7 and 8 for the ProductName and UnitPrice fields.
10. Use the code editor to manually write code to create the fourth field. Hint: Use the code that is automatically generated for the UnitPrice field as a guide; you can also copy and paste the code and then edit it.
11. Return to the form view and use the Property Window to verify that the designer recognizes the new code that you have added. If not, check that you modified the call to the **AddRange** method for the DataTable that adds references to the DataColumns.

After the practice

Questions for discussion after the practice:

- What lessons did you learn during the practice exercise?
- What did you discover as you created the DataSet, DataTable, and DataColumns?

Using Unique Constraints

Technical Notes:

- Visual Studio .NET Beta 2 contains a bug that prevents the interface from writing the code for constraints. Therefore, you must write the code for constraints manually.
- Although you can add code in the same place where the automatically generated code would be generated, using the graphical tools again will delete that code.

Transition to Practice Exercise: Using the example I just showed you as reference, you can practice programmatically creating a unique constraint for the Northwind DataSet.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Practice Solution:

- In Visual Studio .NET, insert code in the **Form1_Load** event to create a unique constraint for the product name column.

Quick solution:

```
Me.dcProductName.Unique = True
```

Recommended solution:

```
Me.dtProducts.Constraints.Add( _
New UniqueConstraint("UC_ProductName", Me.dcProductName))
```

After the practice

Questions for discussion after the practice:

- Why do you think the recommended solution is preferred over the quick solution?
- What lessons did you learn during the practice exercise?
- What did you discover as you created the unique constraint?
- How will you use unique constraints in your data applications?

Using AutoIncrement Columns

Technical Notes:

- AutoIncrement columns cause problems when used with disconnected data that needs to be merged with a central database, because conflicts are likely to occur. The System.Guid and System.Data.SqlTypes.SqlGuid structures can be used as an alternative if the underlying data source uses globally unique identifiers (GUID) in the table.

Discussion Questions: Personalize the following questions to the background of the students in your class.

- Why would AutoIncrement columns cause problems when multiple users add new rows?
- How does the GUID data type solve these problems?
- What else could you do in your applications to prevent conflicts caused by multiple users adding new rows?

Creating Custom Expressions

Technical Notes:

- Custom expressions can reference other columns in the table. They can also use summary functions such as **Count** and **Sum** that apply to all of the values in the specified column. If a DataRelation for a parent table exists, the summary functions can be grouped by parent row by using the **Child** object; otherwise the entire table groups them. For example:
  `=Sum(Child.UnitPrice)`
  `=Sum(UnitPrice)`
  In the first example, the UnitPrice values are grouped by the parent; in the second they are grouped by the table.

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What are some other examples of situations where an expression column would be used?
- Could an expression column be used for concatenation of column values?

Transition to Practice Exercise: Using the syntax printed in the student workbook, you can create a custom expression for the Northwind DataSet.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Practice Solution:

1. In Visual Studio .NET, use the Property Window to add the new column to the **Products** table,

   –or–

2. Write code to add a fifth column with an expression, as shown in the following example:

```
‘ Visual Basic

Dim dcStockValue As New System.Data.DataColumn( _
   "StockValue", GetType(System.Decimal))
dcStockValue.Expression = "UnitPrice*UnitsInStock"
dcStockValue.ReadOnly = True
```

After the practice

Questions for discussion after the practice:

- What lessons did you learn during the practice exercise?
- How can you use expression columns in the applications that you are building at your job?

Lesson: Binding a DataSet to a Windows Application Control

This section describes the instructional methods for teaching each topic in this lesson.

How to Bind Data to a Windows Control

Technical Notes:

- Almost all controls have a **(DataBindings)** property with an **(Advanced)** sub-property that allows any column to be bound to any control property. This is much more flexible than previous data access models that typically only allowed the **Caption** or **Text** properties to be bound.

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What is the advantage of being able to bind a DataSet to a Windows control? Give an example of using this feature in an application.

Instructor Demonstration:

Demonstrate the programmatic and graphical procedures for creating a simple data-bound control. You can choose to ask students to watch you demonstrate this, or instruct them to follow the procedures on their computers as you talk through the steps.

How to Bind a DataSet to a DataGrid

Technical Notes:

- Visual Studio .NET Beta 2 contains a bug that requires the **DataGrid** control to be bound to a DataSet through a DataView rather than directly, for reliable operation. Without an intermediate DataView, the DataGrid cannot track the current filter and sort options set for the data, as well as changes to the data.

Transition to Practice Exercise:

Now that you have seen how to bind a DataSet to a DataGrid both programmatically and by using the graphical tools, choose the method that you would like to use, and turn to the practice exercise at the end of this topic in the student workbook.

Practice Solution:

1. Open the Visual Studio .NET development environment and the project that includes the Northwind DataSet.
2. Add five **TextBox** controls and a DataGrid control to the form.
3. Use the Property Window to set the DataBindings for the **TextBox** controls. Bind the **Text** property of each box to the five columns in the **Products** table in the Northwind DataSet.

Caution Do not bind to the dtProducts variable. If you do, you will only see the first record.

4. Set the DataSource for the DataGrid to the Northwind DataSet.
5. Set the DataMember of the DataGrid to the DataTable. Notice that the **TextBox** controls display the same information as the currently selected row in the DataGrid.
6. Notice that you cannot have two records with the same ProductName, but that you can have two records with the same ProductID.

After the practice

Questions for discussion after the practice:

- How many of you used the graphical tools to bind the DataSet to the DataGrid, and how many of you did it programmatically?
- What are the differences between binding a DataSet to a simple Windows control and binding a DataSet to a DataGrid?

Lesson: Creating a Custom DataSet

This section describes the instructional methods for teaching each topic in this lesson.

Benefits of Inheritance

Discussion Questions: Personalize the following question to the background of the students in your class.

- Name an example of a situation where you could use inheritance in your applications.

How to Create a Custom DataSet by Using Inheritance

Instructor Demonstration:

Demonstrate how to create a custom DataSet by using the **Inherits** statement in Visual Basic and C#. Depending on the needs of your audience, you might only need to demonstrate in one language. You can choose to ask students to watch while you demonstrate, or instruct them to follow the procedures on their computers as you talk through the steps.

Transition to Practice Exercise:

Now that you understand how to use the **Inherits** statement, you can use it to create a custom DataSet. In this practice exercise, we are going to inherit from an existing DataSet. First I'll open that DataSet by using Notepad, so that we can see what it contains.

Open the DataSet file located at <install folder>\Practices\Mod04_1\catprodnone.ds. The Categories and Products tables are both defined with their structures and data in this file.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook to create a custom DataSet based on inheritance from the **catprodnone** DataSet.

Practice Solution:

1. Create a Windows Application project, and add a new class called **CatProdDataSet**. Add the following code, changing the path to point to the correct file:

   ```
   Public Class CatProdDataSet
      Inherits System.Data.DataSet

      Public Sub New()
          Me.ReadXml("<install
   folder>\Practices\Mod04_1\catprodnone.ds",
   XMLReadMode.ReadSchema)
      End Sub
   End Class
   ```

2. Add a DataGrid to the form. Add code to the **Form1_Load** event that creates an instance of the **CatProdDataSet** class and then binds the **Categories** table to the existing DataGrid.
3. Run and test the application. You should see a list of eight categories in the grid, but no way to access the products.

After the practice

Questions for discussion after the practice:

- Why is there no way to access the products? What do you need to do to solve this problem?
- We will talk about DataSets and XML later in this course, but what do you think the **ReadXml** method is doing in this example?

Lesson: Defining Data Relationships

This section describes the instructional methods for teaching each topic in this lesson.

How to Create a Primary Key Constraint

Discussion Questions: Personalize the following questions to the background of the students in your class.

- When would you use the **PrimaryKey** property to define a primary key versus using the unique constraint?
- Describe an example of a situation where two columns need to be defined as a primary key.

Using Foreign Key Constraints to Restrict Actions

Discussion Questions: Personalize the following questions to the background of the students in your class.

- In what situations would you set the **DeleteRule** property to **SetNull**? When would you set it to **SetDefault**?
- How would you handle the exception raised when an update is made and you have set the **DeleteRule** property to **None**?

How to Create a Foreign Key Constraint

Discussion Questions: Personalize the following question to the background of the students in your class.

- With the **DeleteRule** property set to **None**, what will happen if you try to delete a customer who has orders in the Orders table? What would we need to do in this situation?

What Is a DataRelation Object?

Technical Notes:

- It is important to differentiate between a ForeignKeyConstraint (maintains data integrity) and a DataRelation (provides navigation, grouping, and so on).

Discussion Questions: Personalize the following questions to the background of the students in your class.

- How would a DataRelation be used with primary key and foreign key constraints?
- Could you use primary key and foreign key constraints without using a DataRelation? Why would you want to do this, or why not?

How to Create a DataRelation Object

Discussion Questions: Personalize the following questions to the background of the students in your class.

- You can use either the DataRelation constructor or the **Add** method. Why would you use one or the other? What is the difference in results between the two?

Transition to Practice Exercise: You can now choose to use either the DataRelation constructor or the **Add** method and practice creating constraints and a DataRelation object.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Practice Solution:

In Visual Studio .NET, open the file that contains the **DSCatProd** DataSet, and define the following relationships for the **DSCatProd** DataSet on the **Form1_Load** event:

1. Create a **PrimaryKey** on the **CategoryID** column for the **Categories** table.

   ```
   With ds.Tables("Categories")
      .Constraints.Add("PK_Categories", _
          .Columns("CategoryID"), True)
   End With
   ```

2. Create a **PrimaryKey** on the **ProductID** column for the **Products** table.

   ```
   With ds.Tables("Products")
      .Constraints.Add("PK_Products", _
          .Columns("ProductID"), True)
   End With
   ```

3. Create a DataRelation and **ForeignKeyConstraint** between **Categories** and **Products**. (Copy and paste code to create these DataTables.)

   ```
   ds.Relations.Add("FK_CategoriesProducts", _
      ds.Tables("Categories").Columns("CategoryID"), _
      ds.Tables("Products").Columns("CategoryID"), _
      True)
   ```

How to Navigate Related DataTables

Discussion Questions: Personalize the following question to the background of the students in your class.

- How can you make use of navigating related DataTables when you begin to modify data in a DataSet?

Lesson: Modifying Data in a DataTable

This section describes the instructional methods for teaching each topic in this lesson.

How to Insert a New Record

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What is the advantage of calling the **Add** method and passing an array of values typed as Object?
- What would happen if you created a new record but did not add it to the **DataRowCollection**?

How to Position on a Record

Discussion Questions: Personalize the following question to the background of the students in your class.

- Why do you think that navigation through records is managed by the data-binding layer?

Modifying Data in a Table

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What is the difference between the **EndEdit** and **CancelEdit** methods?
- How could you programmatically use the BeginEdit and EndEdit methods to modify multiple records?

How to Delete a Record

Discussion Questions: Personalize the following questions to the background of the students in your class.

- If you used the **Delete** method of the DataRow object and marked the row for deletion, what would you then need to do to permanently delete the row?
- When would you use the **Delete** method of the DataRow object?
- What precautions might you want to take when using the **Remove** method of the **DataRowCollection** object?

How to Handle the DataTable Events

Discussion Questions: Personalize the following questions to the background of the students in your class.

- How are handling DataTable events and setting the **DeleteRule** property on the foreign key constraint different? When would you need to set the **DeleteRule** property, and when would you need to handle a DataTable event?
- Give a business use example of why you would want to programmatically handle DataTable events in a business application.

Transition to Practice Exercise: Using the example I just showed you as reference, you can practice handling the **ColumnChanging** DataTable event in the Northwind DataSet.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Practice Solution:

In Visual Studio .NET, handle the **ColumnChanging** DataTable event by displaying a message box that shows the proposed new value of a modified row in the Northwind DataSet **Products** DataTable.

1. Declare a DataTable variable and set it to the **Products** table in the DataSet so that it can handle events.
2. Add the following code to the **ColumnChanging** event:

```
MessageBox.Show("From: " & e.Row.Item(e.Column) & _
", To: " & e.ProposedValue.ToString(), _
e.Column.ColumnName)
```

After the practice

Questions for discussion after the practice:

- How would you change your code to handle the **ColumnChanged** event?
- How would you change your code to handle the **RowChanging** event?

What Are the RowState and RowVersion Properties?

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What can you deduce about the row by using these properties?
- Can you think of any real-world situations where you might want to use the **RowState** and **RowVersion** properties?

How to Accept or Reject Changes

Discussion Questions: Personalize the following question to the background of the students in your class.

- How will accepting or rejecting changes affect updating data in the data source?

Lesson: Using a DataView

This section describes the instructional methods for teaching each topic in this lesson.

What Is a DataView?

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What might be some of the differences between a DataView and a view in Microsoft SQL Server™?
- Give another example of a situation where a DataView would be useful in your applications.

How to Define a DataView

Discussion Questions: Personalize the following question to the background of the students in your class.

- What data results would you see if you ran the code example for programmatically creating a DataView?

How to Sort and Filter a DataTable Using a DataView

Discussion Questions: Personalize the following questions to the background of the students in your class.

- Give an example of a situation where you might want to filter based on the version or state of a record? Can you sort based on the version or state of a record?
- Why would you use the default DataView?

Transition to Practice Exercise: Using the examples I just showed you as a reference, you can practice sorting and filtering by using a DataView.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Practice Solution:

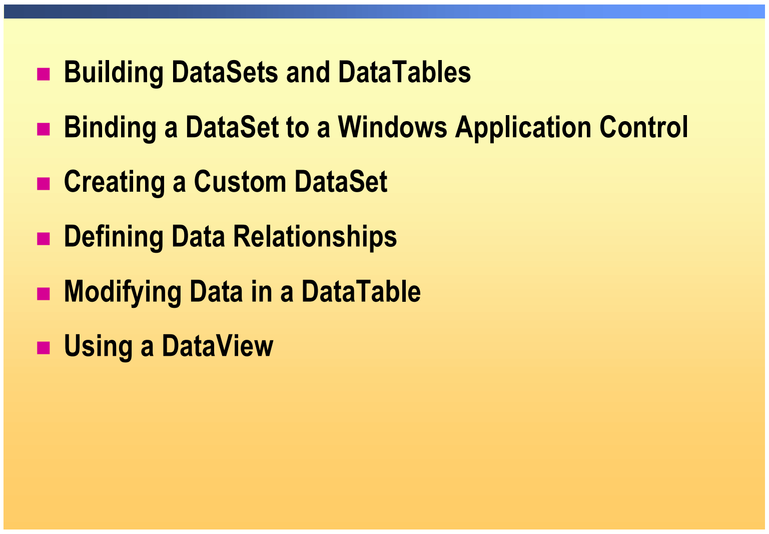
- Using the Visual Studio .NET development environment, build a DataView for the **Products** DataTable in the Northwind DataSet.

After the practice

Questions for discussion after the practice:

- What are some of the different DataViews that you created?
- Did any of you sort and filter the Products DataTable by using the DataView? What kinds of filters did you create?

Overview



- Building DataSets and DataTables
- Binding a DataSet to a Windows Application Control
- Creating a Custom DataSet
- Defining Data Relationships
- Modifying Data in a DataTable
- Using a DataView

Introduction

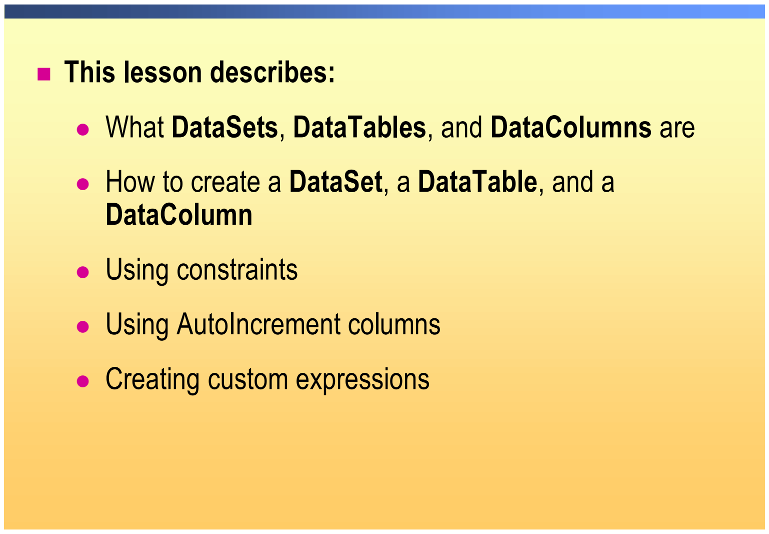
This module presents the concepts and procedures you need to create and use **DataSet**s and related objects. **DataSet**s allow you to store, manipulate, and modify data in a local cache while disconnected from the data source.

Objectives

After completing this module, you will be able to:

- Build a **DataSet** and a **DataTable**.
- Bind a **DataSet** to a **DataGrid**.
- Create a custom **DataSet** by using inheritance.
- Define a data relationship.
- Modify data in a **DataTable**.
- Find and select rows in a **DataTable**.
- Sort and filter a **DataTable** by using a **DataView**.

Lesson: Building DataSets and DataTables



- **This lesson describes:**
  - What **DataSets**, **DataTables**, and **DataColumns** are
  - How to create a **DataSet**, a **DataTable**, and a **DataColumn**
  - Using constraints
  - Using AutoIncrement columns
  - Creating custom expressions

Introduction

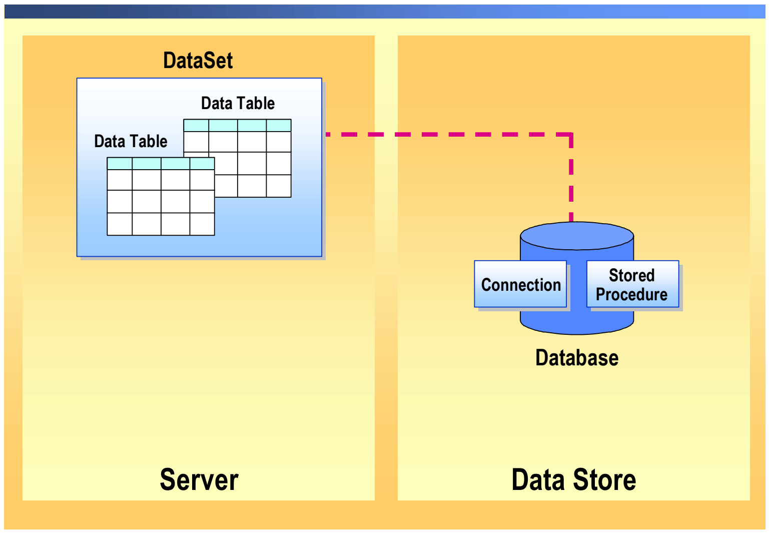
This lesson explains what **DataSet**s, **DataTable**s, and **DataColumn**s are, how to create them programmatically, and how to include exception handling, constraints, AutoIncrement columns, and custom expressions in your Microsoft® ActiveX® Data Objects (ADO) .NET **DataSet**.

Lesson objectives

After completing this lesson, you will be able to:

- Explain what **DataSet**s, **DataTable**s, and **DataColumn**s are.
- Create a **DataSet** and a **DataTable**.
- Use unique constraints.
- Create AutoIncrement columns.
- Create custom expressions.

What Are DataSets, DataTables, and DataColumns?



Introduction

In ADO .NET, **DataSet**s, **DataTable**s, and **DataColumn**s allow you to represent data in a local cache and provide a relational programming model for the data regardless of its source.

Definitions

The ADO .NET **DataSet** is an in-memory cache of data and functions as a disconnected relational view of the data. The connection to the data source does not need to be active to view and manipulate data in a **DataSet**. This disconnected architecture enables greater scalability by using database server resources only when reading from or writing to the data source.

DataSets store data similarly to the way data is stored in a relational database with a hierarchical object model of tables, rows, and columns. Additionally, you can define constraints and relationships for the data in the **DataSet**.

DataTable objects are used to represent the tables in a **DataSet**. A **DataTable** represents one table of in-memory relational data; the data is local to the .NET application in which it resides, but it can be populated from an existing data source. A **DataTable** is composed of **DataColumns**.

A **DataColumn** is the building block for creating the schema of a **DataTable**. Each **DataColumn** has a **DataType** property that determines the kind of data that each **DataColumn** contains. For example, you can restrict the data type to integers, strings, or decimals. Because data contained in the **DataTable** is typically merged back into the original data source, you must match the data types to those in the data source.

DataSets and XML

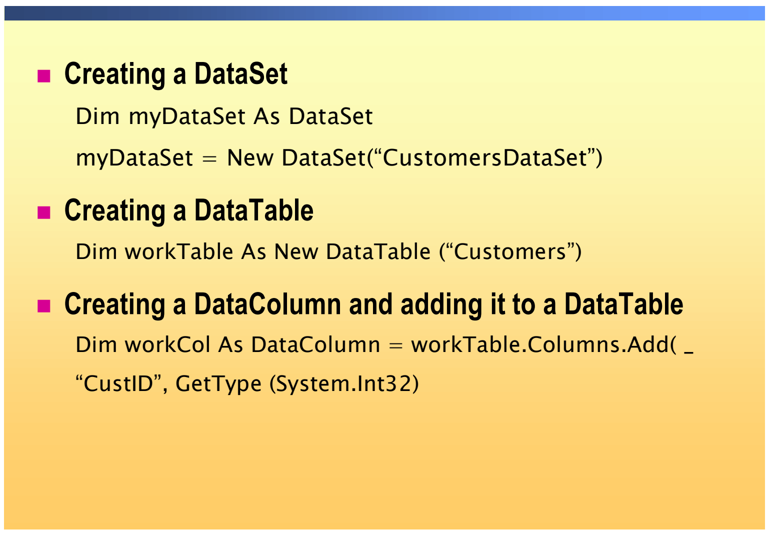
DataSets represent data in a relational view regardless of its source. However, data in a **DataSet** can be represented in XML format. The integration of **DataSets** with XML allows you to define the structure of a **DataSet** schema. For more information about the relationship between **DataSet**s and XML, see Module 5, “Using XML With ADO .NET,” in Course 2389A, *Programming with ADO .NET*.

The System.Data namespace

To create a **DataSet** or manipulate data in a **DataSet**, you use the following classes in the System.Data namespace:

- **System.Data.DataSet**
- **System.Data.DataTable**
- **System.Data.DataColumn**
- **System.Data.Constraint**
- **System.Data.DataRelation**
- **System.Data.DataRow**
- **System.Data.DataView**

How to Create a DataSet, a DataTable, and a DataColumn



- **Creating a DataSet**

```
Dim myDataSet As DataSet
myDataSet = New DataSet(“CustomersDataSet”)
```

- **Creating a DataTable**

```
Dim workTable As New DataTable (“Customers”)
```

- **Creating a DataColumn and adding it to a DataTable**

```
Dim workCol As DataColumn = workTable.Columns.Add( _
“CustID”, GetType (System.Int32)
```

Introduction

You can create **DataSet**s and **DataTable**s in the following ways:

- Programmatically
- By using the graphical tools in the Microsoft Visual Studio® .NET development environment
- By using a **DataAdapter** and filling the **DataSet** with data from a relational data source
- By loading and persisting **DataSet** contents using XML

In this topic, you will learn how to create a **DataSet** and a **DataTable** programmatically, and by using the graphical tools in the Visual Studio .NET development environment.

For information about filling a **DataSet** by using a **DataAdapter**, see Module 6, “Building DataSets From Existing Sources,” in Course 2389A, *Programming with ADO .NET.* For information about loading and persisting data in a **DataSet** by using XML, see Module 5, “Using XML with ADO .NET,” in Course 2389A, *Programming with ADO .NET*.

The DataSet and DataTable constructors

To create a **DataSet** and a **DataTable** programmatically, you use the **DataSet** constructor to initialize a new instance of the **DataSet** class, and use the DataTable constructor to initialize a new instance of the **DataTable** class. You can name the **DataSet** or, if the name is omitted, the name is set by default to **NewDataSet**.

The **DataSet** must have a name to ensure that its XML representation always has a name for the document element, which is the highest-level element in an XML Schema definition.

You can create a **DataTable** object by using the DataTable constructor, or by passing constructor arguments to the **Add** method of the **DataSet** object’s **Tables** property, which is a **DataTableCollection**.

You can set parameters for a **DataTable** or a DataColumn constructor at the time that the **DataTable** or **DataColumn** is created. This is recommended because you can create the **DataTable** and define parameters for it by using only one line of code.

After you have added a **DataTable** as a member of the Tables collection of one **DataSet**, you cannot add it to the collection of tables of any other **DataSet**. You can use the **Clone** method of a **DataTable** to create a new **DataTable** with the same structure (but no data), or you can use the **Copy** method to create a new **DataTable** with the same structure and data.

Example

The following example programmatically creates a **DataSet** named Northwind with a variable called dsNorthwind that can be used to reference it.

```
Dim dsNorthwind As DataSet
dsNorthwind = New DataSet("Northwind")
```

Example

The following example creates an instance of a **DataTable** object and assigns it the name Customers.

```
Dim dtCustomers As New DataTable("Customers")
```

When creating most ADO .NET objects, you can separate the declaration statement from the instantiation statement (as shown in the preceding **DataSet** example) or combine the statements (as shown in the preceding **DataTable** example).

Note that the **DataTable**s created above are not yet associated with a **DataSet**.

Creating a DataTable programmatically

► **To add a DataTable to a DataSet programmatically**

```
dsNorthwind.Tables.Add(dtCustomers)
```

The following example is the simplest way to create both a **DataSet** and an associated **DataTable**. The code creates an instance of a **DataTable** by adding it to the Tables collection of a newly created **DataSet**.

```
Dim dsNorthwind As New DataSet("Northwind")
Dim dtCustomers As DataTable = _
  dsNorthwind.Tables.Add("Customer")
```

You are not required to supply a value for the **TableName** property when you create a **DataTable**. You can specify the **TableName** property at another time, or you can leave it empty. However, when you add a table without a **TableName** value to a **DataSet**, the table is given an incremental default name of Table*N*, starting with “Table” for Table0.

Creating DataColumns

When you first create a **DataTable**, it does not have a schema. To define the table’s schema, you must create and add **DataColumn** objects to its Columns collection.

You create **DataColumn** objects within a table by using the DataColumn constructor or by calling the **Add** method of the table’s **Columns** property. The **Add** method will either accept optional **ColumnName**, **DataType**, and **Expression** arguments and create a new **DataColumn** as a member of the collection, or it will accept an existing **DataColumn** object and add it to the collection.

Example

The following examples add a column to a **DataTable** using Visual Basic and Visual C#. Notice the use of the **typeof** statement in Visual C# and the **GetType** statement in Visual Basic.

```
' Visual Basic
Dim colCustomerID As DataColumn = _
  dtCustomers.Columns.Add("CustomerID", _
  GetType(System.Int32))

colCustomerID.AllowDBNull = False
colCustomerID.Unique = True


// Visual C#
DataColumn colCustomerID =
  dtCustomers.Columns.Add("CustomerID",
  typeof(Int32));

colCustomerID.AllowDBNull = false;
colCustomerID.Unique = true;
```

Handling exceptions

You will need to programmatically control any exceptions that occur when creating a **DataSet** and a **DataTable**. **DataTable** names must be unique, so that an exception will be thrown when duplicate table names are used. The following example shows how to handle duplicate name exceptions programmatically.

```
Try
  dtCustomers = dsNorthwind.Tables.Add("Customers")

Catch DupXcp As System.Data.DuplicateNameException

  MessageBox.Show("A table called Customers already exists!")

...

End Try
```

Practice

Northwind Traders needs to build an application that includes data that is related to its products. Use the Visual Studio .NET development environment graphical tools to build a Windows Application solution.

1. Create a new Windows Application solution named **CreateDataSets** in the following location.

   <install folder>\Practices\Mod04_1\

2. Drag and drop a **DataSet** control from the toolbox to the form. Name it **Northwind**.
3. Use the Property Window of the **DataSet** to build a **DataTable** called **Products** with fields called **ProductID**, **ProductName**, and **UnitPrice**.
4. Give each of the fields appropriate data types.
5. Open the form in Code view and find the code that was written for you by the design tools.
6. Use the automatically generated code as a guide to write some new code that creates a fourth column named **UnitsInStock** programmatically.

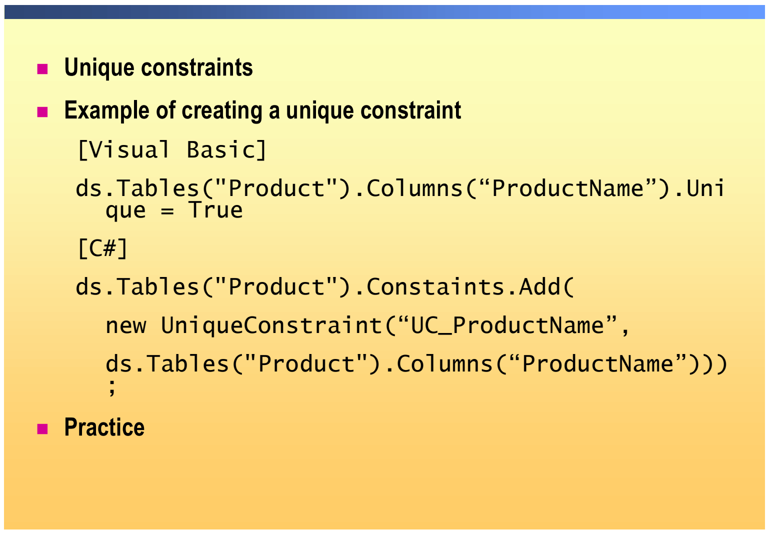
7. Add a reference to your new column to the end of the call to the AddRange method of dtProducts. This will allow the design tools to recognize your new column.

   ```
   Me.dtProducts.Columns.AddRange(...
   ```

8. Return the form to Designer view and use the Property Window to see that the code you have written manually is recognized by the design tools.

The solution for this practice is located at <install folder>\Practices\Mod04_1\Lesson1\CreateDataSets\.

Using Unique Constraints



- Unique constraints
- Example of creating a unique constraint

```
[Visual Basic]
ds.Tables("Product").Columns(“ProductName”).Uni
  que = True
[C#]
ds.Tables("Product").Constaints.Add(
  new UniqueConstraint(“UC_ProductName”,
  ds.Tables("Product").Columns(“ProductName”))
  ;
```

- Practice

Introduction

A relational database must enforce data integrity to ensure the quality of the data in the database. One way to maintain integrity in ADO .NET is by adding constraints to the tables within a **DataSet**. A *constraint* is an automatic rule applied to a column, or related columns, which determines what actions should take place when the value of a row is modified.

There are two kinds of constraints in ADO .NET: the ForeignKeyConstraint and the UniqueConstraint. When you add a **DataRelation** object, which creates a relationship between two or more tables, both constraints can be created automatically. Constraints are not enforced unless the **EnforceConstraints** property of the **DataSet** is set to **true**. Foreign key constraints are primarily intended for use with relationships to primary key columns, and will be discussed later in this module.

Unique constraints

The **UniqueConstraint** object, which can be assigned either to a single column or to an array of columns in a **DataTable**, ensures that all data in the specified column(s) is unique per row. You can create a unique constraint for a single column by setting the **Unique** property of the column to **true**.

You can also create a unique constraint for a column or array of columns by using the UniqueConstraint constructor and passing the **UniqueConstraint** object to the **Add** method of the table’s **Constraints** property (which is a **ConstraintCollection**). You use the **Add** method to add *existing* constraint objects to the **Constraints** collection as well.

Additionally, defining a column or columns as the primary key for a table will automatically create a unique constraint for the specified column(s).

A **UniqueConstraint** triggers an exception when attempting to set a value in a column to a non-unique value.

Examples

The following examples create a **UniqueConstraint** object for an existing column by using two different techniques. In the first example, setting the **Unique** property for a column creates a constraint automatically, but it will be assigned a default name such as Constraint1. In the second example, the name of the constraint can be specified as the first parameter of the **Add** method.

```
' Visual Basic
ds.Tables("Product").Columns("ProductName").Unique = True

// Visual C#
ds.Tables["Product"].Constraints.Add(
  new UniqueConstraint("UC_ProductName",
  ds.Tables["Product"].Columns["ProductName"]));
```

Practice

Northwind Traders needs product names in their online catalog to be unique.

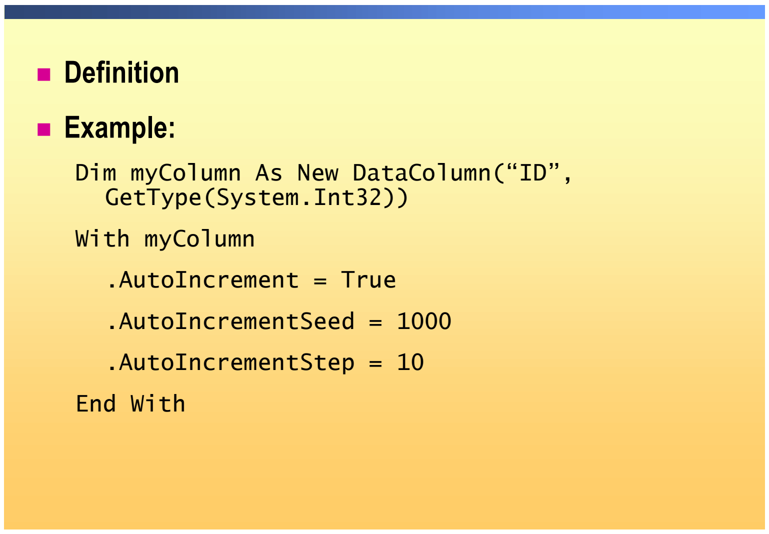
1. Open the solution you built for the previous practice, or open the solution at the following location.

   <install folder>\Practices\Mod04_1\Lesson1\CreateDataSets\

2. Open the form in Designer view and double-click the form to create a handler for the **Form1_Load** event.
3. Write code to add a **UniqueConstraint** object to the **Products** DataTable that prevents duplicate product names.

The solution for this practice is located at <install folder>\Practices\Mod04_1\Lesson1\UsingUniqueConstraints\

Using AutoIncrement Columns



- **Definition**
- **Example:**

```
Dim myColumn As New DataColumn("ID",
  GetType(System.Int32))
With myColumn
  .AutoIncrement = True
  .AutoIncrementSeed = 1000
  .AutoIncrementStep = 10
End With
```

Definition

An AutoIncrement column automatically increments the value of the column for new rows added to the table. AutoIncrement columns are often used as the primary key for a table to help enforce referential integrity. The **AutoIncrement** property is specified on the **DataColumn** object.

You can specify the starting value (**AutoIncrementSeed** property) and the amount by which the value will increment each time a new row is added (**AutoIncrementStep** property).

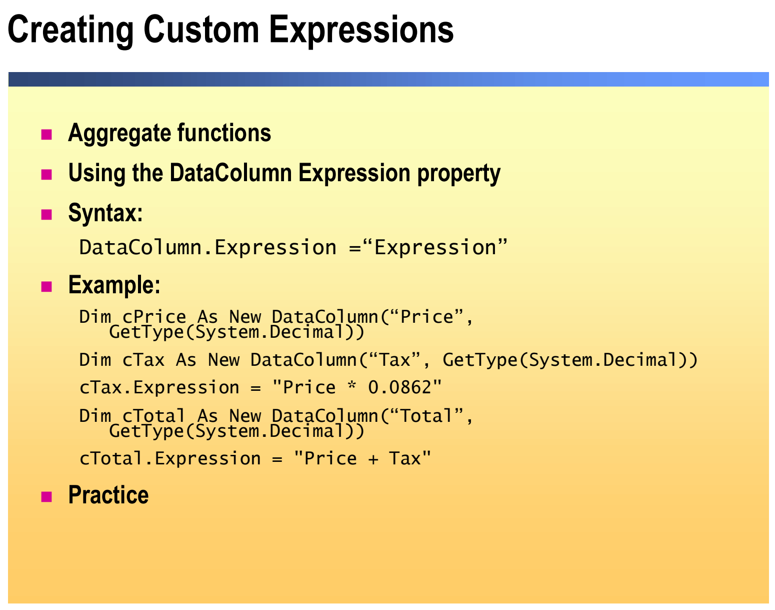
AutoIncrement columns can cause problems in situations where multiple users are adding new rows, because of the likelihood of conflicts. In many situations, a better solution is to use a column with a globally unique identifier (GUID) data type; for example, the **SqlTypes.SqlGuid** data type.

Example

The following example shows how to specify the **AutoIncrement** property.

```
Dim colEmployeeID As New _
  DataColumn("EmployeeID", GetType(System.Int32))

With colEmployeeID

  .AutoIncrement = True
  .AutoIncrementSeed = 1000
  .AutoIncrementStep = 10

End With
```

Creating Custom Expressions



Creating Custom Expressions

- Aggregate functions
- Using the DataColumn Expression property
- Syntax:

```
DataColumn.Expression =“Expression”
```

- Example:

```
Dim cPrice As New DataColumn(“Price”,
   GetType(System.Decimal))
Dim cTax As New DataColumn(“Tax”, GetType(System.Decimal))
cTax.Expression = "Price * 0.0862"
Dim cTotal As New DataColumn(“Total”,
   GetType(System.Decimal))
cTotal.Expression = "Price + Tax"
```

- Practice

Definition

Custom expressions are column values derived from calculations, rather than values retrieved directly from the data source. A custom expression can be a calculation on one column or multiple columns.

Syntax for a custom expression consists of standard arithmetic, Boolean, and string operators and literal values. You can reference a data value by using its column name (as you would in a SQL statement) and include aggregate functions (such as **Sum**, **Count**, **Min**, **Max**, and others).

Example

The Order Details table in the Northwind Traders database contains a column called UnitPrice that tracks the price of each product that is sold, and a column called Quantity that tracks the number of items that are sold. If you need to see the total cost for the purchase of a particular product, multiply the UnitPrice and the Quantity and then display the resulting value in its own column (UnitPrice*Quantity) and name the new column TotalCost.

Aggregate functions

Calculated columns can also include aggregate functions such as **Sum**, **Count**, **Min**, **Max**, and other functions available with the data source. Use aggregate functions when creating an expression based on data that is related to data in another table.

For example, suppose that you want to find the average unit price of products per category. To do this, you need to access two **DataTables**: Categories and Products. These tables are related because the CategoryID column of the Products table is the child of the Categories table. Programmatically, the new column that computes the average price of products per category would be:

```
Avg(Child.UnitPrice)
```

Using the DataColumn Expression property

The DataColumn **Expression** property gets or sets the expression that is used to calculate values in a column or create an aggregate. The **DataType** of the column determines the return type of an expression.

You can use the **Expression** property to:

- Create a calculated column.
- Create an aggregate column.
- Create expressions that include user-defined values.
- Concatenate a string.
- Reference parent and child tables in an expression.

Syntax

The syntax for the **Expression** property of the **DataColumn** object is:

```
DataColumn.Expression = “Expression”
```

Example

The following example creates three columns in a **DataTable**: a price column, a tax column, and a total column. The second and third columns contain expressions. The second column calculates tax by using a variable tax rate, and the third column is the result of adding the tax amount to the price.

```
Dim colPrice As New _
  DataColumn(“Price”, GetType(System.Decimal))

Dim colTax As New _
  DataColumn(“Tax”, GetType(System.Decimal))

colTax.Expression = "Price * 0.0862"

Dim colTotal As New _
  DataColumn(“Total”, GetType(System.Decimal))

colTotal.Expression = "Price + Tax"
```

Practice

The Northwind Traders Sales Director would like to know the value of stock being held for each product.

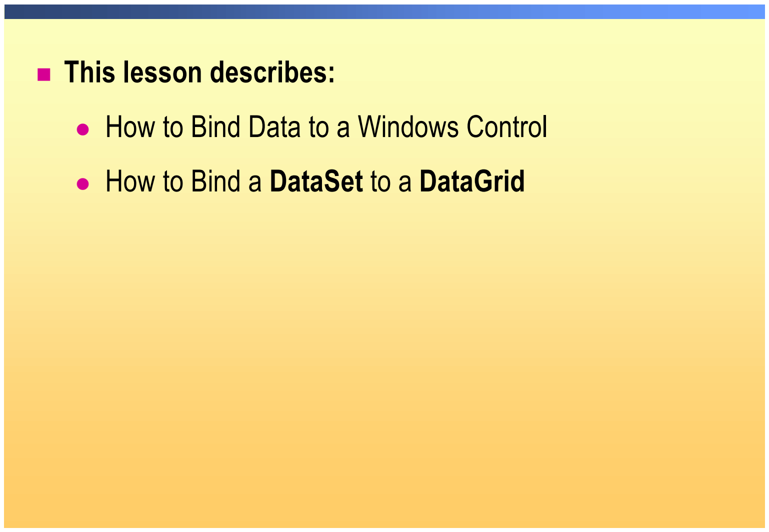
1. Open the solution you built for the previous practice, or open the solution at the following location.

<install folder>\Practices\Mod04_1\Lesson1\UsingUniqueConstraints\

2. Open the form in Code view and find the code that creates the columns in the **Products** DataTable.
3. Manually write code to add a fifth column named **StockValue** that should be the result of the UnitPrice column multiplied by the UnitsInStock column.
4. Use the design tools, for example the Property Window, to view the changes you have made to the Northwind DataSet.

The solution for this practice is located at <install folder>\ Practices\Mod04_1\Lesson1\CreatingCustomExpressions\

Lesson: Binding a DataSet to a Windows Application Control



- **This lesson describes:**
  - How to Bind Data to a Windows Control
  - How to Bind a **DataSet** to a **DataGrid**

Introduction

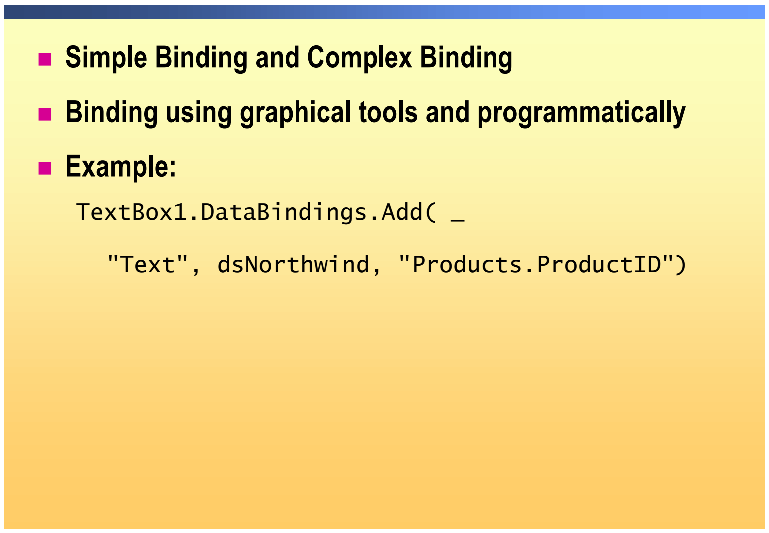
After you have data cached in a **DataSet**, you must bind that **DataSet** to a Microsoft Windows® form **DataGrid** control to display and manipulate or modify the data.

Lesson objectives

After completing this lesson, you will be able to:

- Bind data to a Windows control.
- Bind a **DataTable** to a **DataGrid**.

How to Bind Data to a Windows Control



- Simple Binding and Complex Binding
- Binding using graphical tools and programmatically
- Example:

```
TextBox1.DataBindings.Add( _
  "Text", dsNorthwind, "Products.ProductID")
```

Introduction

Although an ADO .NET **DataSet** allows you to store data in a disconnected cache, the **DataGrid** and other Windows controls visually display data in a Windows Form and support selecting, sorting, and editing the data.

In Windows Forms, you can bind any property of any control to a data source. In addition to binding the display property (such as the **Text** property of a **TextBox** control) to the data source, you can also bind other properties. For example, you may need to bind the graphic of an image control or set the size or color properties of a control based on binding to the data source.

Types of data binding

There are two ways to bind data to a Windows control.

Simple data binding is the ability of a control to bind to a single data element, such as a value in a column in a **DataSet** table. This is typical for binding **TextBox** or **Label** controls, or any control that displays a single value.

Complex data binding is the ability of a control to bind to more than one data element, typically more than one record in a data source. Complex data binding typically uses a **DataGrid**, **ListBox**, or **ErrorProvider** control. Binding a **DataSet** to a **DataGrid** control will be covered later in this module.

Creating a simple data-bound control

► **To bind data to a Windows control graphically**

1. In Visual Studio .NET, open a project form, select a control, and then display the Properties window.
2. Expand the **(DataBindings)** property.
3. If the property you want to bind is not one of the commonly bound properties, click the **Ellipsis** button (. . .) in the **(Advanced)** box to display the **Advanced Data Binding** dialog box with a complete list of properties for that control.
4. Click the arrow next to the property you want to bind.
5. Expand the data source to which you want to bind, until you find the single data element you want. For example, if you are binding to a column value in a table in a **DataSet**, expand the name of the **DataSet**, and then expand the table name to display column names.
6. Click the name of the element to which you want to bind.
7. If you were working in the **Advanced Data Binding** dialog box, click **Close** to return to the Properties window.

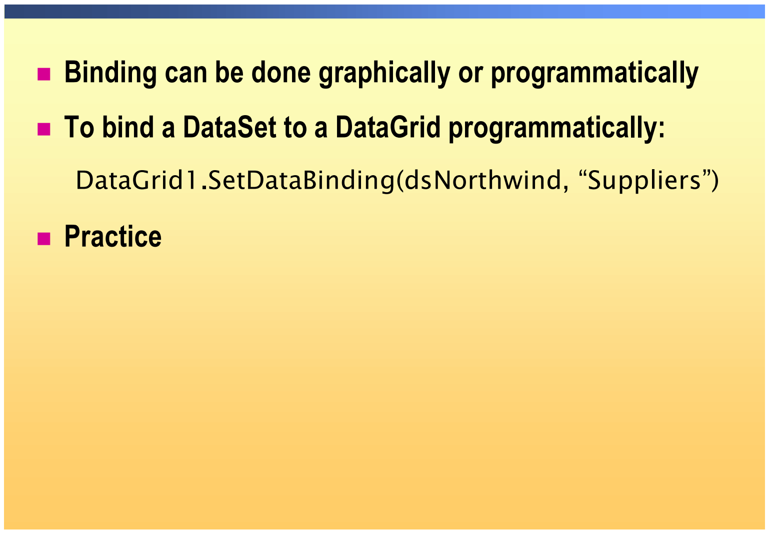
► **To bind a control programmatically**

```
Control.DataBindings.Add( _
  “Property”, DataSource, “Table.Column”)
```

For example:

```
TextBox1.DataBindings.Add( _
  "Text", dsNorthwind, "Products.ProductID")
```

How to Bind a DataSet to a DataGrid



- Binding can be done graphically or programmatically
- To bind a DataSet to a DataGrid programmatically:

  DataGrid1.SetDataBinding(dsNorthwind, "Suppliers")
- Practice

Definition

The Windows Forms **DataGrid** control displays data in a series of rows and columns. The simplest use of this control is when the grid is bound to a data source with a single table containing no relationships. In such a case, the data appears in simple rows and columns, as in a spreadsheet.

If the **DataGrid** is bound to data with multiple related tables, and if navigation is enabled on the grid, the grid will display expanders in each row. An *expander* allows navigation from a parent table to a child table. Clicking a node displays the child table, and clicking the **Back** button displays the original parent table. In this way, the grid displays the hierarchical relationships between tables.

If the **DataSet** contains a series of related tables, you can use two **DataGrid** controls to display the data in a relational format.

Binding a DataSet to a DataGrid

► **To bind a DataSet to a DataGrid graphically**

1. In a Visual Basic .NET project, drag a **DataGrid** control from the **Windows Forms** tab of the **Toolbox** to your form.
2. Select the **DataGrid** control and set its **DataSource** property to a previously created **DataSet**.
3. Set the **DataMember** property to the name of one of the tables. For example, set the property to **Suppliers**.

► **To bind a DataSet to a DataGrid programmatically**

```
DataGrid1.SetDataBinding(dsNorthwind, "Suppliers")
```

Practice

Northwind Traders needs an application that allows the easy editing of product information.

1. Open the solution you built for the previous practice, or open the solution at the following location.

   <install folder>\Practices\Mod04_1\Lesson1\CreatingCustomExpressions\

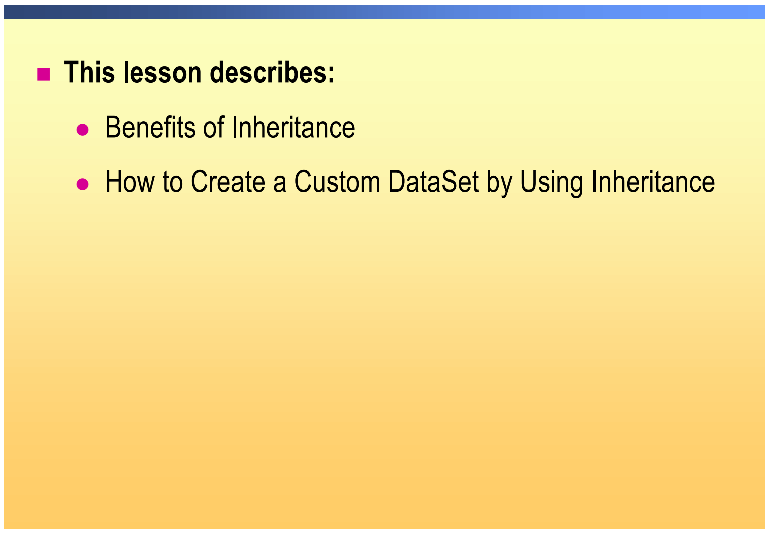
2. Open the form in Designer view.
3. Add five **TextBox** controls and a **DataGrid** control to the form.
4. Use the **Property** window to set the **DataBindings** for the **TextBox** controls. Bind the **Text** property of each box to the five columns in the **Products** table in the **Northwind DataSet**.

   Caution Do not bind to the dtProducts variable. If you do, you will only see the first record.

5. Set the **DataSource** for the **DataGrid** to the **Northwind DataSet**.
6. Set the **DataMember** of the **DataGrid** to the **DataTable**.
7. Run the application.
8. Enter several products into the **DataGrid**. Enter any values you like for the columns. Notice that the StockValue column is read only and calculated automatically. Notice that the numeric fields do not accept alphabetic values. Notice that you cannot have two records with the same **ProductName**, but that you can have two records with the same **ProductID**
9. Use your mouse to select the first product row. Notice that the **TextBox** controls display the same information as the currently selected row in the **DataGrid**.

   The solution for this practice is located at <install folder>\Practices\Mod04_1\Lesson2\DataBinding\

Lesson: Creating a Custom DataSet



- **This lesson describes:**
  - Benefits of Inheritance
  - How to Create a Custom DataSet by Using Inheritance

Introduction

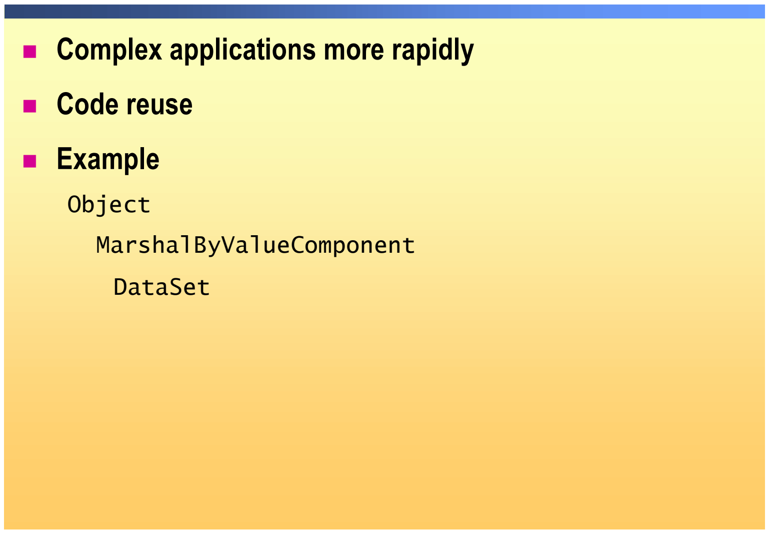
By using inheritance in Visual Studio .NET, you can create a custom **DataSet** based on an existing **DataSet**. This allows for faster application development by reusing code and data.

Lesson objectives

After completing this lesson, you will be able to:

- Describe the benefits of inheritance.
- Create a custom **DataSet** by using inheritance.

Benefits of Inheritance



- **Complex applications more rapidly**
- **Code reuse**
- **Example**

```
Object
  MarshalByValueComponent
    DataSet
```

Introduction

Object-oriented programming allows you to reuse code and data together through *inheritance*. By inheriting from existing or predefined objects, you can construct complex applications more quickly. Because new code may contain bugs, reusing tested code minimizes the probability of introducing additional bugs into the application.

A common example of effective code reuse is in connection with libraries that manage data structures. Object-oriented programming is a development method in which developers create objects that cooperate with one another to form a system. Because the objects are independent, they can be reused in different applications.

Definition

A class inherits the members of its direct base class. Inheritance means that a class implicitly contains all members of its direct base class, except for the instance constructors, static constructors, and destructors of the base class.

Inheritance is transitive. For example, if class **C** is derived from class **B**, and class **B** is derived from class **A**, then class **C** inherits the members declared in class **B** as well as the members declared in class **A**.

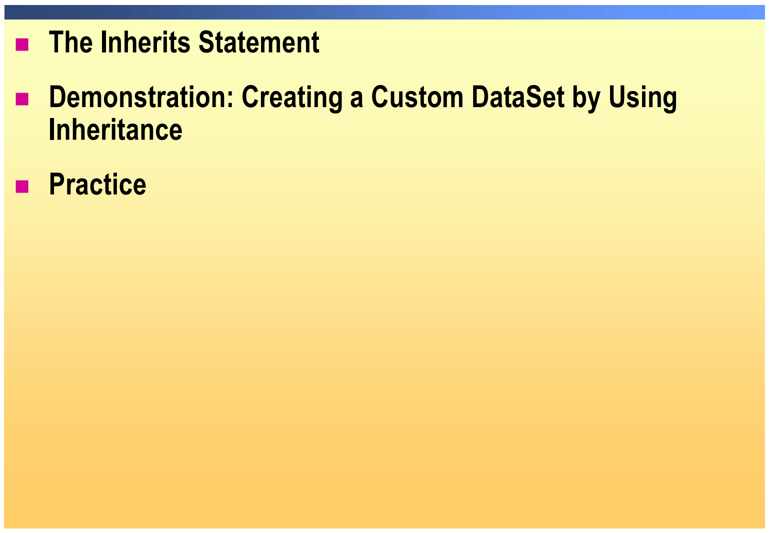
A derived class extends its direct base class. A derived class can add new members to the classes that it inherits, but it cannot remove the definition of an inherited member.

Example

The following is an example of inheritance. The **DataSet** in this example is based on the **MarshalByValueComponent** class that provides the base implementation for components that are marshaled by value (a copy of the serialized object is passed to the **DataSet**).

```
Object
  MarshalByValueComponent
      DataSet
```

How to Create a Custom DataSet by Using Inheritance



- The Inherits Statement
- Demonstration: Creating a Custom DataSet by Using Inheritance
- Practice

Introduction

By using inheritance to create a custom **DataSet**, you can save time and reuse code that already exists for other **DataSet**s, and define a custom schema for your custom dataset that is exposed to components that use your dataset.

The Inherits statement

The **Inherits** statement causes a class to inherit all of the non-private members of the specified class.

To inherit from a class, add an **Inherits** statement with the name of the class that you want to use as a base class as the first statement in your derived class. The **Inherits** statement must be the first non-comment statement after the class statement.

Demonstration

► **To create a custom DataSet by using inheritance in Visual Basic**

1. On the **Project** menu, click **Add Class** to add a new class to your project.
2. When prompted, enter a name for the new class; for example, NorthwindDataSet.
3. Inside the class declaration, add the following code:

   ```
   Inherits System.Data.DataSet
   ```

4. Add code to the **New** method that should run as soon as an instance of your custom class is instantiated. For example, you would write code here to create the schema for your dataset using the DataSet object model.
5. Add any additional properties, methods, fields, or other components that you want your custom **DataSet** to have. If necessary, override existing properties, methods, and so on.

► To create a custom DataSet by using inheritance in Visual C#

1. On the **Project** menu, click **Add Class** to add a new class to your project.
2. When prompted, enter a name for the new class; for example, NorthwindDataSet.
3. At the end of the line that declares the class, add a colon (:) and the class from which you want to inherit.

   ```
   public class NorthwindDataSet : System.Data.DataSet
   ```

4. Add code to the constructor that should run as soon as an instance of your custom class is instantiated. For example, you would write code here to create the schema for your dataset using the DataSet object model.
5. Add any additional properties, methods, fields, or other components that you want your custom **DataSet** to have. If necessary, override existing properties, methods, and so on.

Practice

The Northwind Traders IT Director would like to save money by reusing existing code.

In this practice, you will create a reusable **DataSet** class, bind it to a **DataGrid**, and test the application.

1. Open the following file in Notepad and note that it contains schema and data for the Categories and Products table from the Northwind database in serialized DataSet i.e. XSD/XML format.

\Program Files\MSDNTrain\2389\Practices\Mod04_1\catprodnone.ds

2. Create a new Windows Application project named **CustomDataSets** in the following location.

   <install folder>\Practices\Mod04_1\

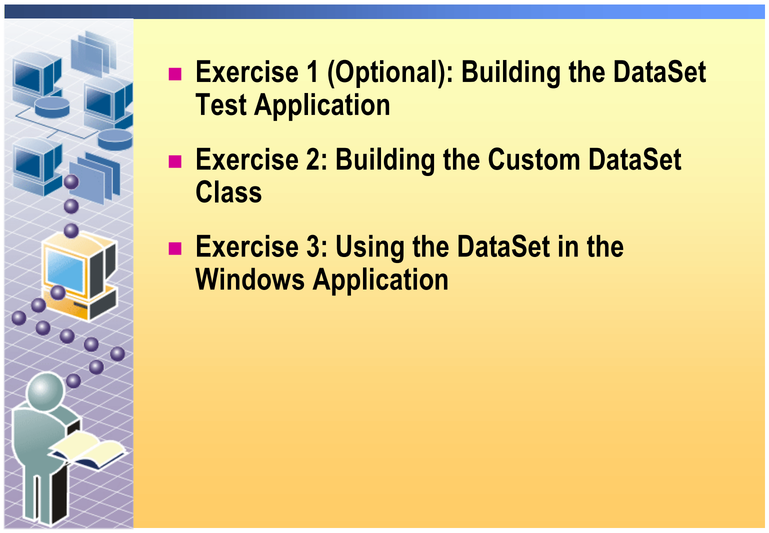
3. Add a new class called **CatProdDataSet** and add the following code:

   ```
   Inherits System.Data.DataSet

   Public Sub New()
     Me.ReadXml("\Program Files\MSDNTrain\" & _
        "2389\Practices\Mod04_1\catprodnone.ds", _
        XmlReadMode.ReadSchema)
   End Sub
   ```

4. Add a **DataGrid** to the form.
5. Set the **Dock** property of the **DataGrid** to **Fill**.
6. Add code to the **Form1_Load** event that creates a private instance of the **CatProdDataSet** class named **dsCatProd** and then binds the **Categories** table to the existing **DataGrid**.
7. Run and test the application. You should see a list of eight categories in the grid, but note that there is no way to access the products.

The solution for this practice is located at <install folder>\
Practices\Mod04_1\Lesson3\CustomDataSets\

Lab 4.1: Building, Binding, Opening, and Saving DataSets



- Exercise 1 (Optional): Building the DataSet Test Application
- Exercise 2: Building the Custom DataSet Class
- Exercise 3: Using the DataSet in the Windows Application

Objectives

After completing this lab, you will be able to:

- Build a DataSet that contains multiple DataTables.
- Bind DataSets to DataGrids.
- Save a DataSet as a file.
- Load a DataSet from an existing file.

Prerequisites

Before working on this lab, you must have:

- Introductory Visual Basic language skills. For example, declaring variables and writing procedures, loops, and branching statements.
- Introductory Windows Forms skills. For example, creating a simple form with multiple controls.

For More Information

See the “Visual Basic Language Features” and “Windows Forms Walkthroughs” topics in the Visual Studio .NET documentation.

Scenario

Northwind Traders has an e-commerce Web site that allows customers to order products from an online catalog. Because it is a publicly available Web site, when a user first visits the site, he or she can browse catalogs and fill a shopping cart before any customer data exists in the Northwind customer database.

The site automatically tracks basic information about customers and the products that they have added to their shopping carts. If a customer wishes to place an order, additional customer information must be gathered. Only during the final checkout stage is the customer and order data saved into the Northwind database. While the customer continues to shop, the data is stored temporarily in the middle tier on the Web server. The DataSet class in ADO .NET provides this capability.

Estimated time to complete this lab: 60 minutes

Exercise 0
Lab Setup

To complete this lab, you must …

►

2.

3.

Exercise 1 (optional)
Building the DataSet Test Application

In this exercise, you will build a Microsoft Windows application that will function as a container for the custom DataSet class that you will build later. The application will be used to track customer shopping carts and customer information on the e-commerce Web site.

Scenario

Before building the Northwind Traders e-commerce Web site, you must build a Windows application that can be used to test the custom DataSet class that you will build in a later exercise.

► To create the Windows application

1. Start the Microsoft Visual Studio .NET development environment.
2. Create a new project by using the **Windows Application** template, and name it **BuildingDataSets**. Set the location to C:\Program Files\MSDNTrain\2389\Labs\Lab04_1.

   Note You can use any language you prefer for this lab, but note that solutions are only provided in Visual Basic and Visual C#.

3. Add a **DataGrid** control to the form and set the following properties.

Property	Value
Name	grd
Dock	Fill

4. Add an **OpenFileDialog** control to the form and set the following properties.

Property	Value
Name	dlgOpen
Filter	DataSet files (*.ds)|*.ds|All files (*.*)|*.*

5. Add a **SaveFileDialog** control to the form and set the following properties.

Property	Value
Name	dlgSave
DefaultExt	ds
Filter	DataSet files (*.ds)|*.ds|All files (*.*)|*.*

► To create the menu

1. Add a **MainMenu** control to the form.
2. Add the following menu items.

Menu	Property	Value
mnuFile	Text	&File
mnuNew	Text	&New
mnuOpen	Text	&Open…
	Shortcut	CtrlO
mnuSave	Text	&Save
	Shortcut	CtrlS
mnuSaveAs	Text	Save &As…
mnuExit	Text	E&xit
mnuView	Text	&View
mnuCustomer	Text	&Customer
	Shortcut	F7
mnuCartItems	Text	Cart &Items
	Shortcut	ShiftF7
mnuProducts	Text	&Products
	Shortcut	CtrlF7

   The Products menu item will not be used until Lab 4.2.

3. (Optional) Add menu separators where appropriate.

► To code the Exit menu item

- Add code to the mnuExit **Click** event to close the form.

► To save the solution

- Save all the files in your solution.

Exercise 2
Building the Custom DataSet Class

In this exercise, you will create the structure of the DataSet programmatically, to save the DataSet to disk and open an existing DataSet file. The DataSet will contain a DataTable called Customer that will hold customer contact details, and a DataTable called CartItems that will hold all of the items that a customer chooses from the online catalog.

Scenario

You will create a custom DataSet class that can track the contents of a shopping cart for a visitor to the Northwind Traders e-commerce Web site.

► **To continue building the application**

- Open the solution that you created in Exercise 1 of this lab, or open the solution **BuildingDataSets** in the folder <install folder>\Labs\Lab04_1\Solution\Ex1\xx\ where xx is either VB or CS.

► **To create a custom DataSet class**

1. Add a new class named **NWShoppingCart** to the project.

Warning Do not choose the **DataSet** file template. You will use XML Schema Definition (XSD) files in Module 6, “Using XML With ADO .NET,” in Course 2389A, *Programming With ADO .NET*.

2. Add code to the class so that it inherits functionality from the **System.Data.DataSet** class.
3. Declare a custom field with the following properties.

Scope	Name	Data type
Public	Filename	System.String

► **To complete the constructor of the custom class**

1. Call the constructor of the base class.

```
‘ Visual Basic
MyBase.New()

// Visual C#
// append the following to the end of the constructor
 : base()
```

2. Set the **DataSetName** property to **NWShoppingCart**.
3. Declare local variables for the two DataTables that you will create.

Name	Data type
dtCustomer	System.Data.DataTable
dtCartItems	System.Data.DataTable

► To create the customer information DataTable

1. Add a new table called **Customer** to the **Tables** collection and assign the result to **dtCustomer**.
2. Add the following DataColumns to the DataTable.

DataColumn	Property	Value
CustomerID	DataType	System.String
	AllowDBNull	False
	MaxLength	5
CompanyName	DataType	System.String
	AllowDBNull	False
	MaxLength	40
	Unique	True
Address	DataType	System.String
	MaxLength	60
City	DataType	System.String
	MaxLength	15

► To create the cart items DataTable

1. Add a new table called **CartItems** to the **Tables** collection and assign the result to **dtCartItems**.
2. Add the following DataColumns to the DataTable.

DataColumn	Property	Value
CustomerID	DataType	System.String
	AllowDBNull	False
	MaxLength	5
ProductID	DataType	System.Int32
	AllowDBNull	False
UnitPrice	DataType	System.Decimal
	AllowDBNull	False
Quantity	DataType	System.Int32
	AllowDBNull	False
Cost	DataType	System.Decimal
	ReadOnly	True
	Expression	UnitPrice * Quantity

► To finish the constructor

1. Add exception handling for common errors such as duplicate table or column names.
2. If a problem occurs, throw a **System.Data.DataException** passing the exception that caused the problem as an inner exception, and a message containing the following text:

   "Failed to create instance of NWShoppingCart. Check innerException for more information."

► To open an existing DataSet file

1. Create a new public method called **OpenFromFile**.
2. Clear any existing tables from the DataSet.
3. Read the XML Schema and data from the file specified in the Filename field.

   Tip Use the **ReadXml** method for the class.

4. Add exception handling for common errors such as an invalid file name.
5. If a problem occurs, throw a **System.Data.DataException** passing the exception that caused the problem as an inner exception, and a message containing the following text:

   "Failed to open NWShoppingCart file. Check innerException for more information."

► To save the DataSet to a file

1. Create a new public method called **SaveToFile**.
2. Write the XML Schema and data to the file name specified in the Filename field.

   Tip Use the **WriteXml** method for the class.

3. Add exception handling for common errors such as an invalid file name.
4. If a problem occurs, throw a **System.Data.DataException** passing the exception that caused the problem as an inner exception, and a message containing the following text:

   "Failed to save NWShoppingCart file. Check innerException for more information."

► To continue building the application

Now that the custom class has been defined, it can be used in the Windows application that you previously created.

- Continue to Exercise 3 to complete this lab.

Exercise 3
Using the DataSet in the Windows Application

In this exercise, you will write code to use the custom DataSet class in the Windows application. To simulate a real-world development environment, you will exchange the class code that you wrote for Exercise 2 with another student. The instructor will pair you with your teammate.

Scenario

On the Northwind Traders e-commerce Web site, the DataSet will be stored temporarily in an ASP .NET session. In the Windows application, you will persist the DataSet to disk to simulate the situation. The Northwind Traders information technology department has a small team of developers working on database projects. The e-commerce Web site development work has been divided between two developers who must use each other's .NET classes to complete the project.

► **To continue building the application**

1. If you do not already have the solution open, open the solution called **BuildingDataSets** in the folder <install folder>\Labs\Lab04_1\Solution\Ex2\xx\ where xx is either VB or CS.
2. Open the **Form1** class module.
3. Declare a variable with the following properties.

Scope	Name	Data type
Private	dsShoppingCart	NWShoppingCart

4. Create a private procedure called **SetFormCaption** that contains code to display the **Filename** property of the **dsShoppingCart** object in the title bar of the form, combined with the fixed string "– Shopping Cart : Test WinApp".
5. Add exception handling to all procedures where appropriate.

► **To code the View menu items**

1. Program the **Click** event of the **Customer** menu item to perform the following actions:
   a. Set the **DataGrid** to bind to the **Customer** table.
   b. Place a check mark next to the **Customer** menu item.
   c. Clear the check mark next to the **Cart Items** menu item.
2. Program the **Click** event of the **Cart Items** menu item to perform the following actions:
   a. Set the **DataGrid** to bind to the **CartItems** table.
   b. Clear the check mark next to the **Customer** menu item.
   c. Place a check mark next to the **Cart Items** menu item.

► To create a new DataSet object

1. Add a private procedure named **NewDataSet** to the form class to perform the following actions:
   a. Set the dsShoppingCart variable to point to a new instance of the **NWShoppingCart** class. This also has the effect of removing any existing DataSet from memory.
   b. Set the **Filename** property of the **dsShoppingCart** object to "ShoppingCart1".
   c. Call the **SetFormCaption** procedure.
   d. Call the **Click** event for the **Customer** menu item to simulate a customer selecting that menu item, as in the following example:

```
' Visual Basic
mnuCustomer_Click(Me, New System.EventArgs())

// Visual C#
mnuCustomer_Click(this, new System.EventArgs());
```

2. Call the **NewDataSet** procedure in the **Load** event for **Form1**.
3. Call the **NewDataSet** procedure in the **Click** event for the **New** menu item.

► To open an existing DataSet file

1. Show the **Open** dialog by adding code to the **Click** event of the **Open** menu item..
2. If the user clicks the **Open** button, perform the following actions:
   a. Set the **Filename** property of the **dsShoppingCart** object to the file name selected in the **Open** dialog.
   b. Call the **OpenFromFile** method of the **dsShoppingCart** object.
   c. Call the **SetFormCaption** procedure.
   d. Call the click event of the **Customer** menu item to simulate a customer selecting that menu item.

► To save a DataSet file

1. Add a private procedure named **SaveDataSet** to the form class to perform the following actions:
   a. Call the **SaveToFile** method of the **dsShoppingCart** object.
   b. Call the **SetFormCaption** procedure.
   c. Program the **Click** event for the **Save** menu item to call **SaveDataSet**.
   d. Program the **Click** event for the **Save As** menu item to show the **Save** dialog.
2. If the user clicks the **Save** button, perform the following actions:
   a. Set the **Filename** property of the **dsShoppingCart** object to the file name selected in the **Save** dialog.
   b. Call the **SaveDataSet** procedure.

