

Building XML-Enabled Applications using Microsoft[®] SQL Server[™] 2000

Delivery Guide

Course Number: 2091A

Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BackOffice, MSDN, MS-DOS, SQL Server, Visual Basic, Visual InterDev, Visual Studio, Windows, Windows Media, Windows NT, and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Contents

Introduction

Course Materials	2
Prerequisites	3
Course Outline	4
Setup	5
Microsoft Certified Professional Program	6
Facilities	8

Module 1: Using XML and Microsoft SQL Server 2000

Overview	1
Building Solutions with XML and SQL Server 2000	2
XML Technology Review	11
XML Support in SQL Server 2000	21
Lab 1: Working with XML	28
Multimedia: Microsoft BizTalk Server 2000 and XML (Optional)	33
Review	34

Module 2: Retrieving XML Data

Overview	1
Using the FOR XML Clause to Retrieve Data	2
Controlling XML Output	13
Lab 2.1: Retrieving XML Data	26
Generating Custom XML Formats	32
Lab 2.2: EXPLICIT Queries	42
Best Practices	49
Review	50

Module 3: Manipulating XML with Transact-SQL

Overview	1
Creating a Rowset from an XML Document	2
Specifying the Structure of a Rowset	13
Lab 3: Using OPENXML	26
Best Practices	33
Review	34

Module 4: Using ADO to Access XML Data

Overview	1
Retrieving XML Data with ADO	2
Lab 4.1: Creating an ADO Client Application	16
Using ADO and XML on an ASP-Based Web Site	22
Lab 4.2: Using ADO on a Web Site	27
Best Practices	31
Review	32

Module 5: Using HTTP to Access XML Data

Overview	1
Enabling HTTP Access to SQL Databases	2
Publishing XML Templates on the Server	14
Lab 5.1: Publishing a SQL Server Database	23
Posting Query Templates from the Client	30
Publishing a Database Object	37
Lab 5.2: Accessing a Database Object	42
Best Practices	45
Review	47

Module 6: Using Mapping Schemas

Overview	1
Using Mapping Schemas	2
Using Schemas to Retrieve Data	11
Lab 6.1: Using Mapping Schemas	21
Mapping Multiple Tables	30
Lab 6.2: Mapping Multiple Tables	37
Annotating an Existing Schema	42
Lab 6.3: Annotating an Existing Schema	52
Best Practices	56
Review	57

Module 7: XML for SQL Tools

Overview	1
Modifying Data with Updategrams	2
Lab 7.1: Using Updategrams	17
Inserting XML Data with the Bulk Load Facility	22
Lab 7.2: Bulk Loading XML Data	31
Best Practices	34
Review	36

About This Course

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

Description

This course provides students with the knowledge and skills they need to build applications that exchange data with Microsoft® SQL Server™ in Extensible Markup Language (XML) format. The course covers the XML-based features of Microsoft SQL Server 2000, including XML-based enhancements to the T-SQL language; generating results in XML format; using query templates; configuring SQL Server for Internet/intranet access; publishing SQL Server data on the Internet by using the HTTP protocol; XML data access by using Microsoft ActiveX® Data Objects (ADO) 2.6 and the SQL Server OLE DB (SQLOLEDB) Provider; writing XML Path Language (XPath) queries; using XML Data-Reduced (XDR) Schemas; and updating a database with XML data.

Audience

The target audience for this course includes:

- Developers of database client applications, including Internet and n-tiered application clients.
- SQL Server database developers who need to query, insert, update, or delete data from a database by using XML.
- SQL Server database administrators who need to publish SQL data as XML.

Student Prerequisites

This course requires that students meet the following prerequisites:

- The ability to write basic Transact-SQL statements that query, insert, update, and delete information from a database. You should be able to use JOIN statements to retrieve information from multiple tables.
- Successful completion of Course 2073A, *Programming a Microsoft SQL Server Database*, or equivalent experience designing relational databases, including the use of primary and foreign keys. You need to know how to create databases, tables, and stored procedures. Students should also be familiar with the SQL Server Query Analyzer and SQL Profiler tools.
- Successful completion of Course 1905A, *Building XML-Based Web Applications*, or equivalent experience using XML, the Extensible Stylesheet Language (XSL), and Extensible Stylesheet Language Transformations (XSLT). This includes knowledge of attributes and elements, schemas, and XPath.
- Experience using ADO to access a data source.
- The ability to write Microsoft Visual Basic® Scripting Edition (VBScript) code.
- The ability to read standard Hypertext Markup Language (HTML) and build a form to gather and present data.
- The ability to create Active Server Pages (ASP) that use the **POST** and **GET** methods, the **Response** object, and the **Session** object.

Course Objectives

The modules in this course provide instruction and learning activities that allow students to achieve the following objectives.

In module	Students learn to:
1	Describe the XML features provided by SQL Server 2000 and the SQL Server-XML architecture.
2	Generate and specify the structure of an XML document from SQL Server data by using Transact-SQL.
3	Insert XML data into SQL Server, and use XML to modify existing data.
4	Use ADO to access SQL Server data and generate XML.
5	Publish a SQL Server database on the Internet/intranet over HTTP, and query the data by using a URL.
5	Create and use XML templates to query the database.
6	Create and use annotated schemas to query and transform SQL Server data.
7	How to use updategrams and the Bulk Load facility..

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
9:30	10:30	Module 1: Using XML and Microsoft SQL Server 2000
10:30	10:45	Break
10:45	11:15	Lab 1: Working with XML
11:15	12:00	Module 2: Retrieving XML Data
12:00	1:00	Lunch
1:00	1:30	Lab 2.1: Retrieving XML Data
1:30	2:45	Module 2: Retrieving XML Data (<i>continued</i>)
2:45	3:00	Break
3:00	3:30	Lab 2.2: EXPLICIT Queries
3:30	4:30	Module 3: Manipulating XML with Transact-SQL

Day 2

Start	End	Module
9:00	9:30	Day 1 review
9:30	10:30	Module 3: Manipulating XML with Transact-SQL (<i>continued</i>)
10:30	10:45	Break
10:45	11:15	Lab 3: Using OPENXML
11:15	12:00	Module 4: Using ADO to Access XML Data
12:00	1:00	Lunch
1:00	1:15	Lab 4.1: Creating an ADO Client Application
1:15	1:30	Module 4: Using ADO to Access XML Data (<i>continued</i>)
1:30	2:00	Lab 4.2: Using ADO on a Web Site
2:00	2:30	Module 5: Using HTTP to Access XML Data
2:30	2:45	Break
2:45	3:30	Lab 5.1: Publishing a SQL Server Database
3:30	4:00	Module 5: Using HTTP to Access XML Data (<i>continued</i>)