► **To test creating a DataSet file**

1. Build the application and correct any build errors.
2. Set a break point at the beginning of the **NewDataSet** and **mnuSaveAs_Click** procedures.
3. Run the application.
4. Step through the code line by line until the form appears.
5. Toggle between the two **DataTables** by using the F7 and SHIFT+F7 keys, or by using the menus. Notice that the **DataGrid** automatically recognizes the **DataTable** structure.
6. Enter some data into the grid for a customer and several cart items. Notice that the rules for data types and NULL values are enforced automatically. The unique constraint on the Customer table prevents a user from adding two customers with the same company name.
7. Use the **Save As** menu item to save the DataSet as a file called **MyFirstDataSet.ds**. (The .ds file extension should be appended automatically if you do not enter it.) Step through the code line by line until the form reappears.
8. Close the application.

► **To test editing and opening a DataSet file**

1. Run Windows Explorer and open the file **MyFirstDataSet.ds** with Notepad.

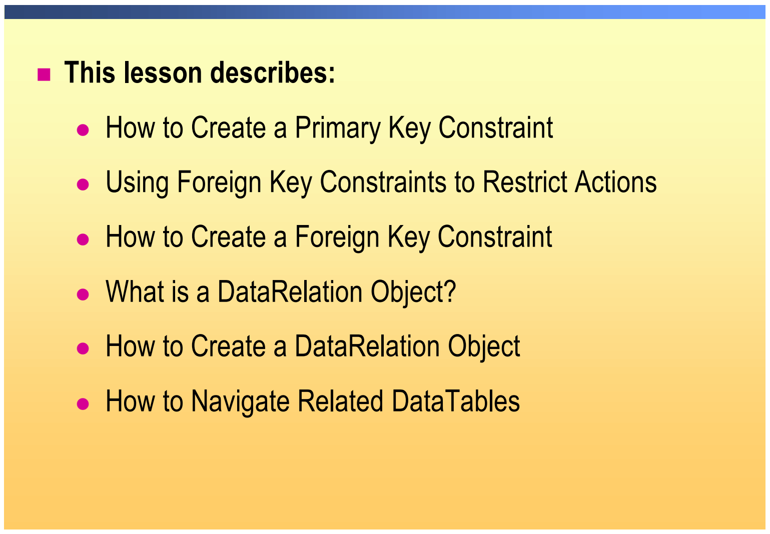
   Notice that using XSD saves the structure of the DataSet, and that the data you entered follows the schema as part of the same file.

2. Use Notepad to edit the **CompanyName** value of the customer you entered. Save the changes to the file and close Notepad.
3. In Visual Studio .NET, clear all existing break points and set a new break point at the beginning of the **mnuOpen_Click** procedure.
4. Run your application again and open the file **MyFirstDataSet.ds**.

   Notice that your application now recognizes the change that you made in Notepad.

5. Choose the **New** menu item. The **DataGrid** should be emptied.
6. Close the application.

Lesson: Defining Data Relationships



- **This lesson describes:**
  - How to Create a Primary Key Constraint
  - Using Foreign Key Constraints to Restrict Actions
  - How to Create a Foreign Key Constraint
  - What is a DataRelation Object?
  - How to Create a DataRelation Object
  - How to Navigate Related DataTables

Introduction

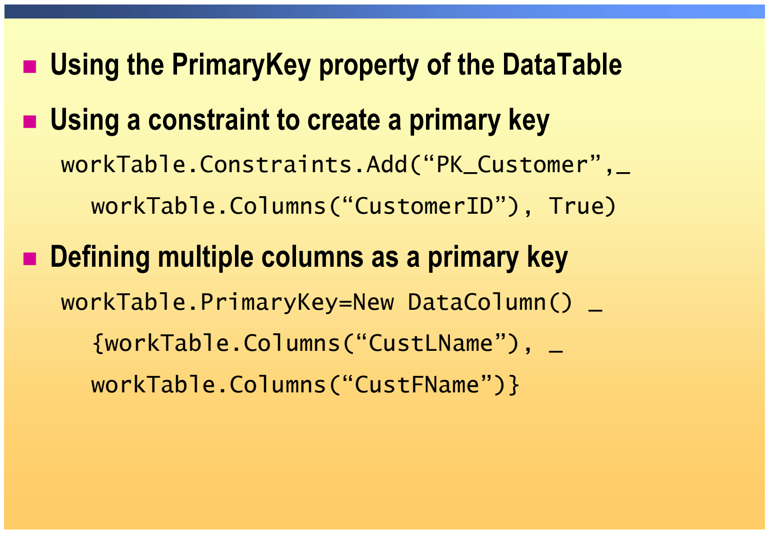
Before you modify data in a **DataTable**, it is necessary to first define the relationships between data to ensure data integrity.

Lesson objectives

After completing this lesson, you will be able to:

- Create a **PrimaryKey** constraint constructor.
- Create a **ForeignKey** constraint constructor.
- Create a **DataRelation** object.
- Navigate related **DataTable**s.

How to Create a Primary Key Constraint



- Using the PrimaryKey property of the DataTable
- Using a constraint to create a primary key

```
workTable.Constraints.Add(“PK_Customer”,_
  workTable.Columns(“CustomerID”), True)
```

- Defining multiple columns as a primary key

```
workTable.PrimaryKey=New DataColumn() _
  {workTable.Columns(“CustLName”), _
  workTable.Columns(“CustFName”)}
```

Introduction

A database table commonly has a column, or group of columns, that uniquely identifies each row in the table. This identifying column, or group of columns, is called the *primary key*.

When you identify a single **DataColumn** as the primary key for a **DataTable**, the table automatically sets the **AllowDBNull** property of the column to **false** and the **Unique** property to **true**. For multiple-column primary keys, only the **AllowDBNull** property is automatically set to **false**.

Using the PrimaryKey property of a DataTable

The **PrimaryKey** property of a **DataTable** receives as its value an array of one or more **DataColumn** objects, as shown in the following examples.

```
dtCustomers.PrimaryKey = New DataColumn() _
  {dtCustomers.Columns("CustomerID")}
```

Using a constraint to create a primary key

An easier method of creating a primary key is to add a unique constraint. When doing so, the last parameter (True) can be used to indicate that a primary key should be created instead of an ordinary unique constraint.

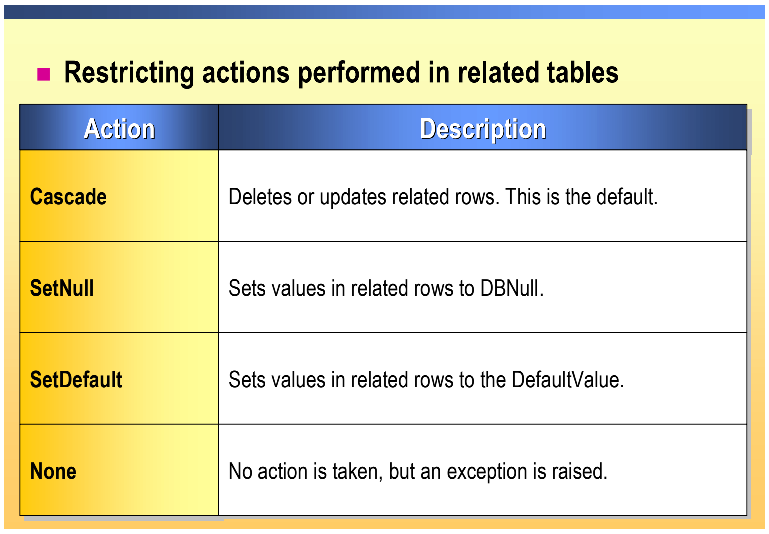
```
dtCustomers.Constraints.Add( _
  "PK_Customer", dtCustomers.Columns("CustomerID"), True)
```

Defining multiple columns as a primary key

The following example defines two columns as a primary key.

```
dtEmployees.PrimaryKey = New DataColumn() _
  {dtEmployees.Columns("LastName"), _
  dtEmployees.Columns("FirstName")}
```

Using Foreign Key Constraints to Restrict Actions



- Restricting actions performed in related tables

Action	Description
Cascade	Deletes or updates related rows. This is the default.
SetNull	Sets values in related rows to DBNull.
SetDefault	Sets values in related rows to the DefaultValue.
None	No action is taken, but an exception is raised.

Introduction

A *foreign key constraint* restricts the action performed in related tables when data in a table is either updated or deleted. For example, if a value in a row of one table is updated or deleted, and that same value is also used in one or more related tables, you can use a **ForeignKeyConstraint** constructor to determine what happens in the related tables.

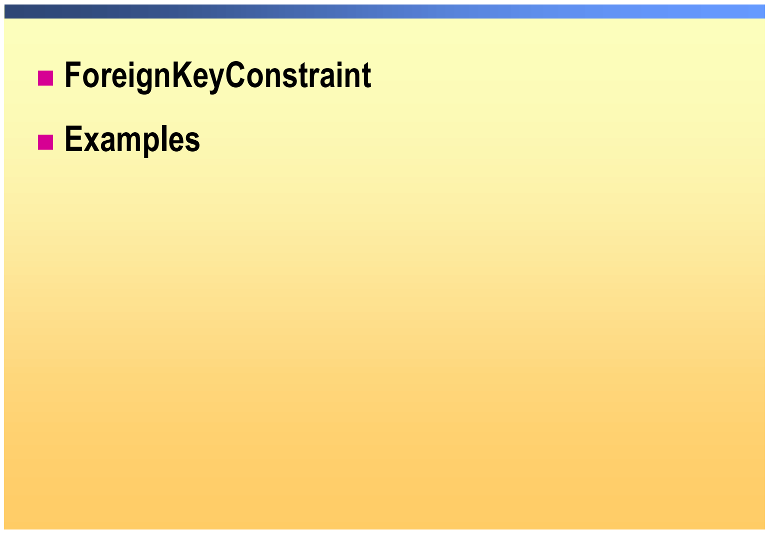
A **ForeignKeyConstraint** can restrict, as well as propagate, changes to related columns. Depending on the properties set for the **ForeignKeyConstraint** of a column, and if the **EnforceConstraints** property of the **DataSet** is true, performing certain operations on the parent row will result in an exception. For example, if the **DeleteRule** property of the **ForeignKeyConstraint** is **None**, a parent row cannot be deleted if it has any child rows.

Restricting actions performed in related tables

In a parent-child relationship between two columns, you establish the action to be taken by setting the **DeleteRule** property of the foreign key constraint to one of the values in the following table.

Action	Description
Cascade	Deletes or updates related rows. This is the default.
SetNull	Sets values in related rows to **DBNull**.
SetDefault	Sets values in related rows to the **DefaultValue**.
None	No action is taken, but an exception is raised

How to Create a Foreign Key Constraint



- ForeignKeyConstraint
- Examples

Visual Basic Example Visual C# Example

Introduction

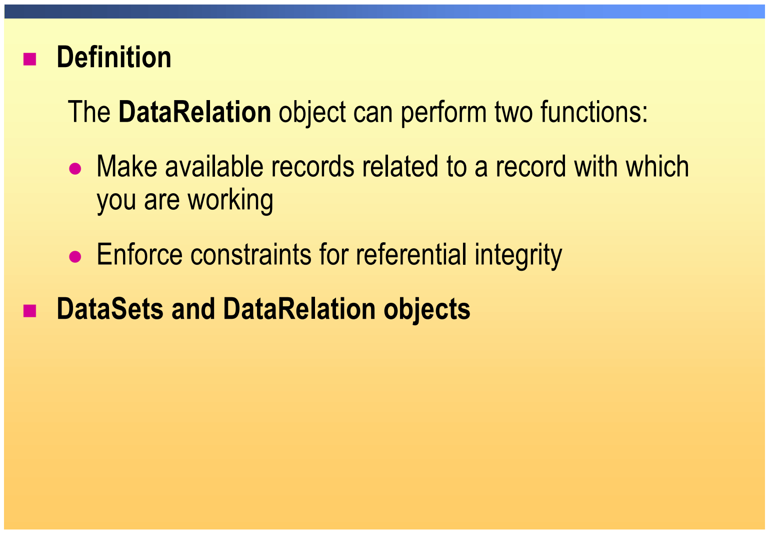
When creating a **ForeignKeyConstraint**, you can pass the DeleteRule and UpdateRule values to the constructor as arguments, or you can set them as properties as in the following example (where the UpdateRule value is left as the default, Cascade).

Example

The following code shows how to create a **ForeignKeyConstraint** on the **CustomerID** column of the Orders table. The **DeleteRule** value is then set to prevent the deletion of any customer who has existing orders

```
dtCustomers = dsNorthwind.Tables("Customers")
dtOrders = dsNorthwind.Tables("Orders")

Dim fkcCustomersOrders As ForeignKeyConstraint = _
  dtOrders.Constraints.Add( _
  New ForeignKeyConstraint("FK_CustomersOrders", _
  dtCustomers.Columns("CustomerID"), _
  dtOrders.Columns("CustomerID")))

' Prevent delete of customer with existing orders
fkcCustomersOrders.DeleteRule = Rule.None
```

What Is a DataRelation Object?



- **Definition**

  The **DataRelation** object can perform two functions:

  - Make available records related to a record with which you are working
  - Enforce constraints for referential integrity

- **DataSets and DataRelation objects**

Definition

A **DataRelation** object defines the relationship between two tables. Typically, two tables are linked through a single field that contains the same data. For example, a table that contains address data might have a single field containing codes that represent countries/regions. A second table that contains country/region data will have a single field that contains the code that identifies the country/region; it is this code that is inserted into the corresponding field in the first table.

The **DataRelation** object can perform the following two functions:

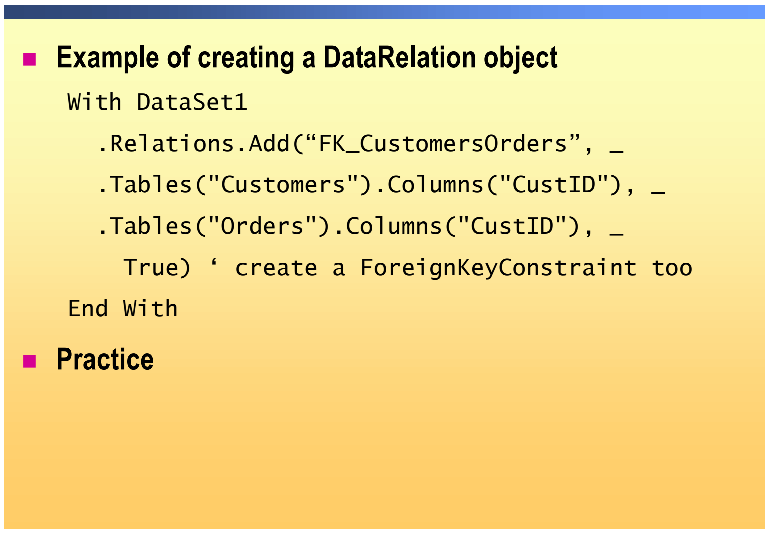
- It can make available the records related to a record that you are working with. It provides child records if you are in a parent record, and a parent record if you are working with a child record.
- It can enforce constraints for referential integrity, such as deleting related child records when you delete a parent record.

DataSets and DataRelation objects

Although a **DataSet** contains tables and columns in a relational structure similar to that of a database, the **DataSet** does not inherently include the ability to relate tables. However, you can create **DataRelation** objects that establish a relationship between a parent (master) and a child (detail) table based on a common key.

For example, a **DataSet** that contains customer data might have a Customers table and an Orders table. Even if the tables contain a key in common (in this example, CustomerID), the **DataSet** itself does not keep track of the records in one table that relate to those in another. However, you can create a **DataRelation** object that references the parent and child tables (and their keys), and then use this object to work with the related tables.

How to Create a DataRelation Object



- **Example of creating a DataRelation object**

```
With DataSet1
  .Relations.Add(“FK_CustomersOrders”, _
  .Tables("Customers").Columns("CustID"), _
  .Tables("Orders").Columns("CustID"), _
    True) ‘ create a ForeignKeyConstraint too
End With
```

- **Practice**

Introduction

To create a **DataRelation** object, you use the **DataRelation** constructor or the **Add** method of the Relations collection of a **DataSet**.

Example

The following example creates a **DataRelation** object.

```
dsNorthwind.Relations.Add(“FK_CustomersOrders”, _
  dtCustomers.Columns("CustomerID"), _
  dtOrders.Columns("CustomerID"), True)
```

Practice

The custom DataSet class you created in the previous exercise does not know about the relationship between the Categories and Products tables.

1. Open the solution you built for the previous practice, or open this solution.

   <install folder>\Practices\Mod04_1\Lesson3\CustomDataSets\

2. Open the form in Code view.
3. Add code to the **Form1_Load** event handler that will perform the following actions.

   e. Create a **PrimaryKey** on the **CategoryID** column for the **Categories** table.

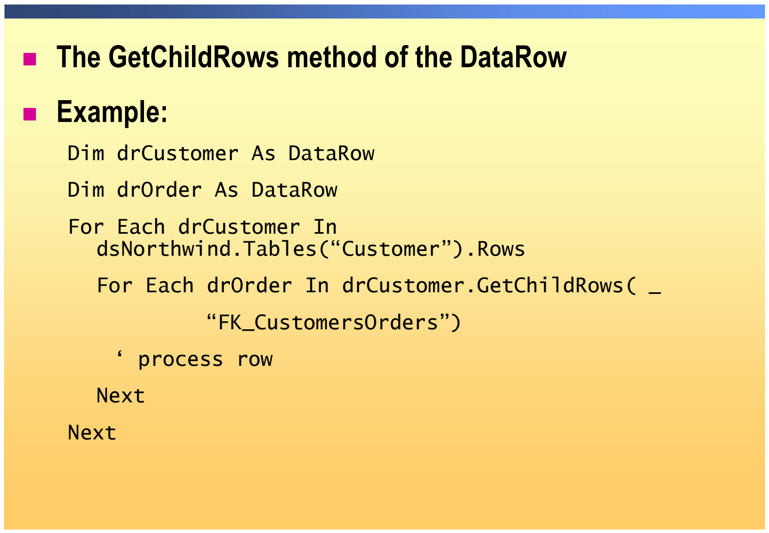
   f. Create a **PrimaryKey** on the **ProductID** column for the **Products** table.

   g. Create a **DataRelation** and **ForeignKeyConstraint** between **Categories** and **Products**.

4. Run and test the application. You should see a list of eight categories in the grid, and note that there is now a way to access the products: the user can click the [+] icon next to a category and drill down to see just the products for that category.

The solution for this practice is located at <install folder>\ Practices\Mod04_2\Lesson1\CreateADataRelation\

How to Navigate Related DataTables



- The GetChildRows method of the DataRow
- Example:

```
Dim drCustomer As DataRow
Dim drOrder As DataRow
For Each drCustomer In
  dsNorthwind.Tables(“Customer”).Rows
  For Each drOrder In drCustomer.GetChildRows( _
          “FK_CustomersOrders”)
   ‘ process row
  Next
Next
```

Introduction

In many application scenarios, you want to work with data from more than one table, and often data from related tables. This is called a *master-detail* relationship between a parent and child table. An example would be retrieving a customer record and also viewing related order information.

The disconnected **DataSet** model allows you to work with multiple tables in your application and to define a relationship between the tables. You can then use the relationship to navigate between related records.

Definition

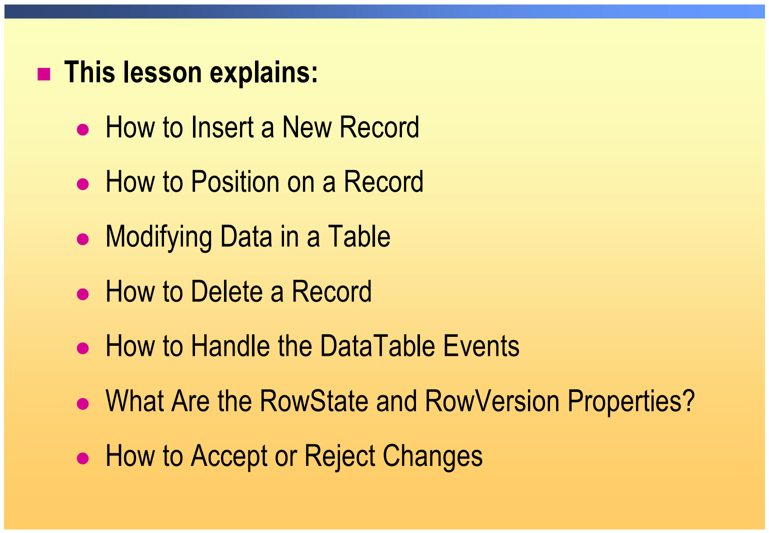
The **GetChildRows** method of the **DataRow** allows you to retrieve related rows from a child table.

Example

The following example loops through each customer and returns the order date and company name for each order in the Orders table that is related to a customer.

```
Dim drCustomer As DataRow
Dim drOrder As DataRow

For Each drCustomer In dsNorthwind.Tables(“Customer”).Rows
  For Each drOrder In drCustomer.GetChildRows( _
        “FK_CustomersOrders”)

      ' process row

  Next
Next
```

Lesson: Modifying Data in a DataTable



- **This lesson explains:**
  - How to Insert a New Record
  - How to Position on a Record
  - Modifying Data in a Table
  - How to Delete a Record
  - How to Handle the DataTable Events
  - What Are the RowState and RowVersion Properties?
  - How to Accept or Reject Changes

Introduction

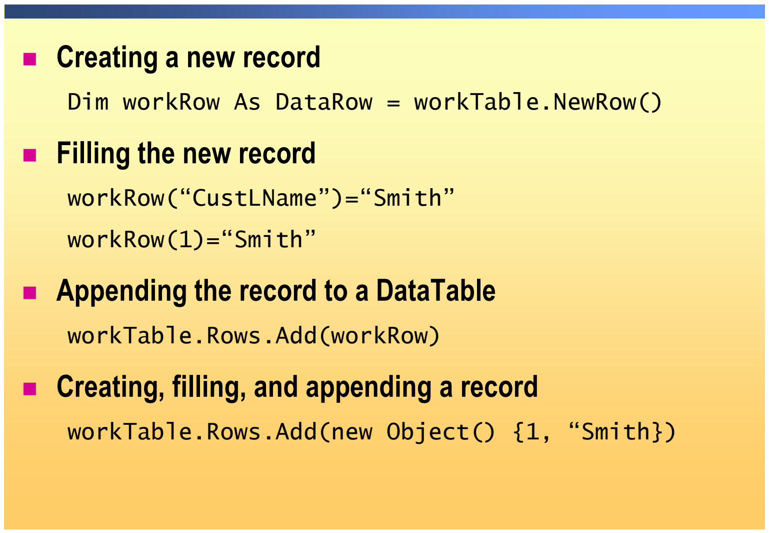
After creating a **DataTable** in a **DataSet**, you can perform the same activities that you would perform when using a table in a database: adding, viewing, editing, and deleting data; monitoring errors and events; and querying the data. When modifying data in a **DataTable**, you can also verify whether the changes are accurate, and determine whether to programmatically accept or reject the changes.

Lesson objectives

After completing this lesson, you will be able to:

- Insert a new record into a **DataTable**.
- Find records in a **DataTable**.
- Update data in **DataTable**s.
- Delete a record in a **DataTable**.
- Handle the **RowDeleted** event.
- Accept or reject changes to **DataTable**s.

How to Insert a New Record



- **Creating a new record**

```
Dim workRow As DataRow = workTable.NewRow()
```

- **Filling the new record**

```
workRow(“CustLName”)=“Smith”
workRow(1)=“Smith”
```

- **Appending the record to a DataTable**

```
workTable.Rows.Add(workRow)
```

- **Creating, filling, and appending a record**

```
workTable.Rows.Add(new Object() {1, “Smith})
```

Introduction

After you create a **DataTable** and define its structure by using columns and constraints, you can add new rows of data to the table.

Creating a new record

To add a new row to a **DataTable**, you declare a new variable of the type **DataRow**. A new **DataRow** object is returned when you call the **NewRow** method. The **DataTable** then creates the **DataRow** object based on the structure of the table, as defined by the **DataColumnCollection**.

```
Dim drNewEmployee As DataRow = dtEmployees.NewRow()
```

Filling the new record

After adding a new row to a **DataTable**, you can manipulate the new row by using an index or the column name.

```
drNewEmployee(0) = 11
drNewEmployee(1) = "Smith"

drNewEmployee("EmployeeID") = 11
drNewEmployee("LastName") = "Smith"
```

Appending the record to a DataTable

After data is inserted into the new row, the **Add** method is used to add the row to the **DataRowCollection**.

```
dtEmployees.Rows.Add(drNewEmployee)
```

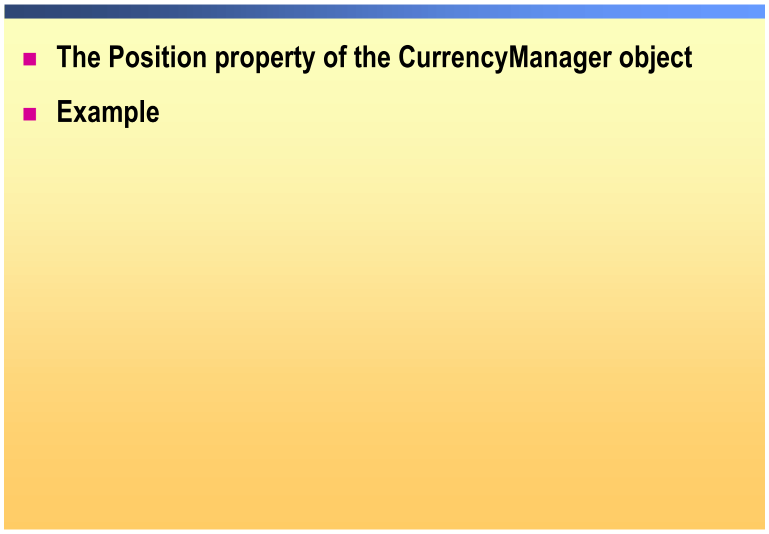
Creating, filling, and appending a record

You can also call the **Add** method to add a new row by passing in an array of values, typed as **Object**.

```
dtEmployees.Rows.Add(New Object() {11, "Smith"})
```

This technique creates a new row inside the table, and sets its column values to the values in the object array. Note that values in the array are matched sequentially to the columns, based on the order in which they appear in the table.

How to Position on a Record



- The Position property of the CurrencyManager object
- Example

Visual Basic Example

Introduction

In order to modify data, you first need to find the records that you want to modify.

In a Windows Form application, the data-binding layer manages navigation through records in a data source. The **CurrencyManager** object associated with a table or view in a **DataSet** includes a **Position** property that can be used to navigate through data.

Because the **DataSet** can contain multiple data sources, or because the controls on a form can be bound to two or more data lists, the form can have multiple currency managers.

Finding records in a Windows Form application

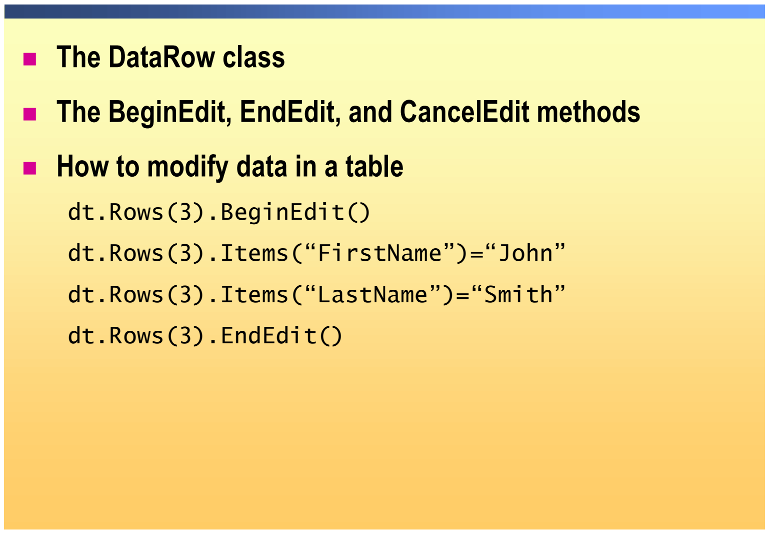
To find a record in a Windows Form, set the **Position** property of the **CurrencyManager** object for the bound data to the record position where you would like to go.

Example

The following is an example of setting the **Position** property of the **CurrencyManager** object

```
Private myCM As CurrencyManager

Private Sub BindControl(myTable As DataTable)
  TextBox1.DataBindings.Add("Text", myTable, "CompanyName")
  myCM = CType(Me.BindingContext(myTable), CurrencyManager)
  myCM.Position = 0
End Sub

Private Sub MoveNext()
  If myCM.Position <> myCM.Count - 1 Then
      myCM.Position += 1
  End If
End Sub

Private Sub MoveFirst()
   myCM.Position = 0
End Sub

Private Sub MovePrevious()
   If myCM.Position <> 0 Then
      myCM.Position -= 1
   End if
End Sub

Private Sub MoveLast()
   myCM.Position = myCM.Count - 1
End Sub
```

Modifying Data in a Table



- The DataRow class
- The BeginEdit, EndEdit, and CancelEdit methods
- How to modify data in a table

```
dt.Rows(3).BeginEdit()
dt.Rows(3).Items(“FirstName”)=“John”
dt.Rows(3).Items(“LastName”)=“Smith”
dt.Rows(3).EndEdit()
```

Introduction

You can modify data in a **DataSet** using code as well as via bound controls.

The DataRow class

The **DataRow** class is used to manipulate individual records.

The BeginEdit, EndEdit, and CancelEdit methods

The **DataRow** class provides three methods for suspending and reactivating the state of the row while editing: **BeginEdit**, **EndEdit**, and **CancelEdit**.

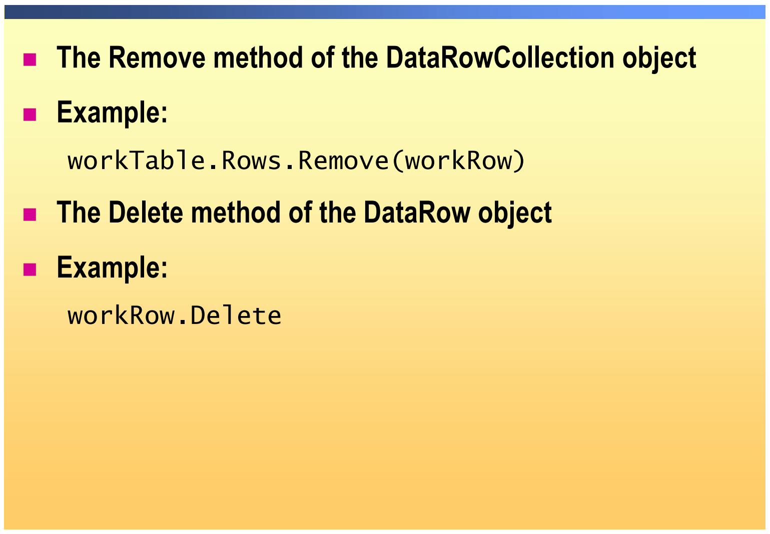
You call **BeginEdit** to suspend any events or exceptions while editing data. You use the **Items** collection to specify the column names of the data you want to modify and the new values. You use **EndEdit** to reactivate any events or exceptions, and **CancelEdit** to abort any changes and reactivate any events or exceptions.

How to modify data in a table

The following example shows how to use the **BeginEdit** method, the **Items** collection, and the **EndEdit** method.

```
' get the third employee
Dim drEmployee As DataRow = dtEmployees.Rows(3)

drEmployee.BeginEdit()
drEmployee("FirstName") = "John"
drEmployee("LastName") = "Smith"
drEmployee.EndEdit()
```

How to Delete a Record



- The Remove method of the DataRowCollection object
- Example:

```
workTable.Rows.Remove(workRow)
```

- The Delete method of the DataRow object
- Example:

```
workRow.Delete
```

Introduction

You can use two methods to delete a **DataRow** object from a **DataTable** object: the **Remove** method of the **DataRowCollection** object, and the **Delete** method of the **DataRow** object. Although the **Remove** method deletes a **DataRow** from the **DataRowCollection**, the **Delete** method only marks the row for deletion.

The **Delete** method is typically used with data in a disconnected environment.

The Remove method and the Delete method

The **Remove** method of the **DataRowCollection** takes a **DataRow** as an argument and removes it from the collection as shown in the following example.

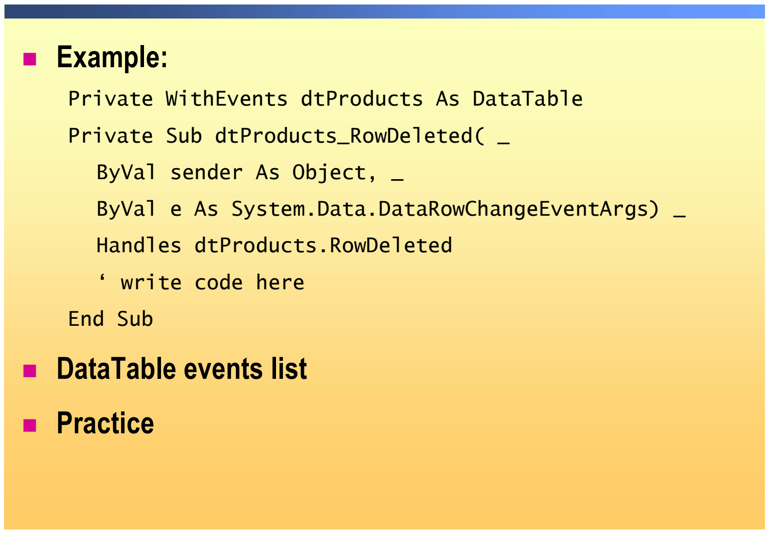
```
Dim drEmployee As DataRow = dtEmployees.Rows(3)

dtEmployees.Rows.Remove(drEmployee)
```

In contrast, the following example demonstrates how to call the **Delete** method on a **DataRow** to change its **RowState** to **Deleted**.

```
drEmployee.Delete
```

Note: You will learn how to find a row in a later lesson.

How to Handle the DataTable Events



- **Example:**

```
Private WithEvents dtProducts As DataTable
Private Sub dtProducts_RowDeleted( _
  ByVal sender As Object, _
  ByVal e As System.Data.DataRowChangeEventArgs) _
  Handles dtProducts.RowDeleted
  ‘ write code here
End Sub
```

- **DataTable events list**
- **Practice**

Introduction

You may need to build functionality into your application that handles events that occur on a **DataTable** when data is being changed or deleted.

To handle DataTable events

To handle **DataTable** events, use the **WithEvents** statement when declaring an object. Then, create a procedure that uses the **Handles** statement to associate the procedure with the event.

Example

The following example shows how to handle the **RowDeleted** event.

```
Private WithEvents dtProducts As DataTable

...

Private Sub dtProducts_RowDeleted( _
  ByVal sender As Object, _
  ByVal e As System.Data.DataRowChangeEventArgs) _
  Handles dtProducts.RowDeleted

  ' write code here

End Sub
```

List of DataTable events

The **DataTable** object provides a series of events that can be processed by an application. The following table describes **DataTable** events.

Event	Description
ColumnChanged	Occurs when a value has been inserted successfully into a column.
ColumnChanging	Occurs when a value has been submitted for a column.
RowChanged	Occurs after a row in the table has been edited successfully.
RowChanging	Occurs when a **DataRow** is changing.
RowDeleted	Occurs after a row in the table has been deleted.
RowDeleting	Occurs when a row in the table is marked for deletion.

Practice

The Northwind Traders Operations Manager wants more complex validation rules to apply when products are changed than can be provided by simple constraints. In this practice you will handle the **ColumnChanging** DataTable event.

We want to see categories in the grid, and have the ability to "drill down" and see the related products, so the grid must stay bound to the **Categories** table. But we also want to see an event fired when a product is changed. Therefore an event handling pointer to the **Products** table needs to be created that responds to the **Column_Changing** event.

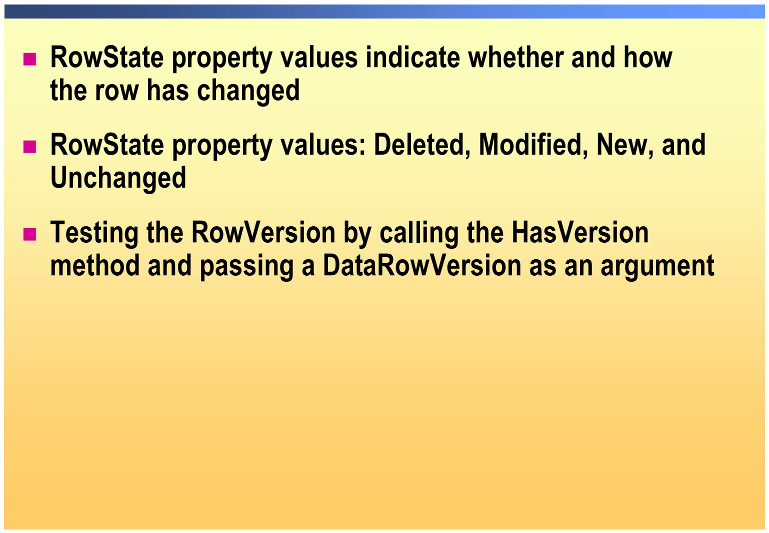
1. Open the solution you built for the previous practice, or open the solution at the following location.

<install folder>\Practices\Mod04_2\Lesson1\CreateADataRelation\

2. Open the form in Code view.
3. Declare a private **DataTable** variable called **dtProducts** with the capability of responding to events.
4. Add code to the **Form1_Load** event handler to point the **dtProducts** variable to the Products table in the Northwind DataSet.
5. Add code to handle the **dtProducts ColumnChanging** event by displaying a message box that shows the original value and the proposed new value of a modified product row.
6. Run and test the application by editing one of the product names. Note: you will need to “drill down” to find a product to edit. If you edit a category name the event will not fire!

The solution for this practice is located at <install folder>\ Practices\Mod04_2\Lesson2\HandlingDataTableEvents\

What Are the RowState and RowVersion Properties?



- RowState property values indicate whether and how the row has changed
- RowState property values: Deleted, Modified, New, and Unchanged
- Testing the RowVersion by calling the HasVersion method and passing a DataRowVersion as an argument

Introduction

The **DataRow** class includes the **RowState** property, whose values indicate whether and how the row has changed since the **DataTable** was first created or loaded from the database. (**RowState** property values include **Deleted**, **Modified**, **New**, and **Unchanged**.)

Changes made to column values in a **DataRow** are immediately placed in the **Current** state of the row. At this point the **RowState** is set to **Modified**.

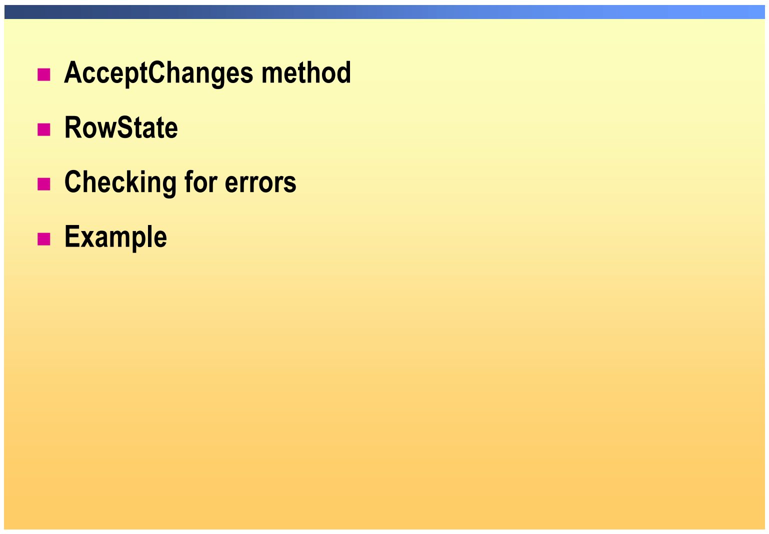
When modifying column values in a **DataRow** directly, the **DataRow** manages column values by using the following row versions: **Current**, **Default**, and **Original**. The **BeginEdit**, **EndEdit**, and **CancelEdit** methods utilize a fourth row version: **Proposed**.

Testing the row version

You can test whether a **DataRow** has a particular row version by calling the **HasVersion** method and passing a **DataRowVersion** as an argument. For example, **DataRow.HasVersion(DataRowVersion.Original)** will return **false** for newly added rows.

The **Proposed** row version exists during an edit operation that is begun by calling **BeginEdit** and ended with either **EndEdit** or **CancelEdit**. During the edit operation, you can apply validation logic to individual columns by evaluating the **ProposedValue** in the **ColumnChanged** event of the **DataTable**. The **DataColumnChangeEventArgs** value passed to the **ColumnChanged** event contains a reference to the changing column and the **ProposedValue**. You can modify the proposed value or trigger the edit to be canceled. The row moves out of the proposed state when the edit is completed.

How to Accept or Reject Changes



- AcceptChanges method
- RowState
- Checking for errors
- Example

Visual Basic Example

Introduction

After verifying the accuracy of changes made to data in a **DataTable**, you can commit the changes by using the **AcceptChanges** method of the **DataRow**, **DataTable**, or **DataSet**. This sets the **Current** row values to be the **Original** values, and sets the **RowState** to **Unchanged**.

Accepting or rejecting changes deletes any **RowError** information and sets **HasErrors** to **false**. Accepting or rejecting changes can also affect updating data in the data source.

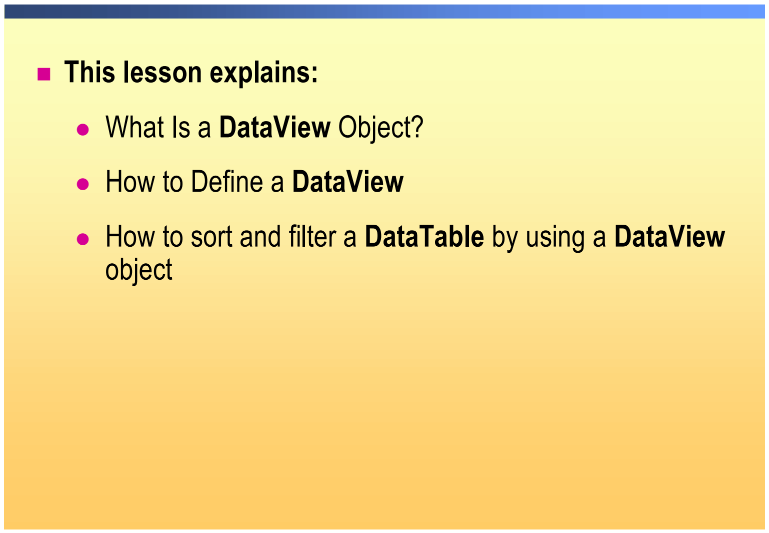
If **ForeignKeyConstraints** exist on the **DataTable**, changes committed or rejected by using **AcceptChanges** and **RejectChanges** are propagated to child rows of the **DataRow** according to the **ForeignKeyConstraint.AcceptRejectRule**.

Example

The following example code checks for rows with errors, resolves the errors where applicable, and rejects the rows where the error cannot be resolved. Note that for resolved errors, the **RowError** value is reset to an empty string, resulting in the **HasErrors** property being set to **false**. When all of the rows with errors have been resolved or rejected, **AcceptChanges** is called to accept all changes for the entire **DataTable**.

```
If workTable.HasErrors Then
  Dim errRows() As DataRow = workTable.GetErrors()

  Dim I As Int32
  Dim errRow As DataRow
  For I = 0 To errRows.Length - 1
      errRow = errRows(I)

      If errRow.RowError = “Total cannot exceed 1000.” Then
          errRow(“Total”) = 1000
          errRow.RowError = ““
      Else
          If errRow.RowState=DataRowState.Added Then I = I - 1
      errRow.RejectChanges()
      End If
  Next
End If

workTable.AcceptChanges()
```

Lesson: Using a DataView



- **This lesson explains:**
  - What Is a **DataView** Object?
  - How to Define a **DataView**
  - How to sort and filter a **DataTable** by using a **DataView** object

Introduction

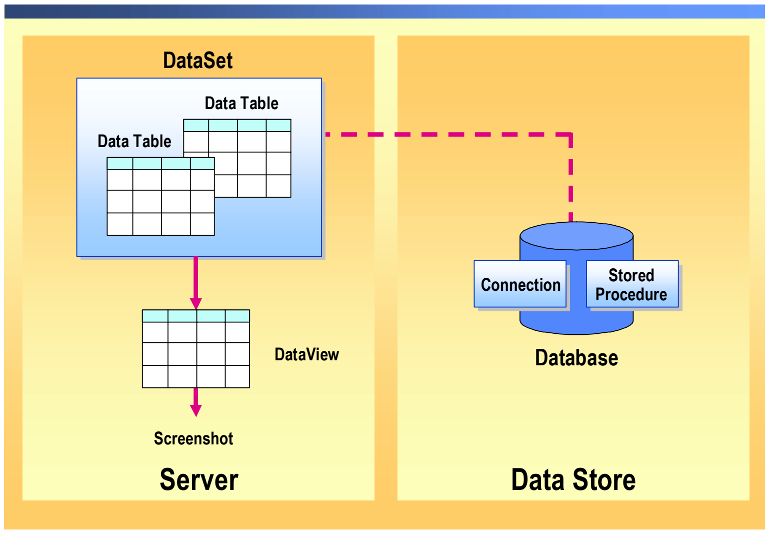
A **DataView** object acts similarly to the way a view acts in Microsoft SQL Server™. It is an object that presents a subset of data from a **DataTable**. A **DataView** object acts like a layer on top of the **DataTable**, providing a filtered and sorted view of the table contents.

Lesson objectives

After completing this lesson, you will be able to:

- Describe what a **DataView** object is.
- Create a **DataView** object.
- Sort and filter a **DataTable** by using a **DataView** Object.

What Is a DataView Object?



Introduction

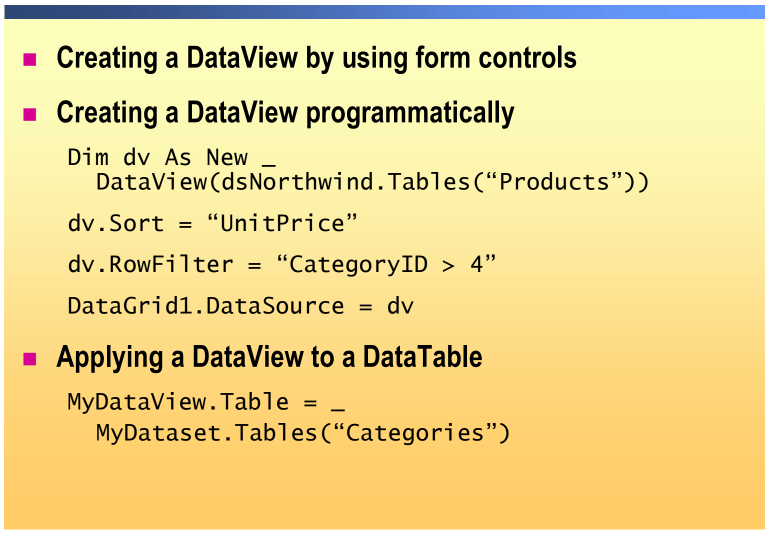
A **DataView** object is similar to a view in SQL Server. It is an object that presents a subset of data from a **DataTable**. A **DataView** object acts like a layer on top of the **DataTable**, providing a filtered and sorted view of the table contents. This capability allows you to have two controls bound to the same **DataTable**, but showing different versions of the data.

Another benefit of the **DataView** object is to allow data binding on both Windows Forms and Web Forms.

Example

Suppose that a sales associate at Northwind Traders travels throughout her assigned region. While in a particular city, she may only want to see the contact information (name, company, phone number, address) for the clients in that city. You could use a **DataView** object to filter clients based on their city where the sales associate will be working on a particular day.

How to Define a DataView



- Creating a DataView by using form controls
- Creating a DataView programmatically

```
Dim dv As New _
  DataView(dsNorthwind.Tables(“Products”))
dv.Sort = “UnitPrice”
dv.RowFilter = “CategoryID > 4”
DataGrid1.DataSource = dv
```

- Applying a DataView to a DataTable

```
MyDataView.Table = _
  MyDataset.Tables(“Categories”)
```

Introduction

You can create a **DataView** object in two ways:

- By using the Visual Studio .NET development environment graphical tools.
- Programmatically.

Creating a DataView by using form controls

► **To add a DataView to a form or component graphically**

1. In the Visual Studio .NET development environment, in the **Toolbox**, drag a **DataView** item from the **Data** tab onto the form or component.

   A new **DataView** with the default name DataView1 is added to the form or component.

2. If you want to configure the **DataView** at design time (rather than in code at run time), select the **DataView** and use the Properties window to configure the **DataView**. Consult the Visual Studio .NET documentation for a list of properties that can be set for a **DataView**.

   You can set all of these properties at run time, except the name of the **DataView**.

Creating a DataView programmatically

The following is an example of creating a **DataView** programmatically.

```
Dim dvProducts As New DataView(dsNorthwind.Tables("Products"))
dvProducts.Sort = "UnitPrice"
dvProducts.RowFilter = "CategoryID > 4"
DataGrid1.DataSource = dvProducts
```

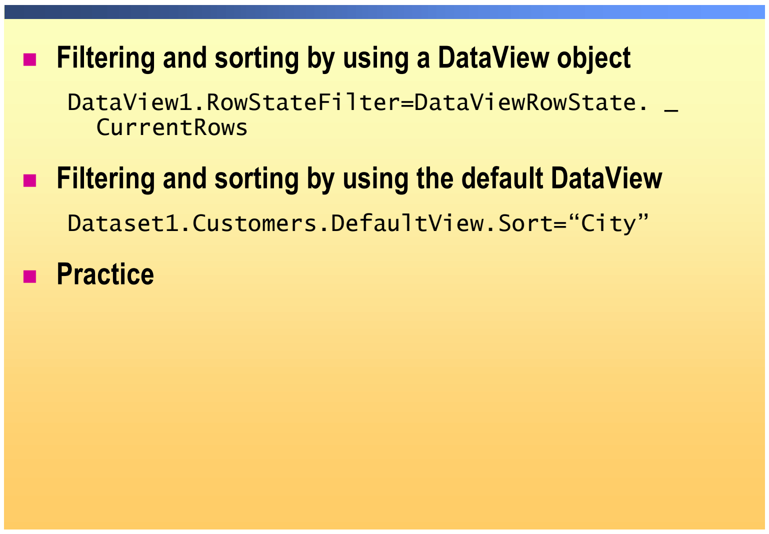
Applying a DataView to a DataTable

A **DataView** can be created independently of a **DataTable** and applied later to different **DataTables** dynamically.

To apply a **DataView** to a **DataTable**, set the **Table** property of the **DataView**, as shown in the following example.

```
Dim dvProducts As New DataView()
dvProducts.Table = dsNorthwind.Tables("Products").
```

How to Sort and Filter a DataTable Using a DataView



- Filtering and sorting by using a DataView object

```
DataView1.RowStateFilter=DataViewRowState. _
  CurrentRows
```

- Filtering and sorting by using the default DataView

```
Dataset1.Customers.DefaultView.Sort="City"
```

- Practice

Introduction

Sorting with a **DataView** object allows you to set sort criteria at design time, and provides an object that you can use for data binding.

You can filter and sort a **DataTable** by using a **DataView** object that you have explicitly added to a form or component. Doing this allows you to set filter and sort options at design time.

Alternatively, you can use the default **DataView**, called **DefaultView**, which is available for every table in a **DataSet**. When you use the default view, you can specify filter and sort options programmatically.

Filtering and sorting by using a DataView object

► **To filter and sort by using a DataView object**

1. If you want to set **DataView** options at design time, add a **DataView** object to the form or component.
2. Set the DataView **Sort** property by using a sort expression. The sort expression can include the names of **DataColumns** or a calculation. If you set the sort expression at run time, the **DataView** reflects the change immediately.
3. Set the DataView **RowFilter** property by using a filter expression. The filter expression should evaluate to **true** or **false**, as in the following expression:

```
dvProducts.RowFilter = "CategoryID = 3"
```

To filter based on a version or state of a record, set the **RowStateFilter** property to a value from the **DataViewRowState** enumeration, such as the following:

```
dvProducts.RowStateFilter = DataViewRowState.CurrentRows
```

Filtering and sorting by using the default DataView

The following example shows how to set the **DataView** filter and sort order at run time by using the default **DataView**:

```
dtCustomers.DefaultView.Sort = "City"
```

Practice

The Northwind Traders Sales Director wants the ability to filter the products in the catalog based on customer requirements.

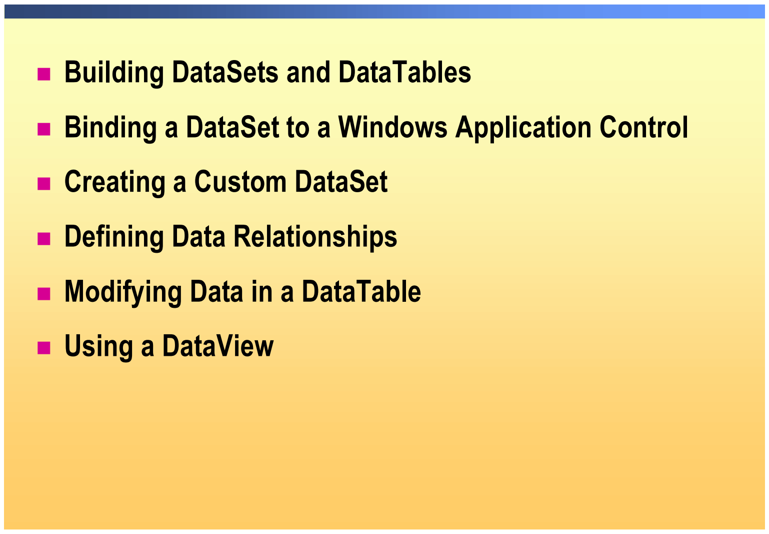
1. Open the solution you built for the previous practice, or open the solution at the following location.

<install folder>\Practices\Mod04_2\Lesson2\HandlingDataTableEvents\

2. Open the form in Code view.
3. Add code to the **Form1_Load** event handler to create a new **DataView** called **dvExpensiveProducts** based on the **dtProducts** variable.
4. Add code to sort the **DataView** on the **UnitsInStock** column.
5. Add code to filter the **DataView** so that only products costing more than $50 are displayed.
6. Run and test the application.

The solution for this practice is located at <install folder>\ Practices\Mod04_2\Lesson3\SortAndFilterUsingADataView\

Review



- Building DataSets and DataTables
- Binding a DataSet to a Windows Application Control
- Creating a Custom DataSet
- Defining Data Relationships
- Modifying Data in a DataTable
- Using a DataView

1.

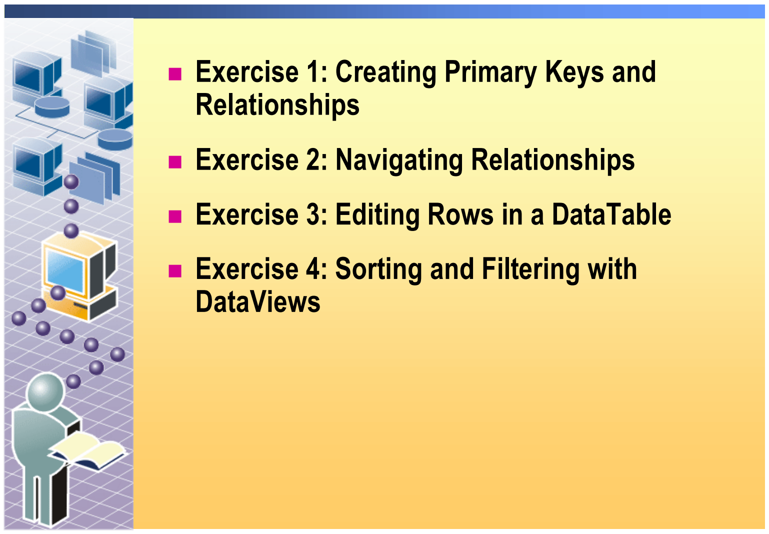
3.

4.

5.

6.

Lab 4.2: Manipulating DataSets and Modifying Data



- Exercise 1: Creating Primary Keys and Relationships
- Exercise 2: Navigating Relationships
- Exercise 3: Editing Rows in a DataTable
- Exercise 4: Sorting and Filtering with DataViews

Objectives

After completing this lab, you will be able to:

- Use a custom DataSet by using inheritance.
- Define a relationship.
- Modify data in a DataTable.
- Find and select rows in a DataTable.
- Sort and filter a DataTable by using a DataView object.

Prerequisites

Before working on this lab, you must have:

-

Scenario

Estimated time to complete this lab: 45 minutes

Exercise 0
Lab Setup

To complete this lab, you must …

►

2.

3.

Exercise 1
Creating Primary Keys and Relationships

In this exercise, you will add primary keys to the DataTables you created in Lab 4.1, "Building, Binding, Opening, and Saving DataSets," and define a relationship between the two tables.

Scenario

The Northwind Traders database is normalized to reduce data redundancy and duplication. To extract useful information, you must join together the data in the disparate tables by using relationships before displaying it to users.

► To continue building the application

- Open the solution you created in Lab 4.1, or open the solution **BuildingDataSets** in the folder in <install folder>\Labs\Lab04_2\Starter\xx\ where xx is either VB or CS.

► To give the Customer table a primary key

1. Open the **NWShoppingCart** class module.
2. After the code that creates the Customer table, programmatically create a primary key by using the following information.

Attribute	Value
Name	PK_CustomerID
Table	Customer
Column	CustomerID

3. Add a new private procedure named **AddSchema** to the class.
4. After the code that creates the CartItems table, insert code to call the **AddSchema** procedure.

► To relate the Customer and CartItems tables

1. Add code to the **AddSchema** procedure to create a relationship (including a foreign key constraint with default rules) between the Customer and CartItems tables by using the following information.

Attribute	Value
Name	FK_Customer_CartItems
Parent Table	Customer
Parent Column	CustomerID
Child Table	CartItems
Child Column	CustomerID

2. Wrap the code that creates the relation in an **If** statement that checks whether the relation already exists before attempting to create the relation. Hint: most collections provide a **Contains** method for this purpose.

► To clear the existing schema

Now that the custom DataSet creates a more complex schema, you must provide a method to dismantle it when necessary.

1. Add a new public procedure named **ClearSchema** to the class.
2. Write a line of code to remove the **ForeignKeyConstraint** created by the relationship.
3. Write a line of code to clear any existing relationships.
4. Write a line of code to clear any existing tables.
5. Modify the **OpenFromFile** method to call the **ClearSchema** method before attempting to open a DataSet file.
6. Modify the **OpenFromFile** method to call the **AddSchema** method after attempting to open a DataSet file.

► To test the relationship

1. Run the application.
2. Add a new customer record, or open a DataSet file that you created previously.
3. Notice that the DataGrid recognizes the relationship and automatically provides the ability to drill down to a customer's shopping cart items.
4. Add a new cart item for a customer. Notice that the customer ID field is automatically completed if the drill down feature is used.
5. Test the primary key by attempting to add another customer with a duplicate CustomerID value.

Exercise 2
Navigating Relationships

In this exercise, you will use the relationship defined between the two tables in Exercise 1 to provide a summation of the total order value of a customer's shopping cart items. You will use two techniques: programmatically navigating the object model, and adding an expression column that can automatically summarize child rows.

Scenario

To make its e-commerce Web site more convenient for users, Northwind Traders wants to give its customers the ability to view a running total of the order value of their shopping cart items.

► **To start with the solution to the previous exercise**

- If you did not complete the previous exercise open the solution **BuildingDataSets** in the folder <install folder>\Labs\Lab04_2\Solution\Ex1\xx\ where xx is either VB or CS.

► **To display a shopping cart order total**

1. Open the **Form1** class module in Designer view.
2. Add a new menu item to the **View** menu using the following information.

Menu	Property	Value
mnuSubtotal	Text	Cart &Subtotal
	Shortcut	F9

(Optional) Add menu separators where appropriate.

► **To calculate the subtotal**

1. Add code to handle the **Click** event for the **Cart Subtotal** menu item.
2. Declare the following local variables.

Name	**Data type**
sCustomerID	System.String
drCustomer	System.Data.DataRow
drCartItem	System.Data.DataRow
dSubtotal	System.Decimal

3. Initialize the subtotal to zero.
4. Set the string variable to be the CustomerID value of the currently selected customer in the DataGrid.

Tip The **CurrentRowIndex** property returns the current row in a DataGrid.

5. Set the customer DataRow variable to point to the correct row in the **Customer** DataTable by using the CustomerID retrieved in the previous line as a criteria value for a DataRow selection filter.

Tip Use the **Select** method provided by the **DataTable** class.

Caution The **Select** method returns an array of DataRows. Even if only one DataRow matches the select filter, you must use array syntax to retrieve an individual DataRow.

6. Write a **For Each** statement to loop through all of the child rows returned by a call to the **GetChildRows** method of the customer DataRow.
7. Inside the loop, increment the subtotal variable by the value in the **Cost** column of the DataRow.
8. Use a message box to show the company name of the customer and the amount they owe based on the items currently in their shopping cart.

► **To add a cart total to the Customer table**

1. Open the **NWShoppingCart** class module.
2. Add code to the **AddSchema** method to check for an existing column in the Customer table named **CartSubtotal**.
3. If the column does not exist, write code to add the column with the following attributes.

DataColumn	**Property**	**Value**
CartSubtotal	DataType	System.Decimal
	Expression	Sum(Child.Cost)

4. Because the **AddSchema** method is called when a new DataSet is instantiated *and* when an existing DataSet file is opened, there is no additional code to write.

► **To test the cart totals**

1. Run the application.
2. Add a new customer record.
3. Add some cart items and then return to the customer view.
4. Notice that the DataGrid automatically calculates the cart subtotal and displays the results in the grid as an additional column.
5. On the **View** menu, click **Cart Subtotal**. The same subtotal value should be returned.

Exercise 3
Editing Rows in a DataTable

In this exercise, you will provide a simple method to add new cart items to the shopping cart. To supply the basic values required for a cart item, you will open an existing DataSet file that contains sample categories and products.

Scenario

When browsing the Northwind Traders e-commerce Web site, a customer will want the ability to add an item to his or her shopping cart with a single click.

► To start with the solution to the previous exercise

- If you did not complete the previous exercise open the solution **BuildingDataSets** in the folder <install folder>\Labs\Lab04_2\Solution\Ex2\xx\ where xx is either VB or CS.

► To open the existing DataSet file

1. Open the **Form1** class module.
2. Declare a class-level variable named dsCatProd by using the standard **DataSet** class as the data type.
3. Add code to the **Form1_Load** event to set the dsCatProd variable to point to a new instance of the **DataSet** class.
4. Call the **ReadXml** method of the **dsCatProd** object to open the existing DataSet file (including the schema) from the following location on your training computer:

   <install folder>\Labs\Lab04_2\CatProd.ds

► To code the View menu items

1. Add code to the **mnuCustomer_Click** event to clear the check mark next to the **mnuProducts** menu item.
2. Add code to the **mnuCartItems_Click** event to clear the check mark next to the **mnuProducts** menu item.
3. Add code to the **mnuProducts_Click** event to place a check mark next to the **mnuProducts** menu item, and then clear the check marks next to the **mnuCustomer** and **mnuCartItems** menu items.
4. Add code to the **mnuProducts_Click** event to set the DataGrid to bind to the **Products** table in the **dsCatProd** DataSet.

► To create the menu to add a product to the cart

1. Use the menu editor to insert an **Edit** menu named mnuEdit with a single **Add To Cart** menu item named mnuAddToCart.

   The **Add To Cart** menu should only be available when the products are displayed in the DataGrid.
2. Add code to handle the **Edit** menu's **Select** event.
3. Add a line of code to set the **Enabled** property of the **Add To Cart** menu item to have the same value as the **Checked** property of the **Products** menu item on the **View** menu.

► **To test the changes**

1. Run the application.
2. Use the **View** menu to toggle the DataGrid to show the products available on the Northwind Traders e-commerce Web site.
3. Ensure that the **Add To Cart** menu is only available when products are shown in the DataGrid.

► **To add a product to the shopping cart**

1. Add code to handle the **Click** event for the **Add To Cart** menu item.
2. Declare local variables to store the CustomerID, ProductID, UnitPrice, and Quantity that the user selects. Use the generic **System.Object** data type for all four.
3. Set the CustomerID to the value in the first column of the first row of the **Customer** DataTable. (To simplify this example, assume that there is only one customer. If you have time at the end of this lab, you can add code to allow the user to select a customer as well as a product and quantity.)
4. Set the ProductID variable to the **ProductID** column of the product currently selected in the DataGrid.
5. Set the UnitPrice variable to the **UnitPrice** column of the product currently selected in the DataGrid.
6. Use the **InputBox** method to prompt the user for a quantity and store the value in the Quantity variable.

Tip C# does not support the **InputBox** method natively. You must add a reference to the **Microsoft Visual Basic .NET Runtime** assembly, and then use a fully-qualified method call. For example, see the code below.

```
// Note: all five parameters are mandatory
Microsoft.VisualBasic.Interaction.InputBox("Prompt",
   "Title", "DefaultResponse", XPos, YPos);
```

7. Try to convert the Quantity variable to an integer. If the conversion fails (for example, if the user entered an invalid value), exit the procedure.
8. Try to add the values as an array of objects using the **Add** method of the **Rows** collection of the **CartItems** DataTable. Catch any exceptions.

► **To test the application**

1. Run the application.
2. Enter a new customer or open an existing file.
3. View the products.
4. Select a product in the DataGrid.
5. On the **Edit** menu, click **Add To Cart**.
6. Enter a quantity when prompted.
7. Repeat the procedure for several products.
8. Switch the view to see the cart items.

Exercise 4
Sorting and Filtering with DataViews

In this exercise, you will add the ability to filter the products list. The DataGrid automatically provides basic sorting capability, but you will provide the user with the ability to view all products, or to filter out discontinued or out-of-stock products, or both.

Scenario

When browsing the Northwind Traders e-commerce Web site, a customer will want the ability to view the products in different ways; for example, products that match some criteria sorted with the cheapest products first. Customers may only want to see products that are in stock, or that have not been discontinued.

► To start with the solution to the previous exercise

- If you did not complete the previous exercise open the solution **BuildingDataSets** in the folder <install folder>\Labs\Lab04_2\Solution\Ex3\xx\ where xx is either VB or CS.

► To open the existing DataSet file

1. Open the **Form1** class module in Designer view.
2. Add two menu items to the **View** menu using the following information.

Menu	Property	Value
mnuDiscontinued	Text	&Discontinued
	Checked	True
mnuOutOfStock	Text	&Out Of Stock
	Checked	True

   (Optional) Add menu separators where appropriate.

   These menu items should only be available when the products are displayed in the DataGrid.

3. Add code to handle the **View** menu's **Select** event.
4. Add some code to set the **Enabled** property of the two menu items to have the same value as the **Checked** property of the **Products** menu item on the **View** menu.

In the following steps, you will add code to handle the **Click** event for these menu items.

► **To toggle the display of discontinued and out-of-stock products**

1. Add a new private procedure named **FilterMenus**. This procedure must be able to handle the **System.EventHandler** delegate.

   ```
   ' Visual Basic
   Public Delegate Sub EventHandler( _
      ByVal sender As Object, _
      ByVal e As EventArgs)

   // Visual C#
   public delegate void EventHandler(
      object sender,
      EventArgs e);
   ```

2. Set the procedure to handle the **mnuDiscontinued.Click**, **mnuOutOfStock.Click**, and **mnuProducts.Click** events.

   Tip Multiple procedures can handle an event. For example, we now have two procedures that handle the **mnuProducts.Click** event; **FilterMenus** and **mnuProducts_Click**.

3. Toggle the **Checked** property of the sender object, if the sender is not **mnuProducts**.

   Tip If you are using C#, you must declare a **System.Windows.Forms.MenuItem** variable, cast the sender object to it, and then use that variable instead, because C# does not support late binding.

4. Declare a local DataView variable named **dv** and instantiate it by using the **Products** DataTable.
5. If the **Discontinued** menu item is not selected, set the **RowFilter** property of the DataView variable to only show rows where the **Discontinued** column is **False.**
6. If the **Out Of Stock** menu item is not selected, set the **RowFilter** property of the DataView variable to only show rows where the **OutOfStock** column is **False.**
7. If both the **Discontinued** and the **Out Of Stock** menu items are not selected, set the **RowFilter** property of the DataView variable to only show rows where the **Discontinued** column is **False** and the **OutOfStock** column is **False.**
8. Set the **DataSource** property of the DataGrid to the DataView named **dv**.
9. In the **mnuProducts_Click** procedure, delete the line of code that sets the **DataSource** of the DataGrid, because the **FilterMenus** procedure does this now.

► **To test the application**

1. Run the application.
2. Use the **View** menu to display the **Products** table.
3. Click the Discontinued column heading in the grid to sort the rows by this value.
4. Use the **View** menu to toggle the display of discontinued and/or out-of-stock products.

   (Optional) Add a menu item to the View menu to provide complex sorting capability programmatically, for example sorting on two columns simultaneously.

msdn® training

Module 5: Using XML with ADO .NET (Prerelease)

Contents

Microsoft®

Instructor Notes

**Presentation:
60 Minutes**

**Lab:
60 Minutes**

Microsoft® ADO.NET uses XML as the format for managing and moving data from a data source to a **DataSet** object and back. In addition, work directly with data in XML format in an ADO.NET application. This module explains how to use XML Schema Definition Language (XSD) to create files that provide structure for working with data in XML documents and in **DataSet**s.

After completing this module, students will be able to:

- Generate an XSD schema from a DataSet by using graphical tools.
- Identify the purpose and uses of the XmlDataDocument object.
- Save a DataSet structure to an XSD schema file.
- Create and populate a DataSet from an XSD schema and XML data.
- Load data and schema simultaneously into a DataSet.
- Save DataSet data as XML.
- Write and load changes by using a DiffGram.
- Manipulate data in an XmlDataDocument object.