Day 3

Start	End	Module
9:00	9:30	Day 2 review
9:30	10:00	Module 5: Using HTTP To Access XML Data (<i>continued</i>)
10:00	10:15	Lab 5.2: Accessing a Database Object
10:15	10:30	Module 5: Best Practices and Review
10:30	10:45	Break
10:45	11:15	Module 6: Using Mapping Schemas
11:15	11:45	Lab 6.1: Using Mapping Schemas
11:45	12:45	Lunch
12:45	1:15	Module 6: Using Mapping Schemas (<i>continued</i>)
1:15	1:45	Lab 6.2: Mapping Multiple Tables
1:45	2:15	Module 6: Using Mapping Schemas (<i>continued</i>)
2:15	2:30	Break
2:30	3:00	Lab 6.3: Annotating an Existing Schema
3:00	3:45	Module 7: XML for SQL Tools
3:45	4:15	Lab 7.1: Using Updategrams
4:45	5:00	Lab 7.2: Bulk Loading XML Data

Trainer Materials Compact Disc Contents

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

- *Autorun.exe*. When the CD is inserted into the CD-ROM drive, or when you double-click the autorun.exe file, this file opens the CD and allows you to browse the Student Materials or Trainer Materials compact discs, or install Microsoft Internet Explorer.
- *Default.htm*. This file opens the Trainer Materials Web page.
- *Readme.txt*. This file contains a description of the compact disc contents and setup instructions in ASCII format (non-Microsoft Word document).
- *2091A_ms.doc*. This file is the Manual Classroom Setup Guide. It contains the steps for manually installing the classroom computers.
- *2091A_sg.doc*. This file is the 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.
- *StudentCD*. This folder contains the Web page that provides students with links to course-related resources, including additional reading, review and lab answers, lab files, multimedia presentations, and course-related Web sites.
- *Errorlog*. This folder contains a template that is used to record any errors and corrections that you find in the course.
- *Fonts*. This folder contains fonts that are required in order to view the Microsoft PowerPoint® presentation and Web-based materials.
- *Mplayer*. This folder contains files that are required to install Microsoft Windows Media™ Player.
- *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.
- *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 in order 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 CD is inserted into the CD-ROM drive, or when you double-click the autorun.exe file, this file opens the CD and allows you to browse the Student Materials CD or install Internet Explorer.
- *Default.htm*. This file opens the Student Materials Web page. It provides you with course-related resources, including additional reading, review and lab answers, lab files, multimedia presentations, and course-related Web sites.
- *Readme.txt*. This file contains a description of the compact disc contents and setup instructions in ASCII format (non-Microsoft Word document).
- *2091A_ms.doc*. This file is the Classroom Setup Guide. It contains a description of classroom requirements, classroom setup instructions, and the classroom configuration.
- *AddRead*. This folder contains additional reading related to this course. If there are no additional reading files, this folder does not appear.
- *Appendix*. This folder contains appendix files for this course. If there are no appendix files, this folder does not appear.
- *Democode*. This folder contains demonstration code. If there is no demonstration code, the Democode folder does not appear.
- *Fonts*. This folder contains fonts that are required in order to view the PowerPoint presentation and Web-based materials.
- *Ie5*. This folder contains Internet Explorer 5.5.
- *Labs*. 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. If this course does not include any multimedia presentations, this folder does not appear.
- *Menu*. This folder contains elements for autorun.exe.
- *Mplayer*. This folder contains files that are required to install Windows Media Player.
- *Practices*. This folder contains files that are used in the hands-on practices. If there are no practices, the Practices folder does not appear.
- *Sampapps*. This folder contains the sample applications associated with this course. If there are no associated sample applications, the Sampapps folder does not appear.
- *Sampcode*. This folder contains sample code that is accessible through the Web pages on the Student Materials compact disc. If there is no sample code, the Sampcode folder does not appear.
- *Sampsite*. This folder contains files that create the sample site associated with this course. If there is no sample site, the Sampsite folder does not appear.

- *Setup*. This folder contains additional files that may be required for lab setup. If no additional files are required, the Setup folder does not appear.
- *Webfiles*. This folder contains the files that are required in order 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. If no Word documents are included, this folder does not appear.

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.
<i>italic</i>	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, <i>[filename]</i> 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.
.	
.	

Introduction

Contents

Introduction	1
Course Materials	2
Prerequisites	3
Course Outline	4
Setup	5
Microsoft Certified Professional Program	6
Facilities	8



Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BackOffice, BizTalk, MSDN, MS-DOS, SQL Server, Visual Basic, Visual C++, Visual InterDev, Visual J++, Visual Studio, Windows, Windows Media, Windows NT, and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Instructor Notes

Presentation:
30 Minutes

This module provides students with an overview of the course content, materials, and logistics.

Course Materials and Preparation

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.

Module Strategy

Use the following strategies to present this module:

- **Course 2091A: *Building XML-Enabled Applications Using Microsoft® SQL Server™ 2000***
Show the slide that displays the course number and course title.
- **Introduction**
Welcome students to the course and introduce yourself. Provide a brief overview of your background to establish credibility.
Have students introduce themselves and provide their background, product experience, and expectations of the course.
Record student expectations on a whiteboard or flip chart that you can reference later in class.
- **Course Materials**
Explain the purpose of all materials used in this course.
Tell students that they will have an opportunity at the end of the class to provide feedback on the course and facilities by using the Microsoft Online Evaluation System.
- **Prerequisites**
Provide the students with the list of prerequisites that they should have met before taking this course. This is an opportunity for you to identify students who may not have the appropriate background or experience to attend this course.
- **Course Outline**
Provide an overview of each module and what students will learn.
Explain how this course will meet students' expectations by relating the information covered in individual modules to their expectations.
- **Setup**
Provide the students with any necessary setup information for the course.
- **Microsoft Certified Professional Program**
Inform students about the Microsoft Certified Professional (MCP) program and the various certification options.
- **Facilities**
Explain the facility information for the training site.

Introduction

Topic Objective

To introduce yourself, establish credibility, meet students, and set student expectations for the course.

Lead-in

Good morning. Welcome to Course 2091A, *Building XML-Enabled Applications Using Microsoft® SQL Server™ 2000*.

My name is...

- Name
- Company affiliation
- Title / function
- Job responsibility
- SQL Server experience
- XML development experience
- Expectations for the course

*****ILLEGAL FOR NON-TRAINER USE*****

Introduce yourself.

Provide a brief overview of your background to establish credibility as an instructor in this set of technologies.

Ask students to introduce themselves, addressing the bulleted items on the slide.

Delivery Tip

As students introduce themselves, use a whiteboard or flip chart to record their expectations of the course.

Course Materials

Topic Objective

To identify and describe the course materials.

Lead-in

We have provided everything you need for this course. You will find the following materials at your desk...

- **Name card**
- **Student workbook**
- **Student Materials compact disc**
- **Course evaluation**

*****ILLEGAL FOR NON-TRAINER USE*****

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

Have students write their names on both sides of the name card.

Tell students that a course evaluation must be completed at the end of the course.

Tell students where they can send comments.

Delivery Tip

Demonstrate how to open the Web page provided on the Student Materials compact disc. On the Trainer Materials compact disc, double-click **Autorun.exe** or **Default.htm** in the StudentCD folder.

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 course-related resources, 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 double-click **Autorun.exe** or **Default.htm** in the root directory of the compact disc.

- *Course evaluation.* At the conclusion of this course, please complete the course evaluation to provide feedback on the instructor, course, and software product. Your comments will help us to improve future courses.

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

Prerequisites

Topic Objective

To present and describe the prerequisites for this course.

Lead-in

The following prerequisite knowledge is needed for this course.

- **Intermediate experience programming in Transact-SQL**
 - Course 2073A, *Programming a Microsoft SQL Server Database*
- **Intermediate experience using XML**
 - Course 1905A, *Building XML-Based Web Applications*
- **Scripting and ADO: VBScript that accesses a data source**
- **Ability to build a form in HTML**
- **ASP pages that use:**
 - **POST** and **GET** methods
 - **Response** and **Session** objects

*****ILLEGAL FOR NON-TRAINER USE*****

Student Prerequisites

To get the most from this course, you should come prepared with the following prerequisites:

- The ability to write basic Transact-SQL statements that query, insert, update, and delete information from a database. You should be able to use JOIN statements to retrieve information from multiple tables.
- Successful completion of Course 2073A, *Programming a Microsoft SQL Server Database*, or equivalent experience designing relational databases, including the use of primary and foreign keys. You need to know how to create databases, tables, and stored procedures. Students should also be familiar with the SQL Server Query Analyzer and SQL Profiler tools.
- Successful completion of Course 1905A, *Building XML-Based Web Applications*, or equivalent experience using the Extensible Markup Language (XML), the Extensible Stylesheet Language (XSL), and Extensible Stylesheet Language Transformations (XSLT). This includes knowledge of attributes and elements, schemas, and the XML Path Language (XPath).
- Experience using Microsoft ActiveX® Data Objects (ADO) to access a data source.
- The ability to write Microsoft Visual Basic®, Scripting Edition (VBScript) code.
- The ability to read standard Hypertext Markup Language (HTML) and build a form to gather and present data.
- The ability to create Active Server Pages (ASP) that use the **POST** and **GET** methods, the **Response** object, and the **Session** object.

Course Outline

Topic Objective

To provide an overview of each module and what students will learn.

Lead-in

In this course, we will cover...

- **Module 1: Using XML and Microsoft SQL Server 2000**
- **Module 2: Retrieving XML Data**
- **Module 3: Manipulating XML with Transact-SQL**
- **Module 4: Using ADO to Access XML Data**
- **Module 5: Using HTTP To Access XML Data**
- **Module 6: Using Mapping Schemas**
- **Module 7: XML for SQL Tools**

*****ILLEGAL FOR NON-TRAINER USE*****

Briefly describe each module.

As you describe each module, acknowledge any information that will meet the student expectations that you recorded earlier.

Module 1, “Using XML and Microsoft® SQL Server™ 2000,” In this module, you will learn the general outlines of how XML and its related technologies fit into the architecture of Microsoft SQL Server 2000.

Module 2, “Retrieving XML Data,” In this module, you will learn how to create and control XML-formatted data from SQL Server 2000. You will also learn how to retrieve a universal table, and how to create linked XML documents.

Module 3, “Manipulating XML with Transact-SQL,” In this module, you will learn how to use the **OPENXML** statement to create rowsets from XML documents. You will also learn how to use XPath expressions in rowpattern and colpattern parameters to specify rowset structure.

Module 4, “Using ADO to Access XML Data,” In this module, you will learn how to create XML query templates that use ADO to return an XML query result to the ASP **Response** object. You will also learn how to control the encoding of an XML query result.

Module 5, “Using HTTP To Access XML Data,” In this module, you will learn how to use Microsoft Internet Information Services (IIS) to publish a SQL Server 2000 database. You will learn how to establish and test a Hypertext Transport Protocol (HTTP) connection to SQL Server 2000. In addition, you will also learn how to run XML templates and retrieve database objects.

Module 6, “Using Mapping Schemas,” In this module, you will learn how to use mapping schemas to specify the structure of an XML document, and to map the elements and attributes the document contains to tables and columns in a SQL Server database.

Module 7, “XML for SQL Tools,” In this module, you will learn how to use Updategrams and the Bulk Load facility to modify data that is stored in a SQL Server 2000 database.

Setup

Topic Objective

To provide any necessary information about the classroom environment.

Lead-in

To replicate this development environment outside of the classroom, you need to assemble the following...

- Microsoft Windows 2000 Server
- Microsoft Windows 2000, Service Pack 1
- Microsoft SQL Server 2000, Standard Edition
- XML for SQL Server 2000 Web Release 1 Beta 2
- Microsoft Visual Basic 6.0, Service Pack 4
- Microsoft Internet Explorer 5.5
- Microsoft XML Parser 3.0
- Internet Explorer Tools for Validating XML and Viewing XSLT Output (optional)

*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

In this course, the Administrator and sa passwords are blank. This is strictly for the classroom environment and should be discouraged in real development settings.

Delivery Tip

If you performed a default installation of the course files, they will be installed in the directory C:\Program Files\MSdntrain\2091.

Course Files

There are files associated with the labs and practices in this course. The lab files are located in the folder <install folder>\Labs\Labxx, where xx represents the lab number. Each lab contains a folder for starter code and another for solution code. If you want to see the solution for a particular lab, open the appropriate file in that lab's solution folder. Many students find it useful to create a link on the desktop or the Start menu to the lab folder.

Software

The following software is installed on each computer:

- Microsoft Windows® 2000 Server
- Microsoft Windows 2000, Service Pack 1
- Microsoft SQL Server 2000, Standard Edition

SQL Server 2000 is set up by using Mixed Mode Authentication with a blank password. This configuration is ideal for classroom use. Never leave administrative passwords blank on production systems.
- XML for SQL Server 2000 Web Release 1 Beta 2

This installs Updategrams and the Bulk Load facility. It also enables you to create virtual directories in IIS that point to SQL Server 2000 databases.
- Microsoft Visual Studio® 6.0, Enterprise Edition:
 - Microsoft Visual Basic 6.0
- Microsoft Visual Studio 6.0, Service Pack 4
- Microsoft Internet Explorer 5.5
- Microsoft XML Parser 3.0 (MSXML parser)

MSXML 3.0 is set to operate in replace mode.
- Internet Explorer Tools for Validating XML and Viewing XSLT Output

Microsoft Certified Professional Program

Topic Objective

To provide students with information about the Microsoft Certified Professional Program.