Required materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2389A_05.ppt
- Module 5, "Reading and Writing XML With ADO .NET"
- Lab 5, Working with XML Data in ADO .NET

Preparation tasks

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices and labs.
- Read the latest .NET Development news at http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000519

How to Teach This Module

This section contains information that will help you to teach this module.

Lesson: Creating XSD Schemas

This section describes the instructional methods for teaching each topic in this lesson.

What is an XSD Schema?

Transition to Practice Exercise:

In this practice, you will create an XML file based on a schema by using the Visual Studio XML Editor.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook. Guide the students through the practice.

What Are Strongly-Typed DataSets?

Discussion Questions: Personalize questions to the background of the students in your class.

How Schema Information Maps to Relational Structure

Discussion Questions: Personalize questions to the background of the students in your class.

Generating an XSD Schema With Visual Studio

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Lesson: Creating a Strongly-Typed DataSet

This section describes the instructional methods for teaching each topic in this lesson.

Loading a Schema Into a DataSet

Discussion Questions: Personalize questions to the background of the students in your class.

Examining Meta Data

Discussion Questions: Personalize questions to the background of the students in your class.

Demonstration: Examining DataSet Structure

Discussion Questions: Personalize questions to the background of the students in your class.

Loading XML Data Into a DataSet

Discussion Questions: Personalize questions to the background of the students in your class.

Lesson: Writing XML From a DataSet

This section describes the instructional methods for teaching each topic in this lesson.

Writing Schema to a File, Reader, or Stream

Discussion Questions: Personalize questions to the background of the students in your class.

Writing DataSet Information to a File or Stream

Discussion Questions: Personalize questions to the background of the students in your class.

Demonstration: Writing Inline Schema and Data to a File

Discussion Questions: Personalize questions to the background of the students in your class.

Writing DataSet Changes

Discussion Questions: Personalize questions to the background of the students in your class.

Lesson: Using the XmlDataDocument Object

This section describes the instructional methods for teaching each topic in this lesson.

What Is an XmlDataDocument Object?

Discussion Questions: Personalize questions to the background of the students in your class.

Synchronizing an XmlDataDocument With a DataSet

Discussion Questions: Personalize questions to the background of the students in your class.

Providing a Hierarchical View of Existing Relational Data

Discussion Questions: Personalize questions to the background of the students in your class.

How to Provide a Relational View of XML Data

Discussion Questions: Personalize questions to the background of the students in your class.

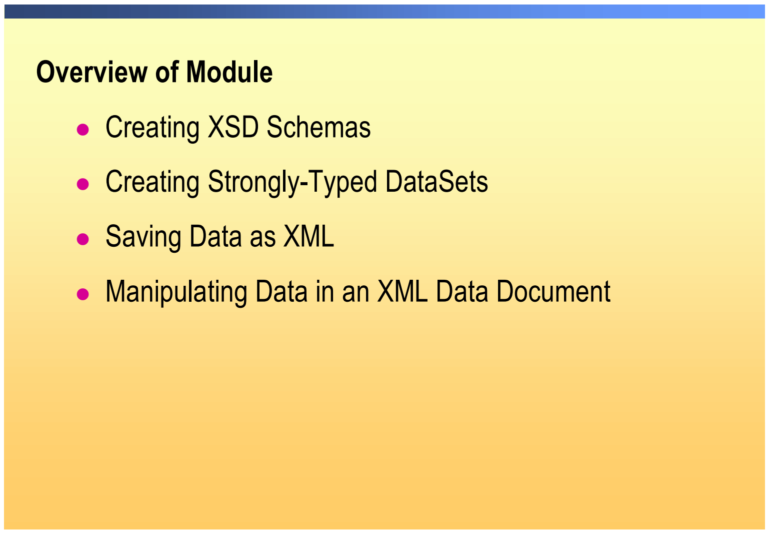
Applying an XSLT Style Sheet to a DataSet

Discussion Questions: Personalize questions to the background of the students in your class.

Performing an XPath Query on a DataSet

Discussion Questions: Personalize questions to the background of the students in your class.

Overview



Overview of Module

- Creating XSD Schemas
- Creating Strongly-Typed DataSets
- Saving Data as XML
- Manipulating Data in an XML Data Document

Introduction

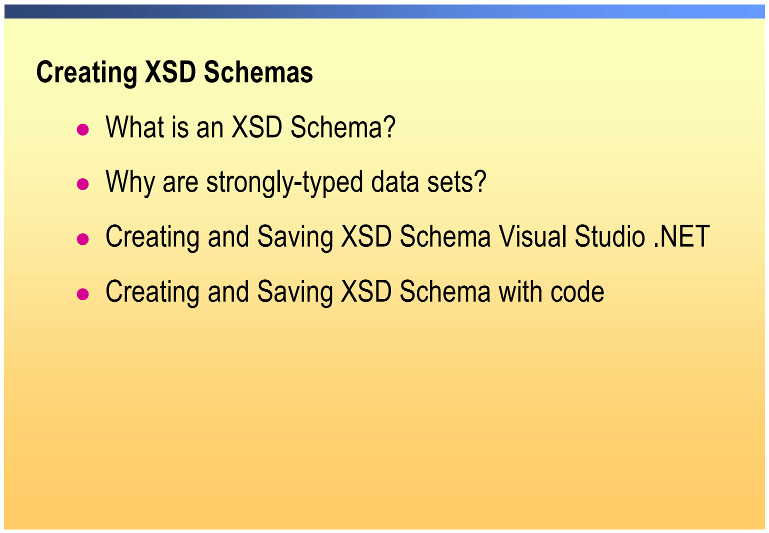
Microsoft® ADO.NET uses XML as the format for managing and moving data from a data source to a **DataSet** object and back. In addition, work directly with data in XML format in an ADO.NET application. This module explains how to use XML Schema Definition Language (XSD) to create files that provide structure for working with data in XML documents and in **DataSet**s.

Objectives

After completing this module, you will be able to:

- Generate an XSD Schema from a **DataSet** by using graphical tools.
- Identify the purpose and uses of the **XmlDataDocument** object
- Save a **DataSet** structure to an XSD Schema file.
- Create and populate a **DataSet** from an XSD Schema and XML data.
- Load data and schema simultaneously into a **DataSet**.
- Save **DataSet** data as XML.
- Write and load changes by using a **DiffGram**.
- Manipulate data in an **XmlDataDocument** object.

Lesson: Creating XSD Schemas



Creating XSD Schemas

- What is an XSD Schema?
- Why are strongly-typed data sets?
- Creating and Saving XSD Schema Visual Studio .NET
- Creating and Saving XSD Schema with code

Introduction

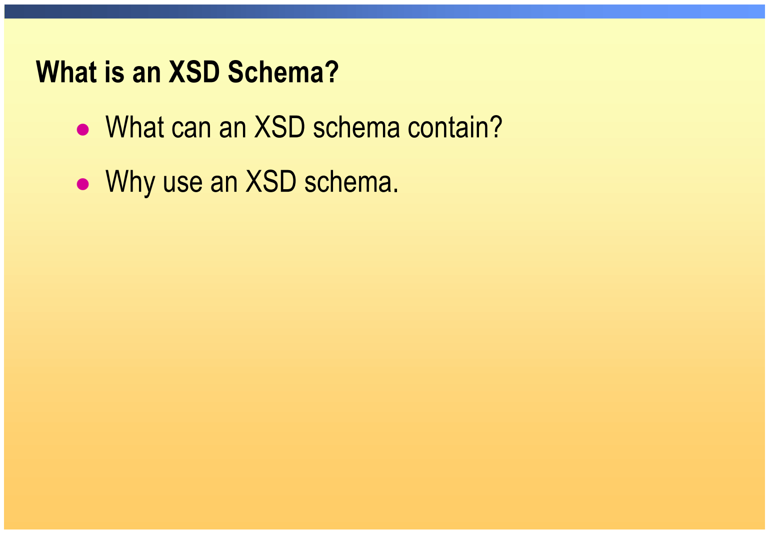
XSD schemas provide the mapping between relational data in **DataSet**s and data in XML documents. XSD schemas define the structure of the data in a dataset, so that the data can be expressed and used in XML format.

Lesson objectives

After completing this lesson, you will be able to:

- Provide a definition of an XSD schema.
- Give examples of when to use an XSD schema.
- Create and save an XSD schema by using Microsoft Visual Studio® .NET.
- Create and save an XSD schema by using code.

What Is an XSD Schema?



What is an XSD Schema?

- What can an XSD schema contain?
- Why use an XSD schema.

XSD example XML document code example XML document in browser

Introduction

When you create a new **DataSet**, no tables or other structures are defined. The structure is usually loaded from an XSD file, or you can create the structure by using code.

Definition

An *XSD schema* is a document that describes the structure of an XML document, as well as the constraints on data in that document.

What can an XSD schema contain?

An XSD schema can contain the following information:

- A representation of relationships between data items similar to the foreign key relationships between tables in a relational database.
- A representation of constraints similar to the primary key and Unique constraints in a relational model.
- A specification of the data types of each individual element and attribute in an XML document that complies with the XSD schema.

Why use an XSD schema?

There are two basic reasons to use XSD schemas:

- To import data and know the structure of the data that you import.

  When you receive data in XML format and want to load that data into a **DataSet,** you can use a schema to define the structure of the data you are reading.
- To describe the structure of data that you are exporting to another consumer.

  When you want to send data from a **DataSet** and put it into an XML data document, you can provide a schema to describe the data.

Example of an XSD schema

The following schema shows a two-level XSD schema.

```
<?xml version="1.0" standalone="yes"?>
<xsd:schema id="PersonPet" targetNamespace="" xmlns=""
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-
com:xml-msdata">
  <xsd:element name="PersonPet" msdata:IsDataSet="true">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="Person">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="ID" msdata:AutoIncrement="true" type="xsd:int" />
              <xsd:element name="Name" type="xsd:string" minOccurs="0" />
              <xsd:element name="Age" type="xsd:int" minOccurs="0" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Pet">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="ID" msdata:AutoIncrement="true" type="xsd:int" />
              <xsd:element name="OwnerID" type="xsd:int" minOccurs="0" />
              <xsd:element name="Name" type="xsd:string" minOccurs="0" />
              <xsd:element name="Type" type="xsd:string" minOccurs="0" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
    <xsd:unique name="Constraint1" msdata:PrimaryKey="true">
      <xsd:selector xpath=".//Person" />
      <xsd:field xpath="ID" />
    </xsd:unique>
    <xsd:unique name="Pet_Constraint1" msdata:ConstraintName="Constraint1"
msdata:PrimaryKey="true">
      <xsd:selector xpath=".//Pet" />
      <xsd:field xpath="ID" />
    </xsd:unique>
  </xsd:element>
  <xsd:annotation>
    <xsd:appinfo>
      <msdata:Relationship name="PersonPet" msdata:parent="Person"
msdata:child="Pet" msdata:parentkey="ID" msdata:childkey="OwnerID" />
    </xsd:appinfo>
  </xsd:annotation>
</xsd:schema>
```

Example of an XML document based on a schema

The following example shows an XML data document that is based on the schema in the preceding example.

```
<PersonPet xmlns = "http://tempuri/PersonPet.xsd">
  <Person>
    <ID>0</ID>
    <Name>Mark</Name>
    <Age>18</Age>
  </Person>
  <Person>
    <ID>1</ID>
    <Name>William</Name>
    <Age>12</Age>
  </Person>
  <Person>
    <ID>2</ID>
    <Name>James</Name>
    <Age>7</Age>
  </Person>
  <Person>
    <ID>3</ID>
    <Name>Levi</Name>
    <Age>4</Age>
  </Person>
  <Pet>
    <ID>0</ID>
    <OwnerID>0</OwnerID>
    <Name>Frank</Name>
    <Type>cat</Type>
  </Pet>
  <Pet>
    <ID>1</ID>
    <OwnerID>1</OwnerID>
    <Name>Rex</Name>
    <Type>dog</Type>
  </Pet>
  <Pet>
    <ID>2</ID>
    <OwnerID>2</OwnerID>
    <Name>Cottontail</Name>
    <Type>rabbit</Type>
  </Pet>
  <Pet>
    <ID>3</ID>
    <OwnerID>3</OwnerID>
    <Name>Sid</Name>
    <Type>snake</Type>
  </Pet>
  <Pet>
    <ID>4</ID>
    <OwnerID>3</OwnerID>
    <Name>Tickles</Name>
    <Type>spider</Type>
  </Pet>
```

(Code continued on next page.)

```
  <Pet>
    <ID>5</ID>
    <OwnerID>1</OwnerID>
    <Name>Tweetie</Name>
    <Type>canary</Type>
  </Pet>
</PersonPet>
```

Practice

Create an XML file based on a schema by using the Visual Studio XML Editor.

► **To create an XML file based on a target schema**

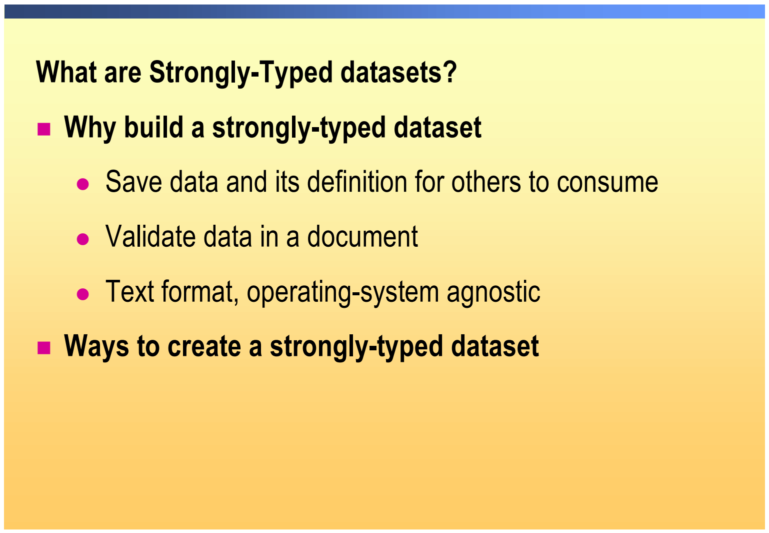
1. Start Visual Studio and create a new empty project.
2. Click **File**, and then click **Add Existing Item**. Add **PhoneList.xsd** to the project. Double-click the file in the Solution Explorer to open an editor window. The editor opens by default in DataSet view showing a graphical representation of the schema for the DataSet.
3. Switch to the XML view and examine the schema. Note that a <Customer> element has three child elements and one attribute.
4. Click **File,** and then click **Add New Item**. Add a new XML file named **PhoneList.xml** to the project.
5. Right-click the new XML document, and then click **Properties**.
6. Change the Target Schema to **http://tempuri.org/PhoneList.xsd**, and then click **OK**. Notice that Visual Studio automatically adds a schema reference and a parent tag to the XML document.

► **To add data to the XML document**

1. Type an '<' between the <CustomerData> tags. Visual Studio suggests an appropriate XML element based on the target schema. Press the TAB key to insert the <Customers> element into the XML document.
2. Type a space. Visual Studio suggests an XML attribute based on the target schema. Press the TAB key to insert the **ContactName** attribute into the XML document.
3. Type an "=". Visual Studio inserts a set of quotation marks. Type your own name as a value for the **Contact Name** attribute between these quotation marks.
4. Type a '>' after your name in quotation marks. Visual Studio adds a closing tag for the <Customers> element. Add some data to the body of the element
5. In between the <**customers**> and **</customers>** tags, type a '<'. Visual Studio suggests appropriate child elements based on the target schema. Double-click the suggested tag to insert a <CompanyName> child element into the XML document.
6. Type an '>' to add a closing tag for the <CompanyName> element. Add data for the company name.

7. Use the Visual Studio XML Editor to complete the information for this customer.
8. Save the changes. Notice that the XML Editor automatically reformats your document.
9. Switch the XML Editor to Data view.
10. Enter another customer.
11. Switch the XML Editor to XML view. Notice that the tags and data are entered for you.

What Are Strongly-Typed DataSets?



What are Strongly-Typed datasets?

- **Why build a strongly-typed dataset**
  - Save data and its definition for others to consume
  - Validate data in a document
  - Text format, operating-system agnostic
- **Ways to create a strongly-typed dataset**

Definition of strongly-typed DataSet

A strongly-typed **DataSet** is an instance of a **DataSet** for which schema information has been defined.

Why build a strongly-typed DataSet?

When you import data from an XML file, you must know how it maps to a relational structure. That information is usually contained in an XSD file, or in an inline schema within the XML document. You use this schema information to build your data structure.

After the structure is defined, Visual Studio .NET creates a class based on this **DataSet** structure. The name that you give to the **DataSet** is the name of the class.

When you create a new **DataSet** based on this class, you create a strongly-typed **DataSet** that inherits the tables, columns, and other structure from the parent class. You can then load data from an XML file into this instance of the **DataSet**.

You use an XSD to create tables, columns, data types, constraints, and relationships in a **DataSet**. When the structure of the **DataSet** is complete, you can load it with data.

The basic working cycle for strongly-typed **DataSet**s follows these steps:

1. Create a blank **DataSet**.
2. Load the schema information from an XSD file.
3. Create a new **DataSet** object based on the schema.

Ways to dynamically load a schema into a dataset

You can dynamically load a schema into a **DataSet** by:

- Using an XSD schema.

  When you load an XSD file into a **DataSet**, the **DataSet** generates tables, relationships, and constraints based on the data structure described in the XSD schema. This relational representation does not capture all of the detail that is represented by an XSD file, but uses only the information that is required to construct tables, columns, data types, unique constraints, and foreign keys in the relational model. The XSD schema can exist in a separate XSD file, or as an inline schema that precedes the data in an XML data file.

- Inferring a schema from XML data.

  If you have some XML data but no schema, you can generate a schema based on the structure of the XML data. In some cases, the data may be ambiguous or inconsistent. Therefore, if an appropriate schema exists, you should use this schema rather than inferring one from the XML data.

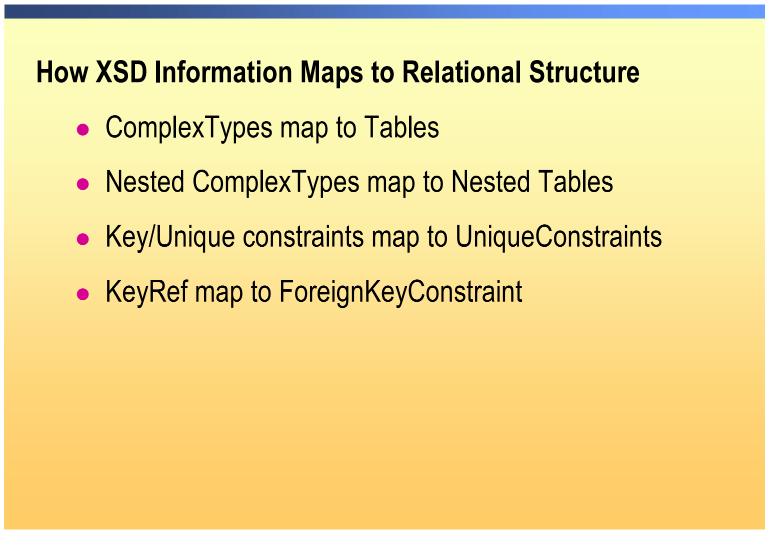
- Manually creating the structure of the **DataSet** by using code to build tables and create relationships.

Save data and its definitions for others to consume

In the .NET world, applications communicate by sending XML documents. When you send an XML document to someone, you must send something that describes the structure of the data, data types, relationships, and constraints that are contained in the document.

You provide this description by including an XSD schema or an inline schema with your data.

How Schema Information Maps to Relational Structure



How XSD Information Maps to Relational Structure

- ComplexTypes map to Tables
- Nested ComplexTypes map to Nested Tables
- Key/Unique constraints map to UniqueConstraints
- KeyRef map to ForeignKeyConstraint

XSD Schema

Introduction

When you use a schema to define your **DataSet**, certain types of schema tags generate certain relational objects.

<complexType> elements map to tables

Within an n XSD schema, there is an element called <complexType>. A **complexType** name usually maps to a table name. A **complexType** definition can contain elements and attributes, which usually map to column names.

Nested <complexType> elements map to nested tables

A **complexType** can also contain other **complexTypes**. In this case, the nested **complexType** maps to a child table in a parent-child relationship in the relational model.

<key> and <unique> constraints map to UniqueConstraint

Schemas can contain **<key>** and **<unique>** elements, often at the end of the schema. These map to primary keys and unique constraints in the relational model.

<keyref> maps to ForeignKeyConstraint

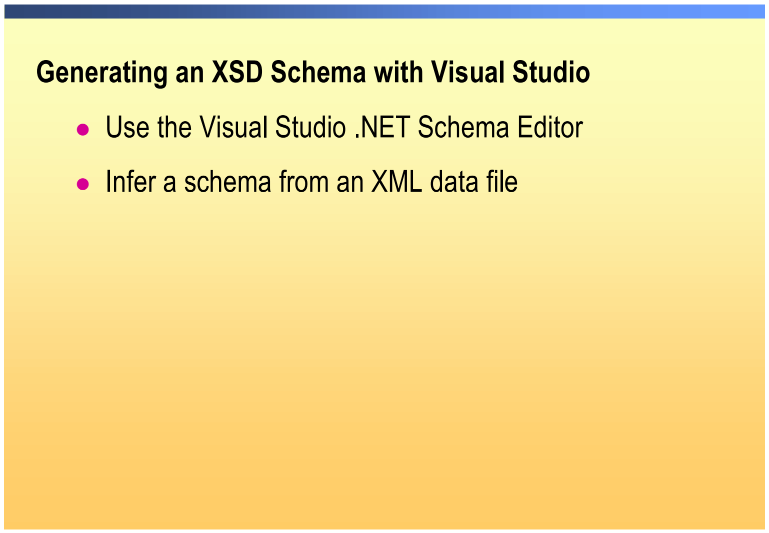
<keyref> elements define the relationships between data items in the schema. **<keyref>** elements map to foreign key constraints in the relational model.

Example

The following example shows how to use an XSD schema that defines the parts of a relational table named "Orders."

```
<!-- The element name followed by a complexType defines the "Orders" table -->

<xsd:element name="Orders" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>

      <xsd:sequence>

         <xsd:element name="OrderID" type="xsd:int"/>

         <!--This next block defines the OrderDetails table -->
         <xsd:element name="OrderDetails" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="ProductID" type="xsd:int"/>
                  <xsd:element name="UnitPrice" type="xsd:number"/>
                  <xsd:element name="Quantity"  type="xsd:short"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>

         <xsd:element name="OrderDate" type="xsd:dateTime" minOccurs="0"/>

      </xsd:sequence>

      <xsd:attribute name="CustomerID" type="xsd:string" use="prohibited" />

   </xsd:complexType>
</xsd:element>

<xsd:unique name="Orders_Constraint">       <!-- Each OrderID is unique -->
   <xsd:selector xpath=".//Orders" />
   <xsd:field xpath="OrderID" />
</xsd:unique>

<xsd:key name="OrderDetails_Constraint">    <!-- Primary key -->
   <xsd:selector xpath=".//OrderDetails" />
   <xsd:field xpath="OrderID" />
   <xsd:field xpath="ProductID" />
</xsd:key>
```

Generating an XSD Schema with Visual Studio



Generating an XSD Schema with Visual Studio

- Use the Visual Studio .NET Schema Editor
- Infer a schema from an XML data file

Introduction

You can use tools in the Visual Studio development environment to create and edit XSD schema documents. You can also use a simple text editor, such as Microsoft Notepad, to create a schema document.

Using the Visual Studio Schema Editor

The XML Designer, which contains the Schema Designer, provides the following advantages:

- Tag completion
- Color coding
- Ability to drag and drop standard XSD tags, such as elements, attributes,, complex types, simple types, keys, and relations into an XSD schema.

Inferring an XSD schema from an XML data file

If you have an XML data file and do not have a schema for the data, you can infer an XSD schema from the data file by following these steps:

1. Add a copy of the XML document into the Solution Explorer, and then view the document.
2. From the XML view of this document, right-click the file, and then select **Create Schema** to add an XSD file to the project.

Practice

Create an XSD schema by using the Visual Studio XML Editor.

► **To create a new schema**

1. Start Visual Studio, and then create a new empty project.
2. Click **File**, and then click **Add New Item**. Add a new XML Schema named **PurchaseOrder.xsd** to the project.
3. Drag a new element from the toolbox onto the designer surface. Change the element name to **PurchaseOrder**.
4. Drag a complex type to the designer surface. Change the complex type name to **Address**.
5. In the **Address** complex type, click the first cell on the first row, and click **element** in the drop-down list. This is a child element.
6. In the second cell on the first row you enter the name of the child element, type **Name**.
7. In the third cell on the first row you choose the data type of the child element, choose **string**.
8. Add the following three additional child elements to the **Address** complex type.

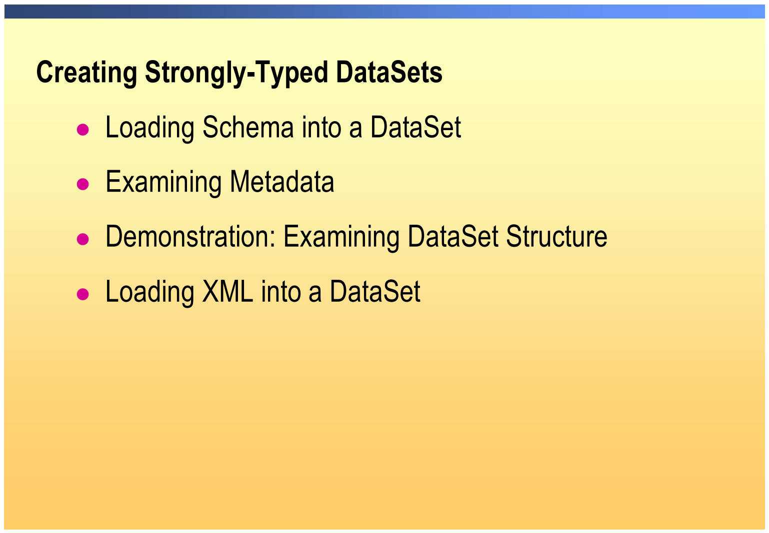
Name	**Data Type**
City	string
Street	string
Zip	integer

9. To the <PurchaseOrder> element, add the following child elements.

Name	**Data Type**
BillTo	Address
ShipTo	Address

10. Add a **Country** attribute to the **Address** complex type. Notice that the <BillTo> and <ShipTo> elements are updated to reflect changes in the data type.
11. Switch to the XML view to view the XSD schema.
12. Save **PurchaseOrder.xsd**, and then quit Visual Studio.

Lesson: Creating a Strongly-Typed DataSet



Creating Strongly-Typed DataSets

- Loading Schema into a DataSet
- Examining Metadata
- Demonstration: Examining DataSet Structure
- Loading XML into a DataSet

Introduction

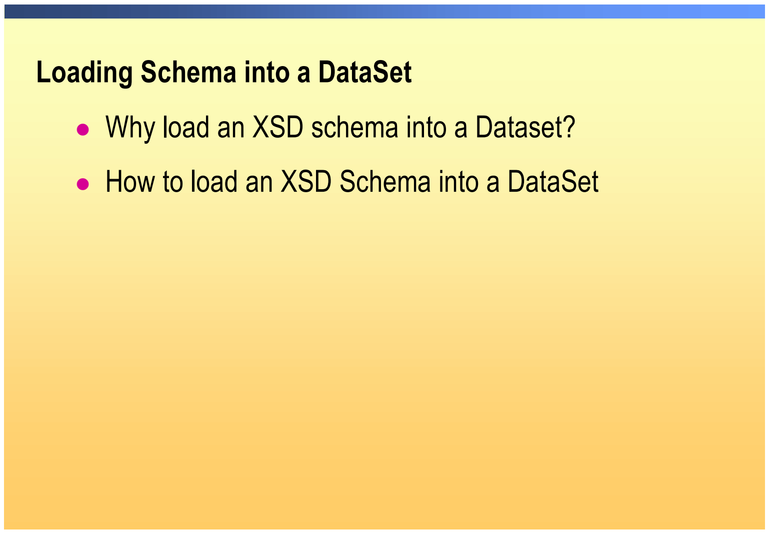
A strongly-typed **DataSet** is one that is based on an XSD schema. In this lesson, you will learn how to create and use strongly-typed **DataSet**s.

Lesson objectives

After completing this lesson, you will be able to:

- Load an XSD schema into a **DataSet**.
- Examine metadata.
- Load data in XML format into a **DataSet**.

Loading a Schema into a DataSet



Loading Schema into a DataSet

- Why load an XSD schema into a Dataset?
- How to load an XSD Schema into a DataSet

Visual Basic – loading an XSD Visual Basic – loading an XSD 2

Why load an XSD schema into a DataSet?

You may need to load data from an XML data file into a **DataSet** object. Before you load the data, you must create a relational data structure in the **DataSet**.

How to load an XSD schema into a DataSet

Use the **ReadXmlSchema** method of the **DataSet** object to load an XSD schema into a **DataSet**. This method is overloaded so that you can you use any of the following to supply the XSD information: a filename, a stream, a TextReader subclass object, or an XmlReader subclass object.

Syntax

The following example shows the syntax for using the **ReadXmlSchema** method of the **DataSet** object.

```
DataSet.ReadXMLSchema (ByVal filename as string | stream as
stream | reader as textreader | reader as xmlreader)
```

Example of loading an XSD from a file

This code loads an XSD schema from a file into a dataset. The XSD schema describes the structure of a purchase order.

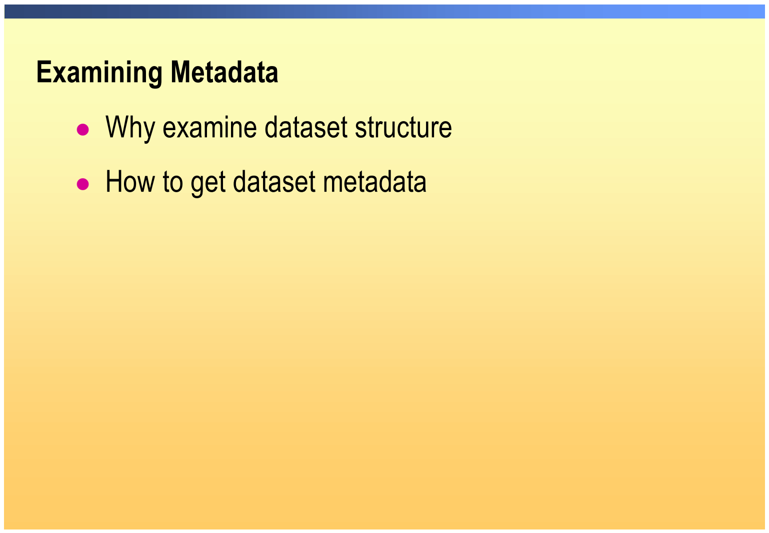
```
Private Const PurchaseSchema As String = _
                                "C:\sampledata\Purchase.xsd"
Private myDS as DataSet

Private Sub Load_XSD()
  Try
      myDS = New DataSet()
      Console.WriteLine ("Reading the Schema file")
      myDS.ReadXmlSchema(PurchaseSchema)
  Catch e as Exception
      Console.WriteLine("Exception: " & e.ToString())
  End Try
End Sub
```

Example of loading an XSD by using a Stream object

The following code loads an XSD into a **DataSet** by using a **Stream** object. In some applications, you may receive schema information in the form of a stream, from a Web service, or another Internet application.

```
Private Const PurchaseSchema As String = _
                                "C:\sampledata\Purchase.xsd"
Private myDS as DataSet

Private Sub Load_XSD()
  Dim myStreamReader As StreamReader = Nothing
  Try
      myStreamReader = New StreamReader(PurchaseSchema)
      myDS = New DataSet()
      Console.WriteLine ("Reading the Schema file")
      myDS.ReadXmlSchema(myStreamReader)
  Catch e as Exception
      Console.WriteLine("Exception: " & e.ToString())
  Finally
      If Not myStreamReader Is Nothing Then
          myStreamReader.Close()
      End If
  End Try
End Sub
```

Examining Metadata



Examining Metadata

- Why examine dataset structure
- How to get dataset metadata

Visual Basic Example

Why examine DataSet structure?

After you have loaded a schema into a **DataSet** object, or created an instance of a strongly-typed **DataSet** subclass, you can examine the structure of the tables and relationships in the dataset.

The dataset object contains information such as table names, column names, and data types. You can use this information to populate user interface controls and data displays.

How to get DataSet metadata

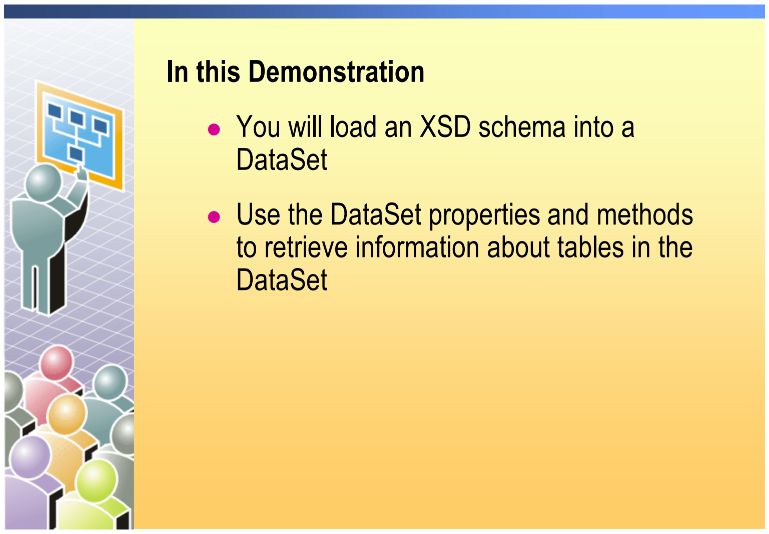
You can use the following properties of the dataset object to return information about the dataset structure

DataSet Property	What this gives you
Tables property	Allows you to reference the **DataTable** collection of a **DataSet**.
Relations property	Allows you to reference the **DataRelation** collection of a **DataSet**.
Tables.Count	Returns the number of tables in a **DataSet**.
Tables(*index*).TableName	Returns the name of a table in the **DataTable** Collection.
Tables(*tablename* | *index*).Columns(*index*)	Allows you to reference the **DataColumn** collection.
Tables(index).Columns(index).ColumnName	Returns the name of a column.
Tables(index).Columns(index).DataType	Returns the data type of a column.
Tables(index).Columns.Count	Returns the number of columns in a table.

Example The following example prints out the structure of a **DataSet** named myDS.

```
Private Sub DisplayTableStructure()

  Console.WriteLine("Table structure")

  'Print the number of tables
  Console.WriteLine("Tables count=" & myDS.Tables.Count.ToString())

  'Print the table and column names
  Dim i, j As Integer

  For i = 0 To (myDS.Tables.Count - 1)
    'Print the table names
    Console.WriteLine("TableName='" & myDS.Tables(i).TableName & "'.")
    Console.WriteLine("Columns count=" & myDS.Tables(i).Columns.Count.ToString())

    For j = 0 To (myDS.Tables(i).Columns.Count - 1)
      'Print the column names and data types
      Console.WriteLine( vbTab & _
        " ColumnName='" & myDS.Tables(i).Columns(j).ColumnName & _
        " DataType='"   & myDS.Tables(i).Columns(j).DataType.ToString() )
    Next
    Console.WriteLine()
  Next
End Sub
```

Demonstration: Examining DataSet Structure



In this Demonstration

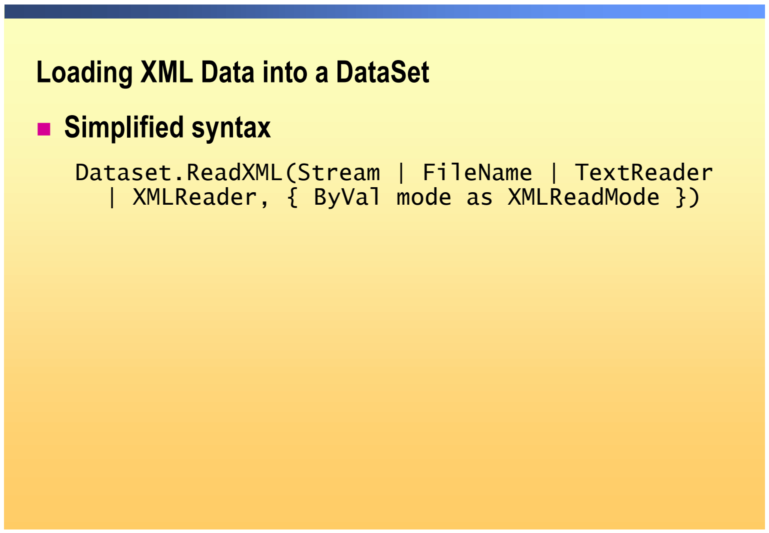
- You will load an XSD schema into a DataSet
- Use the DataSet properties and methods to retrieve information about tables in the DataSet

Introduction

In this demonstration, you will load an XSD schema into a **DataSet**. You will then use **DataSet** properties and methods to retrieve information about the tables in the **DataSet**. (code Walk thru).

1. Open **PersonPet.xsd** and explain how the XSD tags are mapped to relational objects in the **DataSet**.
2. Open the **ExistingSchema** solution file with Visual Studio.
3. Double-click the **Get Schema** button and show the code.
4. Point out that the **ParseSchema** procedure loads the **DataSet myDS** with schema information from **PersonPet.xsd**.
5. Explain the code in the **Button1_click** procedure. Point out that **Tables**, **Columns**, **ColumnName**, and **Count** properties are used to return information about the number of tables and the columns in each table.
6. Point out that a set of nested loops is used to reference each column in each table.
7. Run the application and observe the results.
8. Quit Visual Studio.

Loading XML Data into a DataSet



Loading XML Data into a DataSet

- Simplified syntax

```
Dataset.ReadXML(Stream | FileName | TextReader
  | XMLReader, { ByVal mode as XMLReadMode })
```

Visual Basic Example Visual Basic Example Visual Basic Example

Loading data and schema by using the Dataset.ReadXML method

You can use the **ReadXml** method of the **DataSet** object to load data from an XML file into a **DataSet**. When you use this method, you can load data from XML files that contain only XML data, or from files that contain XML data as well as an inline schema.

An *inline schema* is an XSD schema that appears at the beginning of the XML data file. This schema describes the XML information that appears after the schema in the XML file.

Simplified syntax

The **ReadXml** method is overloaded and can be used to read from a stream object, an XML file, a **TextReader** subclass object, or an **XmlReader** subclass object.

```
Dataset.ReadXml(Stream | FileName | TextReader | XmlReader, {
ByVal mode as XmlReadMode })
```

Use the *XmlReadMode* parameter to specify what the XML file contains and what information should be loaded from the file. This parameter is optional. If no **XmlReadMode** value is supplied, the default value **Auto** is used.

ReadMode parameter values

The following table shows the values for the ***XmlReadMode*** parameter of the **ReadXml** method of the **DataSet** object.

XmlReadMode value	Description
ReadSchema	Reads any inline schema and then loads the schema and data. • If the **DataSet** already contains a schema, any new tables that are defined by an inline schema are added to the **DataSet**. • If the inline schema defines a table that is already in the **DataSet**, an exception is thrown. • If the **DataSet** does not contain a schema, and there is no inline schema, no data is read.

XmlReadMode value	Description
IgnoreSchema	Ignores any inline schema and loads data into the existing **DataSet**. Any data that does not match the existing schema is discarded.
InferSchema	Ignores any inline schema and infers a new schema based on the structure of the XML data. If the **DataSet** already defines a schema, tables are added to this schema. The data is then loaded into the **DataSet**.
DiffGram	Reads a **DiffGram** and adds the data to the current schema in the **DataSet**.
Fragment	Reads XML fragments and appends data to appropriate **DataSet** tables. This setting is typically used to read XML data generated directly from Microsoft SQL Server™.
Auto	Examines the XML file and chooses the most appropriate option. • If the **DataSet** contains a schema or the XML contains an inline schema, **ReadSchema** is used. • If the **DataSet** does not contain a schema and the XML does not contain an inline schema, **InferSchema** is used. For best performance, specify an **XmlReadMode** rather than using **Auto**.

Example of loading a schema and data into a DataSet

The following example first loads a schema into a new **DataSet** by using the **ReadXmlSchema** method, and then loads the data from an XML file by using the **ReadXml** method with the **IgnoreSchema** option of the *XmlReadMode* parameter.

```
Private Const PurchaseSchema As String = _
                                "C:\sampledata\Purchase.xsd"

Private Sub ReadXmlDataOnly()
  Try
     myDS = New DataSet()
     Console.WriteLine("Reading the Schema file")
     myDS.ReadXmlSchema(PurchaseSchema)

     Console.WriteLine("Loading the XML data file")
     myDS.ReadXml("C:\sampledata\PurchaseData.xml", _
               XmlReadMode.IgnoreSchema)

     DataGrid1.DataSource = myDS.Tables(1)

  Catch e as Exception
     Console.WriteLine("Exception: " & e.ToString())
  End Try
End Sub
```

Example of reading inline schema and XML data

The following example reads both inline schema and XML data from an XML file into a **DataSet**. In this case, **PurchaseOrder.xml** contains an inline schema as well as XML data.

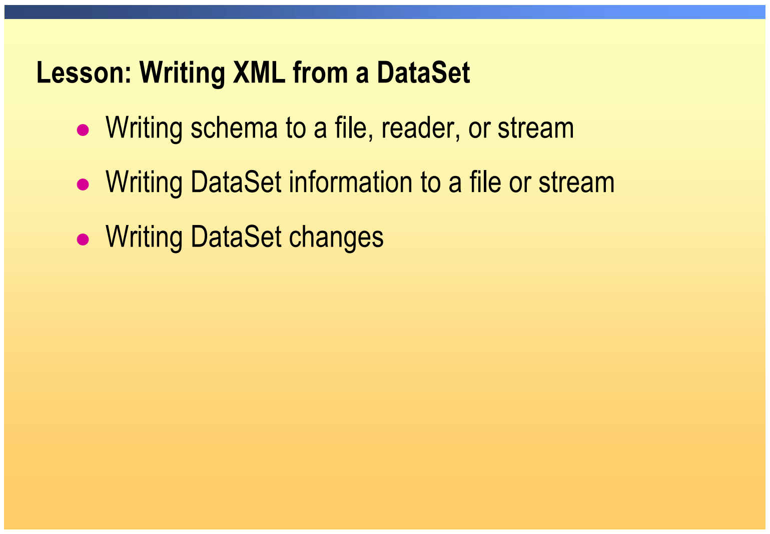
```
Private Sub ReadXmlDataAndSchema()
  Try
     myDS = New DataSet()

     myDS.ReadXml("C:\sampledata\PurchaseOrder.xml", _
                  XmlReadMode.ReadSchema)

  Catch e as Exception
     Console.WriteLine("Exception: " & e.ToString())
  End Try
End Sub
```

Example of inferring a schema

The following example shows how to infer a schema from XML data. In this example, **PurchaseOrder.xml** contains XML data but no inline schema. When the data is read into a **DataSet**, the **DataSet** infers a schema from the structure of the XML data.

```
Private Sub ReadXmlDataInferSchema()
  Try
     myDS = NEW DataSet()

     myDS.ReadXml("C:\sampledata\PurchaseOrder.xml", _
                  XmlReadMode.InferSchema)

  Catch e as Exception
     Console.WriteLine("Exception: " & e.ToString())
  End Try
End Sub
```

Lesson: Writing XML from a DataSet



Lesson: Writing XML from a DataSet

- Writing schema to a file, reader, or stream
- Writing DataSet information to a file or stream
- Writing DataSet changes

Introduction

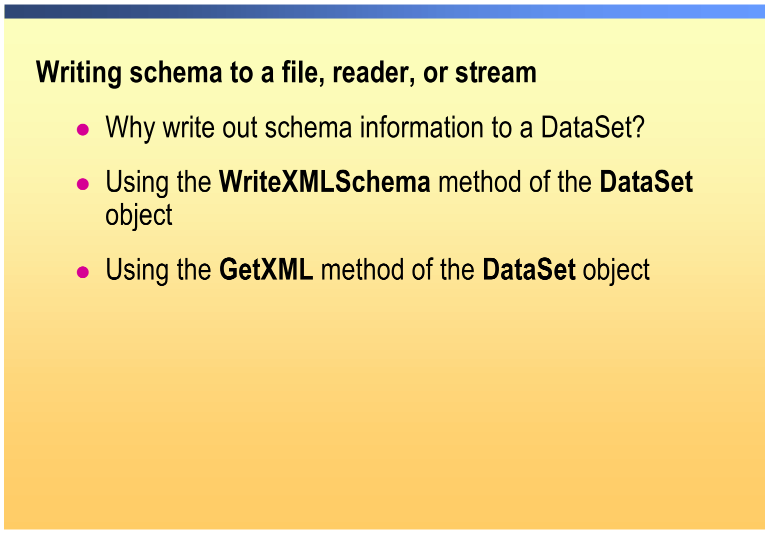
In this lesson you will learn how to write data in XML format from a **DataSet**.

Lesson objectives

After completing this lesson, you will be able to:

- Write a schema to a file, reader, or stream.
- Write information in a **DataSet** object to a file or stream.
- Write **DataSet** changes.

Writing Schema to a File, Reader, or Stream



Writing schema to a file, reader, or stream

- Why write out schema information to a DataSet?
- Using the **WriteXMLSchema** method of the **DataSet** object
- Using the **GetXML** method of the **DataSet** object

Visual Basic Example Visual Basic Example

Why write out schema information from a DataSet?

An application may create and populate a **DataSet** from several different sources. Therefore, the structure of the **DataSet** may be fairly complex.

If a **DataSet** is required to run an application in the future, you may want to save the structure and data that is contained in the **DataSet** to one or more files. By using this strategy, you can easily recreate the structure of the **DataSet** or construct a new strongly-typed **DataSet** from the XSD file and then load the data when it is needed. You can use two different methods of the **DataSet** object to generate an XSD file.:

- WriteXmlSchema
- GetXmlSchema

Using the WriteXmlSchema method of the DataSet object

You can use the **WriteXmlSchema** method of the **DataSet** object to save **DataSet** schema to an XSD file, stream, or reader object. This method takes a single parameter that specifies the destination of the schema information.

Syntax

The following example shows the syntax for the **WriteXmlSchema** method of the **DataSet** object.

```
WriteXmlSchema (ByVal filename As String | stream As Stream |
writer As TextWriter | writer As XmlWriter)
```

Example

This code sample loads a **DataSet** by using an inline schema and data from an XML file. The schema is then saved to an XSD file by using the **WriteXmlSchema** method of the **DataSet** object.

```
Private Sub SaveXSDSchema()
  Try
      myDS = New DataSet()

      'Load an inline schema and data from an XML file
      myDS.ReadXml("C:\sampledata\PurchaseOrder.xml", _
          XmlReadMode.ReadSchema)

      'Save the schema to an XSD file
      myDS.WriteXmlSchema("C:\sampledata\POSchema.xsd")

  Catch e as Exception
      Console.WriteLine("Exception: " & e.ToString())
  End Try
End Sub
```

Using the GetXmlSchema method of the DataSet object

Syntax

To extract schema information from a **DataSet** and store it as a string, use the **GetXmlSchema** method of the **DataSet** object.

```
Public Function GetXmlSchema() as String
```

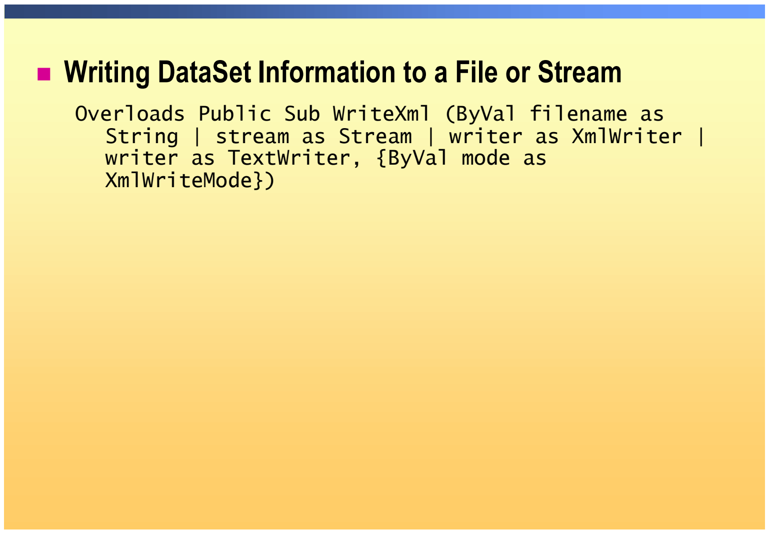
This method has no parameters.

Example

The following code fragment uses the **GetXmlSchema** method of the **DataSet** object to generate a string containing schema information.

```
Private Sub XSDSchemaToString()
  Try
      Dim StrPurchaseSchema as String
      myDS = New DataSet()

      'Load an inline schema and data from an XML file
      myDS.ReadXml("C:\sampledata\PurchaseOrder.xml", _
          XmlReadMode.ReadSchema)

      'Get the schema from the DataSet and load it
      'into a string
      StrPurchaseSchema = myDS.GetXmlSchema()

  Catch e as Exception
      Console.WriteLine("Exception: " & e.ToString())
  End Try
End Sub
```

Writing DataSet Information to a File or Stream



- **Writing DataSet Information to a File or Stream**

```
Overloads Public Sub WriteXml (ByVal filename as
  String | stream as Stream | writer as XmlWriter |
  writer as TextWriter, {ByVal mode as
  XmlWriteMode})
```

Visual Basic Example

Introduction

You can write data and schema information from a **DataSet** to a file or stream by using the **WriteXml** method of the **DataSet** object.

Partial syntax

The following example shows partial syntax for the **WriteXml** method of the **DataSet** object.

```
Overloads Public Sub WriteXml (ByVal filename As String |
stream As Stream | writer as TextWriter | writer as XmlWriter,
{ByVal mode As XmlWriteMode})
```

XmlWriteMode values

When you use the **WriteXml** method, you can specify an optional value for the *XmlWriteMode* parameter. This parameter specifies whether to generate a file that contains only XML data, XML data with an inline XSD schema, or a **DiffGram**.

The following table describes the different values for the *XmlWriteMode* parameter of the **WriteXml** method of the **DataSet** object.

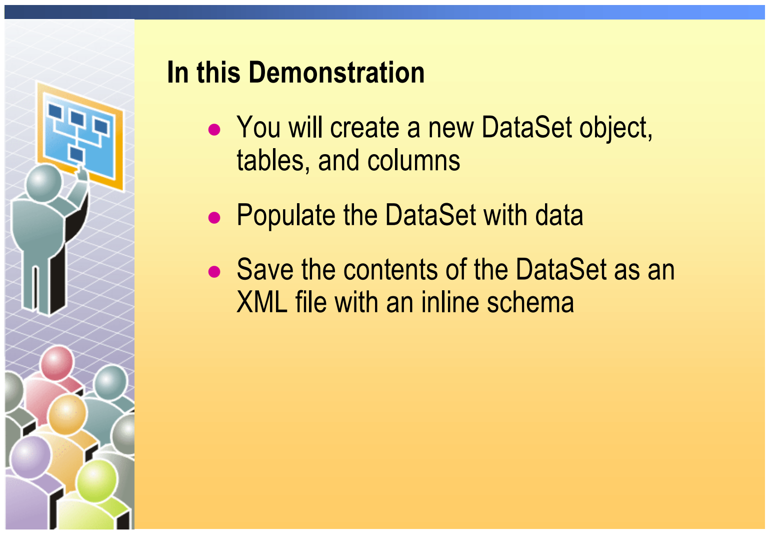
XmlWriteMode value	What is generated
IgnoreSchema	An XML file containing the data from a **DataSet**. No schema information is included. If the **DataSet** is empty, no file is created.
WriteSchema	An XML file containing an inline schema and the data from a populated **DataSet**. If the **DataSet** contains only schema information, an inline schema is written to the output file. If the **DataSet** does not include schema information, no file is created.
DiffGram	An XML file in the form of a **DiffGram**, containing both the original and current values for the data.

Example of writing XML data to a file

The following example saves the data stored in a **DataSet** as an XML file, but does not write any schema information.

```
Private Sub SaveXMLDataOnly()
  Try
      Dim StrPurchaseSchema as String
      myDS = New DataSet()

      'Load an inline schema and data from an XML file
      myDS.ReadXml("C:\sampledata\PurchaseOrder.xml", _
          XmlReadMode.ReadSchema)

      'Save the data portion of the DataSet to a file
      myDS.WriteXml("C:\sampledata\CurrentOrders.xml", _
          XmlWriteMode.IgnoreSchema)

  Catch e as Exception
      Console.WriteLine("Exception: " & e.ToString())
  End Try
End Sub
```

Demonstration



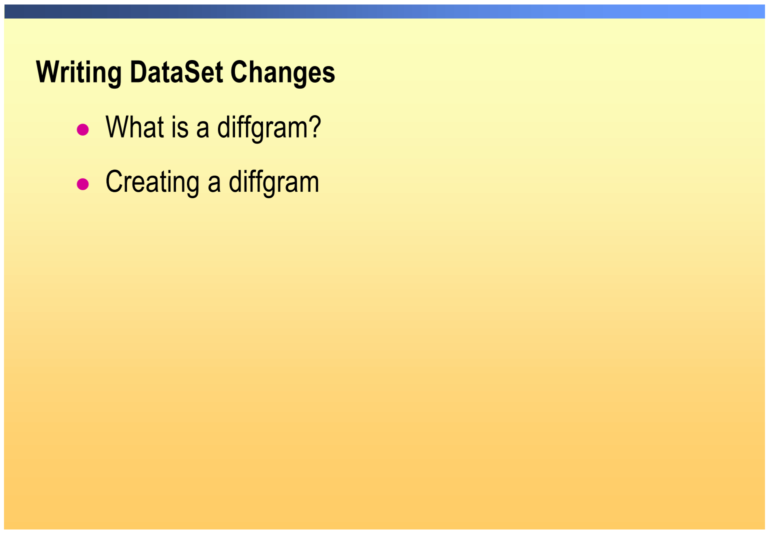
In this Demonstration

- You will create a new DataSet object, tables, and columns
- Populate the DataSet with data
- Save the contents of the DataSet as an XML file with an inline schema

This demonstration:

- Creates a new **DataSet** object, creates tables and columns
- Populates the **DataSet** with data
- Saves the entire contents of the **DataSet** as an XML file with an inline schema

Writing DataSet Changes



Writing DataSet Changes

- What is a diffgram?
- Creating a diffgram

Example of DiffGram Visual Basic Example

Why save DataSet changes?

A **DataSet** is a local information cache. During the lifetime of an application, the data rows in the **DataSet** are often modified or deleted, and new rows may be added.

Although you can send changes in a **DataSet** back to the database by using an **OleDbConnection** or **SqlConnection** object and an **OleDbDataAdapter** or **SqlDataAdapter** object, not every application uses a database to store and retrieve information. You may want to store the changes in an XML file.

Scenario

A salesperson has an application that loads a product stock list from an XML file that he receives in e-mail each morning. The application generates a **DataSet** from this file. During the day, the sales application is not connected to the Internet. At the end of the day, the salesman uses the application to generate an XML document that reflects the changes that he made to the original **DataSet**, and then sends this file in e-mail to the regional sales manager.

What is a DiffGram

A *diffgram* is an XML format, such as a file or stream, that represents changes made to a **DataSet**. It contains the original and current data for an element or attribute, and a unique identifier that associates the original and current versions of an element or attribute to each other.

A **DiffGram** is useful when you want to send data across a network and preserve the various versions of the data (Original or Current), as well as the Row State values (Added, Modified, Deleted, Unchanged) of the DataRows in a **DataSet**.

Example of a DiffGram

In the **CustomerDataSet**, the row with a CustomerID of ALFKI was modified. In the resulting **DiffGram**, the new version of the row appears at the top of the document and the original version appears inside the <diffgr:before> tag near the end of the document.

```
<diffgr:diffgram
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
  xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">

  <CustomerDataSet>
    <Customers diffgr:id="Customers1"msdata:rowOrder="0"
               diffgr:hasChanges="modified">
      <CustomerID>ALFKI</CustomerID>
      <CompanyName>New Company</CompanyName>
    </Customers>
    <Customers diffgr:id="Customers2" msdata:rowOrder="1">
      <CustomerID>ANATR</CustomerID>
      <CompanyName>Ana Trujillo Emparedados y
helados</CompanyName>
    </Customers>
  </CustomerDataSet>

  <diffgr:before>
    <Customers diffgr:id="Customers1" msdata:rowOrder="0">
      <CustomerID>ALFKI</CustomerID>
      <CompanyName>Alfreds Futterkiste</CompanyName>
    </Customers>
  </diffgr:before>
</diffgr:diffgram>
```

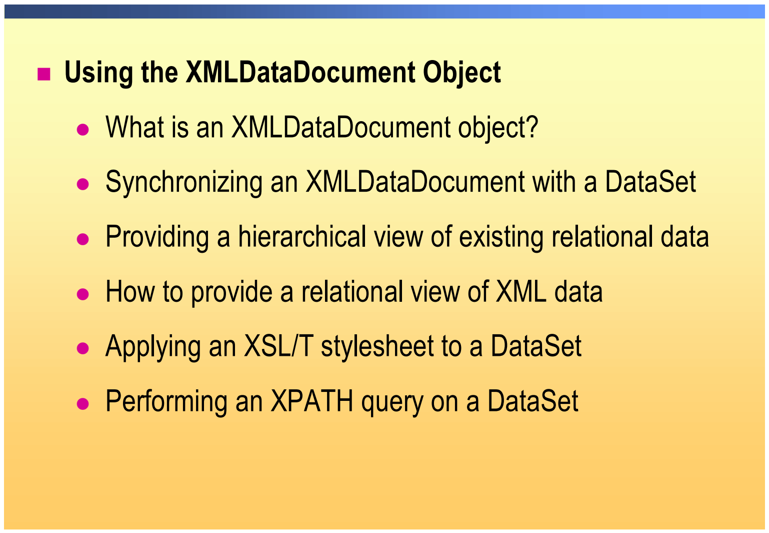
Creating a DiffGram

To generate a **DiffGram** from a **DataSet**, set the **WriteXm**l method of the **DataSet** object with the *XmlWriteMode* parameter set to **DiffGram**.

Example

```
Private Sub SaveDataSetChanges()
  Try
     Dim StrPurchaseSchema as String
     myDS = New DataSet()

     'Load an inline schema and data from an XML file
     myDS.ReadXml("C:\sampledata\Customers.xml", _
        XmlReadMode.ReadSchema)

     'Make a change to information in the DataSet
     'Delete a row
     myDS.Tables(1).Rows(1).Remove

     'Save the data portion of the DataSet as a Diffgram
     myDS.WriteXml("C:\sampledata\CustomerChanges.xml", _
        XmlWriteMode.DiffGram)

  Catch e as Exception
     Console.WriteLine("Exception: " & e.ToString())
  End Try
End Sub
```

Lesson: Using the XmlDataDocument Object



- **Using the XMLDataDocument Object**
  - What is an XMLDataDocument object?
  - Synchronizing an XMLDataDocument with a DataSet
  - Providing a hierarchical view of existing relational data
  - How to provide a relational view of XML data
  - Applying an XSL/T stylesheet to a DataSet
  - Performing an XPATH query on a DataSet

Introduction

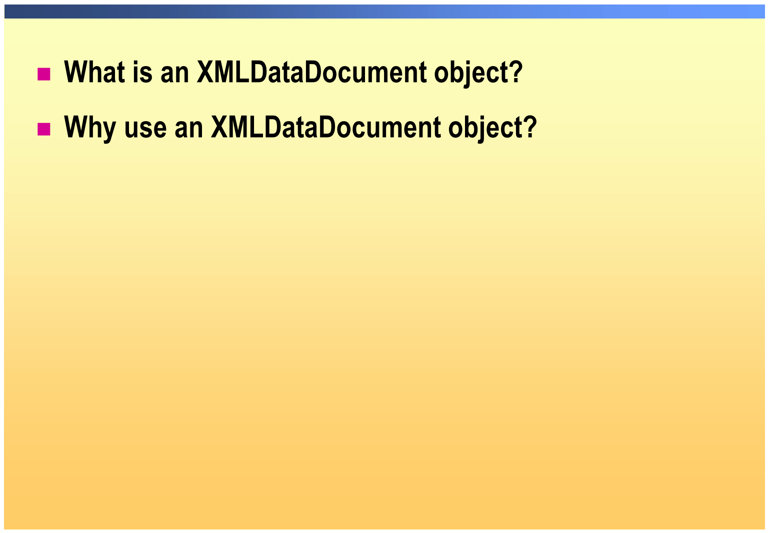
The **XmlDataDocument** object allows you to load relational data or XML data, and process the data by using World Wide Web Consortium (W3C) Document Object Model (DOM) techniques.

Lesson objectives

After completing this lesson, you will be able to:

- Explain what an **XmlDataDocument** object is.
- Synchronize an **XmlDataDocument** with a **DataSet**.
- Diagram a hierarchical view of existing relational data.
- Provide a relational view of XML data.
- Apply an XSL/T style sheet to a **DataSet**.
- Perform an XPath query on a **DataSet**.

What Is an XML DataDocumentObject?



- What is an XMLDataDocument object?
- Why use an XMLDataDocument object?

Introduction

The ADO.NET **DataSet** object provides you with a relational representation of data. For hierarchical data access, you can use the XML classes that are available in the .NET Framework.

Historically, these two representations of data have been used separately. However, the .NET Framework enables access to both the relational and hierarchical representations of the same data by using the **DataSet** object and the **XmlDataDocument** object, respectively.

Definition

The **XmlDataDocument** exposes a dataset as an XML Document Object Model (DOM) tree. This enables you to process the data as if it were an XML document, even if the dataset was populated with relational data.

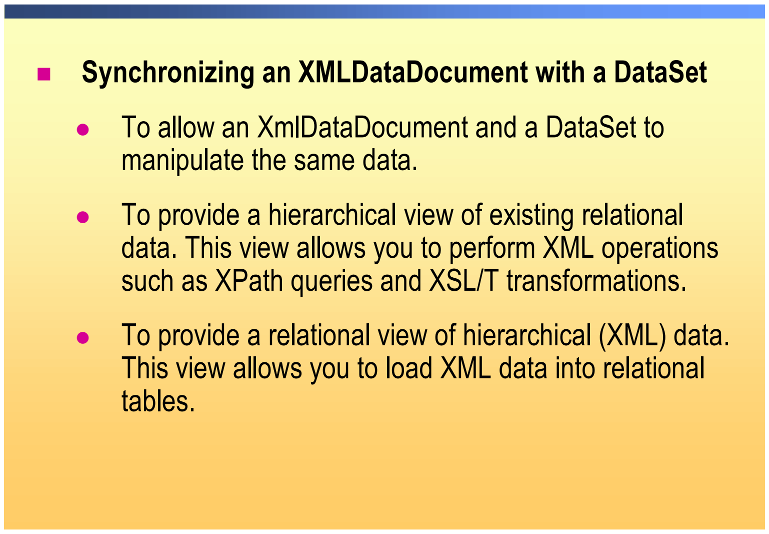
The **XmlDataDocument** object inherits many of its methods and properties from the **XmlDocument** object. As a result, you can use any of the methods of the **XmlDocument** object to manipulate the XML representation of the data in a dataset.

Why use an XMLDataDocument object?

There are three main reasons for using an **XmlDataDocument** object in conjunction with a **DataSet,** rather than using a **DataSet** alone:

- The **XmlDataDocument** enables you to load either relational data or XML data and manipulate that data by using the W3C DOM. The **XmlDataDocument** object has a **DataSet** property. By setting this property to an existing **DataSet**, you can view relational data as an XML hierarchy, and perform XML-oriented operations such as navigating XML nodes, querying nodes using XPath, and transforming the data using an XSL/T style sheet.
- By synchronizing a **DataSet** with an **XmlDataDocument**, changes made to the **DataSet** will be reflected in the **XmlDataDocument**, and vice versa.
- Loading an XML file into an **XmlDataDocument** preserves all of the schema detail that is contained in the original file. If you load or infer an XSD schema into a **DataSet** object, only the information that is required to create a relational representation of the data is captured. Any additional detail is discarded.

Synchronizing an XmlDataDocument with a DataSet



- **Synchronizing an XMLDataDocument with a DataSet**
  - To allow an XmlDataDocument and a DataSet to manipulate the same data.
  - To provide a hierarchical view of existing relational data. This view allows you to perform XML operations such as XPath queries and XSL/T transformations.
  - To provide a relational view of hierarchical (XML) data. This view allows you to load XML data into relational tables.

Why synchronize?

There are three reasons to synchronize an **XmlDataDocument** object with an existing **DataSet**:

1. To allow an **XmlDataDocument** and a **DataSet** to manipulate the same data.
2. To provide a hierarchical view of existing relational data. This view allows you to perform XML operations such as XPath queries and XSL/T transformations.
3. To provide a relational view of hierarchical (XML) data. This view allows you to load XML data into relational tables.

Associating an XMLDataDocument with a DataSet

There are two ways of creating a new **XmlDataDocument** and associating it with an existing **DataSet**:

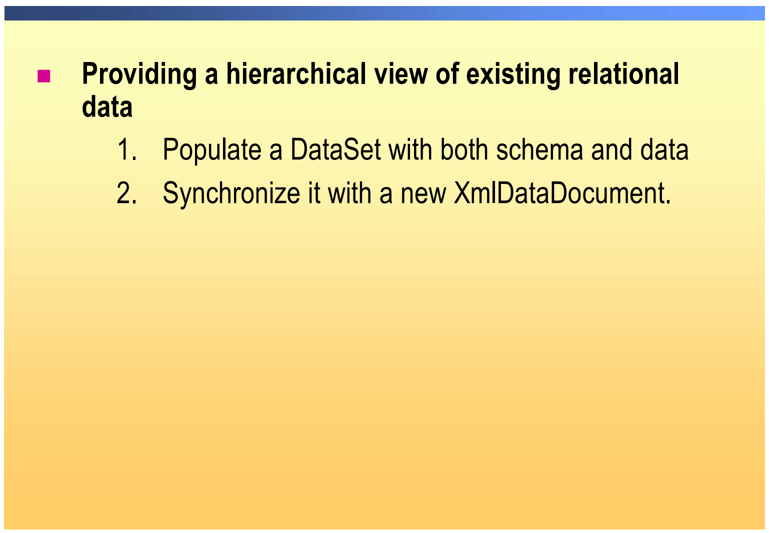
- Use an existing **DataSet** in the constructor for the **XmlDataDocument**.

Example

The following example shows how to create a new **XmlDataDocument** object.

```
Dim xmlDoc As XmlDataDocument
xmlDoc = New XmlDataDocument(myDataSet)
```

Providing a Hierarchical View of Existing Relational Data



- **Providing a hierarchical view of existing relational data**
  1. Populate a DataSet with both schema and data
  2. Synchronize it with a new XmlDataDocument.

Visual Basic Example

Introduction

If the source of your data is an existing **DataSet**, you can already work with the data tables and easily modify, create, and delete data rows and values. If you want to perform XML -related operations on the data, you must associate the **DataSet** with an **XmlDataDocument**.

How to provide a hierarchical view of existing relational data

In this case, the data source is an existing **DataSet**. If you want to use XML operations on a relational **DataSet**:

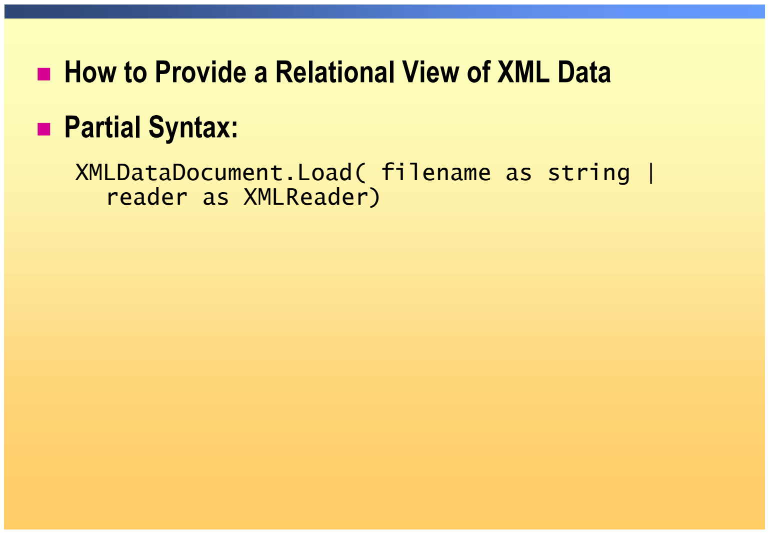
1. Populate a **DataSet** with both schema and data.
2. Synchronize it with a new **XmlDataDocument**.

Example

The following example shows how to use an **XmlDataDocument** object to fill an existing **DataSet** with both schema and data.

```
Dim xmlDoc As XmlDataDocument = New XmlDataDocument
Dim myDataSet As New DataSet()

'Fill a DataSet with schema and data
myDataSet.ReadXmlSchema("c:\PurchaseOrder.xsd")
myDataSet.ReadXML("C:\PurchaseData.xml", _
                   XmlReadMode.IgnoreSchema)

'Create a new XmlDataDocument by using an existing DataSet in
'the constructor.

Dim xmlDoc As XmlDataDocument = New
XmlDataDocument(myDataSet)
```

How to Provide a Relational View of XML Data



- How to Provide a Relational View of XML Data
- Partial Syntax:

```
XMLDataDocument.Load( filename as string |
  reader as XMLReader)
```

Visual Basic Example

Introduction

When you create a new **XmlDataDocument** object, you can load it with data from an XML file, or you can associate it with data in an existing **DataSet**. If you load data directly from an XML file, you can operate on the data by using XML techniques. You do not need to synchronize with a **DataSet**.