Lead-in

The Microsoft Certified Professional Program includes these certifications...



*****ILLEGAL FOR NON-TRAINER USE*****

The Microsoft Certified Professional program includes the following certifications:

- Microsoft Certified Systems Engineer + Internet (MCSE + Internet)
- 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 + Internet (MCP + Internet)
- Microsoft Certified Professional (MCP)
- Microsoft Certified Trainer (MCT)

For More Information See the “Certification” section of the Web page provided on the compact disc or 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.

Exam Preparation Guides

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/>

Important MSDN® Training curriculum helps you to prepare for Microsoft Certified Professional (MCP) exams. However, no one-to-one correlation exists between MSDN Training courses and MCP exams. Passing MCP exams requires real-world experience with the products—MSDN Training courses help get you started.

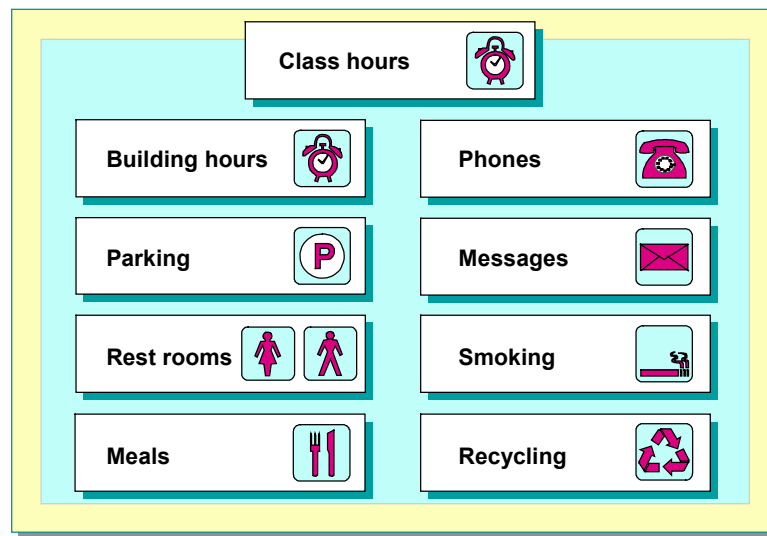
Facilities

Topic Objective

To inform students of class logistics and rules for the training site.

Lead-in

Before we start, let's go over the class logistics.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the class hours, extended building hours for labs, parking, rest room location(s), meals, phones, message posting, and where smoking is or isn't allowed.

Also, make sure that the students are aware of the recycling program if one is available.

Module 1: Using XML and Microsoft SQL Server 2000

Contents

Overview	1
Building Solutions with XML and SQL Server 2000	2
XML Technology Review	11
XML Support in SQL Server 2000	21
Lab 1: Working with XML	28
Multimedia: Microsoft BizTalk Server 2000 and XML (Optional)	33
Review	34



Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BackOffice, BizTalk, MSDN, MS-DOS, SQL Server, Visual Basic, Visual C++, Visual InterDev, Visual J++, Visual Studio, Windows, Windows Media, Windows NT, and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Instructor Notes: Introduction

Presentation:
60 Minutes

This module provides students with an overview of the Extensible Markup Language (XML) and its related technologies, as well as a high-level overview of the XML support in Microsoft® SQL Server™ 2000.

Lab:
30 Minutes

After completing this module, students will be able to:

- Use XML to represent business data.
- Format an XML document by referencing an Extensible Stylesheet Language (XSL) style sheet.
- Use XML Path Language (XPath) expressions in XSL programming.
- List the XML features in SQL Server 2000.
- Describe the SQL Server 2000 XML architecture.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the following materials:

- Microsoft PowerPoint® file 2091A_01.ppt
- The demo code provided for this chapter on the Instructor Materials compact disc

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the lab.
- Practice the demonstration.

Instructor Setup for a Lab

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

Lab 1: Working with XML

► To prepare for the lab

- Verify that the lab files have been installed on each student computer.

Demonstrations

This section provides demonstration procedures that will not fit in the margin notes or are not appropriate for the student notes.

A SQL Server XML-Based Solution

► To prepare for the demonstration

1. Install the sample application by executing SETUP.bat in the folder *<install folder>\SampApps\SQLXMLAPP*.
2. Examine the Active Server Pages (ASP) code in the folder *<system drive>\InetPub\WWWRoot\Retailer*.
3. Examine the templates and schemas in the folders SupplierIntranet, SupplierExtranet, and ShippingCo.
4. Use the SQL Server Query Analyzer to examine the **Retailer**, **Supplier**, and **ShippingCo** databases. In particular, examine the definitions for the tables and stored procedures.

► To demonstrate the Retailer site

1. On the desktop, click the **Retailer Web Site** shortcut to move to the **Retailer** Web site. Explain that this site consists of ASP pages that retrieve data from SQL Server as XML. Style sheets are then applied to the XML to produce the HTML pages. The advantage of this approach is that by applying different style sheets, the presentation logic can be changed. The content remains the same.
2. Click the **Login** link, and then click **Register** to create a user account (or log in as “Graeme” with a password of “password”).
3. View the various product categories, pointing out that there is only one product in the category **Cables etc**.
4. Add any two items to the basket and then view the basket.
5. Check out by providing a fictitious address and credit-card number, and note the order number. Explain that the order has been sent to the supplier as an XML document.

► To demonstrate the supplier intranet application

1. On the desktop, click the **Supplier Intranet Site** shortcut to move to the **Supplier** Web site. Explain that this is a SQL Server Internet Server Application Programming Interface (SQLISAPI) application, hosted by Internet Information Services (IIS), which allows XML data to be retrieved from the Supplier database. In this scenario, the application is an Order Picking application used by warehouse employees to prepare goods for delivery.
2. Right-click in the middle frame where the orders are listed, and choose **View Source**. Point out that this application sends XML to the browser, relying on client-side support for XML. Close Notepad.
3. Click the order that you placed earlier. The details of this order have been inserted into the Supplier database from the XML document sent by the Retailer application. These details were inserted by using SQL Server's **OpenXML** function.
4. Pick all of the items, and then click **Ready for Dispatch**.
5. Explain that clicking **Send Delivery Request** sends an XML Updategram document to the shipping company over the Hypertext Transfer Protocol (HTTP), and inserts a delivery request into the ShippingCo database. Click **Send Delivery Request**.
6. Click the link to the **Shipping Company** Web site and confirm that the delivery details have been sent. Explain that this is another SQLISAPI application, but that this time the style sheet is applied on the server, and therefore any browser can access it.

► To demonstrate the Catalog Import tool

1. On the desktop, click the **Catalog Import Tool** shortcut to view the contents of the CatalogImport folder. Explain that this is a VBScript application used by the retailer to perform the following tasks:
 - a. Download catalog data over HTTP from an annotated schema published in a SQLISAPI application at the supplier.
 - b. Write the catalog data to an XML file for logging purposes.
 - c. Import the XML catalog data into the retailer database by using the SQL XML Bulk Load facility.
2. Double-click the script to execute it. When the script completes, view the catalog.xml file that was produced.
3. Open the **Retailer** Web site and view the various product categories. Note that more products have been imported into the database.

Using an XML Schema

► To prepare for the demonstration

1. Ensure that the XML Validation Tools for Internet Explorer have been installed correctly by opening the file Order.xml in the folder *<install folder>\DemoCode\Mod01\Schema*. Right-click anywhere in the document to verify that the option to **Validate XML** is present
2. Practice the steps in the margin notes.

Using XPath Expressions

► To prepare for the demonstration

1. Practice the steps shown in the margin notes.
2. Examine the document Order.xml and the script OrderData.vbs.

Applying XSL Style Sheets

► To prepare for the demonstration

1. Ensure that the Microsoft XML Parser 3.0 (MSXML3) has been installed correctly, and that XMLInst.exe has been used to replace older versions of the DLL.
2. Practice the steps shown in the margin notes.

SQLOLEDB Services

► To prepare for the demonstration

1. Practice the steps shown in the margin notes.
2. Examine the code in GetProducts.vbs. In particular, note the use of an XPath query to specify all Product elements if no category is supplied, or Product elements containing a CategoryID child element with a specific value if a CategoryID is supplied. The relevant code is shown in the following example:

```
If iCategoryID = "" Then
    CmdProducts.CommandText = "Products"
Else
    CmdProducts.CommandText = "Products[CategoryID="
    & iCategoryID & "]"
End If
```
3. Examine the Transact-SQL statement recorded by the SQL Server Profiler. Note that it is a **SELECT...FOR XML** statement that uses the EXPLICIT modifier. Also note that the CategoryID supplied in the XPath query is used in the WHERE clause.
4. For large classrooms, you may want to increase the font size in Notepad (from the **Format** menu) and in Profiler (on the **Display** tab of the **Options** dialog box, available from the **Tools** menu).

Module Strategy

Use the following strategies to present this module:

■ Building Solutions with XML and SQL Server 2000

Begin by discussing the need to integrate applications and businesses, and the importance of separating data from presentation when dealing with multiple client types. Stress that these are the types of problems that XML can be used to address.

Emphasize that relational databases, such as SQL Server, are an efficient way to store data. Contrast this with XML, which provides an effective way to *exchange* data. The main problems for the developer are:

- Extracting data from a relational database in XML format.
- Inserting data from an XML document into a relational database.

SQL Server 2000 XML-related functionality is designed to help you perform these tasks.

Use the demonstration to discuss the kinds of solutions that you can build by using XML and SQL Server. The technologies used in this sample application will be covered in the remainder of the course.

■ XML Technology Review

This should be review for most students. If some students have not used XML extensively, and are unfamiliar with schemas, XPath, or XSL, use the demonstrations to cover the main points that should be understood when using the XML functionality in SQL Server.

■ XML Support in SQL Server 2000

The information in this module is designed to provide a very high-level overview of the XML-related features in SQL Server 2000. Do not go into too much detail about the use of Microsoft ActiveX® Data Objects (ADO) and HTTP to retrieve XML data, or about the XML-related Transact-SQL statement, templates, annotated schemas, Updategrams, or the Bulk Load facility, as they will be covered in later modules. Instead, emphasize why these features have been added to the product, and show how SQL Server XML support can be a very useful tool in many business integration scenarios.

When discussing the architecture of the XML functionality in SQL Server 2000, emphasize the role of the SQL Server OLE DB Provider (SQLOLEDB Provider). This is the core component in building SQL Server XML-based solutions.

Overview

Topic Objective

To provide an overview of the module topics and objectives.

Lead-in

In this module, you will learn about the various kinds of solutions you can build by using XML and Microsoft SQL Server 2000.

- Building Solutions with XML and SQL Server 2000
- XML Technology Review
- XML Support in SQL Server 2000

*****ILLEGAL FOR NON-TRAINER USE*****

In this module, you will learn how to use the Extensible Markup Language (XML) and Microsoft® SQL Server™ 2000 to create business applications that can be integrated with Web browsers, trading partners, and internal line-of-business (LOB) applications.

The first section describes a sample e-commerce solution, and shows how you can use XML and SQL Server to implement this solution.

The second section reviews XML-related functionality, including XML Schemas, Extensible Stylesheet Language (XSL), and the XML Path Language (XPath).

The final section examines how SQL Server 2000 provides support for XML data through language enhancements, architectural features, and interoperability with Microsoft Internet Information Services (IIS).

After completing this module, you will be able to:

- Use XML to represent business data.
- Format an XML document by referencing an XSL style sheet.
- Use XPath expressions in XSL programming.
- List the XML features in SQL Server 2000.
- Describe the SQL Server 2000 XML architecture.

◆ Building Solutions with XML and SQL Server 2000

Topic Objective

To introduce the topics covered in this section.

Lead-in

XML enables the integration of businesses and applications by providing a neutral, standard language to represent business data.

- Business Application Scenarios
- Comparing Data in XML and Relational Formats
- XML-Based Integration Solutions
- A Typical E-Commerce Scenario
- Demonstration: A SQL Server XML-Based Solution

*****ILLEGAL FOR NON-TRAINER USE*****

As organizations turn to the Internet as a platform for business processes, the task of integrating data-driven business processes with applications becomes a major priority for developers.

One of the main problems facing developers is how to represent business data so that it can easily bridge different computer systems, both within the organization and between partners. XML is rapidly becoming the standard for describing and exchanging data between trading partners and applications, replacing complex and expensive Electronic Data Interchange (EDI) techniques.

In this section, you will learn about the types of solutions you can build by using XML and SQL Server 2000, and how XML provides a common language for exchanging data.

Business Application Scenarios

Topic Objective

To discuss business scenarios where XML can be used effectively.

Lead-in

You can use XML to effectively achieve business solutions within a company and between trading partners.