However, if you want to operate on the same data by using relational techniques, you must first synchronize the **XmlDataDocument** with a **DataSet** that contains schema information.

How to load data into an XmlDataDocument

When you create a new **XmlDataDocument** object, you can load it with data from an XML file by using the **Load** method.

Partial syntax

```
XmlDataDocument.Load(filename As String | reader As XmlReader)
```

Example

The following example shows how to fill an **XmlDataDocument** object from an XML document.

```
myXMLDataDocument.Load("C:\Books.xml")
```

How to provide a relational view of existing hierarchical (XML) data

In this case, the data source is an XML file. If you want to perform relational operations on the XML representation of the data:

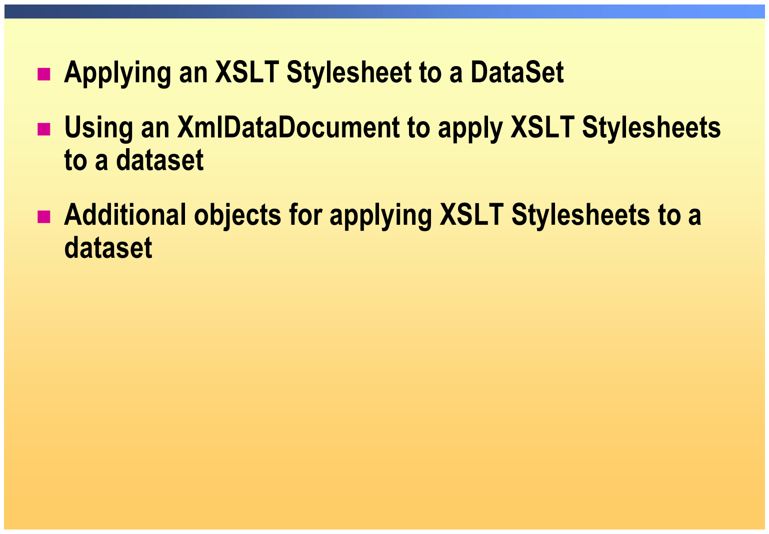
1. Populate a **DataSet** with schema only, such as a strongly-typed **DataSet**.
2. Synchronize the **DataSet** with an **XmlDataDocument**.
3. Load the **XmlDataDocument** from an XML document by using the **Load** method.

Be aware that you cannot load an **XmlDataDocument** if it is synchronized with a **DataSet** that already contains data. This will cause an exception to be thrown.

Example

```
Dim myDataSet As DataSet = New DataSet()

' Populate the DataSet with schema, but not data.
myDataSet.ReadXmlSchema("c:\PurchaseOrder.xsd")

' Create a new XmlDataDocument that uses this DataSet schema
Dim xmlDoc As XmlDataDocument = New XmlDataDocument(myDataSet)

' Load the data into the XmlDataDocument from an XML file
xmlDoc.Load("C:\PurchaseData.xml")
```

Applying an XSLT Style Sheet to a DataSet



- Applying an XSLT Stylesheet to a DataSet
- Using an XmlDataDocument to apply XSLT Stylesheets to a dataset
- Additional objects for applying XSLT Stylesheets to a dataset

Visual Basic Example XSLT Example

Introduction

When you apply an Extensible Stylesheet Language Transformations (XSLT) style sheet to transform data, the source data is usually in XML format. To use an XSLT style sheet with a dataset, you can follow these two steps:

1. Generate XML data from the dataset,
2. Apply the XSLT style sheet to the XML data.

As a more efficient alternative, you can apply an XSLT style sheet directly to the data in a **DataSet** without having to first create an XML file. This is useful when the source of the data is a relational database, or other non-XML source.

Using an XmlDataDocument to apply XSLT style sheets to a DataSet

By associating an existing **DataSet** with an **XmlDataDocument**, you can transform the data without creating an intermediate XML file. The **XmlDataDocument** acts as the input to the transformation.

Additional objects for applying XSLT style sheets to a DataSet

To help you perform XSLT transformations, you can use the following objects and methods.

Object or method	Purpose
XmlDataDocument	The **XmlDataDocument** associated with an existing **DataSet** and acts as the input to the transform.
XslTransform.Load(stylesheet)	Load the style sheet or transform to be applied to the source data.
XslTransform.Transform(output file or writer, parameter list, encoding format)	This method applies the style sheet and saves the result to a file, stream, or writer object.
XmlTextWriter	This object writes XML text to a file or stream. It contains methods to add nodes and format the output.

How to apply an XSLT transform (Procedure)

To apply an XSLT style sheet to a **DataSet**, use the following approach:

1. Create and populate a **DataSet** with both schema and data. The source could be an XML file, a relational database, or a combination.
2. Create a new **XmlDataDocument** object and associate (synchronize) it with the **DataSet**.
3. Create an **XslTransform** object and load a style sheet into the object.
4. Use the **Transform** method of the **XslTransform** object to apply the style sheet and save the results to a file, stream, or writer object. In this step, the **XmlDataDocument** is used as the input for the transform.

Example

The following example populates a **DataSet** with tables and relationships, synchronizes the **DataSet** with an **XmlDataDocument**, and writes a portion of the **DataSet** as an HTML file by using an XSLT style sheet.

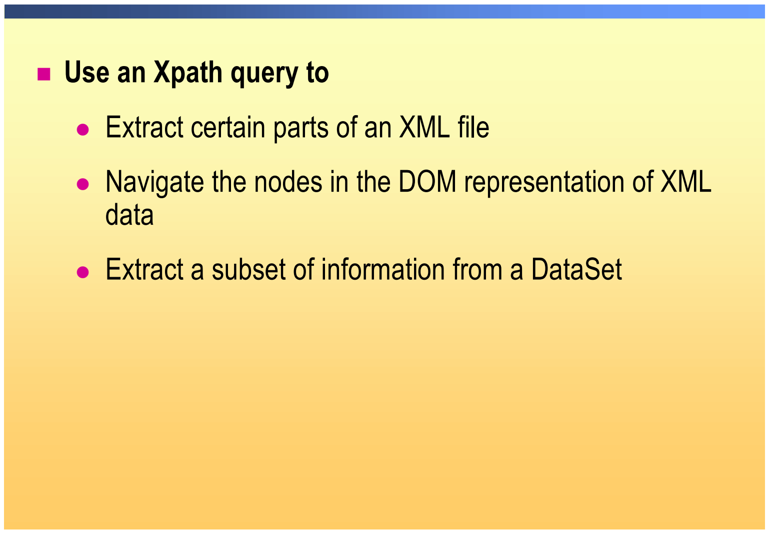
```
Public Shared Sub Main()

   'Create and populate a DataSet from SQL Server

    Dim nwindConn As SqlConnection = New SqlConnection _
       ("Data Source=localhost;Initial Catalog=northwind;Integrated Security=SSPI")
   nwindConn.Open()
   Dim myDataSet As DataSet = New DataSet("CustomerOrders")

   Dim custDA As SqlDataAdapter = New SqlDataAdapter _
       ("SELECT * FROM Customers", nwindConn)
   custDA.Fill(myDataSet, "Customers")

   Dim ordersDA As SqlDataAdapter = New SqlDataAdapter _
       ("SELECT * FROM Orders", nwindConn)
   ordersDA.Fill(myDataSet, "Orders")

   nwindConn.Close()

   myDataSet.Relations.Add("CustOrders", _
                  myDataSet.Tables("Customers").Columns("CustomerID"), _
                  myDataSet.Tables("Orders").Columns("CustomerID")).Nested = true

   'Create a new XmlDataDocument and synchronize it with the DataSet
   Dim xmlDoc As XmlDataDocument = New XmlDataDocument(myDataSet)

   'Create an XslTranform object and load a stylesheet
   Dim xslTran As XslTransform = New XslTransform()
   xslTran.Load("mytransform.xsl")

   'Apply the stylesheet using the XmlDataDocument as input
   'Capture the results in an XmlTextWriter object
   Dim writer As XmlTextWriter = New XmlTextWriter("xslt_output.html", _
                                                  System.Text.Encoding.UTF8)
   xslTran.Transform(xmlDoc, Nothing, writer)
   writer.Close()
End Sub
```

Style sheet used in the previous example

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">

  <xsl:template match="CustomerOrders">
    <html>
      <style>
        body {font-family:verdana;font-size:9pt}
        td   {font-size:8pt}
      </style>
      <body>
        <table border="1">
          <xsl:apply-templates select="Customers"/>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="Customers">
    <tr>
      <td>
        <xsl:value-of select="ContactName"/>,
        <xsl:value-of select="Phone"/><BR/>
      </td>
    </tr>
    <xsl:apply-templates select="Orders"/>
  </xsl:template>

  <xsl:template match="Orders">
    <table border="1">
      <tr>
        <td valign="top">
          <b>Order:</b>
        </td>
        <td valign="top">
          <xsl:value-of select="OrderID"/>
        </td>
      </tr>
      <tr>
        <td valign="top">
          <b>Date:</b>
        </td>
        <td valign="top">
          <xsl:value-of select="OrderDate"/>
        </td>
      </tr>
```
(Code continued on following page.)

```
        <tr>
          <td valign="top">
            <b>Ship To:</b>
          </td>
          <td valign="top">
            <xsl:value-of select="ShipName"/><br/>
            <xsl:value-of select="ShipAddress"/><br/>
            <xsl:value-of select="ShipCity"/>,
            <xsl:value-of select="ShipRegion"/>
            <xsl:value-of select="ShipPostalCode"/><br/>
            <xsl:value-of select="ShipCountry"/>
          </td>
        </tr>
      </table>
    </xsl:template>

  </xsl:stylesheet>
```

Performing an XPath Query on a DataSet



- Use an Xpath query to
  - Extract certain parts of an XML file
  - Navigate the nodes in the DOM representation of XML data
  - Extract a subset of information from a DataSet

Visual Basic Example

Introduction

You can use an XPath query to extract certain parts of an XML file or navigate the nodes in the DOM representation of the XML data. If the data that you want to query is stored in a **DataSet**, you could create a new XML file by using the **WriteXml** method of the **DataSet** object.

However, you can eliminate this step by synchronizing the **DataSet** with an **XmlDataDocument** and then performing the query directly on the **XmlDataDocument**.

Why perform XPath queries on a DataSet?

You may need to extract a subset of information that meets certain criteria from a **DataSet**. One way of filtering the information is by performing an XPath query by using the **SelectNodes** or **SelectSingleNode** methods of the **XmlDataDocument** class.

For example, for a **DataSet** that contains three related tables, Customers, Orders, and OrderDetails, you can use an XPath query to select all customers who ordered product number 35. This approach may be easier than processing the data relationally, that is by using a SQL statement with two join clauses or the **Select** method of the **DataTable** object.

Additional objects needed to perform XPath queries

To help you perform XPath queries, you can use the following objects and methods.

Object or method	Purpose
XmlDataDocument.SelectNodes(xPath)	Select all the nodes in the XML that match the XPath expression.
XmlDataDocument.SelectSingleNode(xPath)	Select the first nodes in the XML that match the XPath expression.
XmlNodeList	Use this collection to store the nodes selected by the XPath query.

How to select XML nodes using XPath(Procedure)

To perform an XPath query on a **DataSet**, use the following approach:

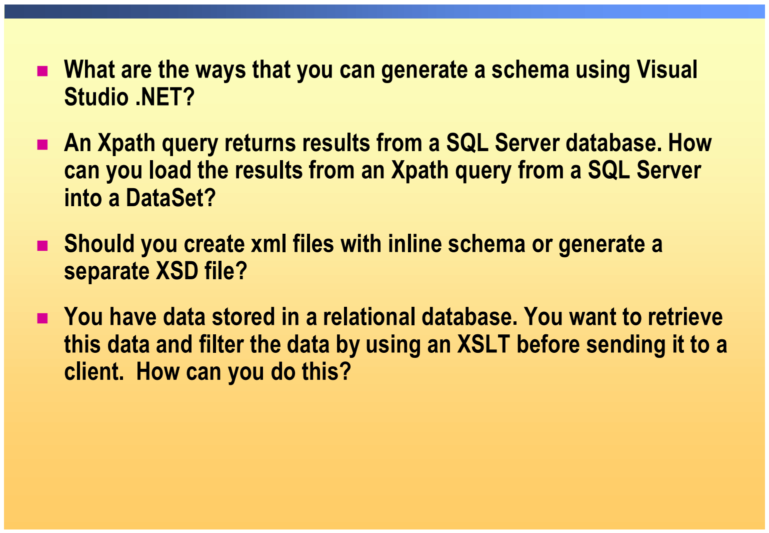
1. Create and populate a **DataSet** with both schema and data. The source could be an XML file, an XSD file, a relational database, or a combination.
2. Create a new **XmlDataDocument** object and associate (synchronize) it with the **DataSet**.
3. Use the **SelectNodes** or **SingleSelectNode** method of the **XmlDataDocument** with an XPath expression to return nodes that match the criteria described by the XPath. Store the results in an **XmlNodeList**.
4. Use a node object to process each node in the **XMLNodeList**.

Example

This example constructs a **DataSet** from an XML file that contains an inline schema. It synchronizes the **DataSet** with an **XmlDataDocument**. The code then selects all Customer nodes where there is a grandchild node containing an order for ProductID number 35.

```
Public Shared Sub Main()

   'Create a new DataSet and load it with schema and data
   Dim myDataSet as New DataSet
   myDataSet.ReadXml("C:\sampledata\Customers.xml", XmlReadmode.ReadSchema)

   'Create a new XmlDataDocument and synchronize it with the DataSet
   Dim xmlDoc As XmlDataDocument = New XmlDataDocument(myDataSet)

   'Perform an XPath query and store the results in an XmlNodeList
   Dim nodeList As XmlNodeList = xmlDoc.DocumentElement.SelectNodes( _
                        "descendant::Customers[*/OrderDetails/ProductID=35]")

                     'Process and print out information from each node
                     Dim myRow As DataRow
                     Dim myNode As XmlNode

    For Each myNode In nodeList
      myRow = xmlDoc.GetRowFromElement(CType(myNode, XmlElement))

      If Not myRow Is Nothing Then Console.WriteLine(myRow(0).ToString())
    Next
  End Sub
```

Review



- What are the ways that you can generate a schema using Visual Studio .NET?
- An Xpath query returns results from a SQL Server database. How can you load the results from an Xpath query from a SQL Server into a DataSet?
- Should you create xml files with inline schema or generate a separate XSD file?
- You have data stored in a relational database. You want to retrieve this data and filter the data by using an XSLT before sending it to a client. How can you do this?

1. You want to create an XSD schema from data in an XML document. The document does not contain an inline schema, and no external XSD document exists. What are the ways that you can generate a schema using Visual Studio .NET?

   You can create a schema by using the XML Designer. First add the XML file to the project, and then infer a schema using visual tools.

2. An XPath query returns results from a SQL Server database. How can you load the results into a **DataSet**?

   SQL server generates document fragments rather than well-formed XML documents as the result of sql-xml queries. Use the ReadXml method of the DataSet object with the XmlReadMode parameter set to fragment.

3. Should you create XML files with inline schema, or generate a separate XSD file?

   It depends. If you are shipping data to a client who does not know the structure of the data, it is a good idea to use an inline schema. The inline schema is easy to read or ignore. It is somewhat faster to load data and schema from a single file rather than from separate files. If you have a choice whether to use separate files or inline schema, use an inline schema. You might want to use separate files when you have multiple sets of information that conform to the same schema, for example, a standard auto parts form that always looks the same.

4. You have data stored in a relational database. You want to retrieve this data and filter the data by using an XSLT style sheet before sending it to a client. How can you do this?

   First, load the data from the database into a DataSet. Next, create an XmlDataDocument object and synchronize it with the DataSet. Use the XmlDataDocument as the information source for an XSLT transform that uses a style sheet to transform the data.

   If the data is stored in an XML data file to begin with, you can apply the style sheet in the browser. Because the data is stored relationally in a DataSet, you must first generate a hierarchical representation by loading the information into an XmlDataDocument object before you can transform it.

Lab 5: Working with XML Data in ADO .NET

Lab x: title

- Exercise_#:_Title
- Exercise_#:_Title
- Exercise_#:_Title
- Exercise_#:_Title
- Exercise_#:_Title

Objectives

After completing this lab, you will be able to:

- Create XSD schemas and XML data files by using Visual Studio .NET.
- Load schema and data into a **DataSet**.
- Write XSD schema and XML data to a file.
- Process relational data by using XML techniques.

Prerequisites

Before working on this lab, you must have:

- Basic knowledge about the structure of an XML document.
- Basic knowledge about the purpose of an XSD schema.

For more information

Estimated time to complete this lab: 90 minutes

Exercise 1
Generating a DataSet Schema by Using Visual Studio XML Designer

In this exercise, you will infer a **DataSet** schema from an XML data file. You will use the tools in the Visual Studio XML Designer to import the data and infer a schema for the data. After you generate a schema, you will use it to display data from the file in a data grid control.

► To infer a schema from an XML file by using the XML Designer

1. Start Visual Studio, and then in Microsoft Windows® create a new project named **CreateSchema**.
2. Drag *install folder*/**labs/lab05/customers.xml** into the new project. Alternatively, you can click **File**, and then click **Add Existing Item…** to add **customers.xml** to the project.
3. Examine the structure of the file. It contains customer, order, and order detail data, but no explicit definition of a schema. Notice that hierarchical data can be mapped to three tables with parent-child relationships in a relational model.
4. Switch to the Data View of the XML Designer.
5. Notice that the XML Designer has identified the three tables. Use the Data Grid to 'drill through' the parent-child relationship of the three tables.
6. Right-click the document, and then click **Create Schema** to generate an XSD file based on the current data.
7. In the **Solution Explorer**, double-click **customers.xsd**.
8. Switch to the XML view and examine the schema generated by Visual Studio.

► To create a new DataSet by using the XML Designer

1. Switch to the DataSet view of the **Customers.xsd** schema.
2. Right-click the design surface, and then click **Generate DataSet**.
3. Click the **View All Files** button in the Solution Explorer. Confirm that a class was generated for the **DataSet**.

► **Display DataSet data in a data grid control**

4. In Project Explorer, double-click **Form1**.
5. Drag and drop a new DataGrid to the form. Set the **Dock** property of the Grid to **Fill** to dock the DataGrid to the entire form.
6. Drag and drop a new **DataSet** from the toolbox to the form. Base this **DataSet** on the **CreateSchema.CustomerData** DataSet.
7. Right-click the DataGrid on the form. Click **properties**, and then set the **data source** property to **CustomerData1.Customers**. Note that the **Customers** table is the parent of the **Orders** and **OrderDetails** tables.
8. Examine the DataGrid on the form. Note that Visual Studio uses the schema information to populate the column headers automatically.
9. View the code for the form. Expand the code generated by the Windows form designer. Find the following code:

```
“InitializeComponent()”
```

10. Add the following code after the call to InitializeComponent().

```
Me.CustomerDataSet1.ReadXml(“\Program
Files\MSDNTrain\2389\Labs\Lab05\customers.xml”)
```

11. Save, build, and then run the application.

Exercise 2
Creating and Loading a DataSet from XML

In this exercise, you will load schema information into a **DataSet** and then load the data from an XML file.

Scenario

A salesman uses a Windows application to collect orders for products that are sold by Northwind Traders. The salesman does not have a reliable Internet connection. Instead, the salesman loads customer and order information from an XML file. This XML file contains only sales data; a separate file contains schema information. You will build a part of a Windows application that allows you to load and view the existing customer and order information.

► **To the Customer Information form**

1. Create a new Visual Basic or Microsoft Visual C#™ Windows application named **LoadingDataSets**.
2. Set the following properties of the **Form1**.

Property	Value
Text	Sales Information

3. Add a DataGrid control with the following characteristics:

Property	Value
Anchor	Top, Left
Dock	Left

4. Add a button to the form with the following characteristics:

Property	Value
Name	btnLoadData
Text	Display Customer Information

5. Add a button to the form with the following characteristics:

Property	Value
Name	btnClose
Text	Close

6. Add code to the **btnClose_Click** event to close the application.

► To declare namespaces, variables, and constants

- Add the following declarations to the form.

Variable or constant name	Type	Value
myDocument	String Constant	\Program Files\MSDNTrain\2389 \Labs\Lab05\customers.xml
myLoadSchema	String Constant	\Program Files\MSDNTrain\2389\Labs\Lab05\customers.xsd
myDS	DataSet Variable	none

► To create and load a DataSet

1. Create a Try/Catch block for the procedure.
2. Create a new instance of a **DataSet** named **myDS**.
3. Write code to load a schema file into the **DataSet**. Use the information in the following table. You may want to code this step as a separate subroutine. Make sure to include code to handle errors. (Hint: use the **ReadXmlSchema** method.)

Parameter	Value
File variable	myLoadSchema

4. Write code to load the XML data into the **DataSet**. Use the information in the following table. (Hint: Use the **ReadXml** method)

Parameter	Value
File variable	MyDocument
XmlReadMode	IgnoreSchema

5. Bind the data grid to the **Customers** table in the **DataSet**.
6. Save, build, and run the application.

► To infer a schema for the Customer DataSet

1. Modify your code to infer a schema for the XML data rather than loading a separate schema file.
2. Save, build, and run the application.

Exercise 3
Saving DataSet Schema and Data as XML

In this exercise, you will add code to the Windows application created in the previous exercise to save the schema and data contained in the **DataSet**. You will save the schema to a separate XSD file and you will save an inline schema with an XML data file.

Scenario

At the end of each business day, the salesman for Northwind traders saves sales information to an XML file and then sends the data in an e-mail message to the company's central office for fulfillment. To accommodate this activity, you will extend the Windows application to generate XML data files and XSD schema files.

► To open the starter code

- For this exercise, you can use the code that you wrote for the previous exercise, or you can use the project *install_path***Labs\Lab05\Solution\Ex2***xx*\ where *xx* is VB or CS. Open the project file in Visual Studio .NET.

► To save schema information

1. Add a button to the form with the following characteristics.

Property	Value
Name	btnSaveSchema
Text	Save Schema

2. Add code to the **Click** event of this button that saves the **DataSet** schema. Use the information in the following table.

File name	\Program Files\MSDNTrain\2389\Labs\Lab05\ResultSchema.xsd

3. Save, build, and then run the application.
4. Examine the resulting XSD file to confirm that it reflects the structure of the Customers DataSet.

► **To save data as XML**

1. Add a button to the form with the following characteristics.

Property	Value
Name	btnSaveData
Text	Save Data

2. Add code to the **Click** event of this button that saves only the **DataSet** data. Use the information in the following table.

File name	\Program Files\MSDNTrain\2389\Labs\Lab05\ResultData.xml

3. Save, build and run the application.
4. Examine the resulting XML file to confirm it reflects the data in the Customers DataSet.

► **To save data and schema as XML**

1. Modify your code to save both the **DataSet** data and an inline schema. Use the information in the following table.

File name	\Program Files\MSDNTrain\2389\Labs\Lab05\ ResultInlineSchema.xml

2. Save, build, and then run the application.
3. Examine the resulting XML file to confirm that it reflects the data and schema information.

If Time Permits
Processing a DataSet by Using XML Techniques

In this exercise, you will query a **DataSet** by using an XPath query.

Scenario

A salesman for Northwind Traders often needs to know which customers have ordered a certain product. You will examine and modify an application that queries information loaded from an XML file into a **DataSet**. You will use the **XmlDataDomcument** object to perform the XPath query.

► **To examine the XML data file**

- Open **CustData.xml** and examine the file.

► **To examine the Common Query application**

1. Open the **Lab05_Ex4 V**isual Basic .NET project. Note that for this exercise, there is no C# solution.
2. Examine Form1. Note that it contains a text box in which to enter search criteria, a list box to display search results and buttons to execute a query and close the application.
3. Examine the code for the **btnQueryByProduct_Click** event. Note that the code
   - Creates and loads a **DataSet** from an XML file.
   - Creates an **XmlDataDocument** and synchronizes it with the **DataSet**.
   - Builds an XPath query string from information in the user interface.
   - Executes the query and captures the results as a collection of nodes.
   - Loops through the node list and populates a list box based on the results.
4. Build and run the application.
5. Enter **42** as the **ProductID** to search for, and then examine the results.

► To execute different queries

1. Find the following line of code:

```
QueryString =
"descendant::Customers[*/OrderDetails/ProductID=" &
CurrentProduct.ToString() & "]"
```

2. Use the information in the following table to modify the query:

Query	Description
`"//Customers[*/OrderDetails/ProductID=" & CurrentProduct.ToString() & "]"`	The nodes of customers who ordered a certain productid This is an alternate syntax for the original query
“//Customers”	All customers

3. Try additional XPath queries and examine the results.

► To change the display information

1. Find the following line of code:

```
ListBox1.Items.Add(myRow(1).ToString())
```

2. Change the index of the **myRow()** collection to **2**.
3. Run the application. What information is displayed in the list box? (ANS: The Contact Name is displayed) What can you conclude from this? (ANS: The Row object maps sub-elements at the same level to columns in a single row)
4. Quit Visual Studio .NET

msdn® training

Module 6: Building DataSets from Existing Data Sources (Prerelease)

Contents

Microsoft®

Instructor Notes

Presentation:
60 Minutes

Lab:
60 Minutes

This module teaches students . . .

After completing this module, students will be able to:

- Configure a DataAdapter to retrieve information.
- Populate a DataSet by using a DataAdapter.
- Configure a DataAdapter to modify information.
- Persist data changes to a server.
- Manage data conflicts.

Required materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2389A_06.ppt
- Module 6, "Building DataSets From Existing Data Sources"
- Lab 6.1, Retrieving Data into a Disconnected Application
- Lab 6.2, Retrieving and Updating Customers and Orders Data

Preparation tasks

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices and labs.
- Read the latest .NET Development news at http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000519

How to Teach This Module

This section contains information that will help you to teach this module.

Lesson: Configuring a DataAdapter to Retrieve Information

This section describes the instructional methods for teaching each topic in this lesson.

What is a DataAdapter?

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What are some examples of when you would not want to use a DataAdapter? Why?

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

DataAdapter Properties and Methods

Discussion Questions: Personalize the following questions to the background of the students in your class.

- How do the DataSet methods GetChanges and Merge

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Practice Solution:

A disconnected application that provides read-only access to a table in the database. For example, an application that allows the user to read the Employees table in the Northwind database.

A disconnected application that provides full read-write access to a table. For example, an application that allows sales people to query customer records, update customer records, add new customers, and delete existing customers.

How to Create a DataAdapter That Uses a New SELECT Statement

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What are some other business scenarios in which this functionality could be used?

How to Create a DataAdapter That Uses an Existing Stored Procedure

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What are some other business scenarios in which this functionality could be used?

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

After the practice

Questions for discussion after the practice:

- What problems did you encounter while completing this practice?

Lesson: Populating a DataSet by Using a DataAdapter

This section describes the instructional methods for teaching each topic in this lesson.

How to Fill a DataSet Table by Using a DataAdapter

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What are the benefits of using a DataAdapter to fill a DataTable? Under what circumstances would you want to do this?
- Why does disabling constraint checking while using the DataAdapter improve performance?
- Are there any circumstances when you would not want to trade off constraint checking or index maintenance for improved performance?

How to Infer Additional Constraints for a DataSet

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What are the differences between the MissingSchemaAction property and the FillSchema method of the DataAdapter?
- When would you want to set the MissingSchemaAction property and when would you want to call the FillSchema method?

How to Fill a DataSet Efficiently

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What are the advantages or disadvantages for creating a strongly-typed DataSet versus creating the DataColumns, DataTables, and DataRelations programmatically?

Lesson: Configuring a DataAdapter to Update the Underlying Data Source

This section describes the instructional methods for teaching each topic in this lesson.

How Does the DataSet Track Changes

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

What Are the Data Modification Commands?

Discussion Questions: Personalize the following questions to the background of the students in your class.

- Why would you use the DataAdapter to modify data rather than using the data modification commands directly?

How to Set the Data Modification Commands Using Existing Stored Procedures With Parameters and the Wizard

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Lesson: Persisting Changes to a Data Source

This section describes the instructional methods for teaching each topic in this lesson.

When to Use the GetChanges Method of a DataSet Object

Instructor Demo: Instructor runs code that has two DataGrids on a form; one contains the original DataSet into which the instructor makes changes. The second shows the DataSet that contains the changes.

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

When to Use the Select Method

Discussion Questions: Personalize the following questions to the background of the students in your class.

- Why is pointing to the rows within the original DataSet efficient?

How to Update a Data Source by Using a DataSet

Discussion Questions: Personalize the following questions to the background of the students in your class.

- How is this method of updating a data source different than using the DataAdapter and data modification commands?

How to Accept Changes Into the DataSet

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook. To save time, you can guide students through this practice.

Lesson: How to Handle Conflicts

This section describes the instructional methods for teaching each topic in this lesson.

What Conflicts Can Occur?

Discussion Questions: Personalize the following questions to the background of the students in your class.

- Why does optimistic concurrency cause the potential for data update conflicts?

How to Resolve Conflicts

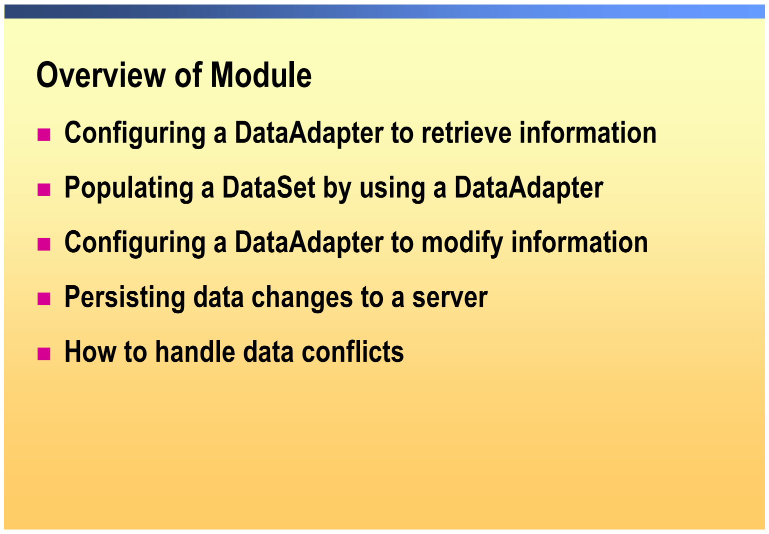
Discussion Questions: Personalize the following questions to the background of the students in your class.

- List examples of when you would not want to use optimistic concurrency.

Transition to Practice Exercise:

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

Overview



Overview of Module

- Configuring a DataAdapter to retrieve information
- Populating a DataSet by using a DataAdapter
- Configuring a DataAdapter to modify information
- Persisting data changes to a server
- How to handle data conflicts

Introduction

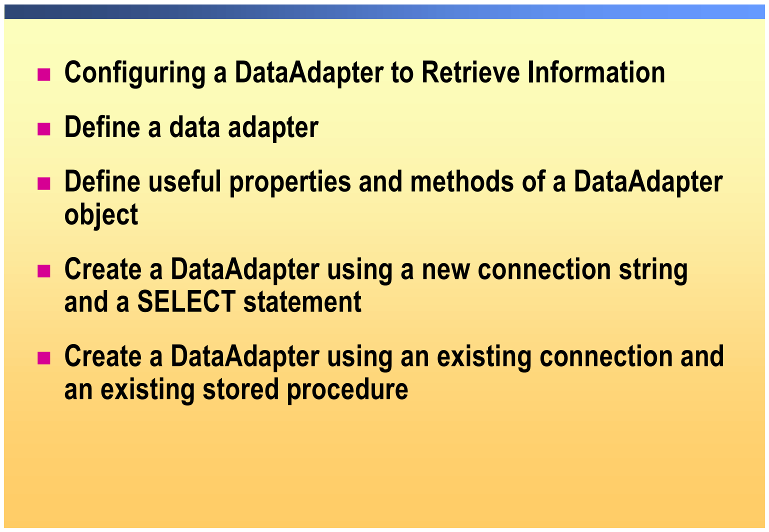
In the .NET environment, data can move from a central data source to a local DataSet. In order to move the data, there must be a bridge from the data source to the DataSet, and that bridge is the DataAdapter.

Objectives

After completing this module, you will be able to:

- Configure a DataAdapter to retrieve information
- Populate a DataSet by Using a DataAdapter
- Configure a DataAdapter to modify information
- Persist data changes to a server
- Manage data conflicts

Lesson: Configuring a DataAdapter to Retrieve Information



- **Configuring a DataAdapter to Retrieve Information**
- **Define a data adapter**
- **Define useful properties and methods of a DataAdapter object**
- **Create a DataAdapter using a new connection string and a SELECT statement**
- **Create a DataAdapter using an existing connection and an existing stored procedure**

Introduction

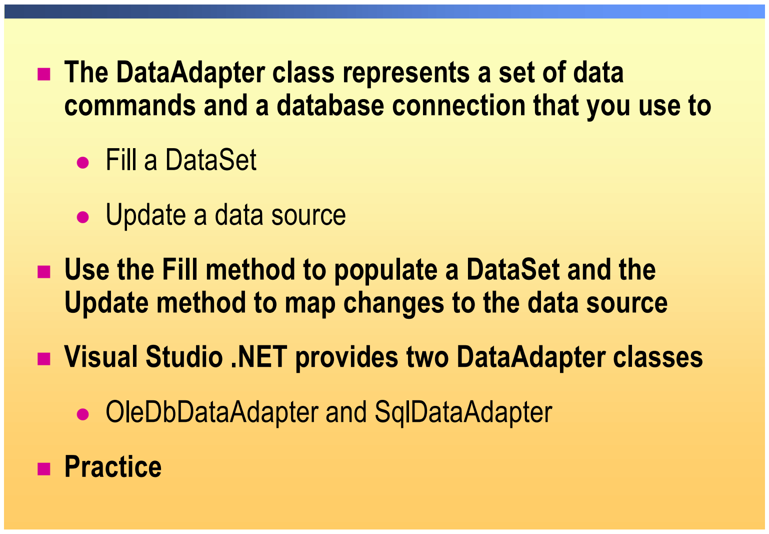
When you create an instance of a DataAdapter object, you can set it up to pull information from an existing data source.

Lesson Objective(s)

After completing this lesson, you will be able to:

- Define a data adapter
- Define useful properties of a DataAdapter object
- Define useful methods of a DataAdapter object
- Create a DataAdapter using a new connection string and a SELECT statement
- Create a DataAdapter using an existing connection and an existing stored procedure

What is a DataAdapter?



- **The DataAdapter class represents a set of data commands and a database connection that you use to**
  - Fill a DataSet
  - Update a data source
- **Use the Fill method to populate a DataSet and the Update method to map changes to the data source**
- **Visual Studio .NET provides two DataAdapter classes**
  - OleDbDataAdapter and SqlDataAdapter
- **Practice**

Introduction

The DataSet object, which represents a local copy of data from a data source, is one of the key innovations of the .NET Framework. By itself, it is useful for reference. However, to serve as a true data management tool, a DataSet must be able to interact with a data source. To accomplish this, .NET provides the **DataAdapter** class.

Definition

A **DataAdapter** object serves as a bridge between a DataSet and a data source for retrieving and saving data. The **DataAdapter** class represents a set of data commands and a database connection you use to fill a DataSet and update the data source. **DataAdapter** objects are part of the ADO.NET data providers, which also include connection objects, data-reader objects, and command objects.

Each **DataAdapter** exchanges data between a single **DataTable** object in a DataSet and a single result set from a SQL statement or stored procedure.

Scenario

You use DataAdapters to exchange data between a DataSet and a data source. A common scenario is that an application reads data from a database into a DataSet, and then writes changes from the DataSet back to the database. A DataAdapter can, however, retrieve and update data from any data source, such as from a BizTalk Server application to a DataSet.

Primary DataAdapters for databases

Visual Studio.NET makes two primary data adapters available for use with databases. In addition, other data adapters can be integrated with Visual Studio. The primary data adapters are:

- **OleDbDataAdapter**, which is suitable for use with any data source exposed by an OLE DB provider
- **SqlDataAdapter**, which is specific to SQL Server 7.0 or later database. It is faster than the OleDbDataAdapter because it works directly with SQL and does not go through an OLE DB layer.

Example

You have a SQL table that you want to make multiple modifications to, so you take a copy of a subset of the table, and store that copy in middle or user tier as a DataSet.

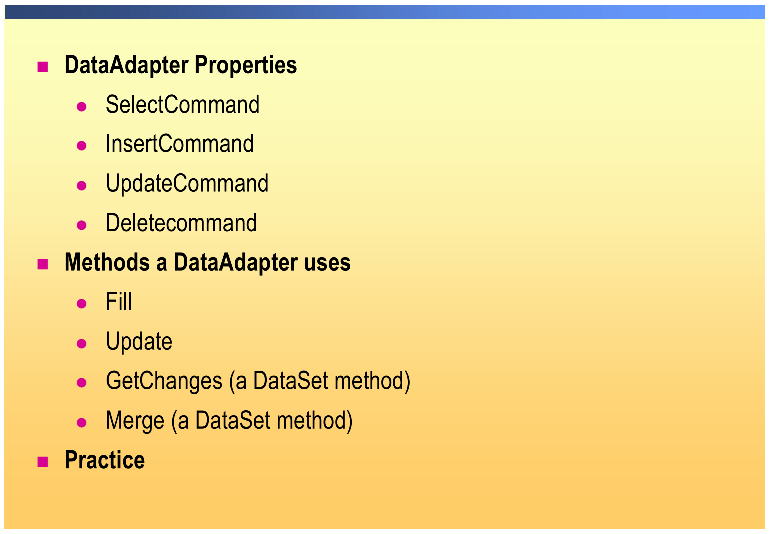
Non-example

A search function on your corporate Web site needs to return a list of matches on a Web page. It would be inappropriate to use a DataAdapter and DataSet because the results will be thrown away as soon as the page is created. There is no point in caching this data in a DataSet.

Practice

Group Discussion: When would you use a DataAdapter? Examples from each participant's company.

DataAdapter Properties and Methods



- **DataAdapter Properties**
  - SelectCommand
  - InsertCommand
  - UpdateCommand
  - Deletecommand
- **Methods a DataAdapter uses**
  - Fill
  - Update
  - GetChanges (a DataSet method)
  - Merge (a DataSet method)
- **Practice**

Introduction

Although the **DataAdapter** class contains a great many properties and methods, you will most likely use a certain subset of each.

DataAdapter Properties

You use DataAdapters to act on records from a data source. You can specify which actions you want to perform by using one of four DataAdapter properties, which executes a SQL statement or call a stored procedure. The properties are actually objects that are instances of the SqlCommand or OleDbCommand class:

- **SelectCommand**. A reference to a SQL statement or stored procedure that retrieves rows from the data source.
- **InsertCommand**. A reference to a command for inserting rows.
- **UpdateCommand**. A reference to a command for modifying rows.
- **DeleteCommand**. A reference to a command for deleting rows.

Methods a DataAdapter Uses

You use **DataAdapter** methods to fill a DataSet or to transmit changes in a DataSet table to a corresponding data store. These methods include:

- **Fill**. Use the **Fill** method of a **SqlDataAdapter** or **OleDbDataAdapter** to add or refresh rows from a data source and place them in a DataSet table. The **Fill** method uses the **SELECT** statement specified by an associated **SelectCommand** property.
- **Update**. Use this method of a **DataAdapter** object to transmit changes to a DataSet table to the corresponding data source. This method calls the respective **INSERT**, **UPDATE**, or **DELETE** statement for each specified row in a DataSet DataTable.

- **GetChanges**. Use this DataSet method to create a new DataSet that contains a copy of changes to a DataSet.
- **Merge**. Use this DataSet method to merge two DataSet objects that have similar schemas, one containing the original data, and the other containing only the changed data. This is useful in a middle-tier application which receives data updates from a client and then needs to merge these changes into its own DataSet.

Practice

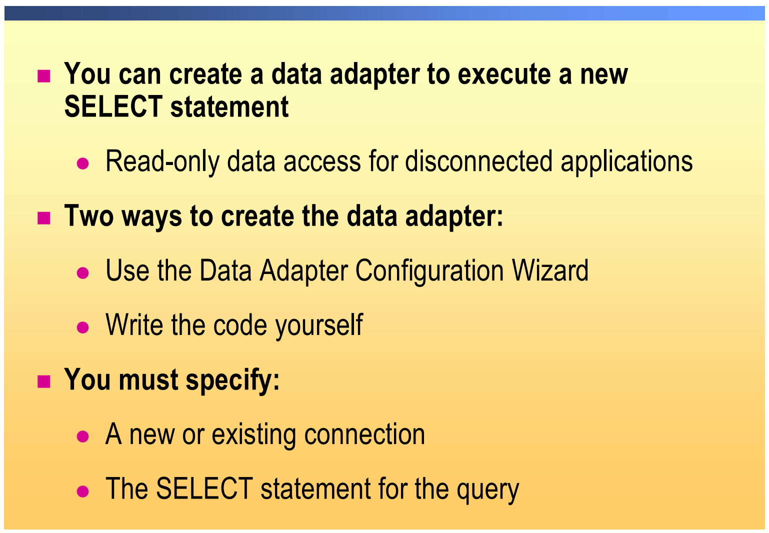
When you create a DataAdapter, you do not necessarily need to create **Command** objects for all the data modification commands (**SelectCommand**, **InsertCommand**, **UpdateCommand**, and **DeleteCommand**).

Describe a scenario where you would only need to create a **Command** object for the **SelectCommand** property.

Describe another scenario where you would only need to create two **Command** objects – one for the **SelectCommand** property, and one for the **UpdateCommand** property.

Describe another scenario where you would need to all four **Command** objects—one each for **SelectCommand**, **InsertCommand**, **UpdateCommand**, and **DeleteCommand**.

How to Create a DataAdapter that Uses a New SELECT Statement



- **You can create a data adapter to execute a new SELECT statement**
  - Read-only data access for disconnected applications
- **Two ways to create the data adapter:**
  - Use the Data Adapter Configuration Wizard
  - Write the code yourself
- **You must specify:**
  - A new or existing connection
  - The SELECT statement for the query

Introduction

You can create a **DataAdapter** object to execute a new **SELECT** statement. This provides disconnected applications with read-only access to the data in the database.

You can create the data adapter by using the Data Adapter Configuration Wizard, or programmatically in your code. You must specify a connection to the required database. You can also specify a new **SELECT** statement, to retrieve data from the database.

Scenario

A mobile worker needs to read addresses and telephone numbers for the company's offices around the world. The worker needs to view this information on the road, where there is no database connectivity available. The worker never needs to update the addresses or telephone numbers.

How to create a data adapter by using the Data Adapter Configuration Wizard

To create a data adapter using the Data Adapter Configuration Wizard, follow these steps:

1. Drag and drop a **SqlDataAdapter** control or **OleDbDataAdapter** control from the toolbox onto your form.
2. In the **Welcome** screen for the Data Adapter Configuration Wizard, click **Next**.
3. In the **Choose Your Data Connection** screen, click **New Connection**.
4. In the **Data Link Properties** dialog box, enter the connection details for the required database. Click **OK**.
5. Back in the **Choose Your Data Connection** screen, click **Next**.
6. In the **Choose a Query Type** screen, choose **Use SQL statements**. Click **Next**.
7. In the **Generate the SQL statements** screen, type an appropriate SQL query statement. Click **Advanced Options**.
8. In the **Advanced SQL Generation Options** dialog box, clear the **Generate Insert, Update, and Delete statements** check box. Click **OK**.

9. Back in the **Generate the SQL statements** screen, click **Next**.
10. In the **View Wizard Results** screen, click **Finish**.

How to create a data adapter programmatically

To create a data adapter programmatically, follow these steps:

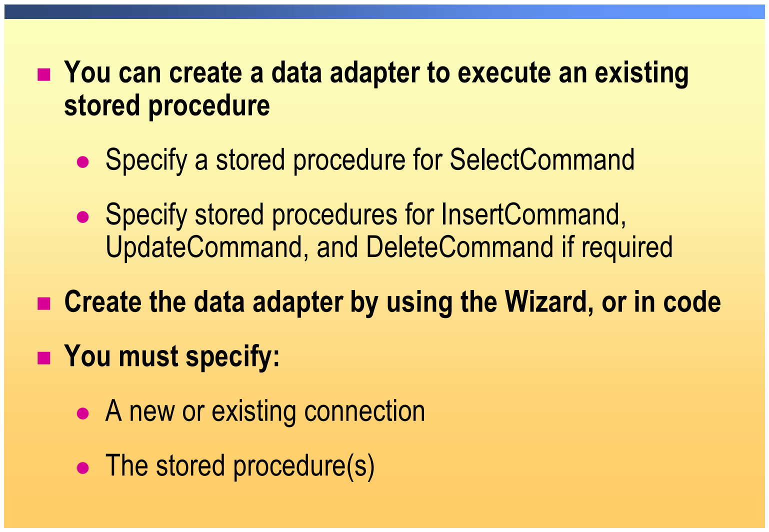
1. Create a new **SqlDataAdapter** object or **OleDbDataAdapter** object.
2. Create a new **SqlConnection** object or **OleDbConnection** object. Specify the connection string, to connect to the required database.
3. Create a new **SqlCommand** object or **OleDbCommand** object. Specify a SELECT statement, to retrieve the required data from the database.
4. Call the **AddParameter** method on the command object, to specify any parameters that are required by the SELECT statement.
5. Assign the new command object to the **SelectCommand** property of the data adapter object.

Example of creating a data adapter programmatically

The following example uses a **SqlDataAdapter** object to define a query on the **Products** table in the Northwind database. The database connection is specified by a **SqlConnection** object, and the query is specified by a **SqlCommand** object:

```
' Visual Basic
Imports System.Data.SqlClient
…
Dim daProducts As New SqlDataAdapter()

Dim cnNorthwind As New SqlConnection( _
  "data source=(local);initial catalog=Northwind;" & _
  "integrated security=SSPI")

Dim cmSelect As New SqlCommand( _
  "SELECT * FROM Products", cnNorthwind)

daProducts.SelectCommand = cmSelect


// Visual C#
using System.Data.SqlClient;
…
SqlDataAdapter daProducts = new SqlDataAdapter();

SqlConnection cnNorthwind = new SqlConnection(
  "data source=(local);initial catalog=Northwind;" +
  "integrated security=SSPI");

SqlCommand cmSelect = new SqlCommand(
  "SELECT * FROM Products", cnNorthwind);

daProducts.SelectCommand = cmSelect;
```

How to Create a Data Adapter that Uses an Existing Stored Procedure



- **You can create a data adapter to execute an existing stored procedure**
  - Specify a stored procedure for SelectCommand
  - Specify stored procedures for InsertCommand, UpdateCommand, and DeleteCommand if required
- **Create the data adapter by using the Wizard, or in code**
- **You must specify:**
  - A new or existing connection
  - The stored procedure(s)

Introduction

You can create a data adapter to execute an existing stored procedure. This enables disconnected applications to retrieve complex table joins, by using existing functionality in the database.

You can create the data adapter by using the Data Adapter Configuration Wizard, or programmatically in your code. You must specify a connection to the required database. You must also specify the name of the stored procedure you wish to call, to retrieve the data from the database.

Scenario

An organization has a suite of stored procedures, which retrieve consolidated data from several tables in the database. Mobile workers need read-only access to this consolidated data, in a disconnected application.

How to create a data adapter using the Data Adapter Configuration Wizard

To create a data adapter using the Data Adapter Configuration Wizard, follow these steps:

1. Drag and drop a **SqlDataAdapter** control or **OleDbDataAdapter** control from the toolbox onto your form.
2. In the **Welcome** screen for the Data Adapter Configuration Wizard, click **Next**.
3. In the **Choose Your Data Connection** screen, select an existing connection (or click **New Connection** and specify a new connection, if necessary).
4. In the **Choose a Query Type** screen, choose **Use existing stored procedures**. Click **Next**.
5. In the **Bind Commands to Existing Stored Procedures** screen, choose an existing stored procedure for the **Select** operation (if the stored procedure does not yet exist, create it now in the Server Explorer). Click **Next**.
6. In the **View Wizard Results** screen, click **Finish**.

How to create a data adapter programmatically

To create a data adapter programmatically, follow these steps:

1. Create a new **SqlDataAdapter** object or **OleDbDataAdapter** object.
2. Create a new **SqlConnection** object or **OleDbConnection** object (or use an existing **XxxConnection** object if you have one available).
3. Create a new **SqlCommand** object or **OleDbCommand** object. Specify the following properties for the command object:

Property	Description
Connection	The **XxxConnection** object
CommandText	The name of the stored procedure you wish to call
CommandType	**System.Data.CommandType.StoredProcedure**

4. Call the **AddParameter** method on the command object, to specify any parameters that are required by the stored procedure.
5. Assign the new command object to the **SelectCommand** property of the data adapter object.

Example of creating a data adapter programmatically

The following example creates a **SqlDataAdapter** object, and uses an existing stored procedure named **GetProductsAndCategories** to query the database. An existing **SqlConnection** object named **cnNorthwind** is used to connect to the database:

```
' Visual Basic
Imports System.Data
Imports System.Data.SqlClient
…
Dim daProdCat As New SqlDataAdapter()

Dim cmSelect As New SqlCommand()
cmSelect.Connection = cnNorthwind
cmSelect.CommandText = "GetProductsAndCategories"
cmSelect.CommandType = CommandType.StoredProcedure

daProdCat.SelectCommand = cmSelect

// Visual C#
using System.Data;
using System.Data.SqlClient;
…
SqlDataAdapter daProdCat = new SqlDataAdapter();

SqlCommand cmSelect = new SqlCommand();
cmSelect.Connection = cnNorthwind;
cmSelect.CommandText = "GetProductsAndCategories";
cmSelect.CommandType = CommandType.StoredProcedure;

daProdCat.SelectCommand = cmSelect;
```

Practice

Northwind Traders needs to build a disconnected data application that allows users to view information that is held in the product catalog.

In this practice, you will create a Windows Application containing two data adapters. The first data adapter will retrieve category information from the Northwind database. The second data adapter will retrieve product information from the same database

1. Create a new Windows Application solution named **CatalogViewer** at the following location.

   <install folder>\Practices\Mod06_1\
2. Drag and drop a **DataGrid** onto the form.
3. Drag and drop a **Button** onto the form. Change the text of the button to **Fill**.
4. Drag and drop a **SqlDataAdapter** control from the toolbox onto the form and use the Data Adapter Configuration Wizard to set the following properties:

Property	**Value**
Server Name	(local)
Log On	Use Windows NT Integrated security
Database	Northwind
Query Type	Use SQL statements
Load Statement	SELECT * FROM Categories
Advanced Options	All options enabled

5. Select the new data adapter. Use the Property Window to change its Name to **daCategories**.
6. In the Server Explorer, create a new stored procedure in the Northwind database as follows:

```
CREATE PROCEDURE dbo.usp_GetProducts
AS
  SELECT * FROM Products
```

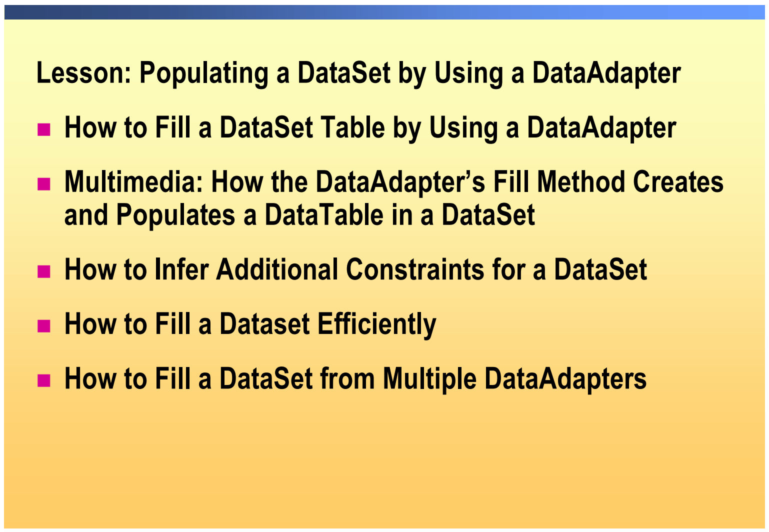
7. In the Form Designer, drag and drop another **SqlDataAdapter** control onto the form. Use the Data Adapter Configuration Wizard to set the following properties:

Property	**Value**
Connection	(Use the connection you created earlier)
Query Type	Use existing stored procedures
Select stored procedure	usp_GetProducts

8. Select the new data adapter. Use the Property Window to change its Name to **daProducts**.
9. Save all the files in your solution.

The solution for this practice is located at <install folder>\
Practices\Mod06_1\Lesson1\CatalogViewer\

Lesson: Populating a DataSet by Using a DataAdapter



Lesson: Populating a DataSet by Using a DataAdapter

- How to Fill a DataSet Table by Using a DataAdapter
- Multimedia: How the DataAdapter's Fill Method Creates and Populates a DataTable in a DataSet
- How to Infer Additional Constraints for a DataSet
- How to Fill a Dataset Efficiently
- How to Fill a DataSet from Multiple DataAdapters

Introduction

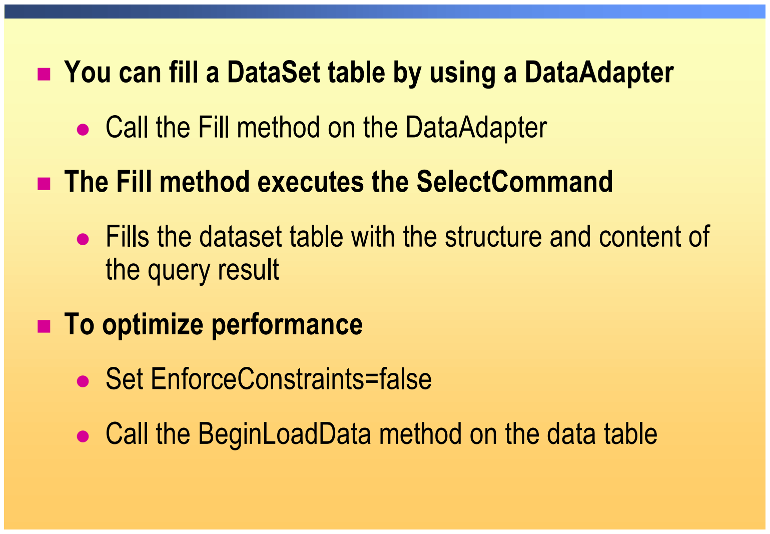
After you choose the type of data adapter you want to use, **SqlDbDataAdapter** or **OleDbDataAdapter**, and configure it to perform the tasks you need, you are ready to populate the DataSet for which you created the DataAdapter.

Lesson objectives

When you complete this lesson, you will be able to:

- Diagram how the **Fill** method works
- Infer additional constraints for a DataSet
- Call the DataAdapter's **Fill** method to populate a DataSet efficiently
- Populate a DataSet from Multiple DataAdapters

How to Fill a DataSet Table by Using a DataAdapter



- **You can fill a DataSet table by using a DataAdapter**
  - Call the Fill method on the DataAdapter
- **The Fill method executes the SelectCommand**
  - Fills the dataset table with the structure and content of the query result
- **To optimize performance**
  - Set EnforceConstraints=false
  - Call the BeginLoadData method on the data table

Introduction

You can fill a DataSet table by using a DataAdapter. Call the **Fill** method on the DataAdapter, specifying the DataSet table you wish to fill.

Definition of the Fill method

The **Fill** method implicitly executes the SQL query in the **SelectCommand** of the DataAdapter. The results of the query are used to define the structure of the DataSet table, and to populate the table with data.

Syntax for the Fill method

The **Fill** method is overloaded. Here are some of the overloaded versions of **Fill**:

```
rowsAffected = aDataAdapter.Fill(aDataSet)
rowsAffected = aDataAdapter.Fill(aDataSet, strDataTableName)
rowsAffected = aDataAdapter.Fill(aDataTable)
```

Performance considerations

When you fill a DataSet, the **DataAdapter** enforces constraints such as primary key uniqueness. To improve performance, set the DataSet property **EnforceConstraints** to **False** before you fill the DataSet. This disables constraint checking while the data is loaded:

```
aDataSet.EnforceConstraints = false
```

Another way to improve performance is to call the **BeginLoadData** method on the data table. This turns off index maintenance and notifications while data is loaded into the table. Call **EndLoadData** after the data has been loaded:

```
aDataTable.BeginLoadData()
…
aDataTable.EndLoadData()
```

Example of filling a DataSet by using a Data Adapter

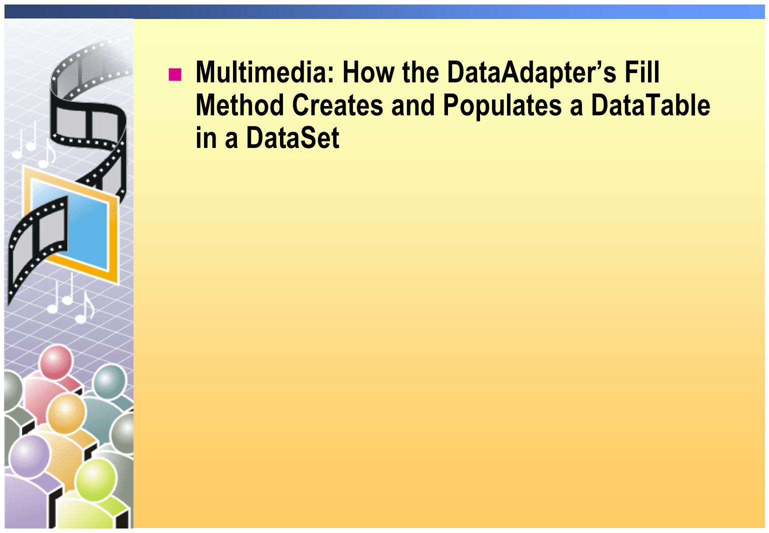
The following example creates a DataSet containing a single table named **Customers**. The table is filled by using a DataAdapter named **daCustomers**. The **BeginLoadData** method is called, to optimize performance.

After the table has been filled, a **DataGrid** control is bound to the table. The **DataGrid** will display the customer information on the screen.

```
' Visual Basic
Dim dsCustomers As New DataSet()
dsCustomers.Tables.Add(New DataTable("Customers"))

dsCustomers.Tables(0).BeginLoadData()
daCustomers.Fill(dsCustomers, "Customers")
dsCustomers.Tables(0).EndLoadData()

DataGrid1.DataSource = dsCustomers.Tables(0).DefaultView
```

```
// Visual C#
DataSet dsCustomers = new DataSet();
dsCustomers.Tables.Add(new DataTable("Customers"));

dsCustomers.Tables[0].BeginLoadData();
daCustomers.Fill(dsCustomers, "Customers");
dsCustomers.Tables[0].EndLoadData();

dataGrid1.DataSource = dsCustomers.Tables[0].DefaultView;
```

Multimedia: How the DataAdapter's Fill Method Creates and Populates a DataTable in a DataSet

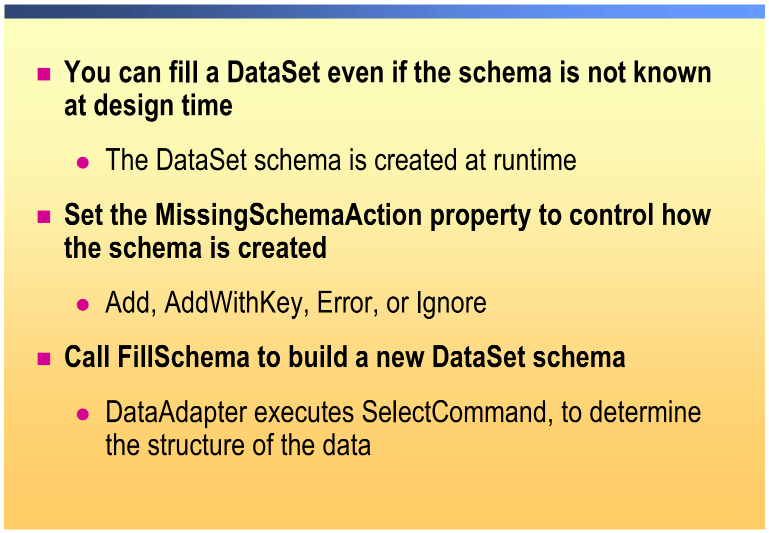


- Multimedia: How the DataAdapter's Fill Method Creates and Populates a DataTable in a DataSet

Introduction

This animation provides an overview of how the **Fill** method of a DataAdapter object creates a **DataTable** in a DataSet, and then populates that **DataTable**.

How to Infer Additional Constraints for a DataSet



- **You can fill a DataSet even if the schema is not known at design time**
  - The DataSet schema is created at runtime
- **Set the MissingSchemaAction property to control how the schema is created**
  - Add, AddWithKey, Error, or Ignore
- **Call FillSchema to build a new DataSet schema**
  - DataAdapter executes SelectCommand, to determine the structure of the data

Introduction

You can fill a DataSet even if the schema is not known at design time. The DataSet schema can be created at runtime, based on the structure of the retrieved data.

You can control how a DataSet schema is created and modified at runtime. Before you fill the dataset, do one of the following:

- Set the **MissingSchemaAction** property on the DataAdapter
- Call the **FillSchema** method on the DataAdapter

Definition of the MissingSchemaAction property

Set the MissingSchemaAction property to control how the schema is created. The **MissingSchemaAction** property specifies the action to take when you retrieve **DataTables** or **DataColumns** that are not present in the DataSet schema.

Use one of the following values for the **MissingSchemaAction** property:

MissingSchemaAction value	Description
Add	Adds extra tables and columns to the DataSet schema, but does not preserve primary key information. If you add the same rows to the DataSet several times, the rows are appended each time rather than being modified. This is because the DataSet does not check for primary keys, and therefore does not realize the same rows are being loaded.
AddWithKey	Extra tables and columns are added to the schema. Primary key information is added to the data table, to overcome the limitations of the **Add** property value described above. The **AllowDBNull**, **AutoIncrement**, **MaxLength**, **ReadOnly**, and **Unique** properties are set for the new columns, as defined in the data source. The **PrimaryKey** property is also set for primary key columns. If there are no primary keys, but the resultset contains unique columns that are all non-nullable, the unique columns are assigned the **PrimaryKey** property. If any unique columns are nullable, a **UniqueConstraint** is added to the **ConstraintCollection** for the DataSet, but the **PrimaryKey** property is not set.
Error	Generates a **SystemException**. This is useful if the retrieved data must comply with a predefined DataSet schema.
Ignore	Ignores extra tables and columns in the resultset.

Syntax for the MissingSchemaAction property

The following example shows the syntax for the **MissingAction** property of a DataAdapter object.

```
aDataAdapter.MissingSchemaAction =
                    MissingSchemaAction.Add |
                    MissingSchemaAction.AddWithKey |
                    MissingSchemaAction.Error |
                    MissingSchemaAction.Ignore
```

Example of using MissingSchemaAction

The following example creates an untyped DataSet, and uses a DataAdapter named **daCustomers** to fill the DataSet. The **MissingSchemaAction** property is set to **AddWithKey**, so that the DataSet schema is amended when the DataSet is filled. This creates the necessary tables and columns in the DataSet, to accommodate the data as it is loaded:

```
' Visual Basic
Dim dsCustomers As New DataSet()
daCustomers.MissingSchemaAction = _
                           MissingSchemaAction.AddWithKey
daCustomers.Fill(dsCustomers)
DataGrid1.DataSource = dsCustomers.Tables(0).DefaultView

// Visual C#
DataSet dsCustomers = new DataSet();
daCustomers.MissingSchemaAction =
                           MissingSchemaAction.AddWithKey;
daCustomers.Fill(dsCustomers);
dataGrid1.DataSource = dsCustomers.Tables[0].DefaultView;
```

Definition of the FillSchema method

Call **FillSchema** to build a new DataSet schema**.** The **FillSchema** method executes the **SelectCommand** object on the DataAdapter, to determine the schema of the data retrieved by that command. The **FillSchema** method takes a **SchemaType** parameter, which can be one of the following values:

SchemaType parameter	Description
Mapped	Applies any existing table mappings to the retrieved schema, and configures the DataSet with the transformed schema.
Source	Ignores any existing table mappings in the DataAdapter, and configures the DataSet with the retrieved schema.

Syntax for the FillSchema method

```
aDataTableArray = aDataAdapter.FillSchema(
                        aDataSet,
                        SchemaType.Mapped | SchemaType.Source)
```

Example of using FillSchema

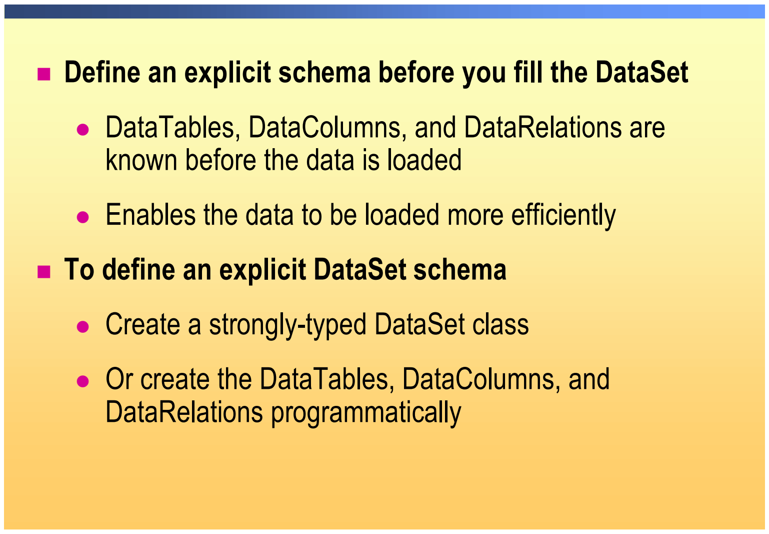
The following example creates an untyped DataSet. The schema for the DataSet is defined by calling the **FillSchema** method on a DataAdapter. The data for the DataSet is retrieved by calling the **Fill** method on the DataAdapter:

```
' Visual Basic
Dim dsCustomers As New DataSet()
daCustomers.FillSchema(dsCustomers, SchemaType.Mapped)
daCustomers.Fill(dsCustomers)
DataGrid1.DataSource = dsCustomers.Tables(0).DefaultView

// Visual C#
DataSet dsCustomers = new DataSet();
daCustomers.FillSchema(dsCustomers, SchemaType.Mapped);
daCustomers.Fill(dsCustomers);
dataGrid1.DataSource = dsCustomers.Tables[0].DefaultView;
```

Performance considerations

The **MissingSchemaAction** property and the **FillSchema** method are slow, because they build the DataSet schema at runtime. You should avoid using these techniques if possible. A more efficient solution is to use strongly typed DataSets, where the schema for the DataSet is defined at design time. This enables the DataSet to retrieve data quickly into a known schema, rather than having to deduce the schema first.

How to Fill a Dataset Efficiently



- **Define an explicit schema before you fill the DataSet**
  - DataTables, DataColumns, and DataRelations are known before the data is loaded
  - Enables the data to be loaded more efficiently
- **To define an explicit DataSet schema**
  - Create a strongly-typed DataSet class
  - Or create the DataTables, DataColumns, and DataRelations programmatically

Introduction

The most efficient way to fill a DataSet is to define an explicit schema before filling the DataSet. This means the DataTables, DataColumns, and DataRelations are already known before the DataSet is filled.

There are two ways to define an explicit schema for a DataSet:

- Create a strongly-typed DataSet in the Form Designer
- Create the DataTables, DataColumns, and DataRelations programmatically

Scenario

A disconnected application retrieves customer information from a central database. The structure of the data is known in advance. You can therefore create a strongly-typed DataSet, with a schema that conforms to the structure of the retrieved data. This enables data to be loaded efficiently at runtime.

How to create a strongly-typed DataSet in the Form Designer

To create a strongly-typed DataSet in the Form Designer, follow these steps:

1. Drag and drop a **SqlDataAdapter** control or **OleDbDataAdapter** control from the toolbox onto your form.
2. Configure the DataAdapter as required, using the Data Adapter Configuration Wizard.
3. Right-click the new DataAdapter object, and choose **Generate Dataset**.
4. In the **Generate Dataset** dialog box, specify a name for the new DataSet class.
5. Choose the tables that you wish to add to the **DataSet**.
6. Ensure the **Add this dataset to the designer** check box is checked.
7. Click **OK**. This will create a strongly-typed DataSet class, inherited from **DataSet**. An instance of this class will also be created and added to your application.
8. Right-click the new DataSet object, and choose **View Schema**.

9. In the XML Designer, examine the XSD schema for the DataSet. Modify and extend the XSD schema if necessary, by dragging XSD schema elements from the toolbox onto the XML Designer.
10. In your application, write code to fill the DataSet by using a Data Adapter.

Example of filling a strongly-typed DataSet

The following example fills a strongly-typed DataSet object named **dsCustomers**. The DataSet has a single table named **Customers**. The **BeginLoadData** method is called before the data is loaded, to optimize performance:

```
' Visual Basic
dsCustomers.Customers.BeginLoadData()
daCustomers.Fill(dsCustomers.Customers)
dsCustomers.Customers.EndLoadData()
DataGrid1.DataSource = dsCustomers.Customers.DefaultView

// Visual C#
dsCustomers.Customers.BeginLoadData();
daCustomers.Fill(dsCustomers.Customers);
dsCustomers.Customers.EndLoadData();
dataGrid1.DataSource = dsCustomers.Customers.DefaultView;
```

How to define a DataSet schema programmatically

To define a DataSet schema programmatically, write the following code:

1. Create a **DataTable** object.
2. Create a **DataColumn** object for each column you require in the table.
3. Add these columns to the table. To do this, call the **Add** method on the **Columns** collection in the **DataTable** object.
4. Define constraints on the table. To do this, call the **Add** method on the **Constraints** collection in the **DataTable** object.
5. Repeat steps 1 to 4 as necessary, to create additional **DataTable** objects.
6. Create a **DataSet** object.
7. Add the **DataTable** objects to the **DataSet**. To do this, call the **Add** method on the **Tables** collection in the **DataSet** object.
8. Define relations between columns in the **DataSet**. To do this, call the **Add** method on the **Relations** collection in the **DataSet** object.

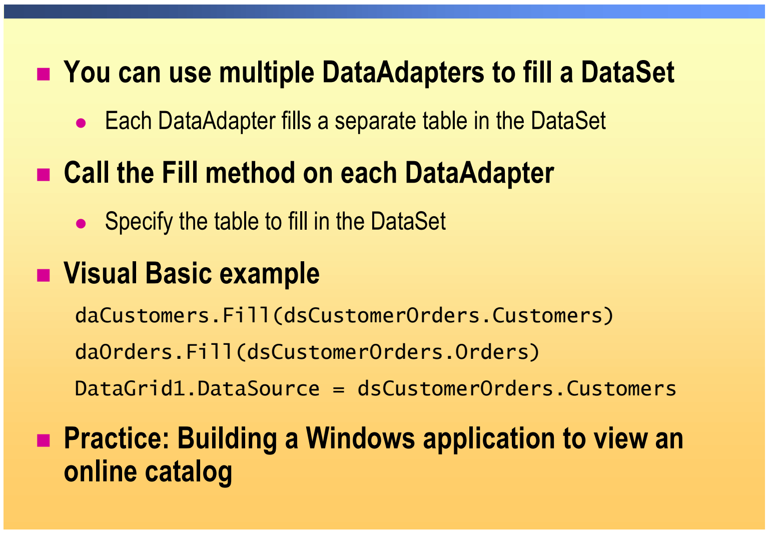
Example of defining a DataSet schema programmatically

The following example shows how to create a DataSet schema programmatically. The DataSet contains a single table named **Customers**. The table has three columns named **CustomerID**, **CompanyName**, and **ContactName** (all strings). The **CustomerID** column is a primary key.

Once the DataSet schema has been defined, the DataSet is filled by using a DataAdapter named **daCustomers**. A **DataGrid** control is then bound to the DataSet:

```
' Visual Basic
' Create the DataTable and DataColumns
Dim table As New DataTable("Customers")
Dim c1 As New DataColumn("CustomerID",  GetType(String))
Dim c2 As New DataColumn("CompanyName", GetType(String))
Dim c3 As New DataColumn("ContactName", GetType(String))

' Add DataColumns and Constraints to the DataTable
table.Columns.Add(c1)
table.Columns.Add(c2)
table.Columns.Add(c3)
table.Constraints.Add("PK_CustomerID", c1, True)

' Create the DataSet, and add the DataTable to it
Dim dsCustomers As New DataSet()
dsCustomers.Tables.Add(table)

' Fill DataSet by using a DataAdapter, and bind to a DataGrid
dsCustomers.Tables(0).BeginLoadData()
daCustomers.Fill(dsCustomers, "Customers")
dsCustomers.Tables(0).EndLoadData()
DataGrid1.DataSource = dsCustomers.Tables(0).DefaultView


// Visual C#
// Create the DataTable and DataColumns
DataTable table = new DataTable("Customers");
DataColumn c1 = new DataColumn("CustomerID",  typeof(String));
DataColumn c2 = new DataColumn("CompanyName", typeof(String));
DataColumn c3 = new DataColumn("ContactName", typeof(String));

// Add DataColumns and Constraints to the DataTable
table.Columns.Add(c1);
table.Columns.Add(c2);
table.Columns.Add(c3);
table.Constraints.Add("PK_CustomerID", c1, true);

// Create the DataSet, and add the DataTable to it
DataSet dsCustomers = new DataSet();
dsCustomers.Tables.Add(table);

// Fill DataSet by using a DataAdapter, and bind to a DataGrid
dsCustomers.Tables[0].BeginLoadData();
daCustomers.Fill(dsCustomers, "Customers");
dsCustomers.Tables[0].EndLoadData();
dataGrid1.DataSource = dsCustomers.Tables[0].DefaultView;
```

How to Fill a DataSet from Multiple DataAdapters



- **You can use multiple DataAdapters to fill a DataSet**
  - Each DataAdapter fills a separate table in the DataSet
- **Call the Fill method on each DataAdapter**
  - Specify the table to fill in the DataSet
- **Visual Basic example**

```
daCustomers.Fill(dsCustomerOrders.Customers)
daOrders.Fill(dsCustomerOrders.Orders)
DataGrid1.DataSource = dsCustomerOrders.Customers
```

- **Practice: Building a Windows application to view an online catalog**

Introduction

You can use multiple DataAdapters to fill a DataSet. Each DataAdapter fills a separate table in the DataSet.

Scenario

A salesperson needs to retrieve customer information, and information about orders placed by each customer, from the central database. To meet this requirement, create a disconnected application that contains two DataAdapters: one to retrieve Customer records, and the other to retrieve order records.

Then, create a strongly-typed DataSet that contains two tables (Customers and Orders), and define a relation to associate orders with customers. After you create the strongly-typed DataSet, use the two DataAdapters to fill the tables in the DataSet.

Example

The following example populates a strongly-typed DataSet by using two DataAdapters named **daCustomers** and **daOrders**. The Dataset has a **Customers** table and an **Orders** table. The **Customers** table is populated with the **daCustomers** DataAdapter. The **Orders** table is populated with the **daOrders** DataAdapter.

Once the DataSet has been populated, a **DataGrid** control is bound to the **Customers** table in the DataSet. The **DataGrid** will display the customers, and the orders placed by each customer.