■ E-Commerce Solutions

- Business-to-consumer (B2C)
- Business-to-business (B2B)

■ Internal Applications

- Intranet applications
- Enterprise Application Integration (EAI) solutions

*****ILLEGAL FOR NON-TRAINER USE*****

Using XML to represent business data makes it easier and less expensive to build the types of applications described in the following section.

E-Commerce Solutions

As the Internet grows in popularity and provides greater opportunities for trading partner integration, more companies are developing e-commerce applications for their business solutions. Typically, e-commerce applications are defined as being in one of the following categories:

■ Business-to-consumer (B2C) applications

These are the most common e-commerce applications. They usually consist of a Web site where customers can purchase goods or services, and the applications that support the Web site. Although most B2C sites use a Web browser as a client application, alternative devices such as Wireless Application Protocol (WAP)-enabled cellular telephones are increasingly used to access the Internet. The challenge for developers of B2C applications is to make the same data available for multiple device formats. Because XML can be easily transformed into different presentation formats, it is the ideal way to describe the data for a B2C solution.

■ Business-to-business (B2B) solutions

Traditional forms of EDI such as EDIFACT and ANSI X12 are generally expensive and difficult to develop. XML provides a relatively inexpensive platform-neutral solution that can be easily used over the Internet. This allows supply chains between trading partners to be automated over the Internet.

Internal Applications

XML can also be extremely useful when building the following types of internal corporate applications:

- Intranet applications

Many companies use an intranet for employee communications, integration with corporate systems, and knowledge-management applications. Because large corporations may require access to data from many different devices, their intranet sites can face many of the same challenges as B2C Web sites. Conversely, if the company has standardized on a single XML-aware browser (such as Microsoft Internet Explorer 5.5), the browser can be used to reduce the load on the intranet servers.

- Enterprise Application Integration (EAI) solutions

Many large companies have made significant investments in existing LOB applications. These applications may run on many different hardware platforms, network architectures, and operating systems throughout the enterprise. XML can be a useful way to integrate and extend existing applications with newer solutions, and to make possible the internal exchange of data between systems.

Comparing Data in XML and Relational Formats

Topic Objective

To explain the similarities and differences between using tables and XML to represent data.

Lead-in

You can represent business data as tables in a relational database, or as XML.

■ Relational format is designed for data storage

- Each instance of an entity is a row
- Each property of an entity is a column
- Relationships are based on common key fields

■ XML is designed for data exchange

- Each instance of an entity is an element
- Properties of an entity can be a value, an attribute, or a child element
- Relationships may be represented hierarchically

*****ILLEGAL FOR NON-TRAINER USE*****

Businesses need to work with data relating to many business entities. These entities can be physical objects such as customers, employees, or products, or abstract concepts such as sales or orders.

Storing Relational Data in Tables

Relational databases are an efficient way to store business data. Most business data is stored in a relational database system such as Microsoft SQL Server. A common approach to designing a relational database is to identify the business entities and create tables to store data relating to each entity. Each instance of an entity (for example, a specific customer order) is represented as a row in the appropriate table, and each property of the entity instance (such as the order date or customer) is represented as a column.

In a relational database, you could create the simple Orders table shown in the following example.

OrderID	CustomerID	OrderDate
10248	VINET	04/07/1996
10249	TOMSP	05/07/1996

When multiple business entities are related—such as when an order entity contains multiple item entities—common key fields are used to join the two tables. For example, the following table contains data from the Orders table and the Order Details table, joined on the OrderID key field for orders with an ID of 10248.

OrderID	ProductID	UnitPrice	Quantity
10248	11	14.00	12
10248	42	9.80	10

This is an efficient way to store the data by eliminating as much duplication as possible. You can retrieve the relevant Order details data for a specific order by using a Transact-SQL **JOIN** operator, as shown in the following example query:

```
SELECT O.CustomerID, O.OrderDate, OD.ProductID, OD.Quantity
FROM Orders O JOIN [Order Details] OD ON O.OrderID=OD.OrderID
WHERE O.OrderID=10248
```

Representing Data in XML

XML provides a neutral way to exchange business data in a hierarchical structure where entities are represented as elements in an XML document. Properties of entities can be represented as element values, attributes, child elements, or a combination of all three. The following example shows a possible XML representation of the Order table shown in the preceding section.

Example

```
<?xml version='1.0' ?>
<Orders>
  <Order OrderDate = '04/07/1996'>
    10248
    <CustomerID>VINET</CustomerID>
  </Order>
  <Order OrderDate = '05/07/1996'>
    10249
    <CustomerID>TOMSP</CustomerID>
  </Order>
</Orders>
```

In the preceding example:

- The OrderID column in the table maps to the value of the Order element.
- The OrderDate column maps to the **OrderDate** attribute of the Order element.
- The CustomerID column maps to the CustomerID child element of the Order element.

Note When an XML document uses attributes to represent all properties of a business entity, it is described as “attribute-centric.” When an XML document uses child elements to represent all of the properties of a business entity, it is described as “element-centric.” Many XML documents use a combination of the two representations.

Consider the following guidelines when mapping relational tables to XML:

- A well-formed XML document must contain a root element, and all attributes must be contained within either single quotes (‘xx’) or double quotes (“xx”).
- The element representing the entity instance (or table row) can have multiple attributes or child elements, but only a single value of its own.
- Attribute-centric mappings result in smaller physical documents.

Representing Relationships in XML

You can use a relational approach to representing related elements in an XML document. You do this by mapping the primary and foreign keys used in the relationship to element values, attributes, or child elements. However, a more common approach to representing relationships in XML is to use a hierarchy of elements in which the element representing the primary key table contains elements representing the foreign key table.

The following example shows how to represent the relationship between an order and the associated order details in XML.

Example

```
<?xml version='1.0' ?>
<Order OrderID='10248' OrderDate='04/07/1996'>
  <CustomerID>VINET</CustomerID>
  <OrderDetail ProductID='11'>
    <UnitPrice>14.00</UnitPrice>
    <Quantity>12</Quantity>
  </OrderDetail>
  <OrderDetail ProductID='42'>
    <UnitPrice>9.80</UnitPrice>
    <Quantity>10</Quantity>
  </OrderDetail>
</Order>
```

In this example, the data from the Orders table is mapped to attributes (for example, **OrderID**) and child elements (for example, <CustomerID>) in the Order element. The data from the Order Details table is mapped to attributes and child elements of the OrderDetail elements, which are themselves child elements of the Order element.

Caution XML element and attribute names are case-sensitive. When working with XML, you must pay particular attention to the case that is used.