```
' Visual Basic
daCustomers.Fill(dsCustomerOrders.Customers)
daOrders.Fill(dsCustomerOrders.Orders)
DataGrid1.DataSource = dsCustomerOrders.Customers.DefaultView

// Visual C#
daCustomers.Fill(dsCustomerOrders.Customers);
daOrders.Fill(dsCustomerOrders.Orders);
dataGrid1.DataSource = dsCustomerOrders.Customers.DefaultView;
```

Practice

In this practice, you will continue to build a Windows Application that allows the users to view the Northwind Traders online product catalog. The solution for this practice is located at <install folder>\Practices\Mod06_1\Lesson2\CatalogViewer\

In the first part of this practice, you will see how the **MissingSchemaAction** property influences how a DataAdapter fills a DataSet:

1. Open the Windows Application solution you created in the previous practice, or the solution named **CatalogViewer.**
2. Open **Form1** in the Form Designer, and then right-click **daCategories** and choose **Preview Data**.
3. Click **Fill DataSet**. This button calls the data adapter's **Fill** method, so it is a useful way of testing a data adaptor. How many bytes of memory does the DataSet require? How many rows are returned?
4. Click **Fill DataSet** again. This simulates refreshing the DataSet with the latest data in the underlying database. How many bytes of memory does the DataSet require now? How many rows are returned? Why are rows being duplicated?
5. Set the **MissingSchemaAction** property of the data adapter to **AddWithKey**.
6. Right-click **daCategories** and choose **Preview Data**.
7. Click **Fill DataSet** twice. Are rows still being duplicated?
8. Set the **MissingSchemaAction** property of the two data adapters to **Error**, because the **Add** and **AddWithKey** values for this property have a negative impact on performance. You will use a DataSet schema instead.

In the next part of this practice, you will generate a strongly typed DataSet based on the structure of the data retrieved by the DataAdapter:

1. Right-click **daCategories** and choose **Generate Dataset**.
2. Set the name of the new DataSet to **CatalogDataSet**, and select both **daCategories** and **daProducts** data adapters.
3. Select the new dataset in the Form Designer. Use the Property Window to change its Name to **dsCatalog**.
4. Right-click **dsCatalog**, and choose **View Schema**. This will open the XSD file that was generated for you by the Wizard.
5. In the **usp_Products** box, change the first field from **usp_Products** to **Products**.
6. Right-click the **Products** box, and choose **Add-New Relation**. Click OK in the Edit Relation dialog box.
7. Click the background of the Schema Designer, to select the DataSet. In the Property Window, expand the **key** collection. Rename the two constraints to **PK_Categories** and **PK_Products**.

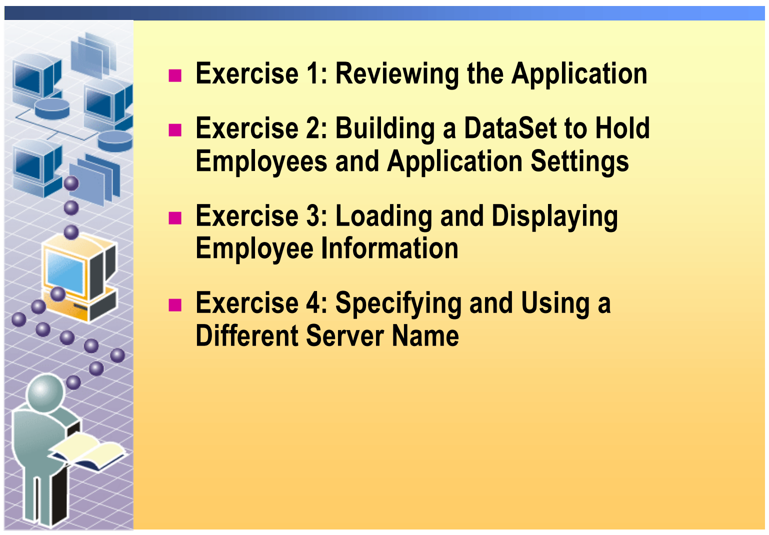
In the final part of this practice, you will use the DataAdapter to fill the DataSet with data from the data store:

1. Return to the Form Designer. Add a Click event handler for the **Fill** button.
2. In the event handler, call the **Fill** method of the two data adaptors. Also bind the DataGrid control to the Categories table in the DataSet:

   ```
   daCategories.Fill(dsCatalog.Categories)
   daProducts.Fill(dsCatalog.Products)
   DataGrid1.DataSource = dsCatalog.Categories
   ```

3. Run and test your application. Verify that the relationship between categories and products is recognized by the DataGrid.
4. Use Server Explorer to change some data in the Products table in the SQL Server Northwind database. Verify that you can use the **Fill** button your form to refresh the DataSet, and see changes made to the underlying data.

Lab 6.1: Retrieving Data into a Disconnected Application



- Exercise 1: Reviewing the Application
- Exercise 2: Building a DataSet to Hold Employees and Application Settings
- Exercise 3: Loading and Displaying Employee Information
- Exercise 4: Specifying and Using a Different Server Name

Objectives

After completing this lab, you will be able to:

- Create and configure a DataAdapter.
- Generate a strongly-typed DataSet from the DataAdapter.
- Use the XML Designer to adjust the schema in the DataSet.
- Use the DataAdapter to fill the DataSet.
- Save the DataSet data as an XML diffgram.

Prerequisites

Before working on this lab, you must have:

- .
- .

For More Information

See the DataSet and SqlDataAdapter topics in the Visual Studio .NET documentation.

Scenario

Northwind Traders has many sales persons on the road visiting customers. They need to be able to update customer data including orders while away from the office. Each sales person typically has responsibility for a limited subset of the total central sales database, so it unnecessary to give every sales person a complete copy of the central database.

The application must allow sales persons to update the data while on the road, and then synchronize when they return to the office.

In this lab, you will retrieve data into DataSets in a disconnected application. In Lab 6.2, you will update the database by using the data in the DataSets.

Estimated time to complete this lab: 60 minutes

Exercise 0
Lab Setup

To complete this lab, you must …

►

2.

3.

Exercise 1
Reviewing the Application

In this exercise, you will review a complete solution to this lab so that you clearly understand how it works. This solution will show how the application loads data into the disconnected application.

Then you will review a starter solution that will be the starting point for the application you will complete in the other exercises of Labs 6.1 and 6.2.

Scenario

The "On The Road" Windows application is used by Northwind Traders sales people to track customer orders while the user is on the road, and do not therefore have access to the central database servers.

The application will run on the users' laptops. While in the office, the user can connect to the corporate network and get the latest order data for their customers. This will be a subset of the order data stored on the central database server. The user creates the subset of data by choosing their name from a list of employees, and this information is used to only return customer and order data for that employee.

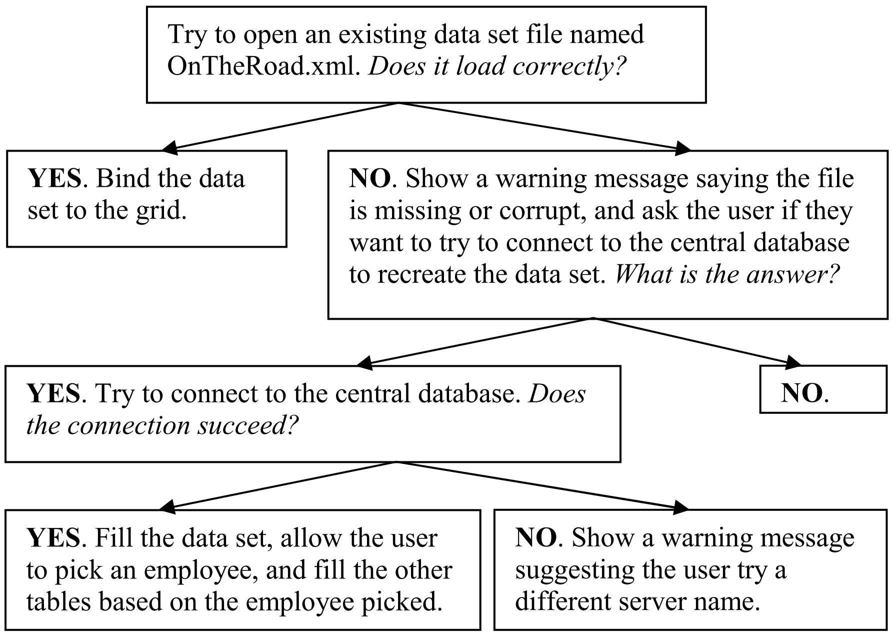
When the application closes, it automatically saves a copy of the current data set to the local disk drive of the laptop. When the application is next executed, it automatically opens the saved data set so that the user can immediately continue working on the data.

While on the road, users can add new orders, and edit and delete existing orders. When the user returns to the office, they can choose a menu item to update the central database with the changes they have made to the data set. You will implement this functionality in Lab 6.2.

The application allows the user to specify the server name that hosts the central database. It also has an About dialog to display copyright information.

Application Startup Decision Tree

This is the decision tree for when the "On The Road" application starts up.



Try to open an existing data set file named OnTheRoad.xml. *Does it load correctly?*

YES. Bind the data set to the grid.

NO. Show a warning message saying the file is missing or corrupt, and ask the user if they want to try to connect to the central database to recreate the data set. *What is the answer?*

YES. Try to connect to the central database. *Does the connection succeed?*

NO.

YES. Fill the data set, allow the user to pick an employee, and fill the other tables based on the employee picked.

NO. Show a warning message suggesting the user try a different server name.

Exercise Steps

Now you will open the solution and test the complete application.

► **To open and rebuild the complete solution**

1. Start the Microsoft Visual Studio .NET development environment.
2. Open an existing project named **OnTheRoad**. The location is <install path>\2389\Labs\Lab06_2\Solution\Ex3\xx\ where xx is either VB or CS. This project contains the complete solution for all the work you will do in Labs 6.1 and 6.2.
3. Rebuild the solution.
4. Exit the Microsoft Visual Studio .NET development environment.
5. Start the Microsoft SQL Server Query Analyzer.
6. Open the script named **lab6setup.sql** in this folder:

   <install path>\2389\Labs\Lab06_1\
7. Run the script.
8. Exit the Microsoft SQL Server Query Analyzer.

► **To test the application settings**

1. Open **Windows Explorer** and go to one of the following folders:

   <install path>\2389\Labs\Lab06_2\Solution\Ex3\VB\OnTheRoad\bin\

   - or -

   <install path>\2389\Labs\Lab06_2\Solution\Ex3\CS\OnTheRoad\bin\debug\
2. There should be two files, named **OnTheRoad.exe** (the application executable) and **OnTheRoad.pdb** (program debug database). If there is a file named **OnTheRoad.xml**, delete it (this is where the data set is saved while on the road).
3. Double-click the executable **OnTheRoad.exe** to run it.
4. You will see a warning message saying that a data set was not found, and offering to connect to the central database to create one. Click **No**.
5. Choose the **Tools – Options** menu item. Notice you can change the server name for the central database, and that it is currently set to **(local)**. Click **Cancel**. We do not want to change this option yet.
6. Close the application.
7. In Windows Explorer, notice that a file was created named **OnTheRoad.xml**. Double-click the file to open it in Internet Explorer.
8. Review the contents of the **OnTheRoad.xml** file in Internet Explorer. Notice that it currently contains the ID of the currently selected employee (it defaults to zero) and the server name for the central database.
9. Close Internet Explorer.
10. Double-click the executable **OnTheRoad.exe** to run it again.
11. Choose **No** to the warning message, because you are still not ready to connect to the central database.
12. Choose the **Tools – Options** menu item, and change the server name to the name of your computer.

13. Close the application.
14. Double-click the **OnTheRoad.xml** file to open it in Internet Explorer again, and note that the server name has changed.

► To test the local data set caching

1. Rerun the executable, and choose **Yes** to the warning message. This will connect to the central database, and download a list of employees from the database.
2. In the **Get from central database** dialog box, choose **Dodsworth, Anne** for the employee name and click **OK**. You will then see all the customers (and their orders and order details) managed by Anne.
3. Close the application. This will automatically save the data set into the same XML file that stored the application settings.
4. Reopen the **OnTheRoad.xml** file using Internet Explorer.
5. Choose the **Edit – Find (on This Page)…** menu item to search for the XML elements that begin with: <Products, <Employees, <Customers, <Orders, <OrderDetails, and <AppSettings. Review the contents.
6. Rerun the executable. Notice you are no longer shown the warning message because the XML file contains a complete and valid data set.
7. In the data grid, expand the customer with the company name of **Around The Horn**. Notice it currently has two orders. Change the order date of the first order to today's date.
8. Expand the first order and add a third order detail row, for a product ID 1, with a unit price of 25 and a quantity of 4. Click on the first or second row to make sure the change is made to the data set.
9. Choose the **Update to central database** menu item. In the central database, one row will be added to the OrderDetails table and one row in the Orders table will be modified.
10. Use the Server Explorer to check that the changes were successfully made.

Note Lab 6.1 only deals with retrieving data from the central database. Lab 6.2 deals with updating the central database.

► To remove the stored procedures used by the solution

1. Start the Microsoft SQL Server Query Analyzer.
2. Open the script named **lab6reset.sql** in this folder:

   <install path>\2389\Labs\Lab06_1\
3. Run the script.
4. Exit the Microsoft SQL Server Query Analyzer.

► To review the starter solution

1. Start the Microsoft Visual Studio .NET development environment.
2. Open the existing project named **OnTheRoad**. The location is <install path>\2389\Labs\Lab06_1\Starter\xx\ where xx is either VB or CS.
3. Open each of the following files using the Designer view and notice that they each provide a very simple dialog box user interface for performing certain tasks.

Form	Description
About	Shows copyright and version information.
Logon	Allows the user to pick a named employee from a list and then retrieves the customer data associated with that employee from the central database.
Options	Allows the user to change the SQL Server name of the central database.

4. Open the file named **MainForm** in Designer view, and review the menu and its items. The menu items will perform the following tasks.

Menu	Task
File – Get from central database…	Shows the Logon form
File – Update to central database	Updates the central database with the latest changes made in the grid
File – Exit	Ends the application
Tools – Options…	Shows the Options form
Help – About…	Shows the About form

► To test the starter solution

- Run the starter solution and click each of the menu items.

Exercise 2
Building a DataSet to Hold Employees and Application Settings

In this exercise, you will build a custom data set that initially contains two tables: one table for storing a list of all the employees IDs and full names, and one table to store application settings. You will create and configure a data adapter, so that it populates the employees table. You will populate the application settings table programmatically in your code.

You will also write code to save the DataSet to an XML document when the application closes.

Scenario

You will create a custom DataSet class and schema that can track the application specific options i.e. the employee using the application and the server name for the central database.

► To open the starter solution

1. Start the Microsoft Visual Studio .NET development environment.
2. Open the existing project named **OnTheRoad**. The location is <install path>\2389\Labs\Lab06_1\Starter\xx\ where xx is either VB or CS.

► To change the project settings

You will start by changing some project settings so that your code strictly enforces data type conversions, and allows the debugging of SQL Server stored procedures called by the code.

1. Right-click the project name in the Solution Explorer and choose Properties.
2. *For Visual Basic projects only*. Select **Build** properties and switch **Option Strict** On. This will enforce the explicit conversion of data types.
3. Select **Configuration Properties**, **Debugging** and switch **SQL Server debugging** on.

► To build the data adapter for filling the employees table

1. Open the **MainForm** class in Designer view and drag a **SqlDataAdapter** from the Toolbox onto the form. This will run the Data Adapter Configuration Wizard.
2. Choose a data connection to the Northwind database on your local SQL Server.
3. Choose to **Use SQL statements** and type the following statement.

```
SELECT
    EmployeeID, LastName + ', ' + FirstName AS FullName
FROM
    Employees ORDER BY LastName, FirstName
```

4. Click the **Advanced Options** button and clear the **Generate Insert, Update and Delete statements** check box. This application will not allow changes to be made to the employees table.
5. Click **Finish**. The wizard will now create a data adapter, a connection and a command that will be used to populate the employees table in the data set.

6. Change the name of the new data adapter to **daEmployees**. Change the name of its associated **SelectCommand** to **cmSelectEmployees**. Change the name of the new connection to **cnNorthwind**.
7. Review the code written by the wizard.

► **To generate the custom data set schema and class**

1. Right-click the data adapter and choose **Generate Dataset**. Change the name to **NWDataSet**, and select the **Add the data set to the designer** check box. Click OK.

Note This will add a new XSD (data set schema) file to the project named **NWDataSet.xsd**. An associated class file will also be created, but by default it is hidden. Use the **Show All Files** button in the Solution Explorer's toolbar to toggle the display of hidden files.

2. Change the **(Name)** property of the data set named **NWDataSet1** to **dsNorthwind**.

► **To store the two application settings**

While the application is running, the two application settings can be held in memory using simple fields that can be added to the form class. When the application is not running, these settings will be stored with the data set in an XML file.

1. Declare two variables named **EmployeeID** and **ServerName** with appropriate data types:

```
' Visual Basic
Friend EmployeeID As System.Int32 = 0
Friend ServerName As System.String = "(local)"

// Visual C#
internal System.Int32 EmployeeID = 0;
internal System.String ServerName = "(local)";
```

2. Right-click the data set named **dsNorthwind** and choose **View Schema**. This will launch the XML Designer and allow you to change the schema.
3. Drag a new **element** from the **XML Schema** section of the Toolbox onto the Designer and name it **AppSettings**.
4. Add two sub-elements to **AppSettings** named **EmployeeID** and **ServerName**.
5. Change the data type of **EmployeeID** to **int**.
6. Save your changes and close the XSD file.

► To save the data set when the application closes

You will now add code to the MainForm Closing event, to save the application settings stored in the data set to an XML file.

1. Locate the code for the MainForm Closing event.
2. Write a line of code to clear any existing rows in the **AppSettings** table.

```
' Visual Basic
Me.dsNorthwind.AppSettings.Clear()

// Visual C#
this.dsNorthwind.AppSettings.Clear();
```

3. Write a line of code to add a new row to the **AppSettings** table using the values stored in the **EmployeeID** and **ServerName** fields:

```
' Visual Basic
Me.dsNorthwind.AppSettings.AddAppSettingsRow( _
   Me.EmployeeID, Me.ServerName)

// Visual C#
this.dsNorthwind.AppSettings.AddAppSettingsRow(
   this.EmployeeID, this.ServerName);

Write a line of code to call the AcceptChanges method to
accept the changes made to the AppSettings table:
' Visual Basic
Me.dsNorthwind.AppSettings.AcceptChanges()

// Visual C#
this.dsNorthwind.AppSettings.AcceptChanges();
```

4. Write a line of code to save the data set using the filename **OnTheRoad.xml** and the DiffGram format. This will ensure that changes to the data set are recorded as well as the original values.

```
' Visual Basic
Me.dsNorthwind.WriteXml( _
   "OnTheRoad.xml", XmlWriteMode.DiffGram)

// Visual C#
this.dsNorthwind.WriteXml(
   "OnTheRoad.xml", XmlWriteMode.DiffGram);
```

► To test the application settings code

1. Run and then immediately close the application. An XML file named **OnTheRoad.xml** should have been created in the same folder that contains the executable file.
2. Open the file and review its contents using Internet Explorer.

Exercise 3
Loading and Displaying Employee Information

In this exercise you will write code for the **Get from central database** menu item, to load employee information from the central database. Errors may occur as you attempt to fill the data set so you will work with a temporary data set, and then only if it is successfully created will you store the data set in the application.

Scenario

You will use the data adapter named **daEmployees** to fill the employees table in the data set. You will also display the employee information in a list box on the Logon form.

► **To start with the solution to the previous exercise**

- If you did not complete the previous exercise, open the solution **OnTheRoad** in the folder <install folder>\Labs\Lab06_1\Solution\Ex2\xx\ where xx is either VB or CS.

► **To fill the data set with employees**

1. In MainForm.vb, locate the mnuFill Click event handler.
2. Declare a data set named **tempNW** based on the **NWDataSet** schema and class.

```
' Visual Basic
Dim tempNW As New OnTheRoad.NWDataSet()

// Visual C#
OnTheRoad.NWDataSet tempNW = new OnTheRoad.NWDataSet();
```

3. Write an **If** statement to check the current state of the connection, and if it is not open, then try to open the connection.

```
' Visual Basic
If Me.cnNorthwind.State <> ConnectionState.Open Then
   Try ' to open the database connection
       Me.cnNorthwind.Open()

// Visual C#
if (this.cnNorthwind.State != ConnectionState.Open)
{
   try // to open the database connection
   {
       this.cnNorthwind.Open();
   }
```

4. If opening fails, catch the exception and display a warning message that suggests the user try changing the server name, and then exit the sub routine.

```
' Visual Basic
   Catch Xcp As System.Exception
       MessageBox.Show("Failed to connect because:" & _
           vbCrLf & Xcp.ToString() & vbCrLf & vbCrLf & _
           "Try a different server name.", _
           "Get from central database", _
           MessageBoxButtons.OK, MessageBoxIcon.Error)
       Exit Sub
   End Try
End If

// Visual C#
   catch (System.Exception Xcp)
   {
       MessageBox.Show("Failed to connect because:\n" +
           Xcp.ToString() +
           "\n\nTry a different server name.",
           "Get from central database",
           MessageBoxButtons.OK, MessageBoxIcon.Error);
       return;
   }
}
```

5. Write code to try to fill the employees table using the data adapter named **daEmployees**.

```
' Visual Basic
Try ' to fill the Employees DataTable
   Me.daEmployees.Fill(tempNW.Employees)

// Visual C#
try // to fill the Employees DataTable
{
   this.daEmployees.Fill(tempNW.Employees);
}
```

6. Catch any exceptions and display a warning message that informs the user the employee list failed to be retrieved.

```
' Visual Basic
Catch Xcp As System.Exception
   MessageBox.Show( _
       "Failed to retrieve employee list because:" & _
       vbCrLf & Xcp.ToString(), _
       "Get from central database", _
       MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

// Visual C#
catch (System.Exception Xcp)
{
   MessageBox.Show(
       "Failed to retrieve employee list because:\n" +
       Xcp.ToString(),
       "Get from central database",
       MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

7. After the code that instantiates the Logon form, write code to set its data properties so that the list box displays a list of employee names, but the bound value is the Employee ID.

```
' Visual Basic
frmLogon.lstEmployees.DataSource = tempNW.Employees
frmLogon.lstEmployees.DisplayMember = "FullName"
frmLogon.lstEmployees.ValueMember = "EmployeeID"

// Visual C#
frmLogon.lstEmployees.DataSource = tempNW.Employees;
frmLogon.lstEmployees.DisplayMember = "FullName";
frmLogon.lstEmployees.ValueMember = "EmployeeID";

Add code to highlight the current EmployeeID in the list.
' Visual Basic
frmLogon.lstEmployees.SelectedValue = Me.EmployeeID

// Visual C#
frmLogon.lstEmployees.SelectedValue = this.EmployeeID;
```

8. Inside the **If** statement that displays the Logon form, write code to change the currently stored employee ID to the selected value in the list box, and store the temporary data set in the dsNorthwind data set, and finally call a method named RefreshUI that you will complete next.

   ```
   ' Visual Basic
   Me.EmployeeID = CInt(frmLogon.lstEmployees.SelectedValue)
   Me.dsNorthwind = tempNW
   Me.RefreshUI()

   // Visual C#
   this.EmployeeID =
   CInt(frmLogon.lstEmployees.SelectedValue);
   this.dsNorthwind = tempNW;
   this.RefreshUI();
   ```

9. After the end of the **If** statement that displays the Logon form, close the connection.

   ```
   ' Visual Basic
   Me.cnNorthwind.Close()

   // Visual C#
   this.cnNorthwind.Close();
   ```

► **To refresh the user interface**

1. Locate the private procedure named **RefreshUI**.
2. Write a line of code to set the title bar of the main form to show the full name of the currently selected employee, and the name of the application. For example, the title bar might show: **Buchanen, Steven – On The Road**.

```
' Visual Basic
Me.Text = Me.dsNorthwind.Employees.Select( _
  "EmployeeID=" & Me.EmployeeID)(0)("FullName").ToString() _
  & " - " & Application.ProductName

// Visual C#
this.Text = this.dsNorthwind.Employees.Select(
  "EmployeeID=" + this.EmployeeID)[0]["FullName"].ToString()
  + " - " + Application.ProductName;
```

► **To test the code**

1. Run and test your application.
2. Choose the **File – Get from central database** menu item.
3. The **Get from central database** dialog box should appear, and display a list of employee names. Pick any employee name, and click **OK**.
4. The name you pick should appear in the title bar of the application, along with the name of the application.
5. Close the application.

Exercise 4
Specifying and Using a Different Server Name

In this exercise you will write code for the **Options** menu item, to allow the user to choose a different server. The application will use this server name when it needs to access the central database.

You will also extend the application start-up code. When the application is launched, it will try to load application settings and employee information from the XML file **OnTheRoad.xml**.

Scenario

You will modify the database connection string, to use the new server name entered by the user in the Options dialog box. You will also use the **ReadXml** method to read application settings and employee information into the DataSet at application start-up.

► To start with the solution to the previous exercise

- If you did not complete the previous exercise, open the solution **OnTheRoad** in the folder <install folder>\Labs\Lab06_1\Solution\Ex3\xx\ where xx is either VB or CS.

► To allow the server name to change

1. In MainForm.vb, locate the mnuOptions Click event hander.
2. Write a line of code to fill the current server name into the text box on the instance of the Options form.

```
' Visual Basic
frmOptions.txtServer.Text = Me.ServerName

// Visual C#
frmOptions.txtServer.Text = this.ServerName;
```

3. After the code that shows the form, write a line of code to retrieve the value in the text box and store it in the **ServerName** field.

```
' Visual Basic
Me.ServerName = frmOptions.txtServer.Text

// Visual C#
this.ServerName = frmOptions.txtServer.Text;
```

4. Write a line of code to retrieve the value in the **ServerName** field and use it to change the data source parameter in the **ConnectionString** property of the connection object.

```
' Visual Basic
Me.cnNorthwind.ConnectionString = _
   "data source=" & Me.ServerName & ";" & _
   "initial catalog=Northwind;" & _
   "integrated security=SSPI;" & _
   "persist security info=False;"

// Visual C#
this.cnNorthwind.ConnectionString =
   "data source=" + this.ServerName + ";" +
   "initial catalog=Northwind;" +
   "integrated security=SSPI;" +
   "persist security info=False;";
```

► **To fill a data set when the application starts**

1. Find the MainForm_Load procedure and add code to try to open an existing XML file named **OnTheRoad.xml**, and that uses the DiffGram format.

```
' Visual Basic
Try ' to open existing local cached DataSet
   Me.dsNorthwind.ReadXml( _
       "OnTheRoad.xml", XmlReadMode.DiffGram)

// Visual C#
try // to open existing local cached DataSet
{
   this.dsNorthwind.ReadXml(
       "OnTheRoad.xml", XmlReadMode.DiffGram);
```

2. If the file is found (and therefore an exception is not thrown), retrieve default values for the **EmployeeID** and **ServerName** fields.

```
' Visual Basic
Me.EmployeeID = _
  CInt(Me.dsNorthwind.AppSettings.Rows(0)("EmployeeID"))

Me.ServerName = _
  Me.dsNorthwind.AppSettings.Rows(0)("ServerName").ToString()

// Visual C#
this.EmployeeID = ConvertTo(
  this.dsNorthwind.AppSettings.Rows[0]["EmployeeID"], int);

this.ServerName =
this.dsNorthwind.AppSettings.Rows[0]["ServerName"].ToString();
```

3. Call the **RefreshUI** method to update the title bar of the form.

```
' Visual Basic
Me.RefreshUI()

// Visual C#
this.RefreshUI();
```

4. Write a **Catch** statement that uses an **If** statement to ask the user if they want to connect to the central database to create the data set and checks their response.

```
' Visual Basic
Catch
   If MessageBox.Show("An existing data set was not " & _
       "found or was corrupt. Do you want to connect " & _
       "to the central database to retrieve a new copy?", _
       "Warning!", MessageBoxButtons.YesNo, _
       MessageBoxIcon.Exclamation) = DialogResult.Yes Then

// Visual C#
catch
{
   if (MessageBox.Show("An existing data set was not " +
       "found or was corrupt. Do you want to connect " +
       "to the central database to retrieve a new copy?",
       "Warning!", MessageBoxButtons.YesNo,
       MessageBoxIcon.Exclamation) == DialogResult.Yes)
   {
```

5. If the user replies **Yes**, write code to try to open the connection and then call the **mnuFill_Click** procedure to simulate the user clicking the **Get from central database** menu item.

```
' Visual Basic
       Try ' to open the connection
           Me.cnNorthwind.Open()
           mnuFill_Click(sender, e)

// Visual C#
       try // to open the connection
       {
           this.cnNorthwind.Open();
           mnuFill_Click(sender, e);
       }
```

6. Write code to catch any exceptions, and if they occur, display a warning message and then exit the procedure.

```
' Visual Basic
      Catch Xcp As System.Exception
          MessageBox.Show("Failed to connect because:" & _
              vbCrLf & Xcp.ToString() & vbCrLf & vbCrLf & _
              "Use Tools, Options to change the name of " & _
              "the SQL Server you are trying to connect to.", _
              "Connect to central database", _
              MessageBoxButtons.OK, MessageBoxIcon.Error)
          Exit Sub
      End Try
  End If
End Try

// Visual C#
      catch (System.Exception Xcp)
      {
          MessageBox.Show("Failed to connect because:\n" +
              Xcp.ToString() +
              "\n\nUse Tools, Options to change the name of " +
              "the SQL Server you are trying to connect to.",
              "Connect to central database",
              MessageBoxButtons.OK, MessageBoxIcon.Error);
          return;
      }
  }
}
```

► **To test the code**

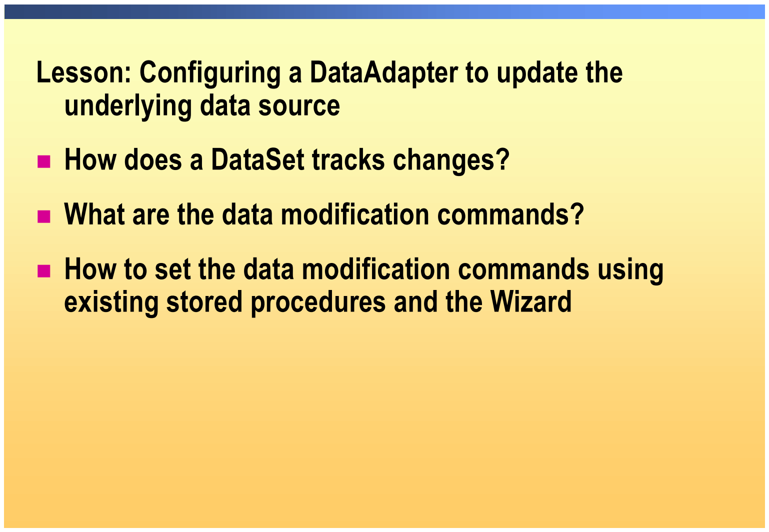
1. In Windows Explorer, delete the XML file named **OnTheRoad.xml** if it exists.
2. Run and test your application. You should get a warning message – choose **Yes** to connect to your local database and retrieve the list of employees.
3. Pick any employee name and click **OK**. The name you pick should appear in the title bar of the application along with the name of the application.
4. Close the application then rerun it and see if it correctly opens the XML and remembers the employee you picked.
5. Try using the **Tools – Options…** menu item to change the server name to **London**, which is the instructor’s computer name.
6. Close the application then rerun it and see if it correctly opens the XML and loads the employees from the instructor's computer.

7. Open **OnTheRoad.xml** and review its contents. It should look something like this.

   ```
   <?xml version="1.0" standalone="yes"?>
   <NWDataSet xmlns="http://www.tempuri.org/NWDataSet.xsd">
     <Employees>
       <EmployeeID>5</EmployeeID>
       <FullName>Buchanan, Steven</FullName>
     </Employees>
     ...
     <Employees>
       <EmployeeID>6</EmployeeID>
       <FullName>Suyama, Michael</FullName>
     </Employees>
     <AppSettings>
       <EmployeeID>7</EmployeeID>
       <ServerName>London</ServerName>
     </AppSettings>
   </NWDataSet>
   ```

8. Close the application.

Lesson: Configuring a DataAdapter to Update the Underlying Data Source



Lesson: Configuring a DataAdapter to update the underlying data source

- How does a DataSet tracks changes?
- What are the data modification commands?
- How to set the data modification commands using existing stored procedures and the Wizard

Introduction

Although a DataSet is typically a local copy of data from a remote data source, you can create and update data in a DataSet and then use a DataAdapter to update the underlying data source.

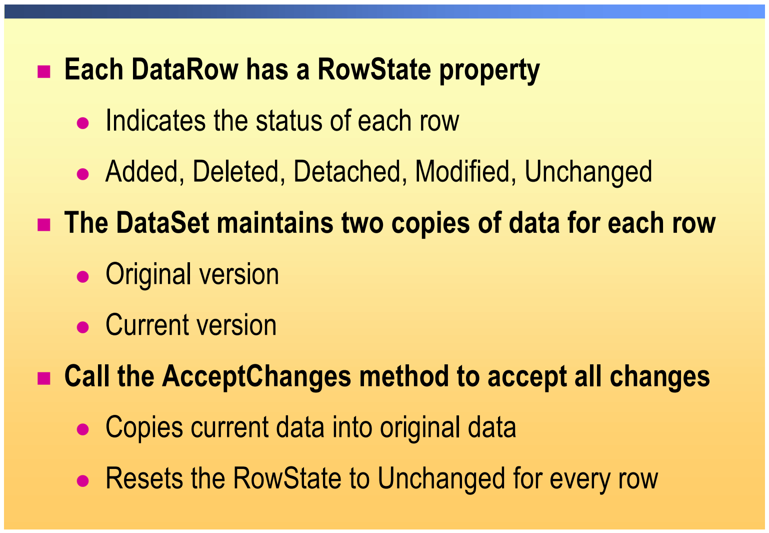
Lesson Objective

Configure a DataAdapter to update the underlying data source.

After completing this lesson, you will be able to:

- Explain how a DataSet tracks changes
- Use the data modification commands
- Set the data modification commands using existing stored procedures with parameters and the Wizard

How Does the DataSet Track Changes?



- **Each DataRow has a RowState property**
  - Indicates the status of each row
  - Added, Deleted, Detached, Modified, Unchanged
- **The DataSet maintains two copies of data for each row**
  - Original version
  - Current version
- **Call the AcceptChanges method to accept all changes**
  - Copies current data into original data
  - Resets the RowState to Unchanged for every row

Introduction

Each DataRow object in a DataTable has a RowState property. The RowState property is read-only, and indicates whether the row has been modified, inserted, or deleted from the DataSet since the DataSet was first populated.

The DataSet maintains two sets of data for each row – the current data and the original data. You can specify which version of data you want when you use the DataSet.

The DataSet also provides an AcceptChanges method. Call this method to accept all the changes made to the DataSet so far, and set the DataSet's original state to the current state.

Definition of the RowState property

The DataSet maintains the current status of each row in the DataSet. Whenever a DataRow is changed in any way, the DataSet sets the RowState property to indicate whether the row has been modified, inserted, or deleted. You can check this property in your code, to examine the status of each row in the DataSet.

Definition of the RowState property

The DataSet maintains the current status of each row in the DataSet. Whenever a DataRow is changed in any way, the DataSet sets the RowState property to indicate whether the row has been modified, inserted, or deleted. You can check this property in your code, to examine the status of each row in the DataSet.

The **RowState** property has one of the following enumeration values:

RowState property value	Description
DataRowState.Added	The row has been added to the DataSet since the AcceptChanges method was called.
DataRowState.Deleted	The row has been deleted from the DataSet since the AcceptChanges method was called.
DataRowState.Detached	The row has been created, but it has not yet been added to a DataRowCollection in a DataSet.
DataRowState.Modified	The row has been modified since the AcceptChanges method was called.
DataRowState.Unchanged	The row has not changed since the AcceptChanges method was called.

Definition of the Current and Original data versions

The DataSet maintains two copies of data for each row – the Current data and the Original data. This enables you to see exactly how each row has changed in the DataSet. When you access data in a DataRow, you can specify a **DataRowVersion** parameter to indicate which version of the data you want:

DataRowVersion value	Description
DataRowVersion.Curent	The current version of data in the DataRow. This is the default data version if you do not specify an explicit version.
DataRowVersion.Original	The original version of data in the DataRow, when the AcceptChanges was last called.

Example of using current and original data in a row

The following example iterates through the rows in a DataSet table, and displays the **RowState** for each row.

If the RowState is **DataRowState.Added** or **DataRowState.Unchanged** the current version of the row data is displayed. If the RowState is **DataRowState.Deleted**, the original version of the row data is displayed. If the RowState is **DataRowState.Modified**, the original and current versions of the row data are displayed to show how they differ:

```
' Visual Basic
Dim row As DataRow
For Each row In Me.dsCustomers.Customers.Rows

  Dim msg As String
  If row.RowState = DataRowState.Added Or _
     row.RowState = DataRowState.Unchanged Then

    msg = "Current data:" & vbCrLf & _
      row("CompanyName", DataRowVersion.Current) & ", " & _
      row("ContactName", DataRowVersion.Current)

  ElseIf row.RowState = DataRowState.Deleted Then

    msg = "Original data:" & vbCrLf & _
      row("CompanyName", DataRowVersion.Original) & ", " & _
      row("ContactName", DataRowVersion.Original)

  ElseIf row.RowState = DataRowState.Modified Then

    msg = "Original data:" & vbCrLf & _
      row("CompanyName", DataRowVersion.Original) & ", " & _
      row("ContactName", DataRowVersion.Original) & vbCrLf

    msg = msg & "Current data:" & _
      row("CompanyName", DataRowVersion.Current) & ", " & _
      row("ContactName", DataRowVersion.Current)

  End If
  MessageBox.Show(msg, "RowState: " & row.RowState.ToString())
Next

// Visual C#
foreach (DataRow row in this.dsCustomers.Customers.Rows)
{
  String msg = "";
  if (row.RowState == DataRowState.Added ||
      row.RowState == DataRowState.Unchanged)
  {
    msg = "Current data:\n" +
      row["CompanyName", DataRowVersion.Current] + ", " +
      row["ContactName", DataRowVersion.Current];
  }
  else if (row.RowState == DataRowState.Deleted)
  {
    msg = "Original data:\n" +
      row["CompanyName", DataRowVersion.Original] + ", " +
      row["ContactName", DataRowVersion.Original];
  }
```

(Code continued on next page.)

```
  else if (row.RowState == DataRowState.Modified)
  {
    msg = "Original data:\n" +
      row["CompanyName", DataRowVersion.Original] + ", " +
      row["ContactName", DataRowVersion.Original] + "\n";

    msg = msg + "Current data:\n" +
      row["CompanyName", DataRowVersion.Current] + ", " +
      row["ContactName", DataRowVersion.Current];
  }
  MessageBox.Show(msg, "RowState: " + row.RowState);
}
```

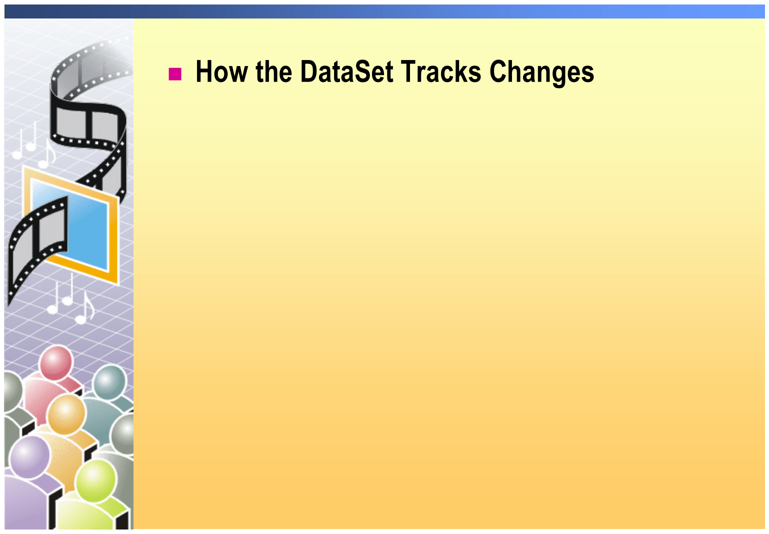
Practice

If you do not explicitly specify a **DataRowVersion** parameter when you access a column in a **DataRow**, do you get the "current" data or the "original" data for the row?

What happens if you try to display the "original" row data for a new added row?

What happens if you try to display the "current" row data for a deleted row?

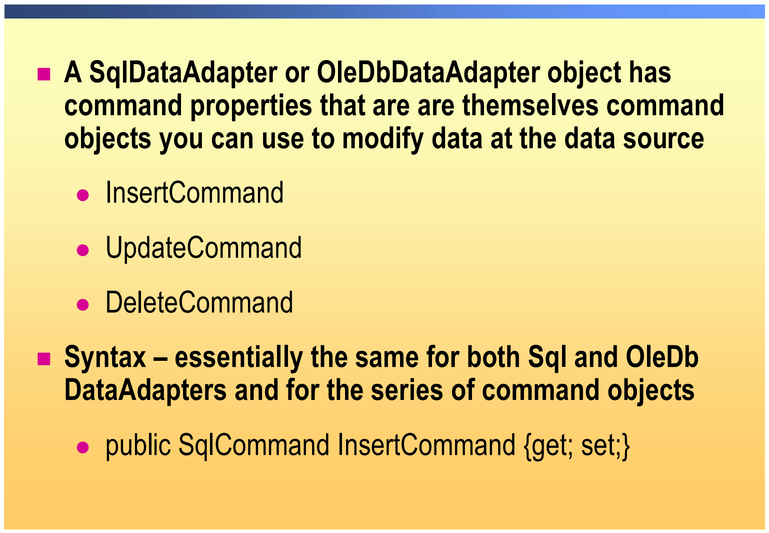
Multimedia: How the DataSet Tracks Changes



- How the DataSet Tracks Changes

This animation shows how a DataSet uses the RowState property to identify the state of each row. The animation also shows how the DataSet maintains two versions of data for each row – the Current version and the Original version.

What are the Data Modification Commands?



- **A SqlDataAdapter or OleDbDataAdapter object has command properties that are are themselves command objects you can use to modify data at the data source**
  - InsertCommand
  - UpdateCommand
  - DeleteCommand
- **Syntax – essentially the same for both Sql and OleDb DataAdapters and for the series of command objects**
  - public SqlCommand InsertCommand {get; set;}

Introduction

A DataAdapter uses command objects to modify data at the data source. The DataAdapter uses these commands to save changes in a DataSet back to the underlying data source.

Data modification commands

The following table describes the data modification commands, which are used by the DataAdapter.

Command	Description
InsertCommand	Used during call to Update method of a DataAdapter to insert records into the data source that correspond to new rows in the DataSet.
UpdateCommand	Used during call to Update method to update records in the data source that correspond to modified rows in the DataSet.
DeleteCommand	Used during call to Update method to delete records in the data source that correspond to deleted rows in the DataSet.

Example of setting the InsertCommand property

The following example programmatically sets the **InsertCommand** property for a DataAdapter. The command inserts a row into a simplified Customers table, which contains columns named **CustomerID** and **CustomerName**. The command requires two **SqlParameter** objects, to set the column values in the new row:

```
' Visual Basic
Dim cmInsert As New SqlCommand( _
   "INSERT INTO Customers VALUES (@ID, @Name)", _
   cnNorthwind)
cmInsert.Parameters.Add(New SqlParameter("@ID", _
   SqlDbType.NChar, 5, ParameterDirection.Input, False, _
   0, 0, "CustomerID", DataRowVersion.Current, Nothing))
cmInsert.Parameters.Add(New SqlParameter("@Name", _
   SqlDbType.NVarChar, 40, ParameterDirection.Input, False, _
   0, 0, "CompanyName", DataRowVersion.Current, Nothing))
daCustomers.InsertCommand = cmInsert

// Visual C#
SqlCommand cmInsert = new SqlCommand(
    "INSERT INTO Customers VALUES (@ID, @Name)",
    cnNorthwind);
cmInsert.Parameters.Add(new SqlParameter("@ID",
    SqlDbType.NChar, 5, ParameterDirection.Input, false,
    0, 0, "CustomerID", DataRowVersion.Current, null));
cmInsert.Parameters.Add(new SqlParameter("@Name",
    SqlDbType.NVarChar, 40, ParameterDirection.Input, false,
    0, 0, "CompanyName", DataRowVersion.Current, null));
daCustomers.InsertCommand = cmInsert;
```

Example of setting the UpdateCommand property

The following example sets the **UpdateCommand** property for a DataAdapter, to update a row in the simplified Customers table. The command requires three **SqlParameter** objects: the new **CustomerID**, the new **CustomerName**, and the original **CustomerID** (to locate the customer record in the data source):

```
' Visual Basic
Dim cmUpdate As New SqlCommand( _
    "UPDATE Customers SET CustomerID = @ID, " & _
    "CompanyName = @Name WHERE (CustomerID = @OrigID)", _
    cnNorthwind)
cmUpdate.Parameters.Add(New SqlParameter("@ID", _
    SqlDbType.NChar, 5, ParameterDirection.Input, False, _
    0, 0, "CustomerID", DataRowVersion.Current, Nothing))
cmUpdate.Parameters.Add(New SqlParameter("@Name", _
    SqlDbType.NVarChar, 40, ParameterDirection.Input, False, _
    0, 0, "CompanyName", DataRowVersion.Current, Nothing))
cmUpdate.Parameters.Add(New SqlParameter("@OrigID", _
    SqlDbType.NChar, 5, ParameterDirection.Input, False, _
    0, 0, "CustomerID", DataRowVersion.Original, Nothing))
daCustomers.UpdateCommand = cmUpdate
```

```
// Visual C#
SqlCommand cmUpdate = new SqlCommand(
    "UPDATE Customers SET CustomerID = @ID, " +
    "CompanyName = @Name WHERE (CustomerID = @OrigID)",
    cnNorthwind);
cmUpdate.Parameters.Add(new SqlParameter("@ID",
    SqlDbType.NChar, 5, ParameterDirection.Input, false,
    0, 0, "CustomerID", DataRowVersion.Current, null));
cmUpdate.Parameters.Add(new SqlParameter("@Name",
    SqlDbType.NVarChar, 40, ParameterDirection.Input, false,
    0, 0, "CompanyName", DataRowVersion.Current, null));
cmUpdate.Parameters.Add(new SqlParameter("@OrigID",
    SqlDbType.NChar, 5, ParameterDirection.Input, false,
    0, 0, "CustomerID", DataRowVersion.Original, null));
daCustomers.UpdateCommand = cmUpdate;
```

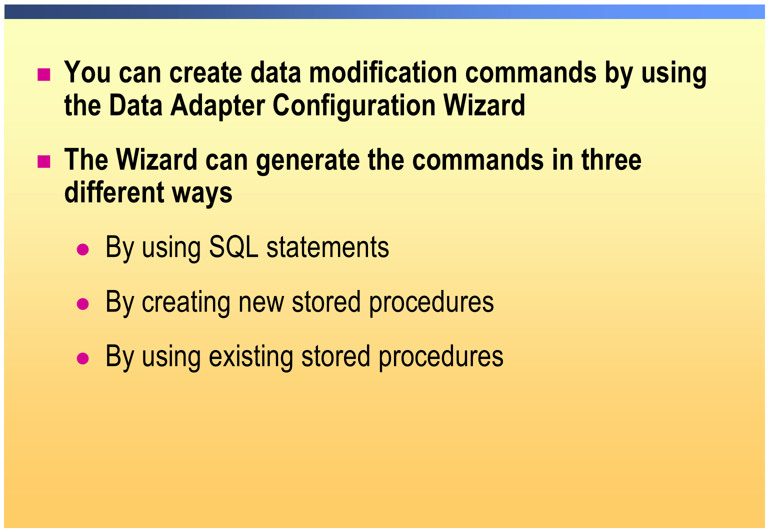
Example of setting the DeleteCommand property

The following example sets the **DeleteCommand** property for a DataAdapter, to delete a row in the simplified Customers table. The command requires one **SqlParameter** object, to specify the **CustomerID** of the row to be deleted:

```
' Visual Basic
cmDelete = New SqlCommand( _
    "DELETE FROM Customers WHERE (CustomerID = @ID)", _
    cnNorthwind)
cmDelete.Parameters.Add(New SqlParameter("@ID", _
    SqlDbType.NChar, 5, ParameterDirection.Input, False, _
    0, 0, "CustomerID", DataRowVersion.Original, Nothing))
daCustomers.DeleteCommand = cmDelete
```

```
// Visual C#
SqlCommand cmDelete = new SqlCommand(
    "DELETE FROM Customers WHERE (CustomerID = @ID)",
    cnNorthwind);
cmDelete.Parameters.Add(new SqlParameter("@ID",
    SqlDbType.NChar, 5, ParameterDirection.Input, false,
    0, 0, "CustomerID", DataRowVersion.Original, null));
daCustomers.DeleteCommand = cmDelete;
```

How to Set the Data Modification Commands using Existing Stored Procedures and the Wizard



- **You can create data modification commands by using the Data Adapter Configuration Wizard**
- **The Wizard can generate the commands in three different ways**
  - By using SQL statements
  - By creating new stored procedures
  - By using existing stored procedures

Introduction

You can create data modification commands by using the Data Adapter Configuration Wizard. The Wizard can generate the commands in three different ways:

- By using SQL statements
- By creating new stored procedures
- By using existing stored procedures

How to create data modification commands by using the Wizard

To create data modification commands by using the Data Adapter Configuration Wizard, follows these steps:

1. Drag and drop a **SqlDataAdapter** control or **OleDbDataAdapter** control from the toolbox onto your form.
2. In the **Welcome** screen for the Data Adapter Configuration Wizard, click **Next**.
3. In the **Choose Your Data Connection** screen, select an existing connection (or click **New Connection** and specify a new connection, if necessary).
4. Click **Next**, to move to the **Choose a Query Type** screen.
5. If you want to use SQL statements for the data modification commands, follow these steps:
   a. Choose **Use SQL statements**, and click **Next**.
   b. In the **Generate the SQL statements** screen, type the SQL statement for the **SelectCommand**. Click **Advanced Options**, and ensure **Generate Insert, Update, and Delete statements** is checked. Click **OK**.
   c. In the **Generate the SQL statements screen**, click **Next**.
   d. In the **View Wizard Results** screen, click **Finish**.
   e. Examine the generated code in your application, to see how the Wizard created the data modification commands.

6. If you want to create new stored procedures for the data modification commands, follow these steps:
   a. Choose **Create new stored procedures**, and click **Next**.
   b. In the **Generate the SQL statements** screen, type the SQL statement for the **SelectCommand**. Click **Advanced Options**, and ensure **Generate Insert, Update, and Delete statements** is checked. Click **OK**.
   c. In the **Generate the SQL statements** screen, click **Next**.
   d. In the **Create the Stored Procedures** screen, enter names for the new stored procedures. Click **Next**.
   e. In the **View Wizard Results** screen, click **Finish**.
   f. Examine the stored procedures in the Server Explorer.
   g. Also examine the generated code in your application, to see how the Wizard created the data modification commands.
7. If you want to use existing stored procedures for the data modification commands, follow these steps:
   a. Choose **Use existing stored procedures**, and click **Next**.
   b. In the **Bind Commands to Existing Stored Procedures** screen, choose existing stored procedures for the **Select**, **Insert**, **Update**, and **Delete** commands. Click **Next**.
   c. In the **View Wizard Results** screen, click **Finish**.
   d. Examine the generated code in your application, to see how the Wizard created the data modification commands.

Practice

Northwind Traders needs to allow users to make changes to the product catalog that is published on the company's web site. In this practice, you will create a new Windows Application that uses a **SqlDataAdapter** to query and modify the Products table.

You will use the Data Adapter Configuration Wizard to generate four new stored procedures to achieve this task. The data adapter will call these stored procedures in its data modification commands (**SelectCommand**, **InsertCommand**, **UpdateCommand**, and **DeleteCommand**).

1. Create a new Windows Application solution named **CatalogEditor** at the following location.

   <install folder>\Practices\Mod06_1\

2. Drag and drop a **SqlDataAdapter** control from the toolbox onto the form. Use the Data Adapter Configuration Wizard to set the following properties:

Property	Value
Server Name	(local)
Log On	Use Windows NT Integrated security
Database	Northwind
Query Type	Create new stored procedures
Load Statement	SELECT * FROM Products
Advanced Options	All options enabled

Property	Value
Stored Procedure Names	usp_SelectProducts usp_InsertProducts usp_UpdateProducts usp_DeleteProducts Let the Wizard create them in the database for you.

3. Select the new data adapter named **SqlDataAdapter1**. Use the Property Window to set the following properties:

Property	Value
(Name)	daProducts
DeleteCommand	
(Name)	cmDeleteProducts
InsertCommand	
(Name)	cmInsertProducts
SelectCommand	
(Name)	cmSelectProducts
UpdateCommand	
(Name)	cmUpdateProducts

4. In Server Explorer, examine the four new stored procedures in the Northwind database. The stored procedures are named **usp_SelectProducts**, **usp_InsertProducts**, **usp_UpdateProducts**, and **usp_DeleteProducts**. Note the following points:

   usp_SelectProducts returns all the columns in the Products table.

   usp_InsertProducts receives a number of parameters, which hold the values for a new product. The stored procedure inserts these values into a new row. The stored procedure returns a record set containing the new row, using the clause **WHERE (ProductID = @@IDENTITY)** to obtain this new row.

   usp_UpdateProducts receives parameters that indicate the new and original values of a particular row. The new values are used to update the data in the row. The original values are used to ensure that the row has not been changed by another application or user, since it was fetched by your application. This prevents conflicting row updates in a disconnected architecture.

   usp_DeleteProducts receives parameters that indicate the original values of the row to be deleted. The stored procedure ensures that the row has not been changed by another application or user, since it was fetched by your application. This prevents conflicting row deletions in a disconnected architecture.

5. In the Code View window, examine the code that has been generated by the Data Adapter Configuration Wizard. Note the following points:

   SqlDataAdapter1 is assigned four data modification commands. The **SelectCommand** property is assigned **cmSelectProducts**; the **InsertCommand** property is assigned **cmInsertProducts**; and so on.

   cmSelectProducts is initialized to call the **usp_SelectProducts** stored procedure. A parameter is added to the command, to receive the return value from the stored procedure.

   cmInsertProducts is initialized to call **usp_InsertProducts**. Several parameters are added to the command, using the current versions of data in the row (the **System.Data.DataRowVersion.Current** flag is used for each parameter).

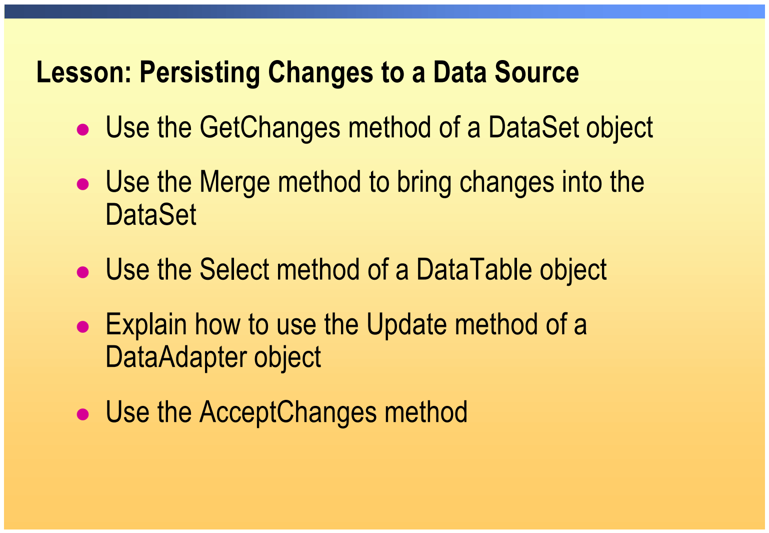
   cmUpdateProducts is initialized to call **usp_UpdateProducts**. Parameters are added for the new values in the row (using **System.Data.DataRowVersion.Current**), and for the original values (using **System.Data.DataRowVersion.Original**).

   cmDeleteProducts is initialized to call **usp_DeleteProducts**. Parameters are added for the original values in the row, using the flag **System.Data.DataRowVersion.Original**.

6. Save all the files in your application.

The solution for this practice is located at <install folder>\Practices\Mod06_1\Lesson3\CatalogEditor\

Lesson: Persisting Changes to a Data Source



Lesson: Persisting Changes to a Data Source

- Use the GetChanges method of a DataSet object
- Use the Merge method to bring changes into the DataSet
- Use the Select method of a DataTable object
- Explain how to use the Update method of a DataAdapter object
- Use the AcceptChanges method

Introduction

When you have created a DataSet in a typical multiple-tier implementation, made changes, and are ready to persist changes to a data source, the steps are the following:

1. Invoke the **GetChanges** method to create a second **DataSet** that features only the changes to the data.
2. Invoke the **Merge** method to merge the changes from the second **DataSet** into the first.
3. Call the **Update** method of the **SqlDataAdapter** (or **OleDbDataAdapter**) and pass the merged **DataSet** as an argument.
4. Invoke the **AcceptChanges** method on the **DataSet** to persist changes. Alternatively, invoke **RejectChange** to cancel the changes.

Lesson objectives

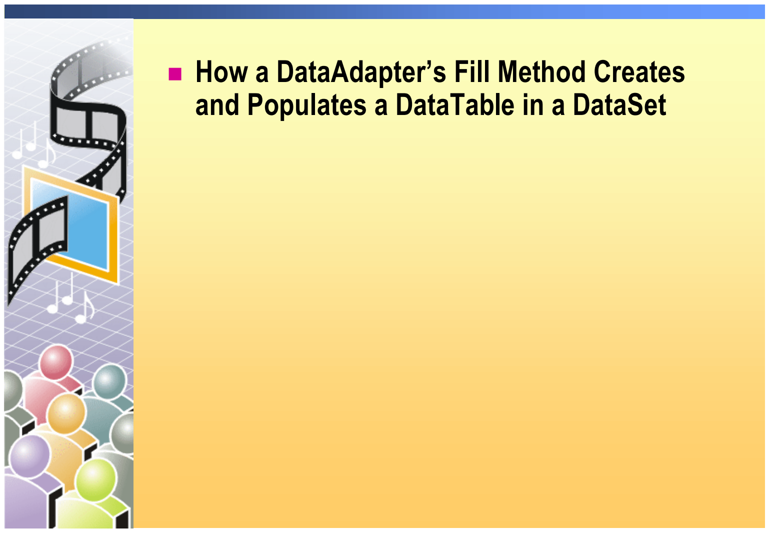
After completing this lesson, you will be able to persist changes to a data source and be able to:

- Use the **GetChanges** method of a **DataAdapter** object
- Use the **Select** method of a **DataAdapter** object
- Use the **Update** method of a **DataAdapter** object
- Use the **Merge** method to changes into the DataSet
- Use the **AcceptChanges** method

Overloading a method

Many of the methods that you use on a DataAdapter object allow different combinations of parameters and datatypes for a method. The ability to create different versions of a method is called *overloading*. The methods you will learn about in this lesson all have the ability to be overloaded.

MultiMedia: How a DataAdapter's Fill Method Creates and Populates a DataTable in a DataSet

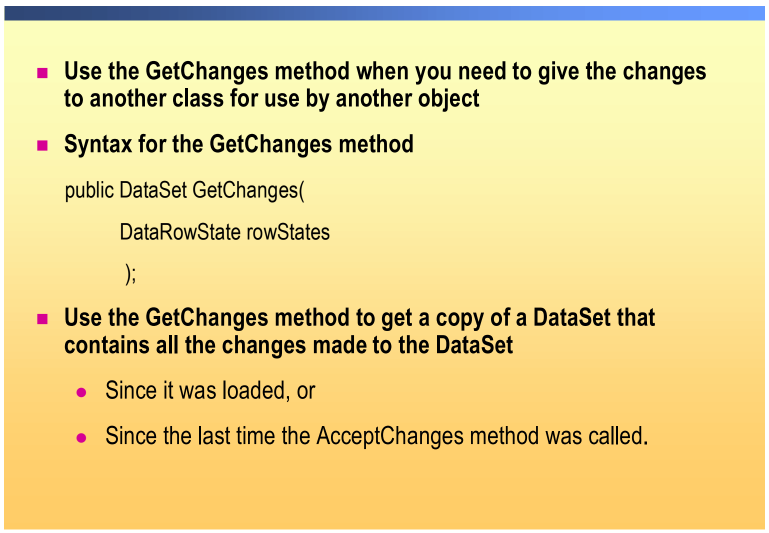


- How a DataAdapter's Fill Method Creates and Populates a DataTable in a DataSet

Introduction

A DataAdapter object is the bridge between a DataSet and a data source. A DataSet contains one or more DataTables, each of which contains DataRows. This animation shows how the Fill method of a DataAdapter object both creates and populates a DataTable.

When to use the GetChanges Method of a DataSet Object



- **Use the GetChanges method when you need to give the changes to another class for use by another object**
- **Syntax for the GetChanges method**

  public DataSet GetChanges(

  DataRowState rowStates

  );
- **Use the GetChanges method to get a copy of a DataSet that contains all the changes made to the DataSet**
  - Since it was loaded, or
  - Since the last time the AcceptChanges method was called.

Introduction

When you work in a disconnected environment, you can make changes to data in a DataSet and then transmit those changes to a data source. You use the **GetChanges** method of a DataSet object to produce a new DataSet object that contains a copy of only the changed rows in the original DataSet. You then can merge the new copy back into the original **DataSet**.

Syntax

The following is the Visual C# Syntax for the **GetChanges** method of a **DataAdapter** object.

```
public DataSet GetChanges(
   DataRowState rowStates
);
```

Use the *rowStates* argument to specify the type of changes the new object should include. You can create sub-subsets with only changes of a certain type, e.g. deleted rows.

When to use GetChanges

You would use GetChanges when you need to give the changes to another class for use by another object

While updating, the order in which inserts and deletes are performed is important for parent/child related tables, such as Customers and Orders. When inserting a new order for a new customer, the customer (parent) record must be inserted before the order (child) record. But when deleting a customer, child orders must be deleted before the customer (parent) record.

Example of getting changes in a dataset

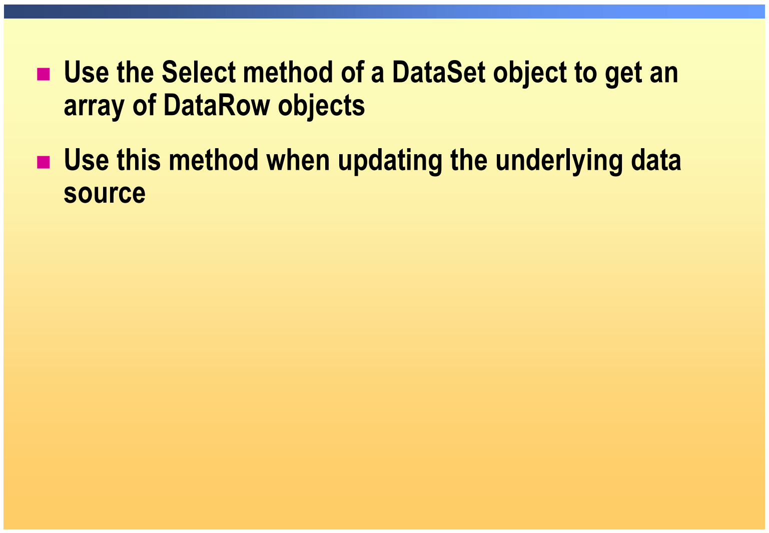
The following example tests a DataSet named **dsCustomers**, to see if it has any modified rows. If it does, the modified rows are copied to a temporary DataSet named **dsTemp**. The modified rows are displayed in a DataGrid.

```
' Visual Basic
If dsCustomers.HasChanges(DataRowState.Modified) Then
  Dim dsTemp As DataSet
  dsTemp = dsCustomers.GetChanges(DataRowState.Modified)
  DataGrid1.DataSource = dsTemp.Tables(0).DefaultView
End If

// Visual C#
if (dsCustomers.HasChanges(DataRowState.Modified))
{
  DataSet dsTemp;
  dsTemp = dsCustomers.GetChanges(DataRowState.Modified);
  dataGrid1.DataSource = dsTemp.Tables[0].DefaultView;
}
```

Practice

Modify the code in the previous example, so that it gets the deleted rows rather than the modified rows from the DataSet, **dsCustomers**. How can you display the deleted rows in a DataGrid control?

When to use the Select method?



- Use the Select method of a DataSet object to get an array of DataRow objects
- Use this method when updating the underlying data source

Introduction

The **Select** method of a DataSet object gets an array of **DataRow** objects. This method creates a set of pointers to rows within the original DataSet, not actually copying anything, but rather just pointing to the changes, so this is very efficient. You use this method when updating the underlying data source.

Note that you can create sub-subsets with only changes of a certain type, such as deleted rows.

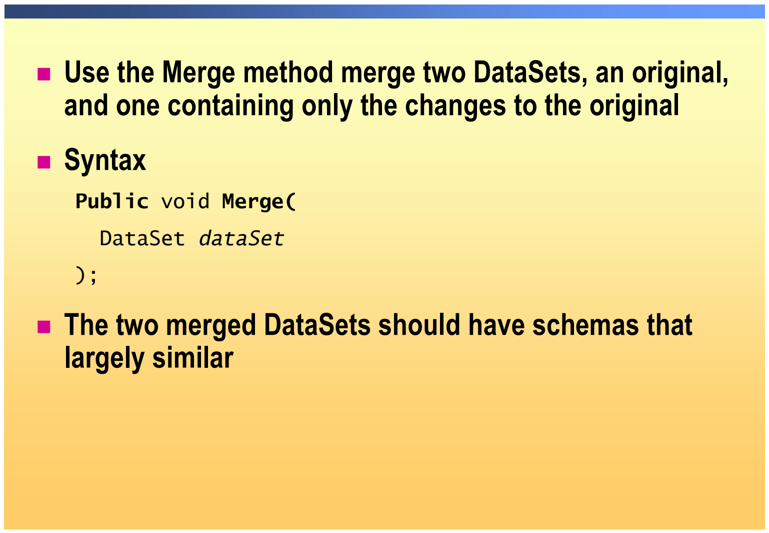
Example of selecting modified rows in a dataset

The following example selects rows from the **Customers** table in the **dsCustomers** DataSet. The **Select** method gets all the deleted customers whose **City** is London. The example loops through these customers, and displays the original **CompanyName** of each customer:

```
' Visual Basic
Dim strFilter As New String("City='London'")
Dim strSort As New String("CompanyName ASC")

Dim selRows As DataRow()
selRows = dsCustomers.Customers.Select( _
  strFilter, strSort, DataViewRowState.Deleted)

Dim row As DataRow
For Each row In selRows
  MessageBox.Show( _
    "Company name: " & _
     row("CompanyName", DataRowVersion.Original), _
    "Deleted company in London")
Next
```

```
// Visual C#
string strFilter = "City='London'";
string strSort = "CompanyName ASC";

DataRow[] selRows;
selRows = dsCustomers.Customers.Select(
  strFilter, strSort, System.Data.DataViewRowState.Deleted);

foreach (DataRow row in selRows)
{
  MessageBox.Show(
    "Company name: " +
     row["CompanyName", DataRowVersion.Original],
    "Deleted company in London");
}
```

How to Merge Changes into the DataSet



- Use the Merge method merge two DataSets, an original, and one containing only the changes to the original
- Syntax

```
Public void Merge(
  DataSet dataSet
);
```

- The two merged DataSets should have schemas that largely similar

Introduction

When you make changes to a DataSet, you typically create a new DataSet that contains only the changes you made to the original DataSet.

Definition of the Merge method

The **Merge** method of a DataSet object merges the contents of the DataSet to which the method is applied, with a second DataSet that typically contains only the changes to the original DataSet. Like other methods that deal with changes to a DataSet, the Merge method can be overloaded.

Syntax

The following is the C# syntax for the Merge method, where *dataSe*t is the DataSet whose data and schema will be merged.

```
public void Merge(
  DataSet dataSet
);
```

When to use the Merge method

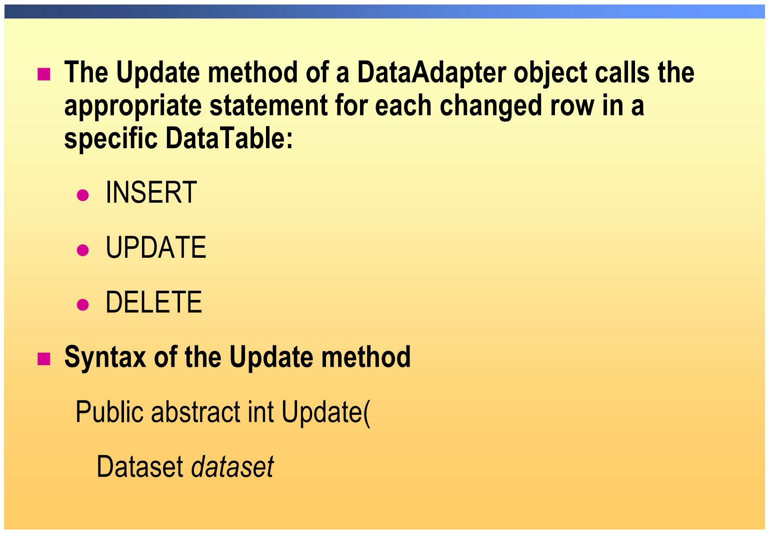
You use the **Merge** method to merge two DataSet objects that have largely similar schemas. You typically use a Merge on a client application to incorporate the latest changes from a data source into an existing **DataSet**. This allows the client application to have a refreshed **DataSet** with the latest data from the data source.

The **Merge** method is typically called at the end of a series of procedures that involve validating changes, reconciling errors, updating the data source with the changes, and finally refreshing the existing **DataSet**.

Example of Using the Merge Method

TBS

How to Update a Data Source by Using a DataSet



- **The Update method of a DataAdapter object calls the appropriate statement for each changed row in a specific DataTable:**
  - INSERT
  - UPDATE
  - DELETE
- **Syntax of the Update method**

  Public abstract int Update(

  Dataset *dataset*

Introduction

The Update method of a DataAdapter object calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet from a DataTable named "Table".

The Update method of a DataAdapter object is distinct and different from the UpdateCommand property of a DataAdapter object, which gets or sets a SQL statement or OleDBbCommand that updates records in the data source.

Syntax of the Update method

The C# syntax for the Update method of the DataAdapter class is the following:

```
Public abstract int Update(
   Dataset dataset
```

In this syntax, *dataSet* is the DataSet that is used to update the data source.

How to call the Update Method

When an application calls the **Update** method, the DataAdapter examines the RowState property, and executes the required INSERT, UPDATE, or DELETE statements based on the order of the indexes configured in the DataSet. For example, **Update** might execute a DELETE statement, followed by an INSERT statement, and then another DELETE statement, due to the ordering of the rows in the DataTable Data. An application can call the GetChanges method in situations where you must control the sequence of statement types (for example, INSERTs before UPDATEs). For more information, see Updating the Database with a DataAdapter and the DataSet.

If INSERT, UPDATE, or DELETE statements have not been specified, the **Update** method generates an exception. However, you can create a SqlCommandBuilder or OleDbCommandBuilder object to automatically generate SQL statements for single-table updates if you set the **SelectCommand** property of a .NET data provider. Then, any additional SQL statements that you do not set are generated by the CommandBuilder. This generation logic requires key column information to be present in the **DataSet**. For more information see Automatically Generated Commands.

The **Update** method retrieves rows from the table listed in the first mapping before performing an update. The **Update** then refreshes the row using the value of the UpdatedRowSource property. Any additional rows returned are ignored.

After any data is loaded back into the **DataSet**, the OnRowUpdated event is raised, allowing the user to inspect the reconciled **DataSet** row and any output parameters returned by the command. After a row updates successfully, the changes to that row are accepted.

Example of updating a data source

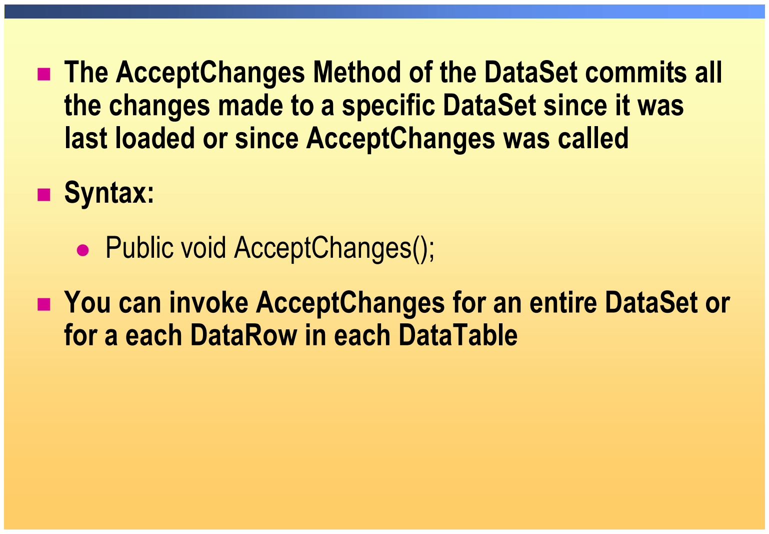
The following example shows how to use the **Update** method to update a data source. The example uses a DataSet named **dsCustomerOrders**, which has two tables named **Customers** and **Orders**. The **Customers** table is initially filled by the **daCustomers** DataAdapter, and the **Orders** table is filled by the **daOrders** DataAdapter.

The objective is to allow the user to delete customers, and all the orders placed by those customers. The orders must be deleted first, to avoid foreign key constraint errors when the customers are deleted.

To achieve this effect, the **GetChanges** method is called to get the deleted rows in the **Orders** table. These rows are deleted first. The **GetChanges** method is then called a second time, to get the deleted rows in the **Customers** table. These rows can now be safely deleted with any errors:

```
' Visual Basic
' Fill the Customers and Orders tables initially
daCustomers.Fill(dsCustomerOrders.Customers)
daOrders.Fill(dsCustomerOrders.Orders)
DataGrid1.DataSource = dsCustomerOrders.Customers.DefaultView
…
' Update the data source with any changes
Dim deletedOrders As DataTable = _
  dsCustomerOrders.Orders.GetChanges(DataRowState.Deleted)
daOrders.Update(deletedOrders)

Dim deletedCustomers As DataTable = _
  dsCustomerOrders.Customers.GetChanges(DataRowState.Deleted)
daCustomers.Update(deletedCustomers)

// Visual C#
// Fill the Customers and Orders tables initially
daCustomers.Fill(dsCustomerOrders.Customers);
daOrders.Fill(dsCustomerOrders.Orders);
dataGrid1.DataSource = dsCustomerOrders.Customers.DefaultView;
…
// Update the data source with any changes
DataTable deletedOrders =
  dsCustomerOrders.Orders.GetChanges(DataRowState.Deleted);
daOrders.Update(deletedOrders);

DataTable deletedCustomers =
  dsCustomerOrders.Customers.GetChanges(DataRowState.Deleted);
daCustomers.Update(deletedCustomers)
```

How to Accept Changes into the DataSet



- **The AcceptChanges Method of the DataSet commits all the changes made to a specific DataSet since it was last loaded or since AcceptChanges was called**
- **Syntax:**
  - Public void AcceptChanges();
- **You can invoke AcceptChanges for an entire DataSet or for a each DataRow in each DataTable**

Introduction

When you make changes to a DataSet, you typically create a new DataSet that contains only the changes you made to the original DataSet. When you have merged the two DataSets and updated the contents, you then can call the AcceptChanges method of any of the following objects: DataSet, DataTable, and DataRow.

Choosing an AcceptChanges method

There is an AcceptChanges method for the DataSet, DataTable, DataRow objects.

When you call AcceptChanges on a DataSet, you also invoke the AcceptChanges method on all subordinate objects with a single call. A call to AcceptChanges on a DataSet object, also calls AcceptChanges on each DataTable in the DataSet, and calls AcceptChanges on each DataRow object in each DataTable.

You can, however, call AcceptChanges on an individual DataTable or DataRow.

Syntax

The following is the Visual C# syntax for the AcceptChanges method of the DataSet class:

```
Public void AcceptChanges ();
```

The syntax is the same for the DataTable and DataRow objects.

Example of merging DataSets

The following example shows how to use the **Merge** and **AcceptChanges** methods in a client application.

The client application has a DataSet named **dsCustomers**. The DataSet is bound to a DataGrid (for example), to allow the user to change the data locally. When the user is ready to send the changes to the data source, the application calls **GetChanges** to get the changes to the DataSet. The application sends this (smaller) DataSet to a middle-tier component, such as a Web Service method.

The code for the Web Service method is not shown, but it could use stored procedures to update the data source with the DataSet changes. The Web Service method returns a new DataSet, which contains the latest data from the data source (for example, the data source might have assigned default values to null columns in the DataSet).

The client application receives this sanitized DataSet, and merges it into the main **dsCustomers** DataSet. The client application then calls **AcceptChanges**, to mark these new records as "unchanged" in the **dsCustomers** DataSet:

```
' Visual Basic
' Get changes made by the user to the dsCustomers DataSet
Dim dsChanges As DataSet = dsCustomers.GetChanges()

' Send changes to a Web Service, get latest data back again
Dim service As New MyWebService()
Dim dsLatest As DataSet = service.MyUpdateMethod(dsChanges)

' Merge latest data back into the dsCustomers DataSet
dsCustomers.Merge(dsLatest)

' Mark all rows as "unchanged" in the dsCustomers DataSet
dsCustomers.AcceptChanges()
```

```
// Visual C#
// Get changes made by the user to the dsCustomers DataSet
DataSet dsChanges = dsCustomers.GetChanges();

// Send changes to a Web Service, get latest data back again
MyWebService service = new MyWebService();
DataSet dsLatest = service.MyUpdateMethod(dsChanges);

// Merge latest data back into the dsCustomers DataSet
dsCustomers.Merge(dsLatest);

// Mark all rows as "unchanged" in the dsCustomers DataSet
dsCustomers.AcceptChanges();
```

Practice

In this practice, you will continue to build a Windows application that allows the user to edit the Northwind Traders online product catalog.

1. Open the Windows Application solution you used in the previous practice, or the solution named **CatalogEditor** at the following location:

   <install folder>\Practices\Mod06_1\Lesson3\CatalogEditor\

2. In the form designer, right-click the **SqlDataAdapter1** control and choose **Generate Dataset**. Set the name of the new dataset to **ProductDataSet**, and select the **Products (SqlDataAdapter1)** table.
3. Drag and drop a **DataGrid** onto the form.
4. Drag and drop a **Button** onto the form. Change the text of the button to **Fill**.
5. Add the following code, to handle the Click event of this button:

```
SqlDataAdapter1.Fill(ProductDataSet1.Products)
DataGrid1.DataSource = ProductDataSet1.Products
```

6. Drag and drop another **Button** onto the form. Change the text of the button to **Get modified rows**.
7. Add the following code, to handle the Click event of this button. This code gets a copy of all the modified rows, and displays the current and original **ProductName** and **UnitPrice** for each row:

```
If (ProductDataSet1.HasChanges(DataRowState.Modified)) Then
  Dim ds As ProductDataSet = _
     ProductDataSet1.GetChanges(DataRowState.Modified)

  Dim row As DataRow
  For Each row In ds.Products.Rows
    Dim str As String = "Current:  " & _
     row("ProductName", DataRowVersion.Current) & ", " & _
     row("UnitPrice",   DataRowVersion.Current) & vbCrLf

    str = str & "Original:  " & _
     row("ProductName", DataRowVersion.Original) & ", " & _
     row("UnitPrice",   DataRowVersion.Original)

    MessageBox.Show(str, "Modified row")
  Next
Else
  MessageBox.Show("No modified rows", "Information")
End If
```

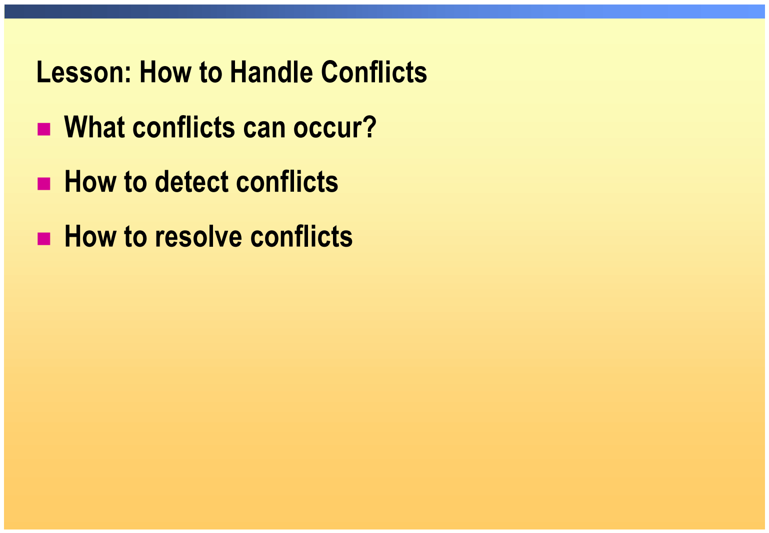
8. Drag and drop a third **Button** onto the form. Change the text of the button to **Update**.
9. Add the following code, to handle the Click event of this button. This code updates the data source, using the current and original data in the data set:

```
SqlDataAdapter1.Update(ProductDataSet1.Products)
```

10. Build and run the application.
11. Click **Fill**, to fill the dataset and display the data in the DataGrid.
12. Change the **ProductName** and **UnitPrice** values for some rows. Then click **Get modified rows**, to display the current and original data in these rows.
13. Click **Update**, to send all updates to the data source.
14. Use the Server Explorer to check that the products have been updated.
15. Back in your application, click **Get modified rows**. There are no modified rows in the data set now, because any pending modifications have been saved to the data source.

The solution for this practice is located at <install folder>\
Practices\Mod06_1\Lesson4\CatalogEditor\

Lesson: How to Handle Conflicts


Lesson: How to Handle Conflicts

- What conflicts can occur?
- How to detect conflicts
- How to resolve conflicts

Introduction

When you write a disconnected application, you might experience data conflicts when you try to update the data source. This happens if the data source has been changed by another application or service, while your application was disconnected from the data source.

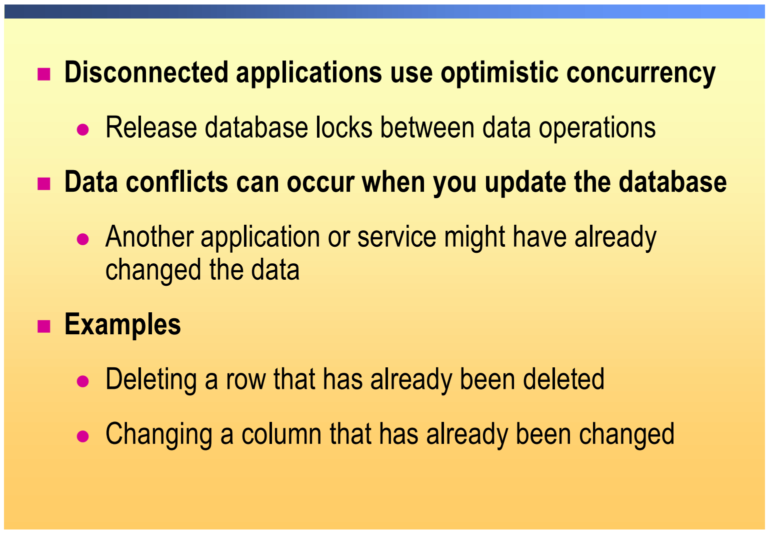
In this lesson, you will learn how to detect potential data conflicts before they happen. You will see how to use the **HasErrors** property to detect errors in a **DataSet**, **DataTable**, or **DataRow**. You will also learn how to resolve these conflicts in your application.

Lesson objectives

After completing this lesson, you will be able to:

- Explain when conflicts can occur
- Define optimistic concurrency
- Detect and resolve conflicts by using the **HasErrors** property

What Conflicts Can Occur?



- **Disconnected applications use optimistic concurrency**
  - Release database locks between data operations
- **Data conflicts can occur when you update the database**
  - Another application or service might have already changed the data
- **Examples**
  - Deleting a row that has already been deleted
  - Changing a column that has already been changed

Introduction
Disconnected applications in ADO.NET use optimistic concurrency. This can cause conflicts when the application tries to update the data source. You can write code to detect these conflicts, and handle them accordingly.

Definition of optimistic concurrency
In optimistic concurrency, database locks are released as soon as data retrieval operations or data update operations are complete. Disconnected applications use optimistic concurrency so that other applications can query and update the database concurrently.

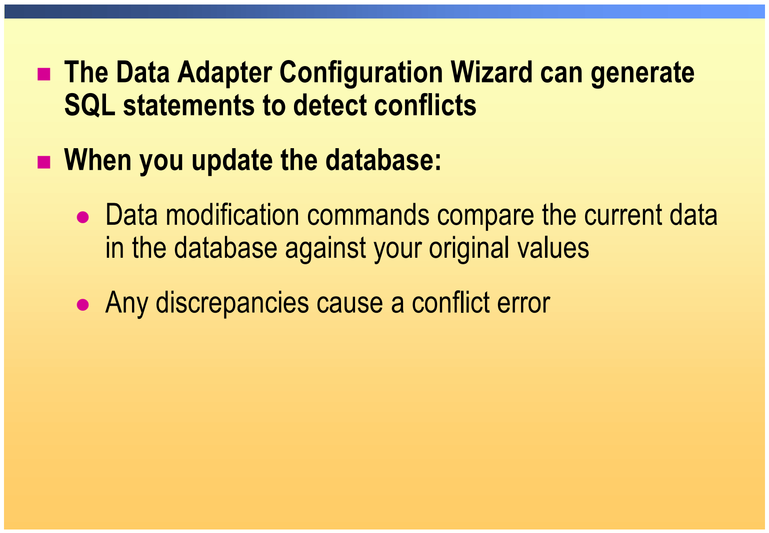
This is different from the situation in connected applications, which often use pessimistic concurrency. The database is kept locked while a series of related data operations are performed. This stops other applications from accessing the database until the related operations have been completed, preventing conflicts at the expense of temporarily denying database access to other applications.

Scenario
A disconnected application retrieves customer records from the central database at the start of the day. During the day, a mobile worker modifies these records, adds new records, and deletes records while disconnected from the database.

At the end of the day, the mobile worker connects to the corporate network and tries to update the central database with these changes. Unfortunately, a co-worker has already modified some of the customer records in the database. The application needs to detect which customer records are in conflict, and must resolve these conflicts in a sensible manner.

Practice
Group Discussion: What specific conflicts can occur when the disconnected application tries to update the data source? How can the disconnected application resolve these conflicts?

How to Detect Conflicts



- **The Data Adapter Configuration Wizard can generate SQL statements to detect conflicts**
- **When you update the database:**
  - Data modification commands compare the current data in the database against your original values
  - Any discrepancies cause a conflict error

Introduction

The Data Adapter Configuration Wizard can generate SQL statements to detect conflicts. The Wizard adds SQL tests to the **InsertCommand**, **UpdateCommand**, and **DeleteCommand**. These tests check that the data in the database is unchanged since you retrieved it into your application.

How the Wizard supports optimistic concurrency

When you use the Data Adapter Configuration Wizard to create a DataAdapter that uses SQL statements, the **Generate the SQL statements** screen lets you specify **Advanced Options**. One of these options is **Use optimistic concurrency**:

- If you choose this option, the Wizard will add tests to your SQL statements to detect conflict errors that arise due to optimistic concurrency.
- If you do not choose this option, the Wizard will not add conflict tests to your SQL statements. Any changes your application makes to data in the database will overwrite changes made by other users.

Example of how the Wizard supports optimistic concurrency

The following example shows how the Data Adapter Configuration Wizard helps detect conflicts that arise due to optimistic concurrency. The example sets the **UpdateCommand** for a DataAdapter. For simplicity, the example uses a simplified Customers table containing just two columns, **CustomerID** and **CustomerName**.

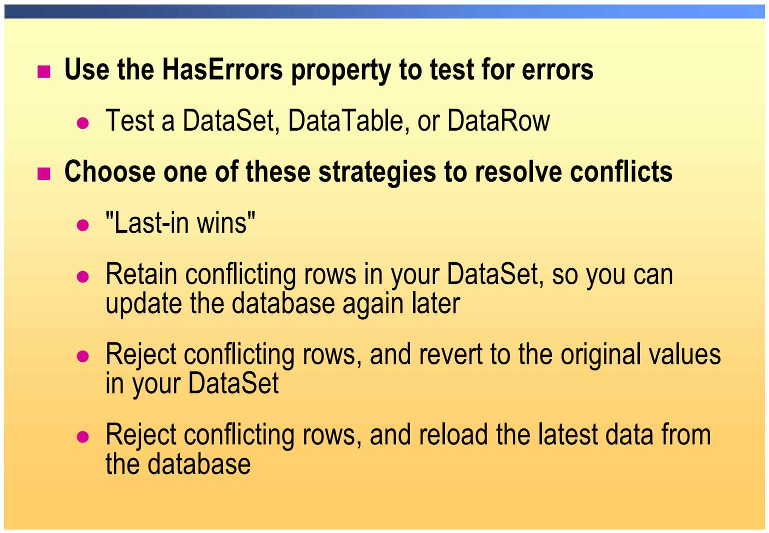
The **UpdateCommand** object requires five parameters:

- The first and second parameters specify the current **CustomerID** and **CompanyName** for the row.
- The third and fourth parameters specify the original **CustomerID** and **CompanyName** for the row. The SQL statement has a **WHERE** clause, to ensure the row in the database still contains these original values.
- The final parameter is used in a **SELECT** statement, to retrieve the updated row from the database. This ensures that the application has the very latest row data, after any trigger operations or default values assignments by the database.

```
' Visual Basic
Me.cmUpdate.CommandText = _
   "UPDATE Customers " & _
   "SET CustomerID=@CustomerID, CompanyName=@CompanyName " & _
   "   WHERE (CustomerID  = @Original_CustomerID) " & _
   "   AND   (CompanyName = @Original_CompanyName); " & _
   "SELECT CustomerID, CompanyName FROM Customers " & _
   "   WHERE (CustomerID = @Select_CustomerID)"

Me.cmUpdate.Parameters.Add(New SqlParameter( _
  "@CustomerID", _
   SqlDbType.NChar, 5, ParameterDirection.Input, False, _
   0, 0, "CustomerID", DataRowVersion.Current, Nothing))

Me.cmUpdate.Parameters.Add(New SqlParameter( _
  "@CompanyName", _
   SqlDbType.NVarChar, 40, ParameterDirection.Input, False, _
   0, 0, "CompanyName", DataRowVersion.Current, Nothing))

Me.cmUpdate.Parameters.Add(New SqlParameter( _
  "@Original_CustomerID", _
   SqlDbType.NChar, 5, ParameterDirection.Input, False, _
   0, 0 , "CustomerID", DataRowVersion.Original, Nothing))

Me.cmUpdate.Parameters.Add(New SqlParameter( _
  "@Original_CompanyName", _
   SqlDbType.NVarChar, 40, ParameterDirection.Input, False, _
   0, 0, "CompanyName", DataRowVersion.Original, Nothing))

Me.cmUpdate.Parameters.Add(New SqlParameter( _
  "@Select_CustomerID", _
   SqlDbType.NChar, 5, ParameterDirection.Input, False, _
   0, 0, "CustomerID", DataRowVersion.Current, Nothing))
```

```
// Visual C#
this.cmUpdate.CommandText =
   "UPDATE Customers " +
   "SET CustomerID=@CustomerID, CompanyName=@CompanyName " +
   "   WHERE (CustomerID  = @Original_CustomerID) " +
   "   AND   (CompanyName = @Original_CompanyName); " +
   "SELECT CustomerID, CompanyName FROM Customers " +
   "   WHERE (CustomerID = @Select_CustomerID)";

this.cmUpdate.Parameters.Add(new SqlParameter(
  "@CustomerID",
   SqlDbType.NChar, 5, ParameterDirection.Input, false,
   0, 0, "CustomerID", DataRowVersion.Current, null));

this.cmUpdate.Parameters.Add(new SqlParameter(
  "@CompanyName",
   SqlDbType.NVarChar, 40, ParameterDirection.Input, false,
   0, 0, "CompanyName", DataRowVersion.Current, null));

this.cmUpdate.Parameters.Add(new SqlParameter(
  "@Original_CustomerID",
   SqlDbType.NChar, 5, ParameterDirection.Input, false,
   0, 0 , "CustomerID", DataRowVersion.Original, null));

this.cmUpdate.Parameters.Add(new SqlParameter(
  "@Original_CompanyName",
   SqlDbType.NVarChar, 40, ParameterDirection.Input, false,
   0, 0, "CompanyName", DataRowVersion.Original, null));

this.cmUpdate.Parameters.Add(new SqlParameter(
  "@Select_CustomerID",
   SqlDbType.NChar, 5, ParameterDirection.Input, false,
   0, 0, "CustomerID", DataRowVersion.Current, null));
```

How to Resolve Conflicts



- **Use the HasErrors property to test for errors**
  - Test a DataSet, DataTable, or DataRow
- **Choose one of these strategies to resolve conflicts**
  - "Last-in wins"
  - Retain conflicting rows in your DataSet, so you can update the database again later
  - Reject conflicting rows, and revert to the original values in your DataSet
  - Reject conflicting rows, and reload the latest data from the database

Introduction

Use the HasErrors property to resolve conflicts when you update data in a disconnected application. You can use this property to find the location and nature of the error in your DataSet.

Definition

The **DataSet**, **DataTable**, and **DataRow** classes each provide a **HasErrors** property. You can use this property on any of these objects, to identify conflicts and other errors at any level of granularity in your data. The **DataRow** class also has a **GetColumnsInError** method, to get the columns in error for a particular row.

How to resolve conflicts

To resolve conflicts, choose one of the following strategies:

- Use a "last in wins" approach, so that data changes made by your application overwrite any database changes made by other applications. This approach is effective for administrative applications that need to force changes through a database.

  To achieve this effect, do not choose **Use optimistic concurrency** when you create the DataAdapter in the Data Adapter Configuration Wizard.

- Do not force conflicting data changes on the database. Retain the conflicting changes locally, in your DataSet, so that the user can try to update the database again later.

  This is the default behavior when you choose the **Use optimistic concurrency** option in the Data Adapter Configuration Wizard.

- Reject the conflicting data changes in the local DataSet, and revert to the data originally loaded from the database.

  To achieve this effect, call the **RejectChanges** method on the conflicting DataSet, DataTable, or DataRow.

- Reject the conflicting data changes in the local DataSet, and reload the latest data from the database.

  To achieve this effect, call the **Clear** method on the DataSet. Then call the **Fill** method on the DataAdapter, to reload the latest data.

Example of resolving conflicts

The following example shows how to resolve conflicts in a disconnected application.

After an **Update** operation, the **HasErrors** property is tested to see if the DataSet has any errors. If there are errors, a loop is used to check each table in turn. If a table has errors, another loop is used to check each of its rows. If a row has errors, the **GetColumnsInError** method is used to find which columns are in error. The **ClearError** and **RejectChanges** methods are then called, to clear the error status and reject the conflicting data in each row:

```
' Visual Basic
Try
  daCustomers.Update(dsCustomers)
Catch ex As System.Exception
  If dsCustomers.HasErrors Then
    Dim table As DataTable
    For Each table In dsCustomers.Tables

      If table.HasErrors Then
        Dim row As DataRow
        For Each row In table.Rows

          If row.HasErrors Then
            MessageBox.Show("Row: " & row("CustomerID"), _
                             row.RowError)

            Dim column As DataColumn
            For Each column In row.GetColumnsInError()
              MessageBox.Show(column.ColumnName, _
                              "Error in this column")
            Next
            row.ClearErrors()
            row.RejectChanges()
          End If
        Next
      End If
    Next
  End If
End Try
```

```
// Visual C#
try
{
  daCustomers.Update(dsCustomers);
}
catch(System.Exception ex)
{
  if(dsCustomers.HasErrors)
  {
    foreach(DataTable table in dsCustomers.Tables)
    {
      if(table.HasErrors)
      {
        foreach(DataRow row in table.Rows)
        {
          if(row.HasErrors)
          {
            MessageBox.Show("Row: " + row["CustomerID"],
                             row.RowError);

            foreach(DataColumn col in row.GetColumnsInError())
            {
              MessageBox.Show(column.ColumnName,
                               "Error in this column");
            }
            row.ClearErrors();
            row.RejectChanges();
          }
        }
      }
    }
  }
}
```

Practice

In this practice, you will continue to build a Windows application that allows the user to edit the Northwind Traders online product catalog.

1. Open the Windows Application solution you used in the previous practice, or the solution named **CatalogEditor** at the following location:

   <install folder>\Practices\Mod06_1\Lesson4\CatalogEditor\

2. Run the application. Change the name of a product. Do NOT click **Update** yet.
3. Use the Server Explorer to change the same product name to a different value.
4. Switch back to the running application, and click **Update**.

   What happens? Why? What does the user have to do in order to force their change through to the underlying data source?

5. Stop the application running.

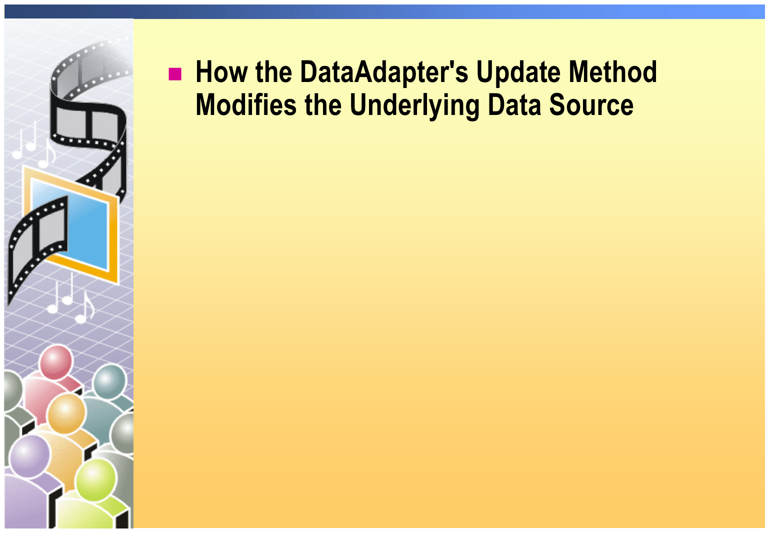
6. Modify the Click event handler for the **Update** button as follows. Check for a DBConcurrencyException, to indicate a conflict error. If this occurs, clear the error status and accept the latest value for the conflicting row from the database:

```
Try

  SqlDataAdapter1.Update(ProductDataSet1.Products)

Catch ex As System.Data.DBConcurrencyException

  MessageBox.Show( _
    "Conflict with an existing record. " & _
    "You have lost your changes for product: " & _
     ex.Row("ProductName").ToString(), _
    "Warning!")

  ' Clear the error status for the conflicting row
  ex.Row.ClearErrors()

  ' Accept the latest value for this row from the database
  ex.Row.AcceptChanges()

End Try
```

7. Run and test your application. Conflicts are now automatically handled by resetting conflicting values to the central version, and the user can immediately reenter the value they want if desired.

The solution for this practice is located at <install folder>\Practices\Mod06_1\Lesson5\CatalogEditor\

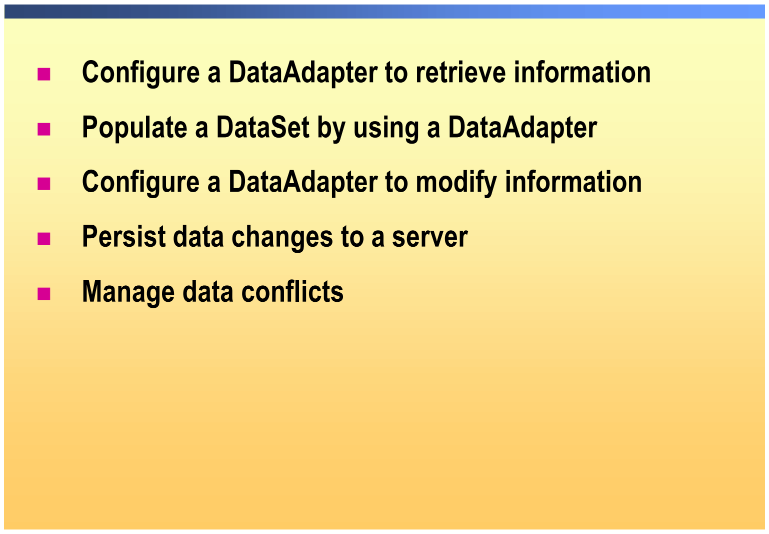
Multimedia: How the DataAdapter's Update Method Modifies the Underlying Data Source



- How the DataAdapter's Update Method Modifies the Underlying Data Source

Text about the multimedia

Review



- Configure a DataAdapter to retrieve information
- Populate a DataSet by using a DataAdapter
- Configure a DataAdapter to modify information
- Persist data changes to a server
- Manage data conflicts

1. How do you create and configure a DataAdapter, to provide a disconnected application with read-only access to a SQL Server 2000 database?

   Create a SqlDataAdapter object, either programmatically in your code or by using the Data Adapter Configuration Wizard.

   Initialize the SelectCommand property for the DataAdapter. You must specify a SqlConnection object. You must also specify a query to retrieve data from the data source. You can define a SQL SELECT to do this, or use a new or existing stored procedure.

2. What is the most efficient way to populate a DataSet by using a DataAdapter?

   First, create a strongly-typed DataSet with the same structure as the data retrieved by the DataAdapter.

   When you are ready to fill the DataSet, call the BeginLoadLoad method to disable constraint checks and index maintenance while the data is being loaded. Then call Fill on the DataAdapter, to fill a specific DataTable in the DataSet. Finally, call EndLoadData when the data has been completely loaded.

3. How do you configure a DataAdapter, to allow a data source to be updated from the contents of a DataSet?

   When you create a DataAdapter object, define SqlCommand or OleDbCommand objects for its InsertCommand, UpdateCommand, and DeleteCommand properties. The DataAdapter uses these command objects implicitly to propagate DataSet changes back to the data source.

4. How do you persist data changes back to the data source? How do you control the order in which different types of changes are persisted?

   Call the Update method on the DataAdapter. If your DataSet contains several DataTables, you might need to perform some updates before others (to avoid foreign key constraint errors). In this case, call the GetChanges method on a DataSet to obtain a subset of changes – modifications, updates, or deletes – on each table. Call the Update method separately on each set of changes, to control the order in which changes are persisted to the data source.

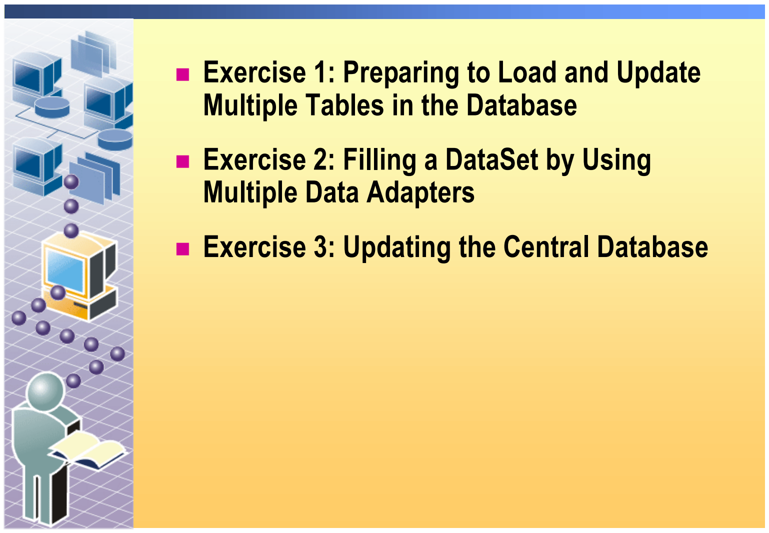
5. What types of conflict can occur when you update a data source in a disconnected application? How do you detect and resolve these conflicts?

   Disconnected applications use optimistic concurrency. This means that rows might be inserted, updated or deleted by other users while your application is disconnected from the data source.

   This can cause conflicts when you try to save your changes to the data source. An exception occurs in this situation. You can catch this exception, and detect the location of the problem by inspecting the HasErrors property on a DataSet, DataTable, and DataRow.

   Once you have located the problem, you can decide whether to reject the proposed change, force the change upon the data source, or keep the change locally in the DataSet so that it can be saved later by the user.

Lab 6.2: Retrieving and Updating Customers and Orders Data



- Exercise 1: Preparing to Load and Update Multiple Tables in the Database
- Exercise 2: Filling a DataSet by Using Multiple Data Adapters
- Exercise 3: Updating the Central Database

Objectives

After completing this lab, you will be able to:

- Create DataAdapters to access multiple tables in the Northwind database.
- Define corresponding tables in a DataSet in your application.
- Specify relationships and constraints in the DataSet tables.
- Populate the DataSet and display its data in a DataGrid.
- Update the data source from the DataSet.

Prerequisites

Before working on this lab, you must have:

- .
- .

For More Information

See the DataSet and SqlDataAdapter topics in the Visual Studio .NET documentation.

Scenario

In Lab 6.1, “Retrieving Data into a Disconnected Application”, you started writing a Windows Application to help sales persons at Northwind Traders deal with customer data while away from the office.

So far, you have written code to download employee names from the central database. You have also written code to save employee names and applications data locally in an XML file.

In this lab, you will extend the application so that it can retrieve and update customers and orders data from the central database. The sales person will download this data at the start of the day, and work with the data while disconnected from the central database. At the end of the day, the sales person will connect to the central database and update any records that have been changed during the day.

Estimated time to complete this lab: 60 minutes

Exercise 0
Lab Setup

To complete this lab, you must …

►

2.

3.

Exercise 1
Preparing to Load and Update Multiple Tables in the Database

Sales persons need to use the Windows Application to create new orders during the working day. To achieve this task, the application needs a local copy of the products, customers, orders, and order details information from the central database.

Scenario

In this exercise, you will create DataAdapters to access the Products, Customers, Orders, and Order Details tables in the Northwind database. You will then add four tables to the DataSet in your application, to correspond with the data returned by these DataAdapters.

► **To continue building the application**

1. Start the Microsoft Visual Studio .NET development environment.
2. Open the solution you created in Lab 6.1, or open the solution **OnTheRoad** in the folder in <install folder>\Labs\Lab06_2\Starter\xx\ where xx is either VB or CS.

► **To create the products table**

1. Open the **MainForm** class in Designer view and drag a **SqlDataAdapter** from the Toolbox onto the form.
2. Choose a data connection to the Northwind database on your local SQL Server.
3. Choose to **Use SQL statements** and click Next.
4. Enter the following statement.

```
SELECT
    ProductID, ProductName, UnitPrice
FROM
    Products
```

5. Click the **Advanced Options** button and clear the **Generate Insert, Update and Delete statements** check box.
6. Click **Finish**. The wizard will now create a data adapter, a connection and a command that will be used to populate the products table in the data set.
7. Change the name of the new data adapter to **daProducts**, and its associated **SelectCommand** to **cmSelectProducts**.
8. Right-click the **daProducts** data adapter and choose **Generate Dataset**.
9. Choose the existing data set called **OnTheRoad.NWDataSet**, and clear the **Add the data set to the designer** check box. Click **OK**.
10. Right-click the **dsNorthwind** data set, then choose **View Schema** to check that the products table has been added to the data set schema.

► **To create the customers table**

1. Open the **MainForm** class in Designer view and drag a **SqlDataAdapter** from the Toolbox onto the form.
2. Choose a data connection to the Northwind database on your local SQL Server.
3. Choose to **Create new stored procedures** and click Next.
4. Type the following statement.

```
SELECT
  Customers.CustomerID, CompanyName, ContactName, City, Phone
FROM Customers INNER JOIN Orders ON
  Customers.CustomerID = Orders.CustomerID
WHERE (Orders.EmployeeID = @EmployeeID)
ORDER BY CompanyName
```

5. Change the names of the stored procedures to: **SelectCustomers**, **InsertCustomers**, **UpdateCustomers**, and **DeleteCustomers**.
6. Click **Finish**. The wizard will now create a data adapter, a connection and four commands that will be used to populate and modify customers.
7. Use the Server Explorer to modify the **SelectCustomers** stored procedure by adding DISTINCT after the SELECT keyword. This will prevent duplicate customer rows. The Wizard cannot auto-generate DML statements based on SELECT DISTINCT statements.
8. Change the name of the new data adapter to **daCustomers**, and its associated **XxxCommands** to **cmSelectCustomers, cmInsertCustomers, cmDeleteCustomers,** and **cmUpdateCustomers**.
9. Right-click the **daCustomers** data adapter and choose **Generate Dataset**, choose the existing data set called **OnTheRoad.NWDataSet**, and clear the **Add the data set to the designer** check box. Click **OK**.

► **To create the orders table**

1. Open the **MainForm** class in Designer view and drag a **SqlDataAdapter** from the Toolbox onto the form.
2. Choose a data connection to the Northwind database on your local SQL Server.
3. Choose to **Create new stored procedures** and click Next.
4. Type the following statement.

```
SELECT OrderID, OrderDate, EmployeeID, CustomerID
FROM Orders WHERE (EmployeeID = @EmployeeID)
```

5. Change the names of the stored procedures to: **SelectOrders**, **InsertOrders**, **UpdateOrders**, and **DeleteOrders**.
6. Click **Finish**.
7. Change the name of the new data adapter to **daOrders**, and its associated **XxxCommands** to **cmSelectOrders, cmInsertOrders, cmDeleteOrders,** and **cmUpdateOrders**.
8. Right-click the **daOrders** data adapter and choose **Generate Dataset**, choose the existing data set called **OnTheRoad.NWDataSet**, and clear the **Add the data set to the designer** check box. Click **OK**.

► **To create the order details table**

1. Open the **MainForm** class in Designer view and drag a **SqlDataAdapter** from the Toolbox onto the form.
2. Choose a data connection to the Northwind database on your local SQL Server.
3. Choose to **Create new stored procedures** and click Next.
4. Type the following statement.

```
SELECT
   [Order Details].OrderID, ProductID, UnitPrice, Quantity
FROM [Order Details] INNER JOIN Orders ON
   [Order Details].OrderID = Orders.OrderID
WHERE
   (Orders.EmployeeID = @EmployeeID)
```

5. Change the names of the stored procedures to: **SelectOrderDetails**, **InsertOrderDetails**, **UpdateOrderDetails**, and **DeleteOrderDetails**.
6. Click **Finish**.
7. Change the name of the new data adapter to **daOrderDetails**, and its associated **XxxCommands** to **cmSelectOrderDetails, cmInsertOrderDetails, cmDeleteOrderDetails,** and **cmUpdateOrderDetails**.
8. Change the **TableMappings** property of the **daOrderDetails** data adapter so that the data set table name does not include a space between Order and Details.
9. Right-click the **daOrderDetails** data adapter and choose **Generate Dataset**, choose the existing data set called **OnTheRoad.NWDataSet**, and clear the **Add the data set to the designer** check box. Click **OK**.

► **To add primary keys to the custom data set schema and class**

1. Right-click the **dsNorthwind** data set, then choose **View Schema** to check that the customers, orders, and order details tables have been added to the data set schema.
2. Right-click the **CustomerID** field in the **Customers** table, and choose **Add – New key**.
3. Change the name to **PK_Customers** and click **OK**.
4. Right-click the **OrderID** field in the **Orders** table, and choose **Edit key**.
5. Change the name to **PK_Orders** and click **OK**.
6. Right-click the **OrderID** field in the **OrderDetails** table, and choose **Add – New key**.
7. Change the name to **PK_OrderDetails**, add **ProductID** to the list of fields, and click **OK**.
8. Right-click the **EmployeeID** field in the **Employees** table, and choose **Edit key**.
9. Change the name to **PK_Employees** and click **OK**.
10. Right-click the **ProductID** field in the **Products** table, and choose **Edit key**.
11. Change the name to **PK_Products** and click **OK**.

► **To add relationships to the custom data set schema and class**

1. Right-click the **Orders** table and choose **Add – New Relation**.
2. Select **Employees** for the parent element and **Orders** for the child element, and click **OK**.
3. Right-click the **Orders** table and choose **Add – New Relation**.
4. Select **Customers** for the parent element and **Orders** for the child element, and click **OK**.
5. Right-click the **OrderDetails** table and choose **Add – New Relation**.
6. Select **Orders** for the parent element and **OrderDetails** for the child element, change the name to **OrdersOrderDetails** and click **OK**.
7. Right-click the **OrderDetails** table and choose **Add – New Relation**.
8. Select **Products** for the parent element and **OrderDetails** for the child element, change the name to **ProductsOrderDetails** and click **OK**.
9. Save and close the XSD file.

► **To test your code**

- Build your application. You are not ready to execute the application yet, because you have not written any code to fill the DataSet from the DataAdapers.

Exercise 2
Filling a DataSet by Using Multiple Data Adapters

When the sales person is connected to the central database, he or she can download data for products, customers, orders, and order details. This information will be held in a local DataSet within the application, so that the user can continue to use the data while the application is disconnected from the database.

Scenario

In this exercise, you will use the **Fill** method on the DataAdapters to fill the various tables in the DataSet. You will also bind the DataSet to a DataGrid, to display the data on the screen.

► To start with the solution to the previous exercise

- If you did not complete the previous exercise, open the solution **OnTheRoad** in the folder <install folder>\Labs\Lab06_2\Solution\Ex1\xx\ where xx is either VB or CS.

► To refresh the data grid

1. In MainForm.vb, add code to the **RefreshUI** method to bind the data grid to the Customers table in the data set.

   ```
   ' Visual Basic
   Me.grd.DataSource = Me.dsNorthwind.Customers

   // Visual C#
   this.grd.DataSource = this.dsNorthwind.Customers;
   ```

2. Locate the **mnuFill_Click** procedure, and insert a new line after the line that retrieves the selected employee ID into the field. For example:

```
' Visual Basic
If frmLogon.ShowDialog(Me) = DialogResult.OK Then
  Me.EmployeeID = CInt(frmLogon.lstEmployees.SelectedValue)
  ' insert new code here

// Visual C#
if (frmLogon.ShowDialog(this) == DialogResult.OK)
{
  this.EmployeeID =
      ConvertTo(frmLogon.lstEmployees.SelectedValue, int);
  // insert new code here
```

3. Write code to try to fill the new tables you just added to the data set (Products, Customers, Orders, OrderDetails) using the data adapters created by the wizard. Catch any exceptions and display a warning message.

```
' Visual Basic
Try
  Me.daProducts.Fill(tempNW.Products)

  Me.daCustomers.SelectCommand.Parameters( _
      "@EmployeeID").Value = Me.EmployeeID
  Me.daCustomers.Fill(tempNW.Customers)

  Me.daOrders.SelectCommand.Parameters( _
      "@EmployeeID").Value = Me.EmployeeID
  Me.daOrders.Fill(tempNW.Orders)

  Me.daOrderDetails.SelectCommand.Parameters( _
      "@EmployeeID").Value = Me.EmployeeID
  Me.daOrderDetails.Fill(tempNW.OrderDetails)

  Me.dsNorthwind = tempNW
  Me.RefreshUI()

Catch Xcp As System.Exception
  MessageBox.Show( _
      "Failed to retrieve data because: " & vbCrLf & _
      Xcp.ToString() & vbCrLf & vbCrLf & _
      "Try a different server name.", _
      "Get from central database", _
      MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try

// Visual C#
try
{
  this.daProducts.Fill(tempNW.Products);

  this.daCustomers.SelectCommand.Parameters[
      "@EmployeeID"].Value = this.EmployeeID;
  this.daCustomers.Fill(tempNW.Customers);

  this.daOrders.SelectCommand.Parameters[
      "@EmployeeID"].Value = this.EmployeeID;
  this.daOrders.Fill(tempNW.Orders);

  this.daOrderDetails.SelectCommand.Parameters[
      "@EmployeeID"].Value = this.EmployeeID;
  this.daOrderDetails.Fill(tempNW.OrderDetails);

  this.dsNorthwind = tempNW;
  this.RefreshUI();
}
catch (System.Exception Xcp)
{
  MessageBox.Show(
      "Failed to retrieve data because:\n" + Xcp.ToString() +
      "\n\nTry a different server name.",
      "Get from central database",
      MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

► **To test the filling and saving of the complete data set**

1. Build the application and correct any build errors.
2. Open the **MainForm** class in Code view and set a break point at the beginning of the **mnuFill_Click** procedure.
3. Use the Server Explorer to set breakpoints on the first line of the **SelectCustomers**, **SelectOrders** and **SelectOrderDetails** stored procedures.
4. Run the application.
5. Choose the **Get from central database** menu item.
6. Step through the code line by line until the form appears.
7. Choose **Fuller, Andrew**, and then click **OK**.
8. Continue to step through the code line by line. Notice the value of the EmployeeID field.
9. When stepping over the code that fills the customers table, notice that the debugger steps into the correct stored procedure, and that the value passed to the stored procedure for the @EmployeeID parameter is the same value as the EmployeeID field.
10. Continue to step through the code line by line until the main form appears with the data grid full of customers and orders taken by Andrew Fuller.
11. Close the application, and check that the **OnTheRoad.xml** file contains all records in the data set using Internet Explorer.
12. Rerun the application and notice that the data set is correctly reloaded automatically.

Exercise 3
Updating the Central Database

When the sales person reconnects to the central database, he or she can persist any changes made to the products, customers, orders, and order details tables in the DataSet.

Scenario

In this exercise, you will use the **Update** method on the DataAdapters to update the central database.

To update the database, you must separate the insert, update and delete changes made to the data set so that they can be applied to the three tables (customer, orders, order details) in the correct order. For example, inserts of customers must occur before inserts of orders, but deletes of customers must come *after* deletes of orders.

► To start with the solution to the previous exercise

- If you did not complete the previous exercise, open the solution **OnTheRoad** in the folder <install folder>\Labs\Lab06_2\Solution\Ex2\xx\ where xx is either VB or CS.

► To update the central database

1. Locate the **mnuUpdate_Click** event handler.
2. Write code to declare three local DataSet variables named **dsInserts**, **dsUpdates**, and **dsDeletes**. Instantiate them with the results of calling the **GetChanges** method of the **dsNorthwind** data set, passing a **DataRowState** parameter to separate insert, update and delete changes.

```
' Visual Basic
Dim dsInserts As DataSet = _
   Me.dsNorthwind.GetChanges(DataRowState.Added)

Dim dsUpdates As DataSet = _
   Me.dsNorthwind.GetChanges(DataRowState.Modified)

Dim dsDeletes As DataSet = _
   Me.dsNorthwind.GetChanges(DataRowState.Deleted)

// Visual C#
DataSet dsInserts =
   this.dsNorthwind.GetChanges(DataRowState.Added);

DataSet dsUpdates =
   this.dsNorthwind.GetChanges(DataRowState.Modified);

DataSet dsDeletes =
   this.dsNorthwind.GetChanges(DataRowState.Deleted);
```

3. Write code to try to call the **Update** method of each of the three data adapters for the insert, update and delete changes made to the data set in the correct order if changes exist.

```
' Visual Basic
Try
  If Not dsInserts Is Nothing Then
      Me.daCustomers.Update(dsInserts.Tables("Customers"))
      Me.daOrders.Update(dsInserts.Tables("Orders"))
      Me.daOrderDetails.Update(dsInserts.Tables( _
          "OrderDetails"))
  End If

  If Not dsUpdates Is Nothing Then
      Me.daCustomers.Update(dsUpdates.Tables("Customers"))
      Me.daOrders.Update(dsUpdates.Tables("Orders"))
      Me.daOrderDetails.Update(dsUpdates.Tables( _
          "OrderDetails"))
  End If

  If Not dsDeletes Is Nothing Then
      Me.daOrderDetails.Update(dsDeletes.Tables( _
          "OrderDetails"))
      Me.daOrders.Update(dsDeletes.Tables("Orders"))
      Me.daCustomers.Update(dsDeletes.Tables("Customers"))
  End If

// Visual C#
try
{
  if (dsInserts != null)
  {
      this.daCustomers.Update(dsInserts.Tables["Customers"]);
      this.daOrders.Update(dsInserts.Tables["Orders"]);
      this.daOrderDetails.Update(dsInserts.Tables[
          "OrderDetails"]);
  }

  if (dsUpdates != null)
  {
      this.daCustomers.Update(dsUpdates.Tables["Customers"]);
      this.daOrders.Update(dsUpdates.Tables["Orders"]);
      this.daOrderDetails.Update(dsUpdates.Tables[
          "OrderDetails"]);
  }

  if (dsDeletes != null)
  {
      this.daOrderDetails.Update(dsDeletes.Tables[
          "OrderDetails"]);
      this.daOrders.Update(dsDeletes.Tables["Orders"]);
      this.daCustomers.Update(dsDeletes.Tables["Customers"]);
  }
```

4. Write code to try to catch any exceptions by displaying a message box and exiting the procedure.

```
' Visual Basic
Catch Xcp As System.Exception
   MessageBox.Show(Xcp.ToString())
  Exit Sub
End Try

// Visual C#
catch (System.Exception Xcp)
{
   MessageBox.Show(Xcp.ToString());
   return;
}
```

5. Write code to ask the user if they want to refresh the data set, and if so, call the **mnuFill_Click** procedure.

```
' Visual Basic
If MessageBox.Show( _
   "Do you want to refresh your local copy of data?", _
   "Update", MessageBoxButtons.YesNo, _
   MessageBoxIcon.Question) = DialogResult.Yes Then

  mnuFill_Click(sender, e)
End If

// Visual C#
if (MessageBox.Show(
   "Do you want to refresh your local copy of data?",
   "Update", MessageBoxButtons.YesNo,
   MessageBoxIcon.Question) == DialogResult.Yes)
{
  mnuFill_Click(sender, e);
}
```

► **To test the updating of the central database**

1. Clear all existing break points in your application.
2. Set a new break point at the beginning of the **mnuUpdate_Click** procedure.
3. Use the Server Explorer to set breakpoints on the first line of the **UpdateCustomers**, **UpdateOrders, UpdateOrderDetails, InsertOrders, InsertOrderDetails, DeleteOrders,** and **DeleteOrderDetails** stored procedures.
4. Run your application.
5. Make changes to the data set. Insert a new order (and order details) for an existing customer. Edit an existing customer and order information.
6. Choose the **Update to central database** menu item.
7. Step through the code and notice which stored procedures are run, and notice the values of parameters passed.
8. Close your application.
9. Restart the application and try to delete the order you added previously.
10. Choose the **Update to central database** menu item.
11. Step through the code and notice which stored procedures are run, and notice the values of parameters passed.
12. Close the application.

msdn® training

Module 7: Building and Consuming a Web Service That Uses ADO .NET (Prerelease)

Contents

Microsoft®

Instructor Notes

Presentation:
45 Minutes

Lab:
60 Minutes

This module teaches students how to build and manage DataSets, define data relationships, modify data, and use DataViews. Because practices in this module build on files built during Lesson 1, the starter file for Lesson 2 is the solution file for Lesson 1. Lesson 3 and practices in other lessons also build upon each other.

After completing this module, students will be able to:

- Build a Web service.
- Consume a Web service in a client application.
- Troubleshoot errors in an ADO .NET application.

Required materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2389A_07.ppt
- Module 7, "Building and Consuming a Web Service That Uses ADO .NET"
- Lab 7, Troubleshooting an ADO .NET Application

Preparation tasks

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices and labs.
- Read the latest .NET Development news at http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000519

Classroom setup

How to Teach This Module

This section contains information that will help you to teach this module.

Lesson: Building and Consuming a Web Service That Returns Data

This section describes the instructional methods for teaching each topic in this lesson.

What is a Web Service?

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What are some other examples of Web services?
- How can you see Web services used in your organization? Give examples.

How to Build a Web Service That Returns Data

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What other ways could you fill the local DataSet with data other than by using a DataAdapter?
- What is the purpose of creating an empty, local instance of a DataSet?
- Why use a strongly-typed DataSet?

Transition to Practice Exercise: Now that you have seen examples of creating a Web service, you can now practice creating a Web service programmatically.

Instruct students to turn to the practice exercise at the end of this topic in the student workbook.

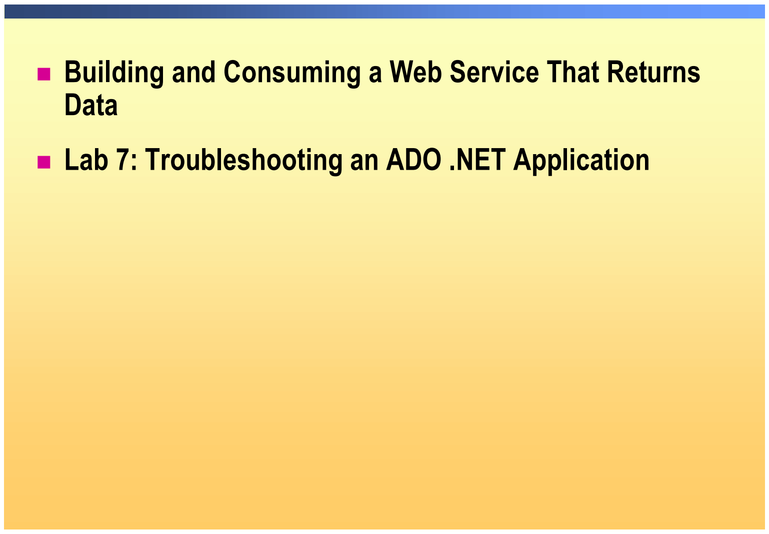
How to Consume a Web Service

Discussion Questions: Personalize the following questions to the background of the students in your class.

- What are some ways to find out the reference to Web services? How do you use Universal Description, Discover, and Integration (UDDI)?

Transition to Practice Exercise: Now that you have seen examples of consuming a Web service, you can now practice consuming a Web service in a client application.

Overview



- Building and Consuming a Web Service That Returns Data
- Lab 7: Troubleshooting an ADO .NET Application

Introduction

Web services allow applications to communicate regardless of operating system or programming language via the Internet. They can be implemented on any platform and are defined through public standards organizations. Sharing data through Web services allows the Web services to be independent of each other while simultaneously giving them the ability to loosely link themselves into a collaborating group that performs a particular task...

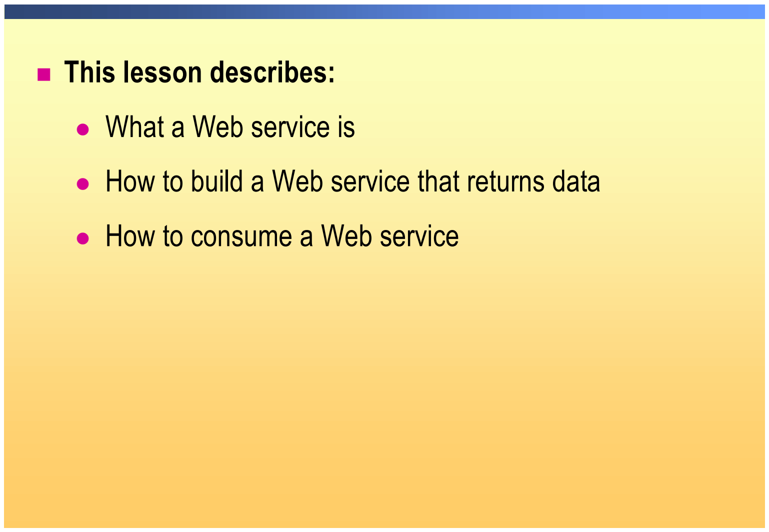
In this module, you will learn to create a Web service that returns data.

Objectives

After completing this module, you will be able to:

- Build and consume a Web service.
- Troubleshoot errors in an ADO .NET application.

Lesson: Building and Consuming a Web Service That Returns Data



- **This lesson describes:**
  - What a Web service is
  - How to build a Web service that returns data
  - How to consume a Web service

Introduction

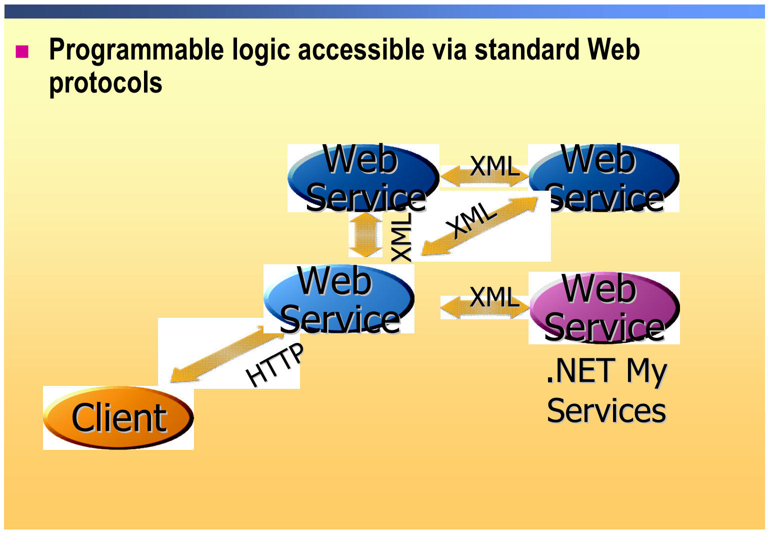
Web services are enabling a new era of distributed application development. By using ADO .NET, you can build Web services that return data, and those Web services can be consumed by multiple applications locally or across the Internet.

Lesson Objective(s)

After completing this lesson, you will be able to:

- Explain what a Web service is.
- Build a Web service that returns data.
- Consume a Web service.

What is a Web Service?



- Programmable logic accessible via standard Web protocols

Definition

A Web service is a piece of programmable logic accessible by standard Web protocols such as HTTP and XML. A Web service can be used locally by a single application or published on the Internet for use by multiple heterogeneous applications.

Web services allow applications to share data and functionality. Web services use XML-based messaging to communicate between systems that use different component models, operating systems, and programming languages. Developers can create applications that weave together Web services from a variety of sources in much the same way that developers traditionally use components when creating a distributed application.

By using XML-based messaging to communicate between a web service and a client application, both the Web service client and the Web service provider are free from needing any knowledge of each other beyond inputs, outputs and location.

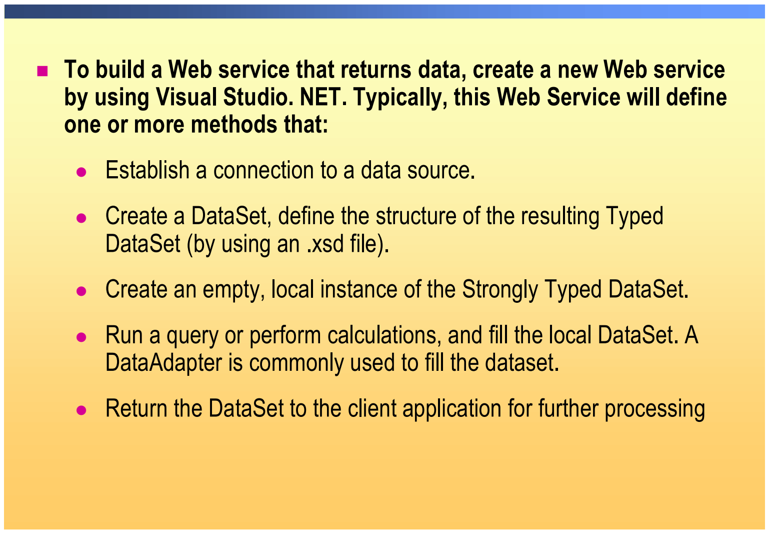
Example

A Web service can provide some reusable functionality that many clients can share.

For example, a challenge faced by e-commerce applications is the need to calculate charges for an assortment of shipping options. Such applications would require current shipping cost tables from each shipping company to use in these calculations.

Alternatively, an application could send an XML-based message over the Internet, using a standard transport protocol such as HTTP, to the shipper's cost calculation Web service. The message might provide the weight and dimensions of the package, ship-from and ship-to locations, and other parameters such as perhaps class of service. The shipper's Web service would then calculate the shipping charge using the latest cost table and return, in a simple XML-based response message, this amount to the calling application for use in calculating the total charge to the customer.

How to Build a Web Service That Returns Data



- **To build a Web service that returns data, create a new Web service by using Visual Studio. NET. Typically, this Web Service will define one or more methods that:**
  - Establish a connection to a data source.
  - Create a DataSet, define the structure of the resulting Typed DataSet (by using an .xsd file).
  - Create an empty, local instance of the Strongly Typed DataSet.
  - Run a query or perform calculations, and fill the local DataSet. A DataAdapter is commonly used to fill the dataset.
  - Return the DataSet to the client application for further processing

Visual Basic Example

Introduction

Creating a Web service is similar to creating any component that provides programmatic access to its application logic. To create a Web service, you need some functionality that constitutes the service you wish to expose, a service description that defines how to use the service, and an infrastructure to support receiving and processing requests, and sending responses. Fortunately, much of the required infrastructure is generated automatically by Visual Studio .NET.

Building a Web service that returns data

To build a Web service that returns data, first create a new Web service by using Visual Studio. NET. Typically, this Web Service will define one or more methods that:

- Establish a connection to a data source.
- Create a DataSet, define the structure of the resulting Typed DataSet (by using an .xsd file).
- Create an empty, local instance of the Strongly Typed DataSet.
- Run a query or perform calculations, and fill the local DataSet. A DataAdapter is commonly used to fill the DataSet.
- Return the DataSet to the client application for further processing.

Example

The following example defines a Web method that takes a customer's city as input, queries the **customers** table in the Northwind database, and returns a DataSet with information about all the customers in that city.

This example assumes that a connection to the Northwind database exists.

```
'Connect to the Northwind DataBase
Dim myCn as new SqlConnection

myCn.ConnectionString = "data source=localhost;initial
catalog=Northwind" & _
"integrated security=SSPI;persist" & _
"security info=false"

myCn.open
```

This example assumes that a SqlDataAdapter has been defined with the following parameterized query:

```
SQLDataAdapter1.SelectCommand.CommandText = _
SELECT CustomerId, CompanyName, ContactName, Address, City,
Region, PostalCode, Country, Phone, Fax
FROM Customers
WHERE (City like @city)
```

Note When you create parameterized queries using the SQLDataAdapter, use named arguments to mark parameters.

When you create parameterized queries using the OLEDBDataAdapter, use the "?" character to mark parameters

```
'Set a query parameter for an OLDB data source

OLEDBDataAdapter1.SelectCommand.CommandText = _
SELECT CustomerId, CompanyName, ContactName, Address, City,
Region, PostalCode, Country, Phone, Fax
FROM Customers
WHERE (City like ?)