XML-Based Integration Solutions

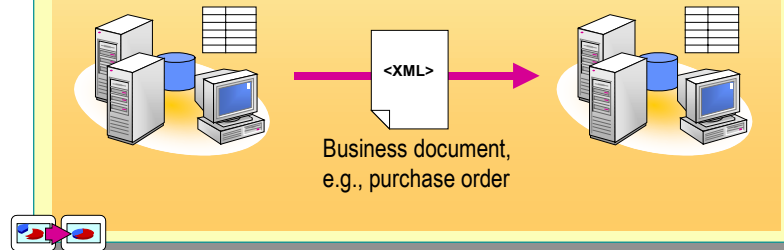
Topic Objective

To describe how XML can be used to build business integration solutions.

Lead-in

XML is a markup language that enables the integration of business and applications through a neutral standard for data representation.

- **Use XML to exchange business documents**
- **Two fundamental programming challenges:**
 - Extract relational data into XML
 - Import XML into a relational data store



*****ILLEGAL FOR NON-TRAINER USE*****

You use XML to represent data for communication between applications. XML provides a format for describing semistructured data, and has become a popular standard for business data integration on the Internet.

Delivery Tip

This slide builds automatically to show relational data from the retailer being sent as XML to the supplier, and then being imported into the relational database at the supplier.

You can restart the build by pressing the left arrow key on the keyboard.

Using XML to Exchange Business Documents

One of the most common uses of XML is to exchange business documents between applications, departments within a company, and trading partners. Most business transactions involve the exchange of physical documents such as purchase orders or invoices. XML makes it easy to represent these transactions electronically by providing a platform-neutral way to describe the data in your business documents.

Programming Challenges in Building XML-Based Integration Solutions

Because most business data is stored in relational form, developers face two major challenges when building XML-based integration solutions:

- **Creating an XML document from data in a relational source**
When building an application that sends a business document, you must provide a mechanism for retrieving the necessary data from the relational database and generating an XML document.
- **Inserting data from an XML document into a relational database**
When building an application that receives and processes a business document, you must provide a mechanism for inserting the data from the XML document into tables in a relational database.

Key Points

The fundamental challenges in using XML for integration are extracting relational data as XML, and inserting XML data into relational tables.

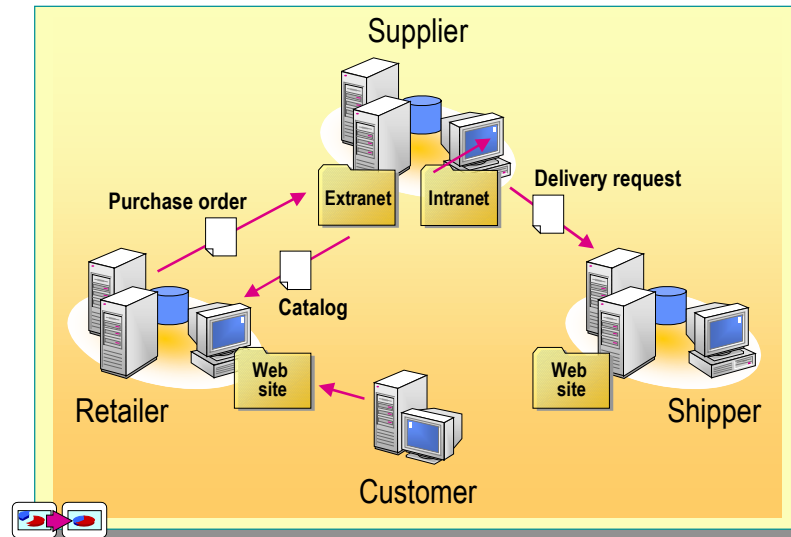
A Typical E-Commerce Scenario

Topic Objective

To describe a typical e-commerce scenario where XML could be used.

Lead-in

A typical e-commerce supply-chain solution involves a combination of B2C and B2B applications.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

This slide builds automatically to show the stages of the e-commerce solution. You can restart the build by pressing the left arrow key on the keyboard.

This scenario is the basis for the demonstration on the next slide.

A typical e-commerce scenario may involve B2C applications that allow customers to place orders, and B2B applications that automate the supply chain by exchanging documents between trading partners. The sample scenario on the slide shows the following e-commerce applications:

1. A customer browses a retailer's Web site, which shows products from a SQL Server database. The product data is extracted as XML, allowing it to be rendered differently for different client devices.
2. The retailer sells goods that are ordered from a supplier. The supplier provides an extranet site to which XML purchase orders can be sent, and from which updated catalogs can be downloaded in XML format.
3. Warehouse employees at the supplier organization use a SQL Server-based intranet application to gather the items in each order for delivery. This application makes use of the native XML support in Internet Explorer, and sends an XML delivery request to a shipping company.
4. The shipping company provides a SQL Server-based Web site for viewing the delivery status of orders.

Delivery Tip

An extranet differs from an intranet because it crosses enterprise boundaries, unlike an intranet, which typically refers to a Web within a single enterprise. Both terms refer to a set of content shared by a clearly defined group.

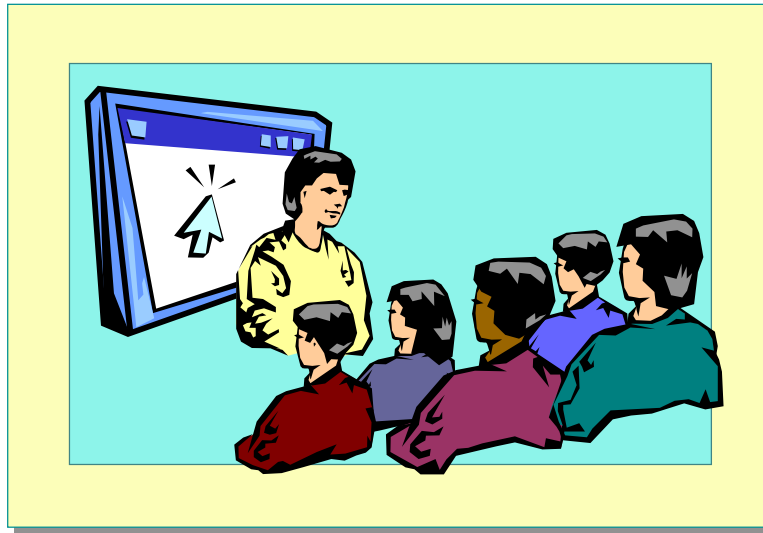
Demonstration: A SQL Server XML-Based Solution

Topic Objective

To demonstrate a sample e-commerce solution.

Lead-in

In this demonstration, you will see a sample e-commerce solution that was created by using XML and SQL Server 2000.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

Detailed instructions for this demonstration can be found in the Instructor Notes section at the beginning of this module.