'Example of a Web Service that returns a DataSet
Imports System.Web.Services
Public Class Service1
  Inherits System.Web.Services.WebService

'This method accepts a city name as a query parameter
<WebMethod()> Public Function GetCustomers(ByVal city as
String) as CustDS

'Create an instance of a Typed DataSet to hold the information
'retrieved from SQL Server
      Dim ds as New CustDS()

'Set the city parameter of the query, 0 is the first in
'the collection
      SqlDataAdapter1.SelectCommand.Parameters(0).Value = city

      'Fill the local DataSet with the results
      SqlDataAdapter1.fill(ds)

      'Pass the results to the calling program
      Return ds

End Function

End Class
```

Practice

A salesperson for Northwind Traders travels to various cities in order to visit customers and take orders. Since customer information changes frequently, the Customer Information application uses an XML Web service to retrieve information from the company's central database about customers in a particular city. The salesperson uses the application to generate the list of customers to visit while in a particular city.

► **To create a Web service**

1. Start Visual Studio.net and create a new project. Use the information in the following table.

Option	Value
Project Type	Visual Basic or Visual C#
Template	ASP.Net Web service
Name	CustomerInfoService

2. Use the Server Explorer to add a new connection to the Northwind database on your local SQL Server.
3. Double Click on Service1.asmx in the Solution Explorer. This displays the design surface for the Web service.
4. Click the Data tab of the toolbox. Drag a SqlDataAdapter to the design surface. Use the information in the following table to configure the SqlDataAdapter by using the wizard.

Option	Value
Connection	Localhost.Northwind.dbo
Query type	Use SQL statement

Query	SELECT CustomerId, CompanyName, ContactName, Address, City, Region, PostalCode, Country, Phone, Fax
	FROM Customers
	WHERE (City like @city)

5. Right click on SQLDataAdapter1 and then click Generate DataSet. Use the information in the following table.

Property	Value
New DataSet Name	CustDS
Tables	Customer
Add DataSet to the designer	Checked

6. Right click CustDS.xsd and then click View Schema. Examine the generated schema.

► To create a Web method that returns a dataset

1. View the code for Service1.asmx.
2. Create a new Web method by inserting the following code after the commented example.

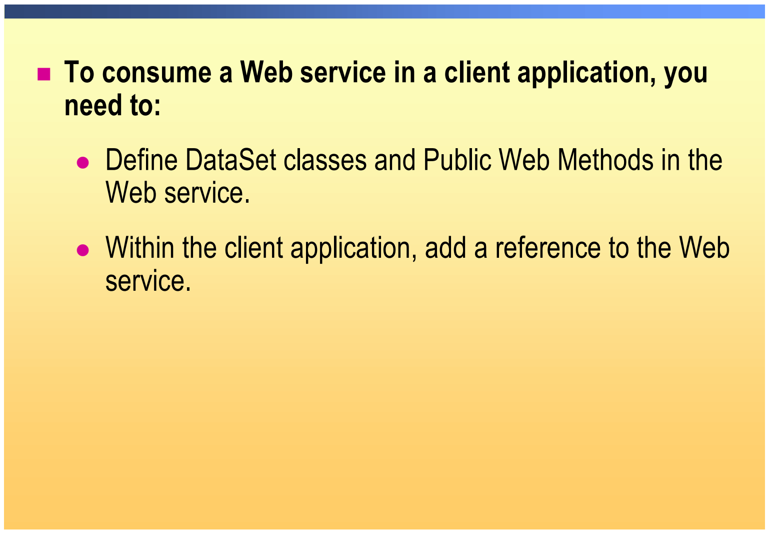
```
<WebMethod()> Public Function GetCustomers(ByVal city as _
String) as CustDS
   Dim ds as New CustDS()
   SqlDataAdapter1.SelectCommand.Parameters(0).Value = city
   SqlDataAdapter1.fill(ds)
   Return ds
End Function
```

3. Build the project.

► To test the Web service

1. Right click Service1.asmx and then click Browse With…Choose Microsoft Internet Explorer from the list.
2. Right click Service1.asmx and then click View in Browser.
3. Examine the default page generated to describe the Web service.
4. Click GetCustomers.
5. Test the Web method by using “London” as the parameter value. Click Invoke.
6. Examine the XML returned by the Web service. Notice that it contains both schema and data.

How to Consume a Web Service



- To consume a Web service in a client application, you need to:
  - Define DataSet classes and Public Web Methods in the Web service.
  - Within the client application, add a reference to the Web service.

Visual Basic Example

Introduction

Because Web services are accessible using URLs, HTTP, and XML it means programs running on any platform and in any language can access Web services. Because the decentralized nature of Web services enables both the client and the Web service to function as autonomous units, there are countless ways to consume a Web service.

For example, a call to a Web service can be included in a Web application, a middleware component, or even another Web service. No matter what form the Web service client may take, all that's needed to call a Web service is to send a properly formatted request message that conforms to the published service description for that Web service. Depending upon the nature of the Web service, it may send a response message in return. The originator of the request must then be capable of extracting the necessary information from this message.

To consume a Web service

To consume a Web service in a client application, you need to:

- Define DataSet classes and Public Web Methods in the Web service.
- Within the client application, add a reference to the Web service.

When you create a Web Reference to a Web service, the classes and methods defined in the Web Service are then available for use in the client application. You can also use Universal Description, Discovery, and Integration (UDDI) to find out what is available on the Web service.

When you find the Web service that you want to access, you need to add a Web reference to that Web service within the client application. This allows you to access the Web services classes and method as if they were local to your client application

Example

The following example retrieves a DataSet from a Web service. In this application, a client form sends the City where customers live as a parameter to a Web service. The Web service connects to SQL Server and executes a query to retrieve a list of the customers in that city. The results are sent to the client as a strongly typed DataSet.

The client application receives the DataSet and uses it to populate the local cache. This cache is a DataSet of the same type defined by the Web service. The application binds a DataGrid control to the local cache to display the results.

The form in the client application contains a textbox to record the choice of city, a datagrid to display results, and a button process the request for information.

A complete code listing for this example is included in ClientList.txt.

```
Public Class Form_ClientList
  Inherits System.Windows.Forms.Form

‘Create a new Strongly Typed DataSet based on the one declared
‘in the Web service.
‘A Web reference to the Web service that defines CustDS
‘already exists.

         Public CustDS1 as New ClientList.localhost.CustDS()
.        Private myCity as String
.
.
‘Contact the Web Service, execute the query and retrieve
‘results

Private Sub Btn_GetClients_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
Btn_GetClients.Click

‘Reference the Web service.
‘This make the Web methods available to the client
application.

Dim ws As New ClientList.localhost.Service1()

‘Get the city parameter from the form
myCity = txt_city.text

‘Use the Web method to retrieve results.
‘Merge the results into the local cache.

CustDS1.Merge(ws.GetCustomers(myCity))

End Sub

End Class
```

Practice

A salesperson for Northwind Traders travels to various cities in order to visit customers and take orders. Since customer information changes frequently, the Customer Information application uses an XML Web service to retrieve information from the company's central database about customers in a particular city. The salesperson uses the application to generate the list of customers to visit while in a particular city.

In this practice, you will build a simple client application to consume the Web service.

► To create a client form

1. Start Visual Studio.net and create a new project. Use the information in the following table

Option	Value
Project Type	Visual Basic or Visual C#
Template	Windows Application
Name	ClientList

2. Add a button to the form. Use the information in the following table:

Property	Value
Name	btn_GetClients
Text	List Customers
Dock	bottom

3. Add a textbox to the form. Use the information in the following table:

Property	Value
Name	Txt_city
Text	Enter a city
Dock	Top

4. Add a data grid to the form. Use the information in the following table:

Property	Value
Name	Dgr_CustGrid
Dock	Fill

► To add a Web reference

1. In the Solution Explorer, right click the References folder in the ClientList project, and then click Add Web Reference.
2. Click Web References on Local Web Server and then click the link to http://localhost/CustomerInfoService/CustomerInfoService.vsdisco.
3. Click Add Reference.
4. In the Solution Explorer, expand the Web References folder of the ClientList project. Notice that CustDS.xsd describes the schema of the DataSet returned by the Web service.
5. In the Class View, expand ClientList. Notice that a class named CustDS has been created. In the next procedure you will create a strongly typed data set based on this class.

► To use the methods and classes from a Web service

1. From the Data tab of the toolbox, add a new DataSet to the form. Use the information in the following table

Option	Value
Typed dataset	Selected
Name	ClientList.localhost.CustDS

2. Set the Data Source property of the data grid to CustDS1.Customers
3. Copy the following code to btn_GetClients_click procedure:

```
Dim ws as New ClientList.localhost.Service1()
CustDS1.Merge(ws.GetCustomers(txt_city.text))
```

4. Save the form and build the solution.

► To test your application

1. Start the Client list application in Debug mode.
2. Type “London” in the text box, and then click Retrieve Customers.
3. Examine the results.
4. Close the application.

Review

- What is a Web Service?
- How to Build a Web Service That Returns Data
- How to Consume a Web Service

1.

2.

3.

4.

5.

Lab 7: Troubleshooting an ADO .NET Application

Objectives

After completing this lab, you will be able to:

-
-
-

Prerequisites

Before working on this lab, you must have:

- Experience debugging .NET solutions.
-

For More Information

Scenario

Estimated time to complete this lab: 60 minutes

Exercise 0 Lab Setup

To complete this lab, you must …

► **To set up the XML Web service**

1. Check that the OnTheRoadWS XML Web service is running correctly by navigating to http://localhost/2389/labs/lab07/OnTheRoadWS/Check that the instructor's machine has a copy of the Employees table called EmployeesLatest that includes a tenth row for John Smith.

Exercise 1
Bug Fixin'

In this exercise, you will fix bugs in a Visual Studio .NET solution. The solution includes an XML Web service and a Windows client application that both use ADO .NET to allow sales people to track customers' orders while away from the office and the central sales database.

Scenario

You have recently been employed in the Northwind Traders IT Department. The previous programmer left for a dot com startup (and is now regretting it). You have inherited a project with bugs, and it is your first job to fix them.

The instructor will play the role of the database administrator (DBA) for the Northwind Traders IT Department. If you find DBA related problems, ask questions and negotiate with the DBA as you would in the real world.

► **To open the buggy solution**

1.

2.

3.

Solution

Here are the solutions to the bugs.

► **To fix all the bugs**

1. *Problem*. The **OnTheRoad** project is missing a reference to the **System.XML.dll** assembly. This causes six build errors related to classes defined in the **System.Xml** namespace.
   Solution. Right-click the **OnTheRoad** project and choose **Add Reference**, then select the **System.XML.dll** assembly and choose OK.
2. *Problem*. The **SalesManager.asmx** class is missing an **Imports** (Visual Basic) or a **using** (C#) statement for the **System.Web.Services** namespace. This causes two "Type is not defined: 'WebMethod'" build errors, because the <WebMethod()> attribute is defined in the **System.Web.Services** namespace.
   Solution. Add the following code to the top of the **SalesManager.asmx** class.

```
' Visual Basic
Imports System.Web.Services

// Visual C#
using System.Web.Services;
```

3. *Problem*. A new employee started recently. Their name is John Smith, but their name is not appearing in the application.
   Solution. The problem is that the connection string for the **cnNorthwindInstructor** connection in the **SalesManager.asmx** class uses **(local)** as the server name. Therefore the database used is the one on the same server as the web service. This must be changed to the instructors machine name, **London**, because that is the only server that contains the **EmployeesLatest** table. This error is very common when moving from a development or test environment to a production environment.

4. *Problem*. The connection string for the **cnNorthwindInstructor** connection in the **SalesManager.asmx** class uses a SQL login named **MaryJoe** with a password of **secret** that does not exist on the server.
   Solution. This problem frequently occurs when moving between servers, for example, when moving from a test server to a production server. The solution would be to either create a new login with the correct permissions, or use another login name and password. Use the **MaryJane** login instead. She has the same password.
5. *Problem*. The code that calls the GetDataSet web method is missing a well-written exception handler, so it is much harder to find out what is going wrong when the code fails.
   Solution. Add the following exception handling code *as a minimum*. If you have time, write more code to catch specific exceptions and display friendly error messages to the user.

```
' Visual Basic
Try
   tempNW = wsSalesMgr.GetDataSet( _
       Me.EmployeeID, Me.ServerName)
Catch Xcp As System.Exception
   MessageBox.Show(Xcp.ToString(), “Exception”)
End Try

// Visual C#
try
{
   tempNW = wsSalesMgr.GetDataSet(
       this.EmployeeID, this.ServerName);
}
catch (System.Exception Xcp)
{
   MessageBox.Show(Xcp.ToString(), “Exception”);
}
```

6. *Problem*. Instructor can put the Northwind database on the London server into single user mode (or run a stored procedure?) to limit the number of concurrent connections to less than the number of students.
   Solution. Contact the DBA and make sure the database does not limit connections for either reason.
7. *Problem*. The **SelectCustomers** command uses the **SelectClients** stored procedure which now has the wrong name (perhaps it was renamed by a DBA without telling the developers after they used the auto-gen. tools).
   Solution. Users can fix the name in their code. It should be changed to **SelectCustomers**.
8. *Problem*. A stored procedure has a parameter type mismatch (the same DBA altered the stored procedure, again without telling the developers!)
   Solution. The InsertCustomers command needs the parameter definition line for the @CustomerID parameter altered from a 4 byte integer to a 5 byte NChar.
9. *Problem*. Before calling the **Fill** method of the **daOrderDetails** data adapter the parameter value for the EmployeeID is not set properly, so the order detail rows displayed to the user are wrong.
   Solution. Change the variable name to use **iEmployeeID** instead of **EmployeeID** (which is always 0).

10. *Problem*. The SqlDataReader loop code for listing employees has a logic error. It uses the Read method in an If statement to check for records (True/False), and then a Do loop, thereby losing the first row.
    Solution.

11. *Problem*. DataGrid binding code uses a child table (Orders) instead of a parent table (Customers).
    Solution. Change the code in the **RefreshUI** method to bind to the Customers table.

12. *Problem*. Indexing problem with filtering/sorting DataView. e.g. n+1
    Solution.

13. *Problem*. XSD file does not define a relationship between the Orders and OrderDetails tables so the data grid does not recognize the relationship either.
    Solution. Users need to use the XSD Editor to manually add the XSD file. Open the NWDataSet.xsd file. Right-click the Orders table. Choose Add-New Relation. Parent element Orders. Child element OrderDetails. Choose OK.

14. *Problem*. Case wrong for an element in the XSD. Will the Editor fix/ignore this?
    Solution.

15. *Problem*. Missing line of code to append a new DataRow to the Rows collection when adding a new AppSettings row.
    Solution.

16. *Problem*. Not using Merge method when trying to combine to DataSets, instead assigning one to the other thereby destroying the first.
    Solution. Change the code to use the **Merge** method.

    ```
    ' Visual Basic
    dsChanges.Merge(Me.dsNorthwind.AppSettings)

    // Visual C#
    dsChanges.Merge(this.dsNorthwind.AppSettings);
    ```

17. *Problem*. Code incorrectly calls **AcceptChanges** method before calling Update meaning that the marked changes are "lost" and not sent to the central database. On the next call to the Fill method the DataSet will revert to the underlying values again.
    Solution. Delete the call to the **AcceptChanges** method before calling the **GetChanges** method in the mnuUpdate_Click procedure.

18. *Problem*. The CommandText property of the SelectCommand of a SqlDataAdapter daEmployees has been mis-fixed so there is a missing space character before the ORDER BY clause.
    Solution. Manually add the space back, or re-run the Wizard to regenerate statement.

19. *Problem*. A call to the Fill method has accidentally swapped the DataSet and DataAdapter giving a compile error e.g. ds.Fill(da, "table") is wrong!
    Solution. Swap the data set and data adapter references.

20. *Problem*. Calls to commit or rollback transactions have been reversed. The Server Explorer shows the data changes have not been made.
    Solution.

msdn® training

Appendix A: Best Practices for Writing SQL Statements and Stored Procedures (Prerelease)

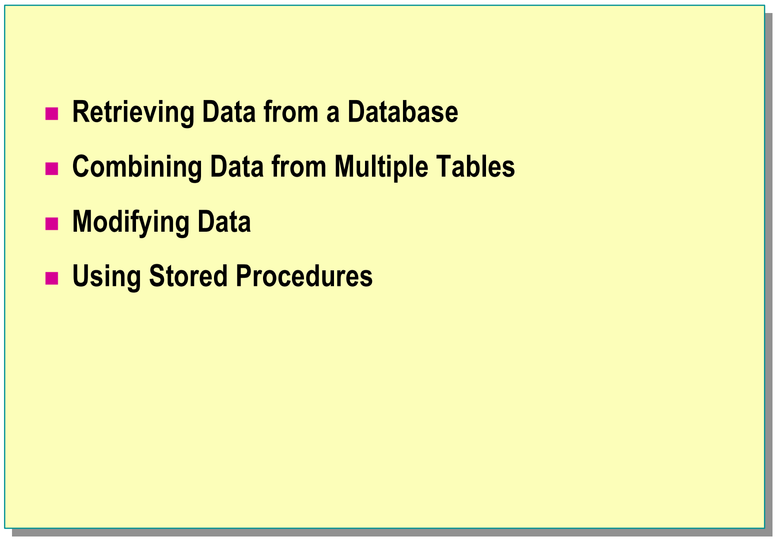
Contents

Microsoft®

Instructor Notes

Instructor_notes.doc

Overview



- Retrieving Data from a Database
- Combining Data from Multiple Tables
- Modifying Data
- Using Stored Procedures

Introduction

This appendix describes the main features of SQL. SQL is a standard language for retrieving data from a relational database and for modifying the data in the database. You use SQL statements in Microsoft® ActiveX® Data Objects (ADO) .NET applications, to retrieve and modify data in the database.

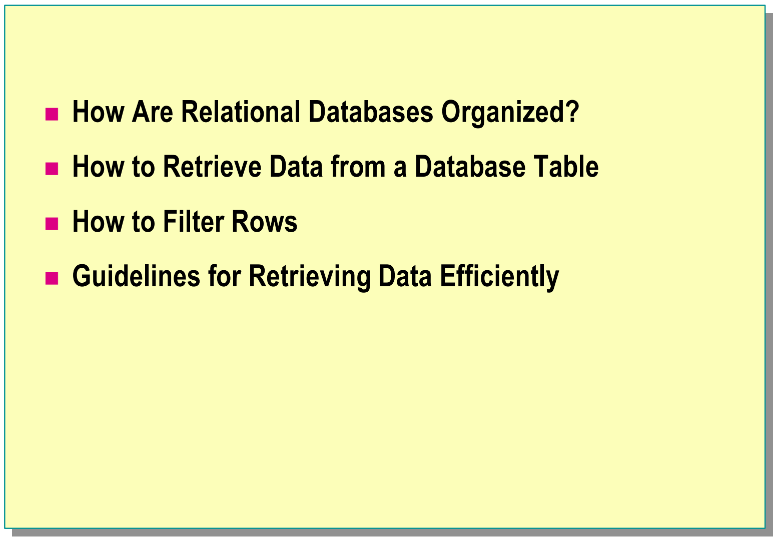
In this appendix, you will learn how to write SQL statements to retrieve data from a single table or from multiple tables.

Objectives

After completing this appendix, you will be able to:

- Retrieve data from a database.
- Join data in different tables.
- Insert, update, and delete data in a database.
- Create and call stored procedures.

Lesson: Retrieving Data from a Database



- How Are Relational Databases Organized?
- How to Retrieve Data from a Database Table
- How to Filter Rows
- Guidelines for Retrieving Data Efficiently

Introduction

In this lesson, you will see how to retrieve data from a relational database.

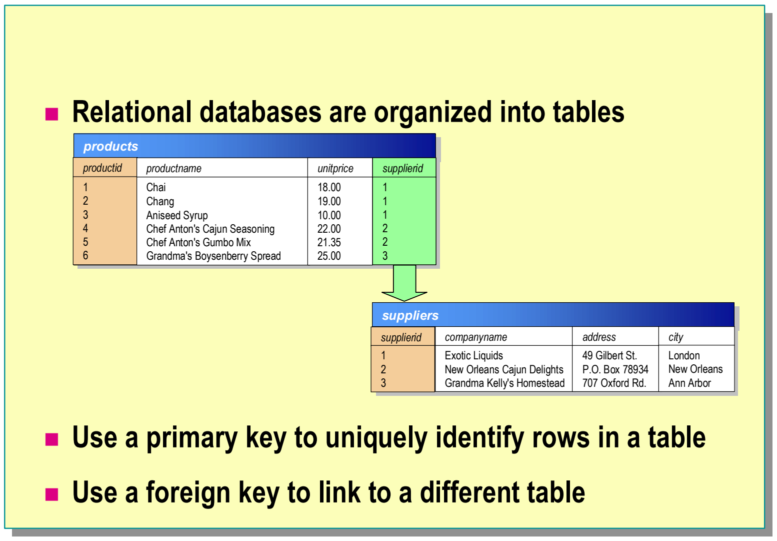
This lesson shows how to use the SELECT statement to retrieve data from a database. The lesson also shows how to filter data by using the WHERE clause, and describes performance considerations that affect retrieving data.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the structure of relational databases.
- Retrieve data from a database by using the SELECT statement.
- Filter data by using search conditions with the WHERE clause.
- Describe performance considerations that affect retrieving data.

How Are Relational Databases Organized?



- Relational databases are organized into tables

products

productid	productname	unitprice	supplierid
1	Chai	18.00	1
2	Chang	19.00	1
3	Aniseed Syrup	10.00	1
4	Chef Anton's Cajun Seasoning	22.00	2
5	Chef Anton's Gumbo Mix	21.35	2
6	Grandma's Boysenberry Spread	25.00	3

suppliers

supplierid	companyname	address	city
1	Exotic Liquids	49 Gilbert St.	London
2	New Orleans Cajun Delights	P.O. Box 78934	New Orleans
3	Grandma Kelly's Homestead	707 Oxford Rd.	Ann Arbor

- Use a primary key to uniquely identify rows in a table
- Use a foreign key to link to a different table

Introduction

Relational databases are organized into tables. Each table contains rows and columns. The rows represent records. The columns define the data items in each record. A key is a column, or group of columns, that contains a unique attribute or attributes that you use to identify a row in a table. For example, each employee must have a unique identification number. In an employee table, the employee's identification number uniquely identifies a row for each employee.

Scenario

When you design a database, you define tables to represent each related group of data. For example, a retailer company might design a database with tables named **Customers**, **Products**, and **Orders**.

Definition of primary key

Use a primary key to uniquely identify rows in a table. When you design a database table, you must specify one or more columns as the primary key. The primary key uniquely identifies rows in the table. Each row has a unique value for its primary key that distinguishes it from all other rows

Definition of foreign key

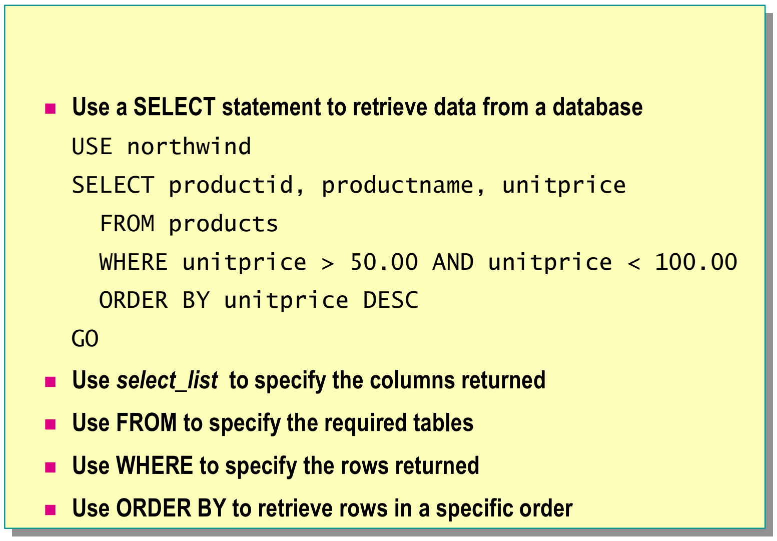
Use a foreign key to link to a different table. When you design a database table, you can specify foreign keys in the table. A foreign key in one table refers to a primary key in another table. The value of the foreign key identifies a particular row in the other table. A foreign key can be a single column or a combination of columns. You can define any number of foreign keys in a table.

Guidelines for defining a table

Consider the following facts and guidelines when defining a table:

- Each column in the table must represent an atomic piece of data.
- You must choose an appropriate name for each column.
- You must choose an appropriate data type for each column.
- You must identify one or more columns in the table as the primary key.
- You can define foreign keys to establish links to primary keys in other tables.

How to Retrieve Data from a Database



- **Use a SELECT statement to retrieve data from a database**

```
USE northwind
SELECT productid, productname, unitprice
  FROM products
  WHERE unitprice > 50.00 AND unitprice < 100.00
  ORDER BY unitprice DESC
GO
```

- **Use *select_list* to specify the columns returned**
- **Use FROM to specify the required tables**
- **Use WHERE to specify the rows returned**
- **Use ORDER BY to retrieve rows in a specific order**

Introduction

Use a SELECT statement to retrieve data from a database. In a **SELECT** statement, you specify the columns and rows that you want a query to return from a table.

Partial syntax for SELECT statement

The partial syntax for the SELECT statement is as follows:

```
SELECT [ALL | DISTINCT] select_list
    FROM {table_source} [,…n]
  [ WHERE search_condition ]
  [ ORDER BY column_name [ASC | DESC] ]
```

Specifying the columns returned

Use *select_list* to specify the columns returned. Consider the following facts and guidelines for the select list:

- The select list retrieves and displays the columns in the specified order.
- Separate the column names with commas. Do not place a comma after the final column name. .
- Avoid or minimize the use of an asterisk (*) in the select list. An asterisk retrieves all columns from a table.

Specifying the required tables

Use a FROM clause to specify the required tables. Consider the following facts and guidelines for the FROM clause:

- The FROM clause is mandatory. You must provide at least one table name.
- You can provide multiple table names to retrieve data from several tables. Use a comma to separate each table name.

Specifying the rows returned

Use a WHERE clause to specify the rows returned. Consider the following facts and guidelines for the WHERE clause:

- The WHERE clause is optional. If you do not provide a WHERE clause, you will retrieve all the rows in the specified table.
- You can define search conditions in the WHERE clause, to restrict the rows to return. The search conditions can include string comparisons, numerical comparisons, and other conditional tests.

Specifying the row order

Use an ORDER BY clause to retrieve rows in a specific order. Consider the following facts and guidelines for the ORDER BY clause:

- The ORDER BY clause is optional. If you do not provide an ORDER BY clause, you will retrieve rows in the order that they appear in the table.
- If you provide an ORDER BY clause, specify which column the database engine must use for sorting the rows.
- You can provide an optional ASC or DESC clause to sort rows in ascending or descending order. The default sorting order is ascending.

Example of using the SELECT statement

The USE statement in this example determines the database that your query acts on. The SELECT statement retrieves the **productid**, **productname**, and **unitprice** columns from the **Products** table in the Northwind Traders database. The WHERE clause restricts the rows returned so that only products costing more than $50 and less than $100 are returned. The ORDER BY clause sorts rows in order of descending unit price.

```
/* Retrieve the product ID, product name, and unit price
   for products costing more than $50 and less than $100.
   Retrieve products in order of descending price */
USE northwind
SELECT productid, productname, unitprice
  FROM products
  WHERE unitprice > 50.00 AND unitprice < 100.00
  ORDER BY unitprice DESC
GO
```

Result

The SELECT statement in the preceding example produces the following result:

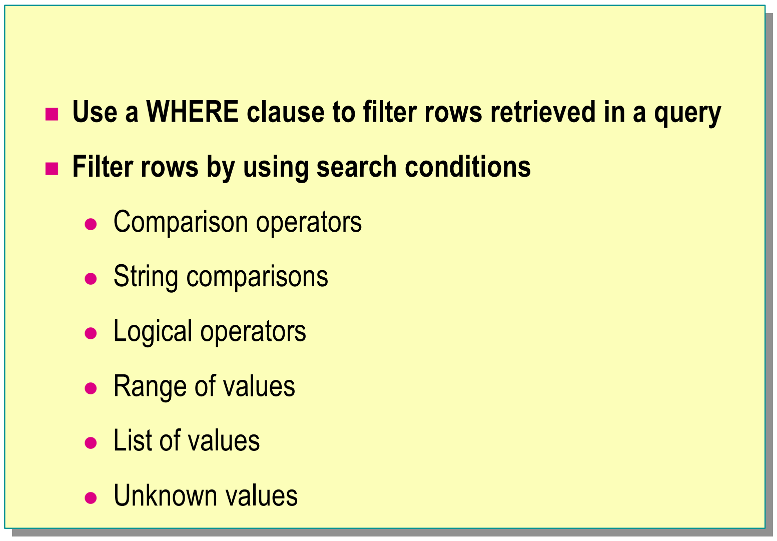
```
productid      productname                unitprice
9              Mishi Kobe Niku            97.00
20             Sir Rodney's Marmalade     81.00
18             Carnarvon Tigers           62.50
59             Raclette Courdavault       55.00
51             Manjimup Dried Apples      53.00
```

Practice

The **Products** table has a column named **unitsinstock**, which gives the number of units currently in stock for each product. There is also a column named **reorderlevel**, which indicates the minimum number of units to maintain in stock for each product.

Rewrite the SELECT statement in the previous example, to retrieve the **productid**, **productname**, **unitsinstock**, and **reorderlevel** columns for all products that have fallen below the reorder level. There is no need to sort the rows in any particular order.

How to Filter Rows



- **Use a WHERE clause to filter rows retrieved in a query**
- **Filter rows by using search conditions**
  - Comparison operators
  - String comparisons
  - Logical operators
  - Range of values
  - List of values
  - Unknown values

Introduction

Use a WHERE clause to filter rows retrieved in a query. You can retrieve just the rows that you need from the database, rather than retrieve all the rows.

Filtering rows

Filter rows by using search conditions. The following table describes the types of filters and the corresponding search condition that you can use to filter data.

Type of filter	Usage	Search condition
Comparison operators	Use comparison operators to compare values in a table to a specified value or expression.	=, >, <, >=, <=, and <>
String comparisons	Use the LIKE and NOT LIKE search conditions with wildcard characters to select rows by comparing character strings.	LIKE and NOT LIKE
Logical operators	Use the AND, OR, and NOT logical operators to combine a series of expressions and to refine query processing. Use parentheses to enforce or emphasize the order of evaluation of the test conditions.	AND, OR, and NOT

(*continued*)

Type of filter	Usage	Search condition
Range of values	Use the BETWEEN and NOT BETWEEN operators to retrieve rows within or outside a specified range of values. Microsoft SQL Server™ includes the end values in the result set. This means you can use BETWEEN rather than an expression such as (>= x AND <= y). Likewise, you can use NOT BETWEEN rather than an expression such as (< x OR > y).	BETWEEN and NOT BETWEEN
Lists of values	Use the IN search condition to retrieve rows that match a specified list of values. Use the NOT IN search condition to retrieve rows that do not match a specified list of values.	IN and NOT IN
Unknown values	Use the IS NULL search condition to retrieve rows where information is missing from a specified column. A column has a null value if no value is entered during data entry and no default values are defined for that column. Use the IS NOT NULL search condition to retrieve rows that do not have a null value for a specified column.	IS NULL and IS NOT NULL

Example of using comparison operators

This example retrieves the last name and city of residence of employees who reside in the United States from the **Employees** table.

```
/* Retrieve information for employees who reside in the US */
USE northwind
SELECT lastname, city
  FROM employees
  WHERE country = 'USA'
GO
```

Using string comparisons

The following table describes characters to use for string comparisons.

Wildcard	Description
%	Any string of zero or more characters
-	Any single character
[ ]	Any single character in the specified range or set
[^]	Any single character not in the specified range or set

Example of using string comparisons

This example retrieves companies from the **Customers** table that have the word "Restaurant" somewhere in their company names:

```
/* Retrieve company names that contain the word restaurant */
USE northwind
SELECT companyname
  FROM customers
  WHERE companyname LIKE '%Restaurant%'
GO
```

Example of using logical operators

This example retrieves all products with product names that begin with the letter T, or have a product ID of 46, and that have a price greater than $10:

```
/* Retrieve products that start with T or have a product ID
   of 46, and cost more than $10 */
USE northwind
SELECT productid, productname, supplierid, unitprice
  FROM products
  WHERE (productname LIKE 'T%' OR productid = 46)
    AND (unitprice > 10)
GO
```

Example of using a range of values

This example retrieves the product name and unit price of all products with a unit price between $10 and $18, inclusive:

```
/* Retrieve products costing between $10 and $18 inclusive */
USE northwind
SELECT productname, unitprice
  FROM products
  WHERE unitprice BETWEEN 10 AND 18
GO
```

Example of using a list of values

This example produces a list of companies from the **Suppliers** table that are located in Japan or Italy:

```
/* Retrieve suppliers in Japan or Italy */
USE northwind
SELECT companyname, country
  FROM suppliers
  WHERE country IN ('Japan', 'Italy')
GO
```

Example of retrieving unknown values

This example retrieves a list of companies from the **Suppliers** table for which the **fax** column contains a null value:

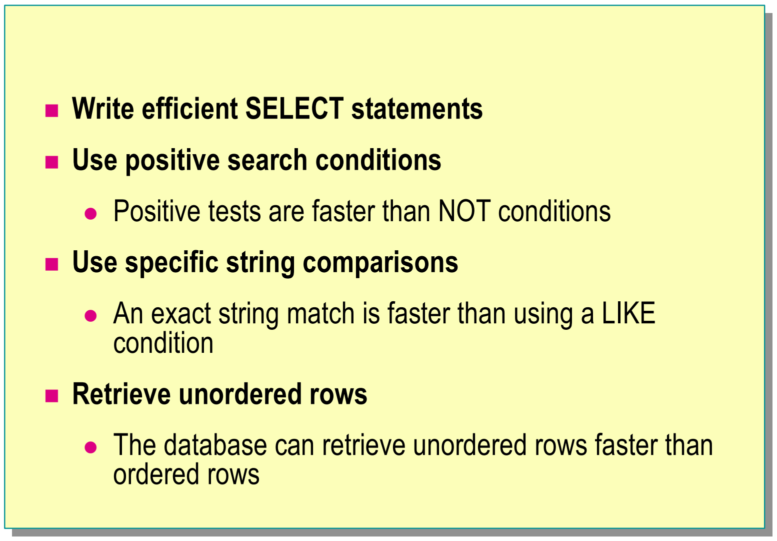
```
/* Retrieve suppliers that have a null fax */
USE northwind
SELECT companyname, fax
  FROM suppliers
  WHERE fax IS NULL
GO
```

Practice

Write an SQL SELECT statement to retrieve the product name, unit price, and number of units in stock for products that match all of these conditions:

- The product name must start with the letter T.
- The unit price must be between $5 and $10, inclusive.
- The number of units in stock must be at least 25.

Guidelines for Retrieving Data Efficiently



- **Write efficient SELECT statements**
- **Use positive search conditions**
  - Positive tests are faster than NOT conditions
- **Use specific string comparisons**
  - An exact string match is faster than using a LIKE condition
- **Retrieve unordered rows**
  - The database can retrieve unordered rows faster than ordered rows

Introduction

Write efficient SELECT statements. This improves the performance of your application, because data is retrieved as quickly as possible. It also lightens the load on the database engine, because it has to do less work to retrieve your data.

Using positive search conditions

Use positive search conditions. Positive search conditions such as BETWEEN, IN, and IS NULL are typically more efficient than negative search conditions such as NOT BETWEEN, NOT IN, and IS NOT NULL.

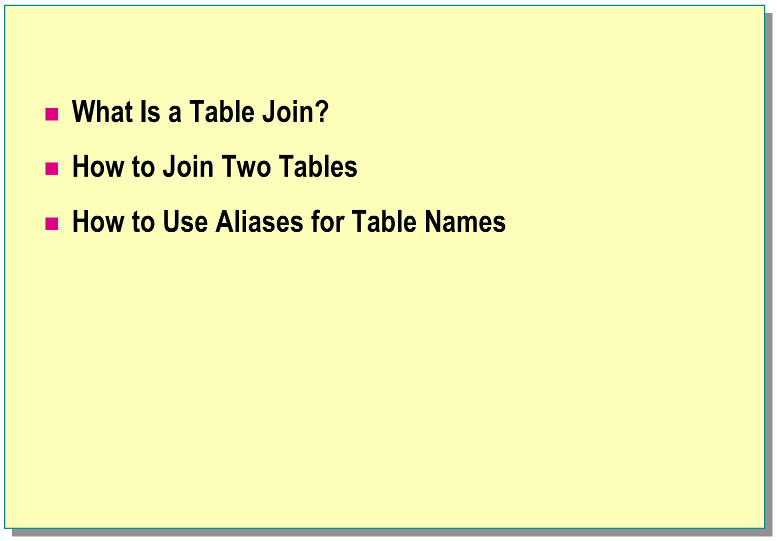
Using specific string comparisons

Use specific string comparisons. String comparisons using = and <> are typically more efficient than those using the LIKE search condition.

Retrieving unordered rows

Retrieve unordered rows. Data retrieval may decrease if you use the ORDER BY clause, because the database engine must determine and sort the result set before it returns the first row.

Lesson: Combining Data from Multiple Tables



- What Is a Table Join?
- How to Join Two Tables
- How to Use Aliases for Table Names

Introduction

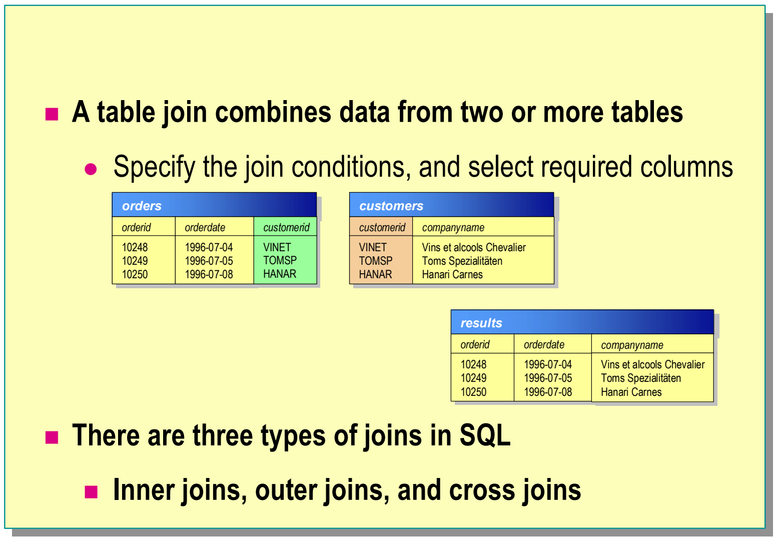
This lesson shows how to use the JOIN keyword to join tables. The result set includes rows and columns from each table. You will see how to use the ON keyword to define the join condition. You will also see how to provide aliases for table names to simplify the syntax in a complex query.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe why joins are important in relational databases.
- Combine data from separate tables by using joins.
- Use aliases for table names.

What Is a Table Join?



- **A table join combines data from two or more tables**
  - Specify the join conditions, and select required columns

orders

orderid	orderdate	customerid
10248	1996-07-04	VINET
10249	1996-07-05	TOMSP
10250	1996-07-08	HANAR

customers

customerid	companyname
VINET	Vins et alcools Chevalier
TOMSP	Toms Spezialitäten
HANAR	Hanari Carnes

results

orderid	orderdate	companyname
10248	1996-07-04	Vins et alcools Chevalier
10249	1996-07-05	Toms Spezialitäten
10250	1996-07-08	Hanari Carnes

- **There are three types of joins in SQL**
  - **Inner joins, outer joins, and cross joins**

Introduction

A table join combines data from two or more tables. Use JOIN statements to query any number of tables in the database to produce a single result set that contains merged data from these tables.

Joins are an essential part of relational database theory. You can write a JOIN statement to combine data from multiple tables in any way you like, depending on the needs of your application.

Using joins

There are three types of joins in SQL: inner joins, outer joins, and cross joins. The following table describes each type of join.

Type of join	Description
Inner join	An inner join combines tables by comparing values in common columns in the tables. The database engine returns only rows that match the join conditions. Inner joins are most often used.
Outer join	An outer join combines rows from two tables that match the join condition, plus any unmatched rows in the first or second table. The LEFT OUTER JOIN clause retrieves all rows from the first-named table, plus the rows in the second-named table that match the join condition. The RIGHT OUTER JOIN clause retrieves all rows from the second-named table, plus the rows in the first-named table that match the join condition.
Cross join	A cross join retrieves every combination of all rows in the joined tables. You do not need to specify a common column to use cross joins. One use of cross joins is to generate test data for a database.

Example of using an inner join

The Northwind Traders database has a table named **Orders**, which contains information about all the orders received by Northwind Traders. The following table shows some of the data in the **Orders** table.

orderid	**orderdate**	**customerid**
10248	1996-07-04	VINET
10249	1996-07-05	TOMSP
10250	1996-07-08	HANAR

The database also has a table named **Customers**, which contains information about all the customers that have placed an order with Northwind Traders. The following table shows some of the data in the **Customers** table.

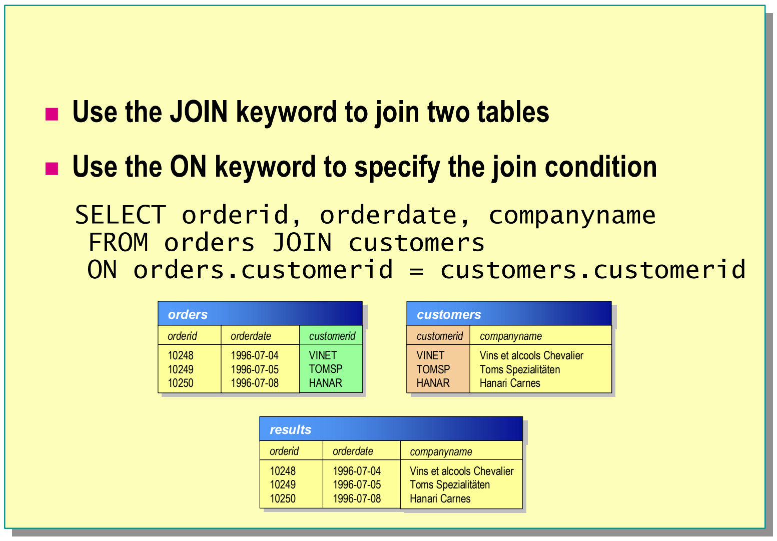
customerid	**companyname**
VINET	Vins et alcools Chevalier
TOMSP	Toms Spezialitäten
HANAR	Hanari Carnes

You can define an inner join on the **Orders** and **Customers** tables, to retrieve detailed information about the customer who placed each order. Use the customerid foreign key in the **Orders** table to identify the customer who placed each order.

Practice

Open the Northwind Traders database in SQL Server Query Analyzer. Examine the **Orders** table, and identify the foreign keys in this table. Consider how you can use these foreign keys to join the **Orders** table to other tables in the Northwind Traders database.

How to Join Two Tables



- Use the JOIN keyword to join two tables
- Use the ON keyword to specify the join condition

```
SELECT orderid, orderdate, companyname
 FROM orders JOIN customers
 ON orders.customerid = customers.customerid
```

orders

orderid	*orderdate*	*customerid*
10248	1996-07-04	VINET
10249	1996-07-05	TOMSP
10250	1996-07-08	HANAR

customers

customerid	*companyname*
VINET	Vins et alcools Chevalier
TOMSP	Toms Spezialitäten
HANAR	Hanari Carnes

results

orderid	*orderdate*	*companyname*
10248	1996-07-04	Vins et alcools Chevalier
10249	1996-07-05	Toms Spezialitäten
10250	1996-07-08	Hanari Carnes

Introduction

Use the JOIN keyword to join two tables. The JOIN keyword specifies which tables are to be joined and how to join them. Inner joins are used in most situations and are the default type of join in SQL Server.

Partial syntax for an inner join

The syntax for an inner join is as follows:

```
SELECT select_list
  FROM {first_table_source} [,…n]
  JOIN joined_table_source
  ON join_condition
```

Defining an inner join

Use the ON keyword to specify the join condition for an inner join. When you define an inner join, consider the following facts and guidelines:

- Specify the tables you want to join.
- Limit the number of tables in a join. The more tables that you join, the longer it takes the database engine to process your query.
- Use a foreign key in one table to link to a primary key in another table. The columns must have the same or similar data types.
- If a table has a composite key, you must reference the entire key when you join tables.
- Select required columns from the joined tables.
- If any columns have the same name in both tables, use the table name to qualify the column name. Use the syntax *table_name.column_name*.

Example of defining an inner join on two tables

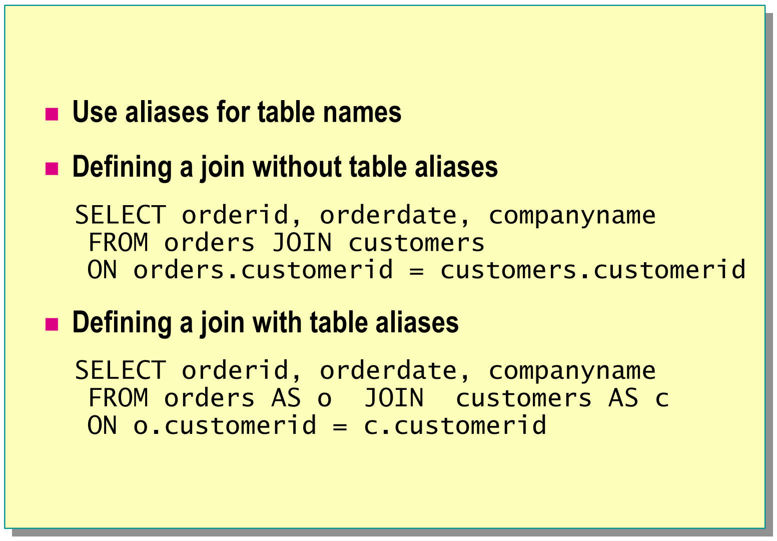
This example performs an inner join on the **Orders** and **Customers** tables to get the name of the customer who placed each order. The join condition uses the **customerid** column in each table.

```
/* Join the orders and customers tables, using the customerid
   as the join condition */
USE northwind
SELECT orderid, orderdate, companyname
  FROM orders JOIN customers
  ON orders.customerid = customers.customerid
GO
```

Practice

Rewrite the SELECT statement in the previous example to perform an inner join on the **Orders** and **Employees** tables. For each order, retrieve the order ID, the order date, and the last name of the employee who took the order.

How to Use Aliases for Table Names



- Use aliases for table names
- Defining a join without table aliases

```
SELECT orderid, orderdate, companyname
 FROM orders JOIN customers
 ON orders.customerid = customers.customerid
```

- Defining a join with table aliases

```
SELECT orderid, orderdate, companyname
 FROM orders AS o  JOIN  customers AS c
 ON o.customerid = c.customerid
```

Introduction

Use aliases for table names, to simplify complex query statements. Using aliases for table names makes Transact-SQL scripts easier to read and to maintain.

You can replace a long and complex fully qualified table name with a simple, abbreviated alias name when writing scripts. You use an alias name in place of the full table name.

Partial syntax for table aliases

The syntax for table aliases is as follows:

```
SELECT select_list
  FROM table_source AS table_alias
```

Example of defining a join without using table aliases

This example performs a join on the **Orders** and **Customers** tables. The example does not use table aliases. This means that you must write the names of the **Orders** and **Customers** tables in full in the join condition.

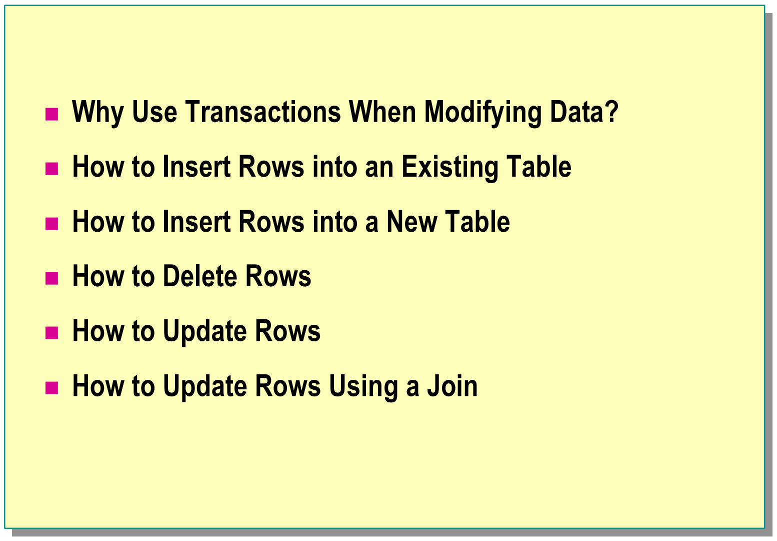
```
/* Perform a join without using table aliases */
USE northwind
SELECT orderid, orderdate, companyname
  FROM orders JOIN customers
  ON orders.customerid = customers.customerid
GO
```

Example of defining a join using table aliases

This example performs the same join as the previous example. However, this example uses aliases for the **Orders** and **Customers** tables. The aliases are used to simplify the join condition.

```
/* Perform a join using table aliases */
USE northwind
SELECT orderid, orderdate, companyname
  FROM orders AS o  JOIN  customers AS c
  ON o.customerid = c.customeridGO
```

Lesson: Modifying Data



- Why Use Transactions When Modifying Data?
- How to Insert Rows into an Existing Table
- How to Insert Rows into a New Table
- How to Delete Rows
- How to Update Rows
- How to Update Rows Using a Join

Introduction

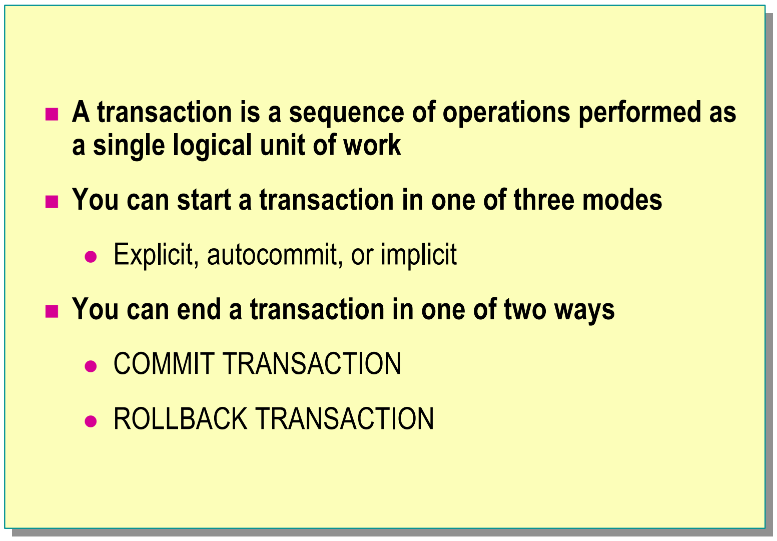
In this lesson, you will learn how to modify data in a database. You have already seen how to query data by using a SELECT statement. In this lesson, you will see how to write INSERT, DELETE, and UPDATE statements to change the data in the database.

Lesson Objectives

After completing this lesson, you will be able to:

- Use transactions to enforce the logical consistency of data.
- Write INSERT statements to insert rows into a new or existing table.
- Write DELETE statements to delete rows.
- Write UPDATE statements to update existing rows.

Why Use Transactions When Modifying Data?



- A transaction is a sequence of operations performed as a single logical unit of work
- You can start a transaction in one of three modes
  - Explicit, autocommit, or implicit
- You can end a transaction in one of two ways
  - COMMIT TRANSACTION
  - ROLLBACK TRANSACTION

Introduction

A transaction is a sequence of operations performed as a single logical unit of work. You are responsible for starting and ending transactions at points that enforce the logical consistency of the data. You must define a sequence of data modifications that leaves the data in a consistent state relative to the organization's business rules.

Scenario

You can use transactions when you make changes to several tables and you need to ensure all of the changes are performed successfully. If any problems arise, you can cancel all of the changes. For example, you might want to withdraw money from a savings account and deposit the money into a checking account. If any errors occur during the transaction, both accounts are restored to their original states.

Starting a transaction

You can start a transaction in one of three modes: explicit, autocommit, or implicit. The following table describes these modes.

Transaction mode	Description
Explicit	To start an explicit transaction, use the BEGIN TRANSACTION statement.
Autocommit	Autocommit transactions are the default in SQL Server. Each individual Transact-SQL statement is committed as soon as it is executed. You do not have to specify any statements to control transactions.
Implicit	Implicit transactions mode is set by an application programming interface (API) function or the Transact-SQL statement SET IMPLICT_TRANSACTIONS ON. Using this mode, the next statement automatically starts a new transaction. When the transaction is complete, the next Transact-SQL statement implicitly starts a new transaction.

Ending a transaction

You can end a transaction in one of two ways. To commit a transaction, use the COMMIT TRANSACTION statement. To rollback a transaction, use the ROLLBACK TRANSACTION statement. The following table describes these statements.

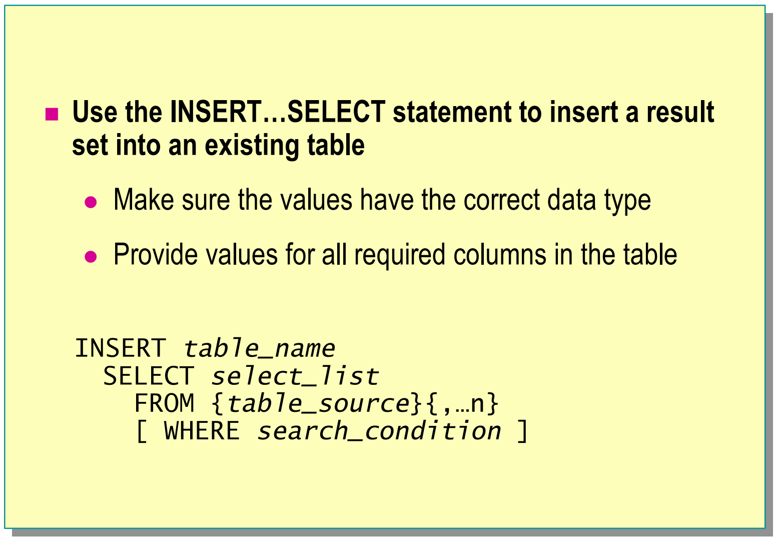
Statement	Description
COMMIT TRANSACTION	The COMMIT TRANSACTION statement guarantees that all of the modifications in the transaction are permanently part of the database. A COMMIT TRANSACTION statement also frees resources, such as database locks, used during the transaction.
ROLLBACK TRANSACTION	The ROLLBACK TRANSACTION statement cancels a transaction. It backs out of all modifications made in the transaction by returning the data to its original state at the start of the transaction. A ROLLBACK TRANSACTION statement also frees resources used during the transaction. If an error occurs in a transaction, SQL Server automatically performs a rollback of the transaction in progress.

Example of using transactions

This example transfers $100 from a savings account to a checking account for a customer by using a transaction. It undoes any data changes if there is an error at any point during the transaction.

```
/* Transfer money from a savings account to a checking
   account. Use a transaction to ensure the logical
   consistency of the data */
BEGIN TRANSACTION

  UPDATE savings
    SET balance = balance - 100
    WHERE custid = 78910

  IF @@ERROR <> 0
    BEGIN
      RAISERROR ('Error, transaction not completed!', 16, -1)
      ROLLBACK TRANSACTION
    END

  UPDATE checking
    SET balance = balance + 100
    WHERE custid = 78910

  IF @@ERROR <> 0
    BEGIN
      RAISERROR ('Error, transaction not completed!', 16, -1)
      ROLLBACK TRANSACTION
    END

COMMIT TRANSACTION
```

How to Insert Rows into an Existing Table



- **Use the INSERT…SELECT statement to insert a result set into an existing table**
  - Make sure the values have the correct data type
  - Provide values for all required columns in the table

```
INSERT table_name
  SELECT select_list
    FROM {table_source}{,…n}
    [ WHERE search_condition ]
```

Introduction

Use the INSERT…SELECT statement to insert a result set into an existing table. All rows that satisfy the SELECT statement are inserted into the specified table.

Using the INSERT…SELECT statement is more efficient than writing multiple, single-row INSERT statements.

Scenario

Using the INSERT…SELECT statement is a convenient way of adding rows to an existing table from other sources. For example, employees of Northwind Traders are eligible to buy company products. You can write an INSERT…SELECT statement to add employee information to the **Customers** table.

Partial syntax for INSERT...SELECT

The syntax for the INSERT…SELECT statement is as follows:

```
INSERT table_name
  SELECT select_list
    FROM {table_source}{,…n}
    [ WHERE search_condition ]
```

Guidelines for using INSERT...SELECT

When you use the INSERT…SELECT statement, consider the following facts and guidelines:

- Make sure the new values have the same or similar data types as the columns in the destination table.
- The result set must include values for all the required columns in the destination table.
- The result set does not have to contain values for columns with default values or for columns that can contain null values.

Example of using INSERT…SELECT

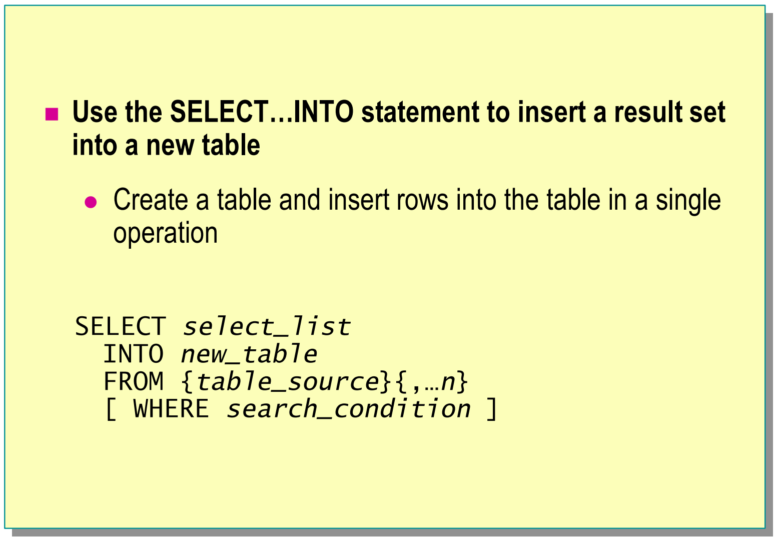
This example adds Northwind Traders employees to the **Customers** table. The new **customerid** column consists of the first three letters of the employee's first name, concatenated with the first two letters of the last name. The employee's last name is used as the new company name, and the first name is used as the contact name.

```
/* Insert employee information into the customers table */
USE northwind
INSERT customers
  SELECT substring (firstname, 1, 3)
         + substring (lastname, 1, 2)
         , lastname, firstname, title, address, city
         , region, postalcode, country, homephone, NULL
    FROM employees
GO
```

Practice

Rewrite the INSERT…SELECT statement in the previous example so that it inserts only Seattle-based employees into the **Customers** table.

How to Insert Rows into a New Table



- **Use the SELECT…INTO statement to insert a result set into a new table**
  - Create a table and insert rows into the table in a single operation

```
SELECT select_list
  INTO new_table
  FROM {table_source}{,…n}
  [ WHERE search_condition ]
```

Introduction

Use the SELECT…INTO statement to insert a result set into a new table. This enables you to populate new tables in a database with imported or computed data.

Scenario 1

You can use the SELECT…INTO statement to break down complex problems that require a data set from various sources. If you first create a temporary table, the queries that you execute on it are simpler than those you execute on multiple tables or databases.

For example, you might want to create a temporary table containing detailed information about orders at Northwind Traders. For each order, the temporary table might contain the product name from the **Products** table, the company name from the **Customers** table, and the last name of the employee that took the order.

Scenario 2

You can also use the SELECT…INTO statement to store computed data temporarily. For example, you might want to create a temporary table containing the unit price and computed sales tax for all products at Northwind Traders.

Partial syntax for SELECT…INTO

The syntax for the SELECT…INTO statement is as follows:

```
SELECT select_list
  INTO new_table
  FROM {table_source}{,…n}
  [ WHERE search_condition ]
```

Guidelines for using SELECT…INTO

When you use the SELECT…INTO statement, consider the following facts and guidelines:

- You can use the SELECT…INTO statement to create a table and to insert rows into the table in a single operation.
- Ensure that the table name in the SELECT…INTO statement is unique. If a table exists with the same name, the SELECT…INTO statement fails.
- You must create column aliases or specify the column names of the new table in the select list.

Creating a temporary or permanent table

You can use the SELECT…INTO statement to create a local temporary table, a global temporary table, or a permanent table. The following table describes each of these types of tables and shows how to specify them using the SELECT…INTO statement.

Type of table	How to specify	Description
Local temporary table	`#table_name`	A local temporary table is visible in the current session only. Space for a local temporary table is reclaimed when the user ends the session.
Global temporary table	`##table_name`	A global temporary table is visible in all sessions. Space for a global temporary table is reclaimed when the table is no longer used in any session.
Permanent table	`table_name`	A permanent table is visible in all sessions and is not reclaimed automatically. Set the **SELECT INTO/BLUKCOPY** database option to **ON** to create a permanent table.

Example of using SELECT…INTO

This example creates a local temporary table based on a query on the **Products** table. Notice that you can use string and mathematical functions to manipulate the result set.

```
/* Retrieve data from the products table, and insert the
   result set into a new temporary table named #pricetable */
USE northwind
SELECT productname AS products
       , unitprice AS price
       , (unitprice * 0.1) AS tax
  INTO #pricetable
  FROM products
GO
```

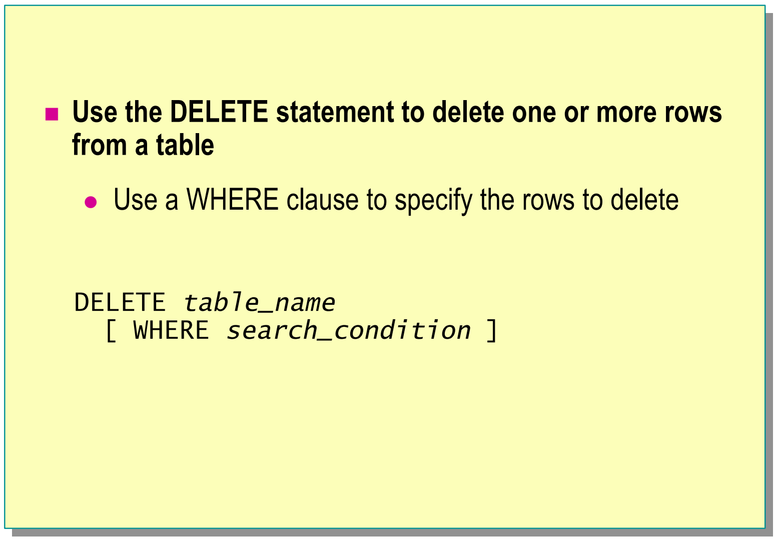
To view your result set, execute the following query:

```
/* Examine the data in the temporary table, #pricetable */
USE northwind
SELECT * FROM #pricetable
GO
```

Practice

Write a SELECT…INTO statement to create a temporary local table named **Stocklevels**. Populate the table with the name of each product, the number of units in stock, and the difference between the values in the **unitsinstock** and **reorderlevel** columns.

How to Delete Rows



- **Use the DELETE statement to delete one or more rows from a table**
  - Use a WHERE clause to specify the rows to delete

```
DELETE table_name
  [ WHERE search_condition ]
```

Introduction

Use the DELETE statement to delete one or more rows from a table. Each deleted row is logged in the transaction log.

Partial syntax for DELETE

The syntax for the DELETE statement is as follows:

```
DELETE table_name
  [ WHERE search_condition ]
```

Guidelines for using DELETE

When you use the DELETE statement, consider the following facts and guidelines:

- Use a WHERE clause to specify which rows to delete.
- If you omit the WHERE clause, SQL Server deletes all the rows in the table.

Example of using DELETE

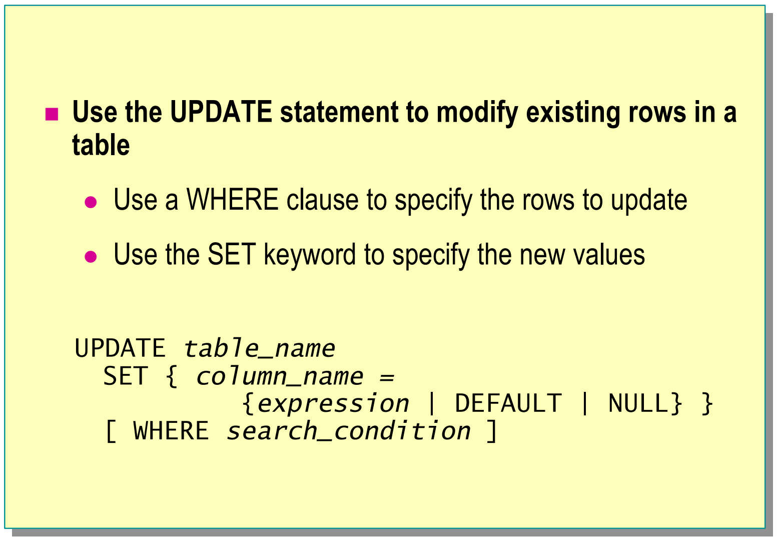
This example deletes order records that are at least six months old:

```
/* Delete orders that are at least six months old */
USE northwind
DELETE orders
  WHERE DATEDIFF (MONTH, shippeddate, GETDATE()) >= 6
GO
```

Practice

Write a DELETE statement to delete rows from the “Order Details” table (you must use quotes around this table name, because the table name contains a space character). Delete each row where the quantity ordered is more than 50.

How to Update Rows



- **Use the UPDATE statement to modify existing rows in a table**
  - Use a WHERE clause to specify the rows to update
  - Use the SET keyword to specify the new values

```
UPDATE table_name
  SET { column_name =
           {expression | DEFAULT | NULL} }
  [ WHERE search_condition ]
```

Introduction

Use the UPDATE statement to modify existing rows in a table. You can change single rows, a group of rows, or all the rows in a table.

Partial syntax for UPDATE

The syntax for the UPDATE statement is as follows:

```
UPDATE table_name
  SET { column_name = {expression | DEFAULT | NULL}  }  [,…n]
  [ WHERE search_condition ]
```

Guidelines for using UPDATE

When you use the UPDATE statement, consider the following facts and guidelines:

- You can change the data in only one table at a time.
- Use a WHERE clause to specify which rows to update.
- Use the SET keyword to specify the new values for columns in the table.
- You can use expressions for the new values. For example, you can use an expression such as (unitprice * 2), add two columns together, and so on.
- Make sure the new values have the correct data types for the columns that you are updating.
- SQL Server does not update any rows that violate any integrity constraints. The changes do not occur, and the statement is rolled back.

Example of using UPDATE

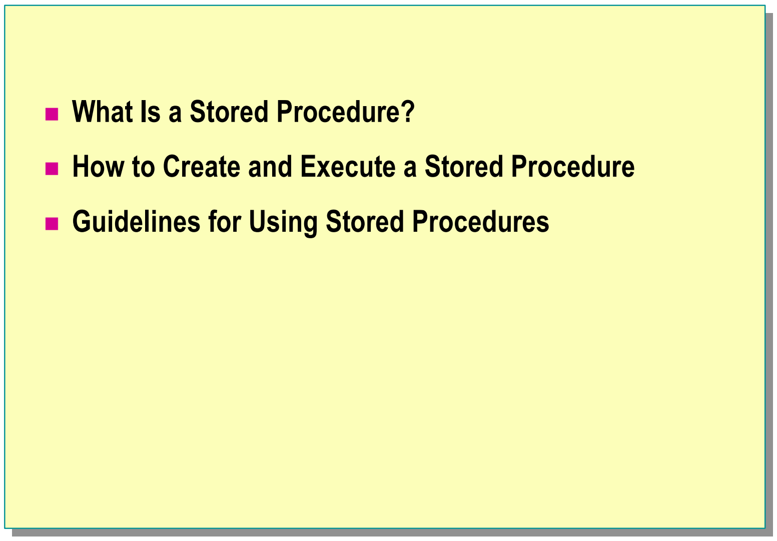
This example adds 10 percent to the current price of all Northwind Traders products:

```
/* Add 10 percent to the price of all products */
USE northwind
UPDATE products
  SET unitprice = unitprice * 1.1
GO
```

Practice

Rewrite the UPDATE statement in the previous example so that it only updates the price for products that cost more than $50.

Lesson: Using Stored Procedures



- What Is a Stored Procedure?
- How to Create and Execute a Stored Procedure
- Guidelines for Using Stored Procedures

Introduction

In this lesson, you will learn how to create and execute a simple stored procedure in Transact-SQL. You can use stored procedures to encapsulate business rules and to execute these rules in procedural code in the database engine.

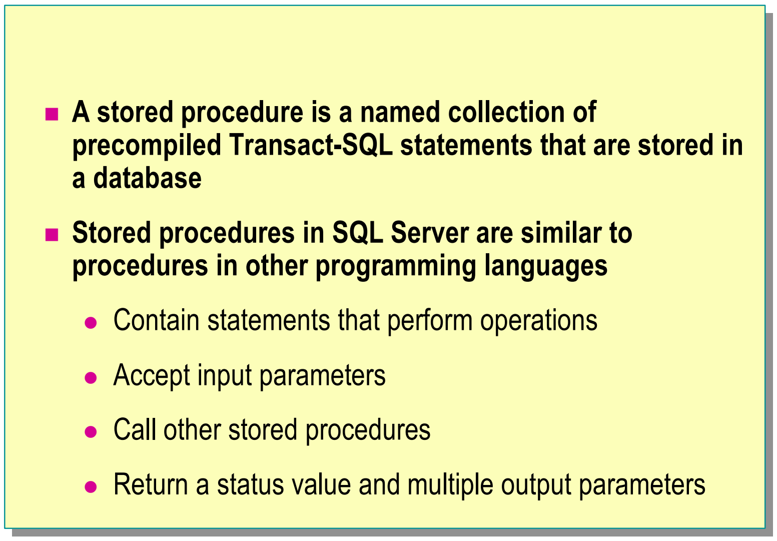
You will also learn how to create triggers in Transact-SQL. You can use triggers to enforce the consistency of related tables in the database.

Lesson Objectives

After completing this lesson, you will be able to:

- Create a simple stored procedure.
- Execute a stored procedure.
- Describe the advantages of using stored procedures.

What Is a Stored Procedure?



- **A stored procedure is a named collection of precompiled Transact-SQL statements that are stored in a database**
- **Stored procedures in SQL Server are similar to procedures in other programming languages**
  - Contain statements that perform operations
  - Accept input parameters
  - Call other stored procedures
  - Return a status value and multiple output parameters

Introduction

A stored procedure is a named collection of precompiled Transact-SQL statements that are stored in a database. Stored procedures support user-defined variables, control-of-flow execution, and other advanced programming features.

You can use stored procedures to encapsulate repetitive tasks in the database engine and to execute these tasks efficiently and securely.

Scenario

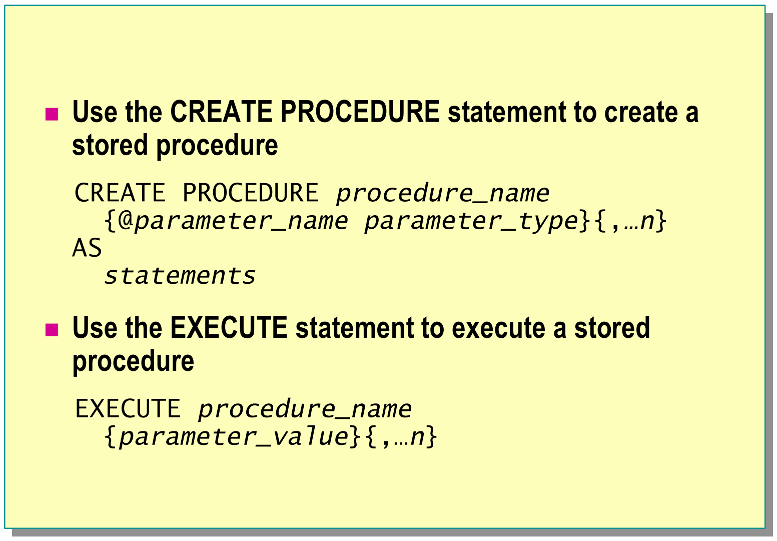
The **Orders** table in the Northwind Traders database contains information about all the orders received. The table has a foreign key identifying the customer that made the order. The table also has a foreign key identifying the employee that took the order. You can create a stored procedure to execute a complex SELECT statement that returns all this information by using a three-table join.

Guidelines for using stored procedures

Stored procedures in SQL Server are similar to procedures in other programming languages. You can perform the following tasks in a stored procedure:

- Define statements that perform operations in the database, such as a SELECT statement.
- Accept input parameters, such as a value to be used in a WHERE clause in the stored procedure.
- Call other stored procedures, to perform related and additional tasks.
- Return a status value to a calling stored procedure or batch, to indicate the success or failure of the stored procedure.
- Return multiple values to the calling stored procedure or batch, using output parameters.

How to Create and Execute a Stored Procedure



- **Use the CREATE PROCEDURE statement to create a stored procedure**

```
CREATE PROCEDURE procedure_name
  {@parameter_name parameter_type}{,…n}
AS
  statements
```

- **Use the EXECUTE statement to execute a stored procedure**

```
EXECUTE procedure_name
  {parameter_value}{,…n}
```

Introduction

Use the CREATE PROCEDURE statement to create a stored procedure. You can define parameters for the stored procedure, if necessary, to make the stored procedure more flexible. Within the body of the stored procedure, specify the statements that you want to perform when the stored procedure is executed.

Syntax for creating a stored procedure

The syntax for creating a stored procedure is as follows:

```
CREATE PROCEDURE procedure_name
  {@parameter_name parameter_type}{,…n}
AS
  statements
```

Example of creating a stored procedure

This example creates a stored procedure named **orders_info_all**. The stored procedure performs a complex SELECT statement, which uses a three-table join on the **Orders**, **Customers**, and **Employees** tables. The query retrieves the order ID, the company name for the customer that made the order, and the last name of the employee that took the order.

This stored procedure does not use any parameters.

```
/* Create a stored procedure with a SELECT statement, using a
   join on the orders, customers, and employees tables */
CREATE PROCEDURE orders_info_all
AS
SELECT orderid, companyname, lastname
  FROM orders AS o
   JOIN customers AS c ON o.customerid = c.customerid
   JOIN employees AS e ON o.employeeid = e.employeeid
GO
```

Executing a stored procedure

Use the EXECUTE statement to execute a stored procedure. If the stored procedure requires parameters, pass these parameters as a comma-separated list.

Syntax for executing a stored procedure

The syntax for executing a stored procedure is as follows:

```
EXECUTE procedure_name
  {parameter_value}{,…n}
```

Example of executing a stored procedure

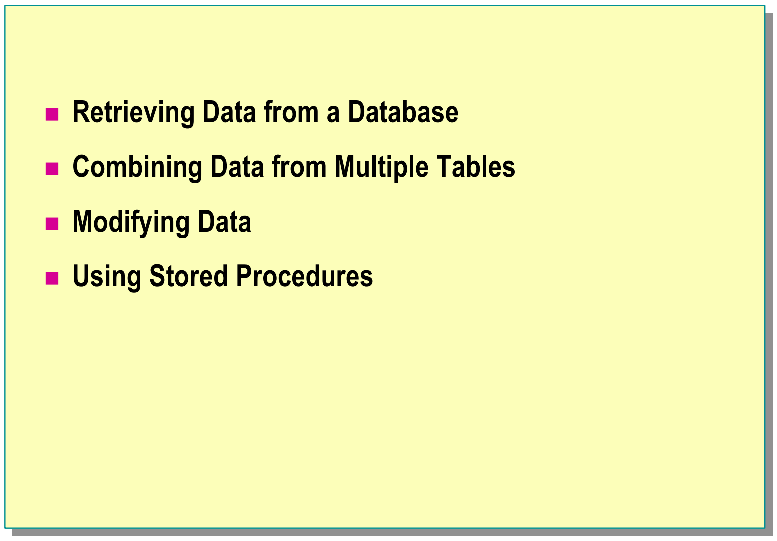
This example executes the **orders_info_all** stored procedure, which was created in the previous example:

```
/* Execute the orders_info_all stored procedure */
USE northwind
EXECUTE orders_info_all
GO
```

Practice

Write a stored procedure to increase the unit price of all products in the **Products** table by 10 percent. Also write an SQL statement to execute the stored procedure.

Review



- Retrieving Data from a Database
- Combining Data from Multiple Tables
- Modifying Data
- Using Stored Procedures

The Duluth Mutual Life health care organization has a database that tracks information about doctors and their patients. The database includes the following tables.

doctors

Column	Description
doc_id	int NOT NULL, PRIMARY KEY
fname	char(20) NOT NULL
lname	char(25) NOT NULL
street	char(50) NULL
city	char(255) NULL
state	char(255) NULL
postal_code	char(7) NULL
specialty	char(25) NOT NULL
charge_rate	money NOT NULL
phone	char(10) NULL

patients

Column	Description
pat_id	int NOT NULL, PRIMARY KEY
fname	char(20) NOT NULL
lname	char(25) NOT NULL
insurance_company	char(25) NOT NULL
phone	char(10) NULL

casefiles

Column	Description	
casefile_id	int	NOT NULL, PRIMARY KEY
admission_date	datetime	NOT NULL
pat_id	int	NOT NULL, FOREIGN KEY to patients.pat_id
doc_id	int	NOT NULL, FOREIGN KEY to doctors.doc_id
diagnosis	char(150)	NOT NULL

1. How would you retrieve information about doctors whose specialty is ‘Pediatrics’, , ‘Physiotherapy’, or ‘Opthalmics’?

   Write a SELECT statement with a WHERE clause of the following type:

   ```
   WHERE specialty = 'Pediatrics'
      OR specialty = 'Physiotherapy'
      OR specialty = 'Opthalmics'
   ```

   Or use a WHERE clause that includes the IN keyword as follows:

   ```
   WHERE specialty IN
    ('Pediatrics', 'Physiotherapy', 'Opthalmics')
   ```

2. How can you generate a list of patient names for a particular doctor?

   You must join all three tables. The relationship between doctors and patients is a many-to-many relationship. Even though you only want information from the doctors and patients tables, you must also use the casefiles table, because this table relates doctors to patients. Join the doctors table to the casefiles table on the doc_id column, and then join the patients table to the casefiles table on the pat_id column. Use a WHERE clause to limit the results for a particular doctor.

3. The participating doctors have increased their costs of services. How can you increase the value in the **charge_rate** column for all doctors by 12 percent?

   Use an UPDATE statement of the following type:

   ```
   UPDATE doctors SET charge_rate = (charge_rate * 1.12)
   ```

4. What is the minimum number of column values that you must supply to add a new row to the **doctors** table?

   You must supply data for at least five columns. At a minimum, the INSERT statement must contain values for the doc_id, fname, lname, specialty, and charge_rate columns. All other columns can have null values.

5. How can you remove rows from the **casefiles** table for cases where the date of admission was more than 12 months ago?

   Use a DELETE statement of the following type:

```
DELETE casefiles
  WHERE DATEDIFF (MONTH, admission_date, GETDATE()) >= 12
```

6. How can you create a stored procedure to encapsulate a query to retrieve full details for each case file? The stored procedure must return the admission date, the full name of the patient, the full name of the doctor, and the diagnosis.

   Use a CREATE PROCEDURE statement of the following type:

```
CREATE PROCEDURE casefiles_info_all
AS
SELECT admission_date,
       p.fname + ' ' + p.lname,
       d.fname + ' ' + d.lname,
       diagnosis
 FROM casefiles AS c
  JOIN patients AS p ON c.pat_id = p.pat_id
  JOIN doctors  AS d ON c.doc_id = d.doc_id
GO
```

Lab A: Best Practices for Writing SQL Statements and Stored Procedures

Lab.doc

Appendix B: ADO and ADO.NET Comparison (Prerelease)

ADO and ADO.NET Comparison Table

Introduction

Microsoft® ActiveX® Data Objects (ADO) .NET is an evolutionary improvement on ADO. One way to quickly understand the advantages of ADO.NET is to compare its features to those of ADO.

Feature	ADO	ADO.NET
Memory-resident data representation	Uses the **Recordset** object, which looks like a single table.	Uses the **DataSet** object, which can contain one or more tables that are represented by **DataTable** objects.
Relationships between multiple tables	Requires the JOIN query to assemble data from multiple database tables in a single result table.	Supports the **DataRelation** object to associate rows in one **DataTable** object with rows in another **DataTable** object.
Data visitation	Scans **DataSet** rows sequentially.	Uses a navigation paradigm for non-sequential access to rows in a table. ADO.NET follows relationships to navigate from rows in one table to corresponding rows in another table.
Disconnected access	Uses the **Recordset** object for disconnected access but typically supports connected access, represented by the **Connection** object. You communicate to a database with calls to an OLE DB provider.	Communicates to a database with standardized calls to the **DataAdapter** object, which communicates to the OLE DB provider (or sometimes directly to application programming interfaces (APIs) that are provided by a database management system).

(*continued*)

Feature	ADO	ADO.NET
Programmability	Uses the **Connection** object to transmit commands addressing a data source's underlying data constructs.	Uses the strongly typed programming characteristic of Extensible Markup Language (XML). Data is self-describing because names for code items correspond to the "real world" problems solved by the code. Underlying data constructs such as tables, rows, and tables do not appear, making code easier to read and to write.
Sharing disconnected data between tiers or components	Uses COM marshalling to transmit a disconnected recordset. This supports only those data types defined by the COM standard. Requires type conversions, which demand system resources.	Transmits a **DataSet** with an XML file. The XML format places no restrictions on data types and requires no type conversions.
Transmitting data through firewalls	Problematic, because firewalls are typically configured to prevent system-level requests such as COM marshalling.	Supported, because ADO.NET **DataSet** objects use text-based XML, which can pass through firewalls.
Scalability	Database locks and active database connections for long durations contend for limited database resources.	There is disconnected access to database data without database locks or active database connections for long durations . This limits contention for limited database resources.

Support for ADO in the .NET Framework

Introduction

With all the features provided by ADO.NET, is there any reason to continue to use ADO? Yes. Here are some potential reasons why.

- You will not be able to rewrite all of your existing software for the Microsoft .NET Framework.
- You will need to interact with existing Component Object Model (COM) components.
- You will not write all of your new software for the .NET Framework.
- You will need your “legacy” applications to be able to interact with .NET components.

Some scenarios where you will want to interact with existing ADO code, or reuse existing code that uses ADO, are when you want to:

- Upgrade an existing ADO project.
- Use server-side cursors directly.
- Use an existing COM component that returns an ADO Recordset.

Upgrading existing ADO projects

If you have existing projects that use ADO, you may want to upgrade the project to use the .NET Framework, while still using the ADO data layer. Although this is not recommended, it can be done through the COM Interop layer.

Using server-side cursors

Another reason for using ADO is if your application requires server-side cursors. The only type of server-side cursor supported by ADO.NET is the forward-only, read-only, “firehose” cursor. ADO.NET does not support dynamic and keyset cursors.

You can create a stored procedure that, because it is executed by the database server, can create and use server-side cursors. Yet, ADO.NET can not create and use server-side cursors directly. This functionality may be added to future versions of ADO.NET. However, misuse of server-side cursors is a major factor in poor scalability of database applications. This is one of the reasons why a design decision was made to exclude this functionality from ADO.NET.

Accessing an ADO recordset or record from ADO.NET

In the .NET Framework, you can access both existing components that return ADO **Recordset** or **Record** objects by using .NET COM Interop services and the OLE DB .NET Data Provider. This enables you to use existing COM objects that return ADO objects, without having to rewrite them entirely by using the .NET Framework. ADO.NET and the OLE DB .NET Data Provider only support filling a **DataSet** from an ADO **Recordset** or **Record** object.

Performance

Performance of COM Interop in the Common Language Runtime (CLR) has improved dramatically in later betas. ADO is one of the COM Interop team's main scenarios, and the team continues to monitor performance and reliability to make sure this is a valid scenario. In any case, you shouldn't be afraid to use COM Interop to ADO; you just shouldn't be as excited about doing so as you should be about using ADO.NET.

How to Expose COM Components to the .NET Framework

Introduction

This section summarizes the process for exposing an existing COM component, such as ADO, to managed code.

Existing COM components are valuable resources in managed code as middle-tier business applications or as an isolated functionality. An ideal component has a primary Interop assembly and closely conforms to the programming standards imposed by COM.

How to expose COM components to the .NET Framework

To expose a COM component to the .NET Framework, perform the following procedure:

1. Import a type library as an assembly.

   The common language runtime requires metadata for all types, including COM types. There are several ways to obtain an assembly containing COM types that are imported as metadata.

2. Create COM types in managed code.

   You can inspect COM types, activate instances, and invoke methods on the COM object in the same way that you do for any managed type.

3. Compile an Interop project.

   The .NET Framework software development kit (SDK) provides compilers for several languages that are compliant with common language specification (CLS), including Microsoft Visual Basic® .NET, C#, and Managed Extensions for C++.

4. Deploy an Interop application.

   Interop applications are best deployed as strong-named, signed assemblies in the global assembly cache (GAC).

For further information

COM Interop is potentially a very complex subject. For more information about COM Interop, search for the topic “Exposing COM Components to the .NET Framework” in the Microsoft Visual Studio .NET documentation for further information.

How to Fill a DataSet with an ADO Recordset or Record

Introduction

To provide access to ADO **Recordset** and **Record** objects from ADO.NET, the OLE DB .NET Data Provider overloads the **Fill** method of the **OleDbDataAdapter** class to accept an ADO **Recordset** or **Record** object. Filling a **DataSet** with the contents of an ADO object is a one-way operation. That is, data can be imported from the ADO **Recordset** or **Record** object into the **DataSet**, but any updates of the data must be handled explicitly by either ADO.NET or ADO.

How to consume a COM component that returns an ADO Recordset or Record

To consume a COM component that returns an ADO **Recordset** or **Record** object by using .NET COM Interop services and ADO.NET, you need to first import the type library information for the COM component and ADO. You do this by using TlbImp.exe or the Visual Studio .NET development environment.

Using the command line

You can use the command line to import the type library information for the COM component and ADO.

For example, an existing COM component with a programmatic identifier (ProgID) of ADOComponent.DataClass is compiled into ADOComponent.dll. It has methods that return objects of type **ADODB.Recordset**. To consume this object from .NET, import both ADOComponent.dll, and msado15.dll, which contains the **ADODB.Recordset** and **ADODB.Record** objects. To import the COM type libraries to .NET, issue the following commands:

```
TlbImp "C:\Program Files\Common Files\System\Ado\msado15.dll"
/out:ADODB.dll

TlbImp ADOComponent.dll /out:ADOCOM.dll
```

You can then pass the resulting .NET libraries, ADODB.dll and ADOCOM.dll, as library references when compiling a .NET-compatible program. The following example shows how to compile a Microsoft Visual Basic® .NET program using vbc.exe and how to supply the imported COM libraries:

```
vbc MyVB.vb /r:system.dll /r:system.data.dll /r:system.xml.dll
/r:ADODB.dll /r:ADOCOM.dll
```

Using the Visual Studio .NET development environment

Alternatively, using the Visual Studio .NET development environment, simply use the References folder in the Solution Explorer.

If the **ADOComponent.DataClass** object has a method named **GetData** that returns an **ADODB.Recordset** object, you can write the following in Visual Basic .NET:

```
Dim adoComponent As ADOCOM.DataClass = New ADOCOM.DataClass
Dim adoRS As ADODB.Recordset = adoComponent.GetData()
```

Using the OLE DB .NET Data Provider, the **ADODB.Recordset** object can be used to fill a **DataSet** as shown in the following sample:

```
Dim myDA As OleDbDataAdapter = New OleDbDataAdapter
Dim myDS As DataSet = New DataSet
myDA.Fill(myDS, adoRS, "MyTable")
```

What Were We Thinking

Introduction

The first question most developers have when they start learning about the .NET Framework, which includes ADO.NET, is "what were they thinking?!?" It is often unclear why a change was made to an existing technology, or why a new feature or technology might be useful.

Educating developers

A major problem with new development tools and technologies is educating developers. You know this. It is why you are reading this.

Often people assume that showing developers the syntax of the new tool or technology is enough, especially for those developers already familiar with a previous version of that tool or technology.

For example, developers who currently write Visual Basic applications that use ADO just need to know the new syntax and object model to write Visual Basic .NET applications that use ADO.NET. *Right?*

Wrong! Those developers need to know why the changes were made, in order to adjust design decisions in applications they write, and therefore get the best out of the new software. If developers attempt to use the design patterns they learned about ADO in the ADO.NET world, they will fail.

Why did they change that?

Developers will also be frustrated. For example, transaction handling has changed in ADO.NET, apparently for no good reason.

For example, here is some ADO code:

```
Dim cn As ADODB.Connection
cn.ConnectionString = "..."
cn.Open
cn.BeginTrans
' perform some database actions
cn.CommitTrans
cn.Close
```

Here is the equivalent ADO.NET code:

```
Dim cn As SqlConnection, tn As SqlTransaction
cn.ConnectionString = "..."
cn.Open
tn = cn.BeginTransaction()
' perform some database actions
tn.Commit
cn.Close
```

Why is there a new class? After all, no new functionality has been added, and the code is now slightly more complex. Superficially it looks like a change for no good reason. The reason is that by breaking transaction handling functionality into a separate class, the **SqlConnection** class can be lighter-weight. So for the majority of applications that do not require transaction support, the applications are smaller and faster.

Another good example is the loss of the ADO **Execute** method. ADO.NET does not have a single **Execute** method. Instead ADO.NET provides multiple **ExecuteX** methods that return different types of information.

At first glance this appears to complicate things, but the **Execute** method in ADO was more complicated than most developers realized. It hides its complexity in its optional parameters. Because the parameters were optional, many developers did not use them, and got bad performance or scalability from ADO as a result.

Another example: the adExecuteNoRecords option

Here is another example. Have you ever used the **adExecuteNoRecords** option? Most ADO developers have not, so when running a Data Manipulation Language (DML) statement (for example, **UPDATE**, **INSERT**, and **DELETE**), an unnecessary **Recordset** object is created. With ADO.NET, the **ExecuteNonQuery** method is used to run DML statements, which does not create unnecessary objects.

ADO is too easy to use (and therefore misuse)

ADO is easy to use, but this leads to misuse. ADO.NET is harder to use, but this leads to better, faster, scalable code. The developer's job is now harder in the beginning, but the long-term benefits are substantial.

ADO.NET Design Decisions

Introduction

Mike Pizzo posted on the public bulletin board **microsoft.public.dotnet.framework.adonet** an excellent commentary on the design decisions and architectural basis for ADO.NET. It was written in 2000, before ADO+ was renamed ADO.NET.

Posting

Many of the differences between ADO and ADO+ come from subtle but important conceptual differences. An admitted lack of documentation for our data components in the beta release of the .NET Framework has left early adopters on their own in trying to understand how to compose a differently factored set of data objects to perform familiar tasks.

There are a number of comments already offered on this discussion group debating the pros/cons of the ADO+ architecture, including references to several articles on the subject. Instead of going into a semi-exhaustive list of feature differences between the two, let me see if I can describe some of the design goals we had in mind in (re)developing ADO for the .NET Framework as ADO+ (ADO.NET).

Where-as OLE DB defines a set of factored interfaces between pluggable components, ADO+ is built as a well-factored set of components. This is a key conceptual difference that permeates the design.

OLE DB was designed as a flexible, extensible interface between pluggable components. A data store can expose its native functionality, semantics, and behavior through a common interface that contains introspection methods for determining an individual store's level of support, behavior, and semantics.

The Rowset object, in particular, was designed to expose all data as a shared buffer over which multiple components could add functionality, such as query processing or cursoring, to a less-capable store. The idea being that applications could specify the functionality they required, and it was up to the data store, and possibly a set of services augmenting the functionality of that store, to decide how best to expose the requested functionality.

ADO is built on top of OLE DB, and inherits OLE DB's strengths as well as its weaknesses. It can expose a rich set of functionality and capabilities on top of a wide variety of data stores. However, important differences in how functionality is implemented or exposed are often inaccessible to the user.

For example, in talking to customers we find that a majority of them end up invoking a service called the Client Cursor Engine, either directly or indirectly. Even if they are not using the cursor engine's rich set of features, having the common implementation of the OLE DB Rowset (ADO Recordset) gives them a predictable target to program against.

The cursor engine reads all of the results from a query, puts them into an in-memory cache, and then lets the user work with the data in the cache. If the user updates data, the cursor engine generates insert, update, and delete commands based on the metadata of the result in order to propagate the changes back to the server; all done under the covers so the user doesn't have to even know it's going on-most of the time. However, if the user's update logic is done through stored procedures, rather than as direct queries against base tables (as is often the case), then, because the generation of the insert/update/delete statements is done under the covers, the user has no way to invoke that server-side update logic.

Similarly, if the cursor engine can't determine the necessary metadata, or if the user wants updates to occur against a different source, or with different logic, than the default, they are at a dead end. And, since the cursor engine holds on to the connection in order to propagate these changes back to the server, persistence, remoting, or storing of the rowset (ADO Recordset) in a cache can be problematic.

In looking at this common usage pattern, and talking to customers, we found that these typical scenarios could be better served by providing an explicit cache implementation for binding to, navigating, and updating data, and separating out the logic for querying and updating data in the database.

This explicit cache is the ADO+ DataSet. Because it is an explicit cache, the user is guaranteed consistent and predictable behavior and semantics, regardless of the source of the data. Further, by defining an explicit cache, rather than a generic interface like IRowset (or the ADO Recordset) that may be implemented over either cached data or a live data stream, the interface can be optimized to expose common cache functionality such as the ability to return the number of records in a table. Also, the programming model over the cache can be made more consistent with arrays, lists, and other collection types within the programming language (indexed access, ForEach support, Contains(), etc.).

We also took a close look at how customers talk to a database. We found that, except for a few generic tools and common components, most applications know the query they are executing and the shape and types of the results. In many cases the result is a single record, such as a customer profile, or even a single value such as an account balance. When the results are a set of records, in order not to hold state on the server, they are generally read sequentially in a forward-only manner, and written to application memory or written out, for example, as formatted HTML tables in a Response.Write.

We rarely found updates being made to the results themselves, but rather through update logic such as stored procedures. For example, if I query an on-line bookstore for titles and find one I want to purchase, I don't update the bookstore's catalog that I've been browsing, but rather I build up a request containing the books I'm interested in, and that request is processed on the server by a series of order processing components that make individual reads and writes to a series of tables and databases.

For all of these uses, the rich cursoring capabilities and other multi-user facilities of the Rowset were unnecessary baggage. Even when such extended functionality was not requested, there was some overhead inherent in the model (such as row handle indirection) necessary in order to provide a common model for accessing the data.

By providing a common relational DataSet that users could fill, bind to, and use to navigate and modify their data, we were free to develop optimized components for connecting to, executing queries against, and streaming data from a database. These "Managed Providers" were not intended to replace OLE DB interfaces to sophisticated stores; instead they were designed to provide optimized access to certain core features we found most used in working with a database.

So we have the DataSet, a custom object designed for navigating, remoting, storing, and working with data, and Managed Providers that expose an optimized set of simplified components for talking to a database. An object called the DataSetCommand, supported by the Managed Provider, provides the glue between the two. The DataSetCommand object encapsulates the logic for populating a DataSet from the results of an ADO+ Command, and pushing changes back to the database using exposed ADO+ Command objects containing commands for Inserting, Updating, and Deleting records in a database. We provide a component for automatically generating the Insert, Update, and Delete statements, similar to the logic found in the ADO Client Cursor Engine, but instead of these commands being generated as part of internal logic, the commands are external. This means that users can specify their own statements for applying updates to the back-end, such as the invocation of stored procedures. Additionally, users can listen for the OnRowUpdating events in order to add more complex business logic.

By putting all of the logic for communicating with the database outside of the DataSet, the DataSet is left with no affinity to any particular back-end store. Whether you populate the DataSet with the results from an Oracle Query, a SQLServer stored procedure, application data, or XML, once the data is in the DataSet it's just data. That means that you can populate a single DataSet with data from various sources, and define relations between each of these different types of data. So, for example, you could navigate from customers sourced from an Oracle database, to orders obtained from a SQLServer database, to product descriptions loaded from an XML file. That's pretty cool.

And, since the DataSet doesn't contain connections or other database state, the DataSet is a perfect candidate for storing in an ASP+ cache, persisting to disk, or remoting. In fact, WebServices know how to remote DataSets as parameters in a WebRequest for passing relational sets of data between tiers. The DataSet itself is persisted, of course, as XML.

Speaking of XML, The ADO+ stack of components, and the DataSet in particular, were designed from the core to be great components for working with XML data. Not only does the DataSet persist and load XML data, but it also saves and loads its schema according to the XSD Schema definition language for XML, and changes as SQLXML-compatible UpdateGrams. And don't get me started on the integration between a live XmlDataDocument and the DataSet.

So what happens to ADO and OLE DB? They continue to exist, and to serve the role for which they were designed. OLE DB continues to be Microsoft's preferred interface for developing a robust, comprehensive interface to an arbitrary data store. We won't, for example, drop our OLE DB interface to SQLServer in favor of a Managed Provider; there is just too much native functionality in the store that Manged Providers weren't intended to represent. Our ADO Managed Provider exposes the ADO+ Connection, Command, and DataReader interfaces directly on top of any OLE DB provider, as well as a DataSetCommand for loading OLE DB data into, and propagating changes out of, the DataSet. OLE DB providers continue to be the way to expose data to things like Microsoft SQLServer's Distributed Query Processor, or for plugging multi-dimensional data into Microsoft Excel.

For functionality not exposed through ADO+, such as an object model for schema manipulations, .NET users are still able to call classic ADO through COM Interop.

Which leads me to a question that I often get; will Managed Providers be extended to support everything that ADO/OLE DB support today? Probably

not. Managed Providers were not intended to be the single, comprehensive interface to any data store. I wouldn't want to add more and more functionality to the Managed Providers simply because it exists today in ADO or OLE DB. I do expect, however, that we will continue to enhance the functionality of Managed Providers, based on customer demand. At the same time, I want to be sure we don't muddy the clean factoring we have between these components. So, for instance, rather than extending Managed Providers to support a schema object model, we may decide to have a separate Database Schema object, perhaps designed around XSD.

The key to the ADO+ architecture is the factoring of components. All communication with the Database; connections, transactions, queries, stored procedure invocation, streaming of data, etc., is done through the Managed Providers. All interactive navigation and management of data is done through the DataSet. Does the fact that the DataSet doesn't hold locks mean that ADO+ doesn't support things like a pessimistic update model? Not at all. It just means that you have to understand which components hold locks (the Managed Providers), and which expose a cursor model (the DataSet).

So, for example, if you want to guarantee that updates made through the DataSet don't fail for reasons of concurrency, just start a transaction on an ADO+ Connection with the appropriate isolation level specified. As long as you use that same connection to execute the query that populates the dataset and to propagate the changes to the backend, they will all be done under the scope of that transaction. When you're done, just commit the transaction on the connection.

More likely you won't want to hold state, such as locks, on the server while the user interacts with the data in the DataSet. That's fine too; you don't have to hold open a transaction or even a connection between populating and updating a DataSet. You can even use a different Managed Provider to handle updates than the one that originally populated the DataSet.

This explicit factoring of objects in ADO+ does mean that your applications may look a little different, and in some cases you'll have to write a little more code to wire the proper components together, but the benefits of having a well factored set of focused, highly optimized, components should pay great dividends in building a clean, well designed application with tightly integrated data support.

Still skeptical? Fair enough. There is some new thinking in this architecture that even I didn't fully appreciate until I starting writing code with it. My suggestion would be to take a look at it. Kick the tires. Take it for a spin. And continue to share thoughts, impressions, and comments, good and bad, on this discussion group. For my part, I'm going to try and devote more time to listening to your comments and feedback, and making sure ADO+ provides the best possible tools and services for working with data in the .NET Framework.”

Appendix C: Additional Resources (Prerelease)

ADO and ADO .NET Comparison

Introduction

This appendix provides Web links to additional resources about Microsoft® .NET, Microsoft® ActiveX® Data Objects (ADO) .NET, and related technologies.

These Web links are subject to change.

ADO.NET and SQL Server

The following resources provide information about ADO.NET and Microsoft SQL Server™:

- Accessing Data with ADO.NET

  http://msdn.microsoft.com/library/?url=/library/en-us/cpguidnf/html/cpconaccessingdatawithadonet.asp?frame=true

- Introducing ADO+: Data Access Services for the Microsoft .NET Framework

  http://msdn.microsoft.com/msdnmag/issues/1100/adoplus/adoplus.asp

- ADO.NET for the ADO Programmer

  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/adonetdev.asp?frame=true

- ADO to XML: Building a Data Access Tier with the DataManager Component

  http://msdn.microsoft.com/msdnmag/issues/01/08/data/data0108.asp

- SQL and XML: Use XML to Invoke and Return Stored Procedures Over the Web

  http://msdn.microsoft.com/msdnmag/issues/01/08/XMLSQL/XMLSQL.asp

- Database Architecture: The Storage Engine

  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/Dnsql2k/html/TheStorageEngine.asp?frame=true

The Microsoft .NET Framework

The following resources provide information about the Microsoft .NET Framework:

- Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web

  http://msdn.microsoft.com/msdnmag/issues/0900/Framework/Framework.asp

- Part 2: Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web

  http://msdn.microsoft.com/msdnmag/issues/1000/Framework2/Framework2.asp

- Avoiding DLL Hell: Introducing Application Metadata in the Microsoft .NET Framework

  http://msdn.microsoft.com/msdnmag/issues/1000/metadata/metadata.asp

- .NET Framework: Building, Packaging, Deploying, and Administering Applications and Types

  http://msdn.microsoft.com/msdnmag/issues/01/02/buildapps/buildapps.asp

- .NET Framework: Building, Packaging, Deploying, and Administering Applications and Types—Part 2

  http://msdn.microsoft.com/msdnmag/issues/01/03/buildapps2/buildapps2.asp

- Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework

  http://msdn.microsoft.com/msdnmag/issues/1100/GCI/GCI.asp

- Garbage Collection—Part 2: Automatic Memory Management in the Microsoft .NET Framework

  http://msdn.microsoft.com/msdnmag/issues/1200/GCI2/GCI2.asp

- Windows Forms: A Modern-Day Programming Model for Writing GUI Applications

  http://msdn.microsoft.com/msdnmag/issues/01/02/winforms/winforms.asp

- .NET: An Introduction to Delegates

  http://msdn.microsoft.com/msdnmag/issues/01/04/net/net0104.asp

- .NET: Delegates, Part 2

  http://msdn.microsoft.com/msdnmag/issues/01/06/net/net0106.asp

- .NET: Implementation of Events with Delegates

  http://msdn.microsoft.com/msdnmag/issues/01/08/net/net0108.asp

- .NET Delegates: Making Asynchronous Method Calls in the .NET Environment

  http://msdn.microsoft.com/msdnmag/issues/01/08/Async/Async.asp

- Security in .NET: Enforce Code Access Rights with the Common Language Runtime

  http://msdn.microsoft.com/msdnmag/issues/01/02/CAS/CAS.asp

- .NET P2P: Writing Peer-to-Peer Networked Apps with the Microsoft .NET Framework

  http://msdn.microsoft.com/msdnmag/issues/01/02/netpeers/netpeers.asp
- Advanced Basics: Using Inheritance in Windows Forms Applications

  http://msdn.microsoft.com/msdnmag/issues/01/06/Basics/Basics0106.asp
- Under the Hood: Displaying Metadata in .NET EXEs with MetaViewer

  http://msdn.microsoft.com/msdnmag/issues/01/03/Hood/Hood0103.asp
- C++ Attributes: Make COM Programming a Breeze with New Feature in Visual Studio .NET

  http://msdn.microsoft.com/msdnmag/issues/01/04/Attributes/Attributes.asp
- House of COM: Migrating Native Code to the .NET CLR

  http://msdn.microsoft.com/msdnmag/issues/01/05/com/com0105.asp
- Microsoft .NET: Implement a Custom Common Language Runtime Host for Your Managed App

  http://msdn.microsoft.com/msdnmag/issues/01/03/clr/clr.asp

ASP.NET

The following resources provide information about Active Server Pages (ASP) .NET:

- Active Server Pages+: ASP+ Improves Web App Deployment, Scalability, Security, and Reliability

  http://msdn.microsoft.com/msdnmag/issues/0900/ASPPlus/ASPPlus.asp
- ASP .NET: Web Forms Let You Drag and Drop Your Way to Powerful Web Apps

  http://msdn.microsoft.com/msdnmag/issues/01/05/WebForms/WebForms.asp
- Cutting Edge: Server-side ASP .NET Data Binding

  http://msdn.microsoft.com/msdnmag/issues/01/03/cutting/cutting0103.asp
- Cutting Edge: Server-side ASP .NET Data Binding, Part 2: Customizing the DataGrid Control

  http://msdn.microsoft.com/msdnmag/issues/01/04/cutting/cutting0104.asp
- Cutting Edge: Server-side ASP .NET Data Binding, Part 3: Interactive DataGrids

  http://msdn.microsoft.com/msdnmag/issues/01/05/cutting/cutting0105.asp
- Cutting Edge: DataGrid In-place Editing

  http://msdn.microsoft.com/msdnmag/issues/01/06/cutting/cutting0106.asp
- Data Points: Revisiting the Ad-Hoc Data Display Web Application

  http://msdn.microsoft.com/msdnmag/issues/01/06/data/data0106.asp

- Cutting Edge: Custom Web Data Reporting

  http://msdn.microsoft.com/msdnmag/issues/01/07/cutting/cutting0107.asp

- Cutting Edge: Reusability in ASP .NET: Code-behind Classes and Pagelets

  http://msdn.microsoft.com/msdnmag/issues/01/08/cutting/cutting0108.asp

- The ASP Column: ASP .NET Connection Model and Writing Custom HTTP Handler/Response Objects

  http://msdn.microsoft.com/msdnmag/issues/01/07/asp/asp0107.asp

XML and Web services

The following resources provide information about Extensible Markup Language (XML) and Web services:

- The Programmable Web: Web Services Provides Building Blocks for the Microsoft .NET Framework

  http://msdn.microsoft.com/msdnmag/issues/0900/WebPlatform/WebPlatform.asp

- Visual Studio .NET: Build Web Applications Faster and Easier Using Web Services and XML

  http://msdn.microsoft.com/msdnmag/issues/0900/VSNET/VSNET.asp

- Web Services: Building Reusable Web Components with SOAP and ASP .NET

  http://msdn.microsoft.com/msdnmag/issues/01/02/webcomp/webcomp.asp

- Documenting Your Web Service

  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dn_voices_webservice/html/service07182001.asp

- XML in .NET: .NET Framework XML Classes and C# Offer Simple, Scalable Data Manipulation

  http://msdn.microsoft.com/msdnmag/issues/01/01/xml/xml.asp

- The XML Files: Understanding XML Namespaces

  http://msdn.microsoft.com/msdnmag/issues/01/07/xml/xml0107.asp

- Wicked Code: CityView App: Build Web Service Clients Quickly and Easily with C#

  http://msdn.microsoft.com/msdnmag/issues/01/04/Wicked/Wicked0104.asp

Visual Basic .NET

The following resources provide information about Microsoft Visual Basic® .NET:

- Visual Basic .NET: New Programming Model and Language Enhancements Boost Development Power

  http://msdn.microsoft.com/msdnmag/issues/01/02/vbnet/vbnet.asp
- Serving the Web: Windows Forms in Visual Basic .NET

  http://msdn.microsoft.com/msdnmag/issues/01/04/serving/serving0104.asp
- Basic Instincts: New Features in Visual Basic .NET: Variables, Types, Arrays, and Properties

  http://msdn.microsoft.com/msdnmag/issues/01/05/Instincts/Instincts0105.asp
- Basic Instincts: Exploiting New Language Features in Visual Basic .NET, Part 2

  http://msdn.microsoft.com/msdnmag/issues/01/08/Instincts/Instincts0108.asp
- Visual Basic .NET: Tracing, Logging, and Threading Made Easy with .NET

  http://msdn.microsoft.com/msdnmag/issues/01/07/vbnet/vbnet.asp
- Advanced Basics: Happy 10th Birthday, Visual Basic

  http://msdn.microsoft.com/msdnmag/issues/01/07/basics/basics0107.asp

C#

The following resources provide information about C#:

- Sharp New Language: C# Offers the Power of C++ and Simplicity of Visual Basic

  http://msdn.microsoft.com/msdnmag/issues/0900/csharp/csharp.asp
- C++ -> C#: What You Need to Know to Move from C++ to C#

  http://msdn.microsoft.com/msdnmag/issues/01/07/ctocsharp/ctocsharp.asp
- Visual Studio .NET: Managed Extensions Bring .NET CLR Support to C++

  http://msdn.microsoft.com/msdnmag/issues/01/07/vsnet/vsnet.asp
- Design Patterns: Solidify Your C# Application Architecture with Design Patterns

  http://msdn.microsoft.com/msdnmag/issues/01/07/patterns/patterns.asp
- C# and the Web: Writing a Web Client Application with Managed Code in the Microsoft .NET Framework

  http://msdn.microsoft.com/msdnmag/issues/01/09/cweb/cweb.asp

Mobile applications

The following resources provide information about mobile applications:

- .NET Mobile Web SDK: Build and Test Wireless Web Applications for Phones and PDAs

  http://msdn.microsoft.com/msdnmag/issues/01/06/Mobile/Mobile.asp

- SQL Server CE: New Version Lets You Store and Update Data on Handheld Devices

  http://msdn.microsoft.com/msdnmag/issues/01/06/sqlce/sqlce.asp

  Pocket PC: Seamless App Integration with Your Desktop using ActiveSync 3.1

  http://msdn.microsoft.com/msdnmag/issues/01/06/PPC/PPC.asp

- Security Models and Scenarios for SQL Server 2000 Windows CE Edition

  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql2K/html/sscesecurity.asp?frame=true

MSDN Voices Columns

The following resources are columns on MSDN:

- Diving Into Data Access

  http://msdn.microsoft.com/columns/data.asp

- Nothin' But ASP .NET

  http://msdn.microsoft.com/columns/aspnet.asp

- Working with C#

  http://msdn.microsoft.com/columns/csharp.asp

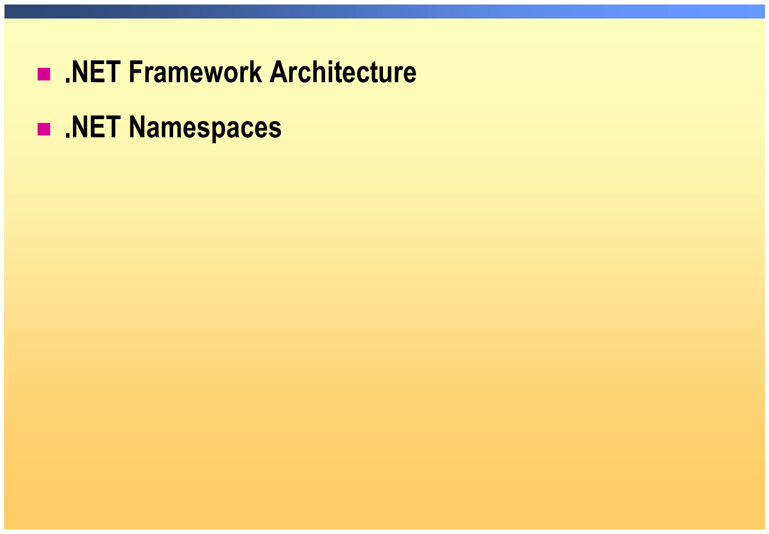
msdn® training

Appendix D: Microsoft .NET Framework Overview (Prerelease)

Contents

Microsoft®

Overview



- .NET Framework Architecture
- .NET Namespaces

Introduction

This appendix provides an overview of the .NET Framework.

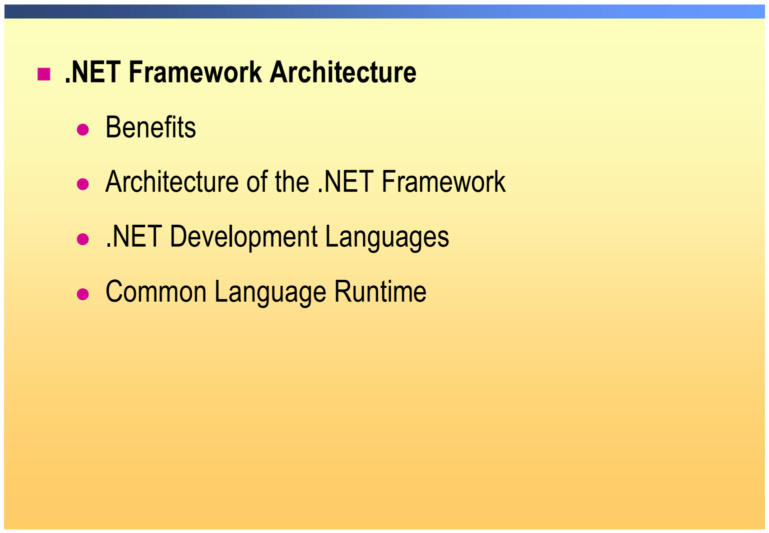
Objectives

1. Describe the .NET Framework architecture, and its major features.

After completing this module, students will be able to:

- Diagram the .NET Framework architecture.
- Reference namespaces in projects.
- Create a new namespace, or extend an existing namespace.

Lesson: .NET Framework Architecture



- **.NET Framework Architecture**
  - Benefits
  - Architecture of the .NET Framework
  - .NET Development Languages
  - Common Language Runtime

Introduction

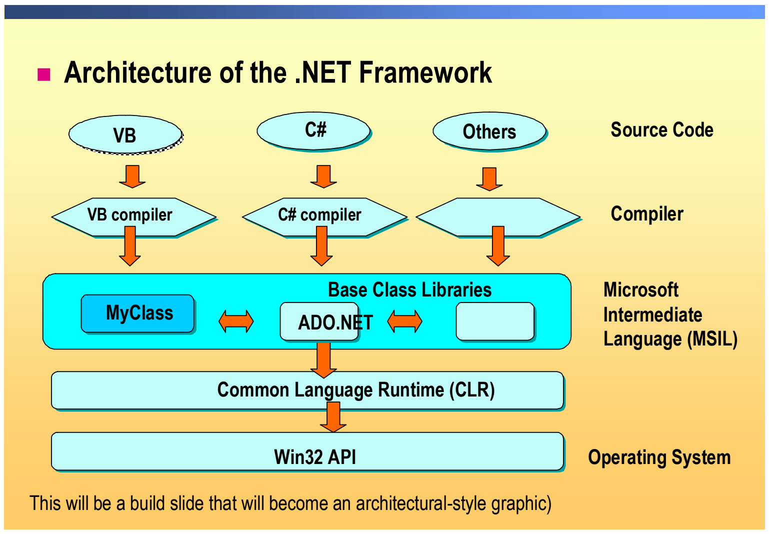
This lesson introduces the .NET Framework and explains how ADO.NET relates to other .NET Framework components.

Lesson Objectives

After completing this lesson, you will be able to:

- List the benefits of using the .NET Framework.
- Diagram the general architecture of the .NET Framework.
- Discuss the .NET languages and the common language runtime.

Architecture of the .NET Framework


■ Architecture of the .NET Framework

This will be a build slide that will become an architectural-style graphic)

Introduction

The .NET Framework is a computing platform that simplifies application development in the highly distributed environment of the Internet. The major components of the .NET Framework are:

- Language compilers
- .NET base class libraries (written in C#, pre-compiled to MSIL)
- Common language runtime
- ASP.NET application and XML Web service platform

Language compilers

The .NET Framework supports many programming languages. The strategic languages are Visual Basic and C#, but you can also use C++, Microsoft JScript®, and more than 30 others such as Perl and COBOL.

The compilers produce Microsoft Intermediate Language code that is then compiled "just in time" by the common language runtime.

The .NET Framework class libraries

The .NET Framework base class libraries include classes for the following:

- Data access (ADO.NET)
- Building Microsoft Win32® applications
- Building web applications (ASP.NET)
- Building XML Web services
- XML support

The common language runtime

The common language runtime simplifies deployment and management of applications. It performs the following:

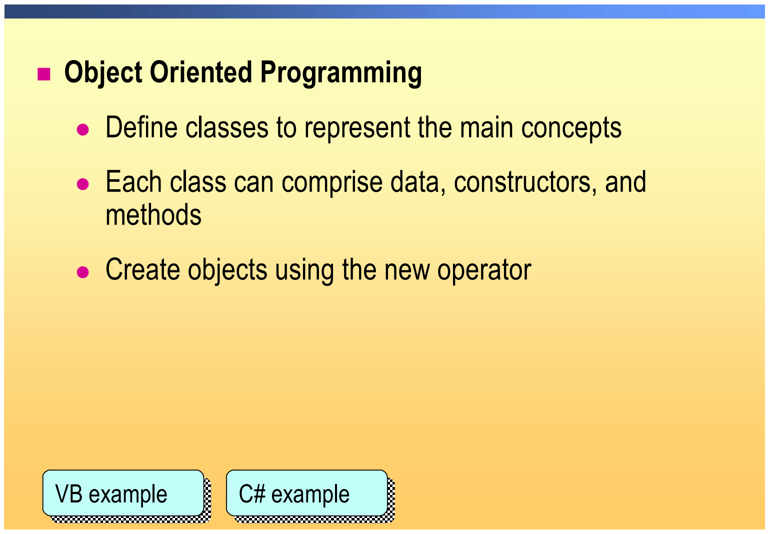
- Manages running code, while:
- Verifying type safety.
- Providing garbage collection and error handling.
- Providing code access security for semi-trusted code.
- Provides a common type system, including:
- Value types (integer, float, user defined, and so on)
- Objects and interfaces
- Provides access to system resources, including native API and COM interoperability.

Benefits

The .NET Framework provides you with the following benefits:

- Exposes a language-independent yet consistent programming model across all tiers of an application.
- Provides seamless interoperability with and easy migration from existing technologies.
- Offers a simplified application development with a consistent set of classes and interfaces.
- Provides a unified programming model with consistent application programming interfaces (API) available across all languages and application types.
- Utilizes Web standards by providing rich XML support, including integrated XML and SOAP support, standard protocols, and a stateless environment.
- Increases productivity because .NET applications are easy to deploy, run, and maintain. The common language runtime makes applications developed with the .NET Framework easy to use because it helps avoid registration and version problems.

Object Oriented Programming



- **Object Oriented Programming**
  - Define classes to represent the main concepts
  - Each class can comprise data, constructors, and methods
  - Create objects using the new operator

VB example
C# example

Introduction

One of the major features of the common language runtime is its support for object oriented programming. A language compiler does not have to support all the features of the common language runtime, but the two strategic .NET languages, Visual Basic and C#, do. This means that you define classes to represent the important concepts in your application. Each class can comprise data, constructors to initialize the data, and methods to encapsulate the data.

Example of defining a class in Visual Basic

The following example defines a class in Visual Basic, with a single instance variable:

```
' Define a class with a balance instance variable

Public Class BankAccount
  Private dblBalance As Double    ' Partial class definition
End Class
```

Example of defining a class in C#

The following example shows how to define the **BankAccount** class in C#:

```
// Define a class with a balance instance variable

public class BankAccount
{
  private double dblBalance;     // Partial class definition
}
```

Defining methods in a class

Methods provide the behavior for a class, and encapsulate the data in the class. You can also define constructors, to initialize the object when it is created.

Example of defining methods in Visual Basic

This example defines a constructor for the **BankAccount** class in Visual Basic. The example also includes a **Credit** subroutine and a **Debit** function. Subroutines do not return a value, but functions return a single value.

```
' Constructor to initialize the state of an object

Public Sub New(ByVal initBalance As Double)
  dblBalance = initBalance
End Sub

' Credit subroutine

Public Sub Credit(ByVal amount As Double)
  dblBalance += amount
End Sub

' Debit function. Return true if the balance is still OK,
' or false if overdrawn

Public Function Debit(ByVal amount As Double) As Boolean
  dblBalance -= amount
  Debit = (dblBalance >= 0)
End Function
```

Example of defining methods in C#

This example defines a constructor, a **Credit** method, and a **Debit** method in C#. In C#, the constructor has the same name as the class. The **void** keyword in the **Credit** method indicates that the method does not return a value.

```
// Constructor to initialize the state of an object

public BankAccount(double initBalance)
{
  dblBalance = initBalance;
}

// Credit the balance

public void Credit(double amount)
{
  dblBalance += amount;
}

// Debit the balance, and return true or false

public bool Debit(double amount)
{
  dblBalance -= amount;
  return (dblBalance >= 0);
}
```

Defining properties in a class

Properties provide the attributes for a class, and encapsulate the data in the class.

Example of defining properties in Visual Basic .NET

This example defines a Balance property for the **BankAccount** class in Visual Basic.

```
Public ReadOnly Property Balance() As Double
  Get
    Balance = dblBalance
  End Get
End Property
```

Example of defining properties in C#

This example defines a Balance property for the **BankAccount** class in C#.

```
public double Balance
{
  get
  {
    return dblBalance;
  }
}
```

Creating and using objects

An object is an instance of a class. To create an object, use the **new** operator. To use an object, call the public methods defined in the class.

Example of creating and using objects in Visual Basic

This example creates a **BankAccount** object in Visual Basic, with an initial balance of $100. The example shows how to call the **Credit** and **Debit** methods.

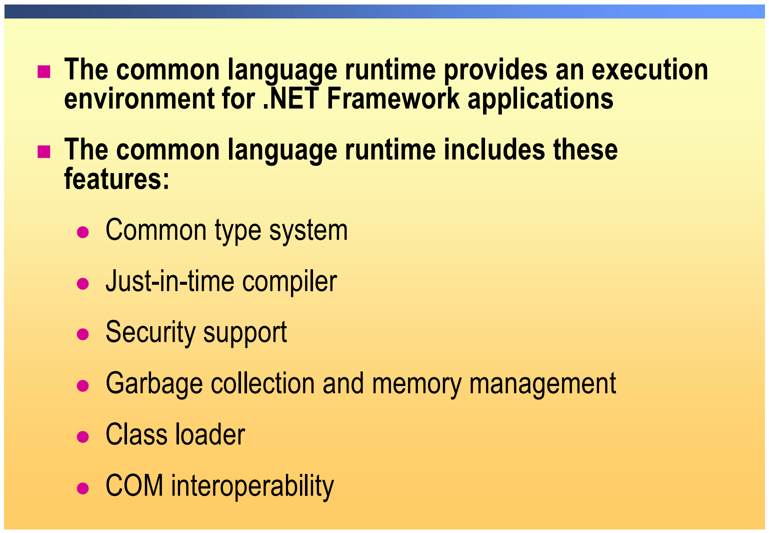
```
' Create a BankAccount object, and call methods on the object

Dim myAccount As New BankAccount( _
  Convert.ToDouble(txtAmount.Text))
myAccount.Credit(50)
myAccount.Debit(75)
```

Example of creating and using objects in C#

This example creates and uses a **BankAccount** object in C#:

```
// Create and use a BankAccount object in C#

BankAccount myAccount = new BankAccount(
  Convert.ToDouble(txtAmount.Text));
myAccount.Credit(50);
myAccount.Debit(75);
```

The Common Language Runtime



- **The common language runtime provides an execution environment for .NET Framework applications**
- **The common language runtime includes these features:**
  - Common type system
  - Just-in-time compiler
  - Security support
  - Garbage collection and memory management
  - Class loader
  - COM interoperability

Introduction

The common language runtime provides an execution environment for .NET Framework code. The runtime gives a secure, robust, CPU-independent environment for managed applications in the .NET Framework.

Definition of common language runtime

The common language runtime includes the following features:

- Common type system
- Just-in-time (JIT) compiler, from Microsoft Intermediate Language (MSIL) to native code
- Security support
- Garbage collection and memory management
- Class loader
- COM interoperability

Common type system

The common type system (CTS) defines a common set of data types that are available in all .NET Framework development languages.

The CTS also defines the common language specification (CLS). The CLS is a set of rules and programming language features that all .NET Framework development languages must support. The CLS enables you to integrate code written different in languages, and ensures type safety between these languages.

Just-in-time compiler

When you develop code targeted at the common language runtime, you compile your code to Microsoft Intermediate Language. MSIL is a CPU-independent instruction set, designed for efficient translation into native code.

When you run your code on a particular platform, the runtime must translate the MSIL instructions into native code. For this purpose, the runtime includes a JIT compiler for each CPU that it supports. This means you can execute your code on any platform for which the runtime is available.

Security

Security is an important issue in the era of Web-enabled applications. The common language runtime supports and enforces a strict security model, to address these needs.

The security model uses code access security to control access to restricted resources. Code access security defines the set of permissions required to access system resources. Administrators create and configure security policy files, which associate these permissions with groups of code. When your code tries to access a protected resource, the runtime security system checks that the code has the required permission.

Garbage collection and memory management

The common language runtime manages how your application allocates and releases memory.

Each time you create a new object, the runtime allocates memory from the heap. When the object is no longer referenced in your application, the object becomes available for garbage collection. The garbage collector periodically sweeps through memory, reclaiming objects that are no longer required.

Garbage collection makes it easier for you to design your applications, because you do not have to delete objects when they are no longer required. The garbage collector deletes objects for you.

Garbage collection also helps to prevent memory leaks, which occur in unmanaged code if you forget to delete an object when it is no longer needed.

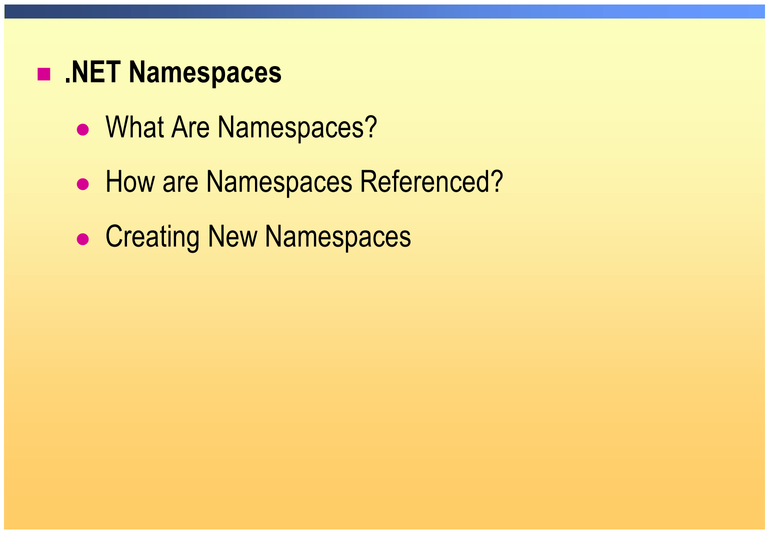
Class loader

The common language runtime includes a class loader, which dynamically loads classes into the runtime when needed.

COM interoperability

The common language runtime includes execution services for COM interoperability. This enables you to write .NET Framework code to interface with existing COM components. You can also expose .NET Framework components as COM interfaces.

Lesson: .NET Namespaces



- **.NET Namespaces**
  - What Are Namespaces?
  - How are Namespaces Referenced?
  - Creating New Namespaces

Introduction

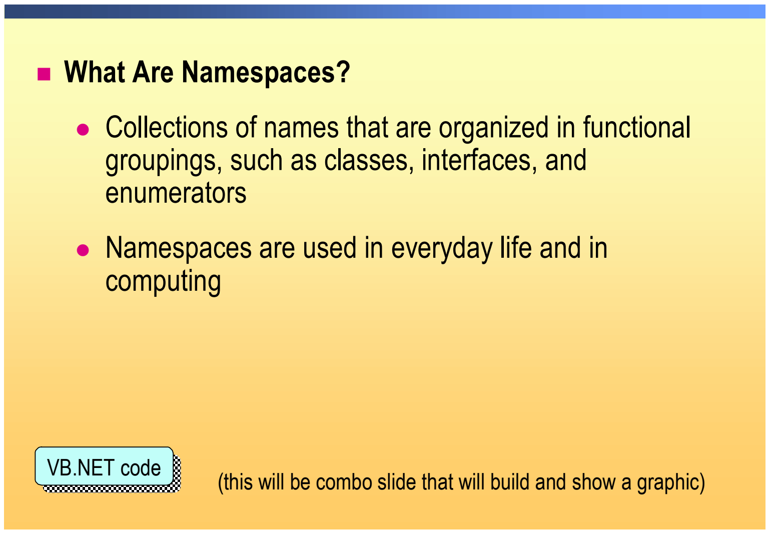
This lesson introduces the .NET namespaces.

Lesson Objectives

After completing this lesson, you will be able to:

- Define a namespace.
- Reference a namespace.

What Are Namespaces?



- **What Are Namespaces?**
  - Collections of names that are organized in functional groupings, such as classes, interfaces, and enumerators
  - Namespaces are used in everyday life and in computing

VB.NET code

(this will be combo slide that will build and show a graphic)

Introduction

The .NET Framework classes and data types are organized into namespaces. A *namespace* is a logical collection of related classes and data types.

Namespaces divide the .NET Framework class library into meaningful partitions. This makes it easy for developers to find the classes that they need. Namespaces also help to prevent name clashes between similar classes in different namespaces.

Definition

In simplest terms, a namespace is a collection of names. In application development, a namespace is a bounded area of an organized collection of names that is accessible to programs. Namespaces are often organized in a hierarchy so that each entry in the namespace below the top level belongs to a group.

Everyday examples

Examples of namespaces exist outside of computers in everyday life. A telephone directory is one type of namespace, where names are organized by business type. A practical example of a namespace in everyday life is a collection of mail slots in a business. These slots and the names that are associated with them are managed to be current and accurate. The names are often organized by group, by function, or in alphabetical order.

Computer examples

In the computer world, namespaces consist of collections of classes, interfaces, enumerations, and related programming tools. Examples of namespaces include the System.Web namespace, which is used by ASP.NET, and the System.Xml namespace, which is used for XML processing.

A good non-example is the Win32 API. This collection of function calls does not use namespaces, so all 600+ functions require unique names (making the names long and complicated) and have no logical grouping. This is one of the reasons writing applications for Windows is so hard using direct calls to the operating system.

Namespace implementation

Namespaces do not indicate physical implementation. For example, a single namespace can be implemented by multiple assemblies (EXEs and DLLs). For example, some classes in the System.Web namespace are implemented in the System.Web.dll assembly, and others are implemented in the System.Web.RegularExpressions.dll assembly.

The reverse is also true: multiple namespaces can be implemented by a single assembly. For example, the System.Data.dll assembly implements the System.Data.SqlClient and System.Data.OleDb namespaces, as well as others.

Namespace hierarchy

The .NET Framework namespaces use a ‘dot’ naming convention to indicate namespace hierarchy. The following table describes some of the standard .NET Framework namespaces.

Namespace	Description
System	General-purpose classes and interfaces, such as the **Math** class.
System.Web.UI	Classes and interfaces for ASP .NET user interfaces, such as the **Page** class that represents a Web form.
System.Web.UI.WebControls	Classes and interfaces for ASP .NET controls, such as the **TextBox** and **Button** classes.
System.Data.SqlClient	Classes and interfaces for accessing SQL Server 7.0 and later databases;. For example, the **SqlCommand** class represents a SQL command.

Class inheritance hierarchy

Do not confuse namespace hierarchy (logical groups) with class inheritance hierarchy. Use the online help to discover the class inheritance hierarchy for a class.

For example, the **SqlDataAdapter** class in the System.Data.SqlClient namespace inherits from the **DbDataAdapter** class in the System.Data.Common namespace.

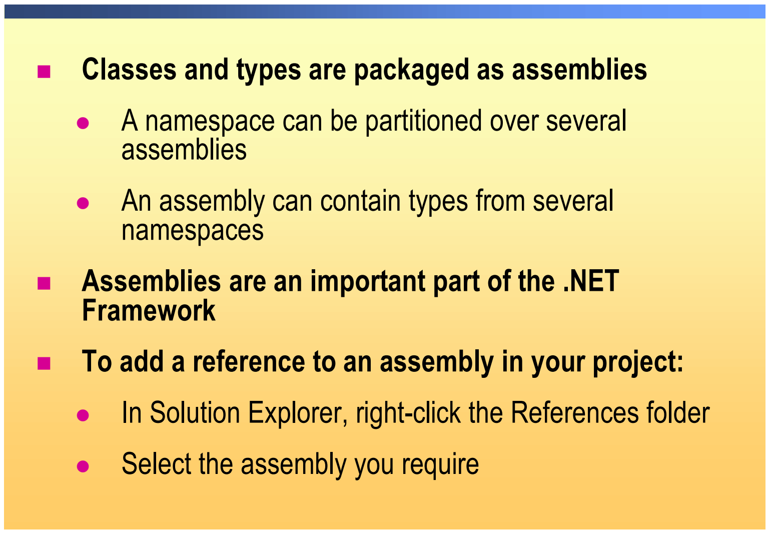
So the namespace hierarchy of SqlDataAdapter is:

```
System
    Data
        SqlClient
            SqlDataAdapter
```

But the class inheritance hierarchy of SqlDataAdapter is:

```
Object
    MarshallByRefObject
        Component
            DataAdapter
                DbDataAdapter
                    SqlDataAdapter
```

How Are Namespaces Referenced?



- **Classes and types are packaged as assemblies**
  - A namespace can be partitioned over several assemblies
  - An assembly can contain types from several namespaces
- **Assemblies are an important part of the .NET Framework**
- **To add a reference to an assembly in your project:**
  - In Solution Explorer, right-click the References folder
  - Select the assembly you require

Introduction

Classes and types are packaged as assemblies. You can partition a namespace over several assemblies. An assembly can contain classes and data types from several related namespaces. Before you can use a namespace, you must reference a physical DLL.

Importance of assemblies

Assemblies provide the following capabilities:

- Assemblies are the unit of deployment in the .NET Framework. To install an assembly, copy the assembly into the file system. There is no need to register the assembly in the registry.
- Assemblies are self-describing. Each assembly has a manifest file, which contains metadata about the classes and data types in the assembly.
- Assemblies contain version information. This makes it possible to install and execute different versions of an assembly side by side, without conflict.
- Assemblies are the unit of security in the .NET Framework. You can specify the permissions required to use and run each assembly.
- Assemblies can be declared as private to an application, or shared among applications. Application-private assemblies are useful if you only want an assembly to be visible in a single application. Shared assemblies are useful for system classes and data types that are used by many applications.

Adding an assembly reference to a project

To add a reference to an assembly in your project, follow these steps:

1. Open the Solution Explorer in Microsoft Visual Studio® .NET.
2. Expand the project in which you want to make a reference.
3. Right-click the References folder.
4. On the menu, choose **Add Reference**.
5. Select the checkbox for the DLL that implements the part of the namespace that contains the class that you want to use.
6. Select **OK**.

Practice

In this practice, you will learn how to add assembly references to a project. You will also see how to import namespaces to simplify your code.

In the first part of the practice, you will create a new Windows application and declare a System.Messaging.MessageQueue variable. You must also add a reference to the System.Messaging assembly.

1. Run Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project** to create a new project.
3. In the **New Project** dialog box, select the options in the following table, and then click **OK**.

Option	Selection
Project Types	Visual Basic Project
Templates	Windows Application

4. In the Solution Explorer, right-click **Form1.vb** and then click **View Code**.
5. Declare the following instance variable at the start of the **Form1** class:

```
Private mq As System.Messaging.MessageQueue
```

6. On the **Build** menu, click **Build**. The following compiler error occurs:

```
Type is not defined: 'System.Messaging.MessageQueue'
```

7. In the Solution Explorer, right-click **References**, and then click **Add Reference**.
8. In the **Add Reference** dialog box, select the **.NET** tab. In this tab, select the component name **System.Messaging.dll**, click **Select**, and then click **OK**.
9. Notice that the References folder now includes the System.Messaging assembly.
10. Build the project. The build succeeds.

In the next part of the practice, you will import the System.Messaging namespace. This will enable you to use the **MessageQueue** data type directly in the code, without specifying its namespace each time.

11. View the code for **Form1.vb**. Modify the declaration of the instance variable at the start of the class, as follows:

```
Private mq As MessageQueue
```

12. Build the project. The following compiler error occurs:

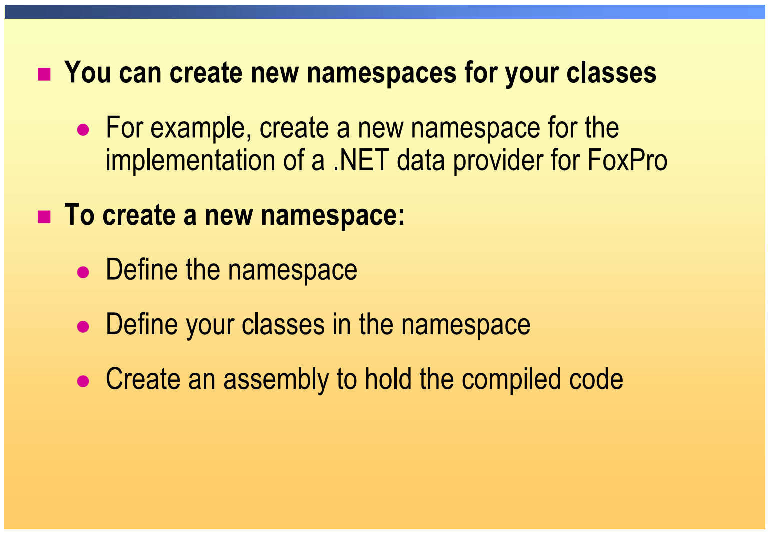
```
Type is not defined: 'MessageQueue'
```

13. Add the following **Imports** statement at the start of **Form1.vb**:

```
Imports System.Messaging
```

14. Build the project again. The build succeeds.

Creating New Namespaces



- **You can create new namespaces for your classes**
  - For example, create a new namespace for the implementation of a .NET data provider for FoxPro
- **To create a new namespace:**
  - Define the namespace
  - Define your classes in the namespace
  - Create an assembly to hold the compiled code

Introduction

You can create a new namespace to act as a logical container for your classes and data types. Placing your code in a distinct namespace helps you to avoid name clashes with code written elsewhere. It also makes it easier for developers to use your classes and data types in their programs.

Scenario

For example, in ADO.NET, each .NET data provider is located in a different namespace. The following table shows the namespaces for the data providers that are packaged in the .NET Framework.

Data provider	Namespace
SQL Server .NET	System.Data.SqlClient
OLE DB .NET	System.Data.OleDb

ADO.NET provides classes to help you to define your own .NET data providers. This might be useful if you want to provide a simplified data access architecture, or to expose provider-specific behavior to consumers.

To define your own .NET data provider, you must first create a new namespace. In this namespace, place the classes and data types for your new data provider.

Creating a new namespace

To create a new namespace, follow these steps:

1. Define the namespace in your source code. Choose a unique name for the namespace.
2. Define your classes and data types in the namespace. You can place the classes and data types in separate source files, as long as each source file includes the same namespace definition.
3. Create an assembly to hold the compiled code. If your code makes use of any other assemblies, include a reference to these assemblies.

Example of creating a data provider in Visual Basic .NET

This example defines a namespace in Visual Basic, to represent a new data provider. The namespace is called System.Data.Fox. The System.Data prefix indicates that the namespace represents a data provider.

```
' Visual Basic .NET

Namespace System.Data.Fox

  Class FoxConnection
      Inherits Component
      Implements IDbConnection, ICloneable

      ' . . .

  End Class

  ' Implement other.NET data provider classes and interfaces,
  ' such as IDbCommand and IDbDataAdapter

End Namespace
```

The following command line compiles all Visual Basic files in the current directory, and creates a DLL assembly named System.Data.Fox.dll. This assembly includes references to the System.dll and System.Data.dll assemblies, which contain classes needed by the new data provider.

```
vbc /target:library
    /out:System.Data.Fox.dll *.vb
    /r:System.dll /r:System.Data.dll
```

Review

- .NET Framework Architecture
- .NET Namespaces