Use this demonstration to discuss the kinds of solutions that can be built by using SQL Server and XML. In general, avoid going into too much detail about the implementation because it will be covered in the rest of the course. However, if you have a particularly advanced group, you may want to spend some time showing students the source code.

Emphasize to the students that they can install this demo from the Student Materials compact disc.

In this demonstration, you will see a sample e-commerce solution based on the scenario described in the preceding section. This solution was built by using XML and SQL Server 2000.

The source code for this demonstration is included on the Student Materials compact disc.

◆ XML Technology Review

Topic Objective

To introduce the topics in this section.

Lead-in

In this section, we will review some of the important XML-related technologies that are required to build solutions similar to the one in the previous section.

- Defining XML Documents Using XML Schemas
- Demonstration: Using an XML Schema
- Navigating an XML Document Using XPath
- Demonstration: Using Xpath Expressions
- Transforming XML Using XSL
- Demonstration: Applying XSL Style Sheets

*****ILLEGAL FOR NON-TRAINER USE*****

In this section, you will review some of the important XML-related technologies that you can use to build XML-based solutions, such as the e-commerce scenario in the previous section.

Schemas are an important aspect of XML-based development, especially when you are building solutions that integrate multiple organizations. They are used to define and validate XML documents by specifying what elements and attributes can legally be used in a specific type of business document.

XPath is an XML document navigation language. You use XPath to extract specific data from an XML document.

You use XSL to convert XML data into a presentation format such as Hypertext Markup Language (HTML), or an alternative XML representation of the same data. XSL is the basis for many Web-based applications that use XML.

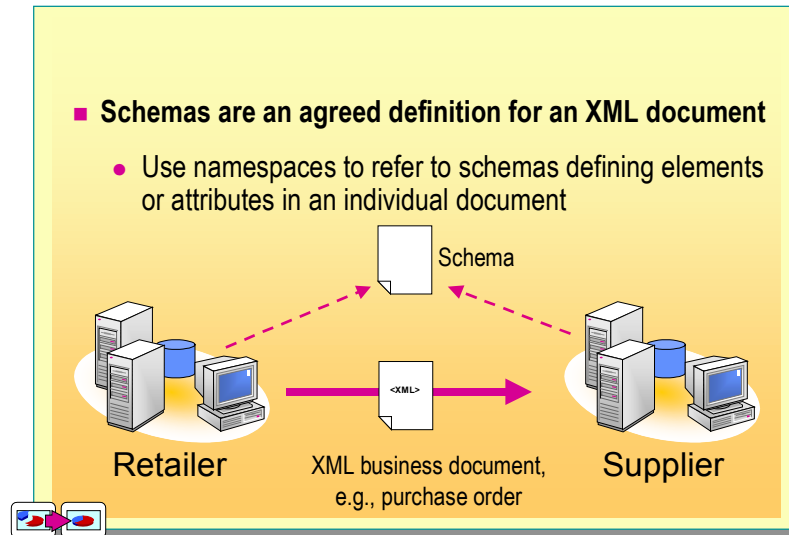
Defining XML Documents Using XML Schemas

Topic Objective

To introduce XML Schemas.

Lead-in

You can use XML Schemas to define business documents.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

This slide builds automatically to show a retailer and a supplier exchanging an XML document based on an agreed schema. To restart the build, press the left arrow key on the keyboard.

In order for businesses to exchange data in a useful way, they must define the format of the business documents to be exchanged. XML Schemas are XML documents that contain a definition for a specific class of business document. If two trading partners create their documents according to the same XML Schema, they can ensure that their respective applications can use the data that they exchange.

Note XML Schemas are a more powerful way to define documents than the Data Type Definition (DTD) standard. The World Wide Web Consortium (W3C) is currently working toward approving a standard syntax for XML Schemas. Microsoft technologies use a syntax called XML-Data Reduced (XDR) to define XML Schemas, and will support the W3C standard when it is approved.

Specifying Aspects of a Business Document

XML Schemas enable business integration architects to specify the following aspects of a particular business document:

- The elements and attributes that must be present in the document
- The data types of those elements and attributes
- How many instances of an element can exist
- The order of elements and attributes in the document
- The minimum and maximum values of numerical elements or attributes
- The minimum and maximum lengths of text elements and attributes
- The content model of the document

An open content model allows additional (undefined) elements or attributes to be present in instances of the document. A closed model ensures that only the elements and attributes defined in the schema can be present in an instance of the document.

Referencing a Namespace

An instance of an XML document that is based on a schema references the XML Schema by using a namespace. You can place the reference in the root element. You can also place the reference in a child element deeper in the hierarchy, thus allowing business documents to be contained within an XML “envelope” for document-routing purposes. Placing the reference in a child element also facilitates nesting of business documents.

The following example shows an XML schema for a purchase order, and an XML document based on that schema.

Example Schema

```
<?xml version='1.0' ?>
<Schema xmlns='urn:schemas-microsoft-com:xml-data'
        xmlns:dt='urn:schemas-microsoft-com:datatypes'>

  <ElementType name='UnitPrice' dt:type='fixed.14.4' />
  <ElementType name='Quantity' dt:type='int' />
  <ElementType name='CustomerID' dt:type='string' />

  <ElementType name='OrderDetail' content='eltOnly'
                model='closed'>
    <AttributeType name='ProductID' dt:type='int' />

    <attribute type='ProductID' required='yes' />
    <element type='ProductID' minOccurs='1' maxOccurs='1' />
    <element type='UnitPrice' minOccurs='1' maxOccurs='1' />
    <element type='Quantity' minOccurs='1' maxOccurs='1' />
  </ElementType>

  <ElementType name='Order' content='eltOnly' model='closed'>
    <AttributeType name='OrderID' dt:type='id' />
    <AttributeType name='OrderDate' dt:type='dateTime' />

    <attribute type='OrderID' required='yes' />
    <attribute type='OrderDate' />
    <element type='CustomerID' minOccurs='1' maxOccurs='1' />
    <element type='OrderDetail' minOccurs='1' maxOccurs='*' />
  </ElementType>
</Schema>
```

Example XML Document

```
<?xml version='1.0' ?>
<Order xmlns="x-schema:OrderSchema.xml"
        OrderID='10248' OrderDate='04/07/1996'>
  <CustomerID>VINET</CustomerID>
  <OrderDetail ProductID='11'>
    <UnitPrice>14.00</UnitPrice>
    <Quantity>12</Quantity>
  </OrderDetail>
  <OrderDetail ProductID='42'>
    <UnitPrice>9.80</UnitPrice>
    <Quantity>10</Quantity>
  </OrderDetail>
</Order>
```

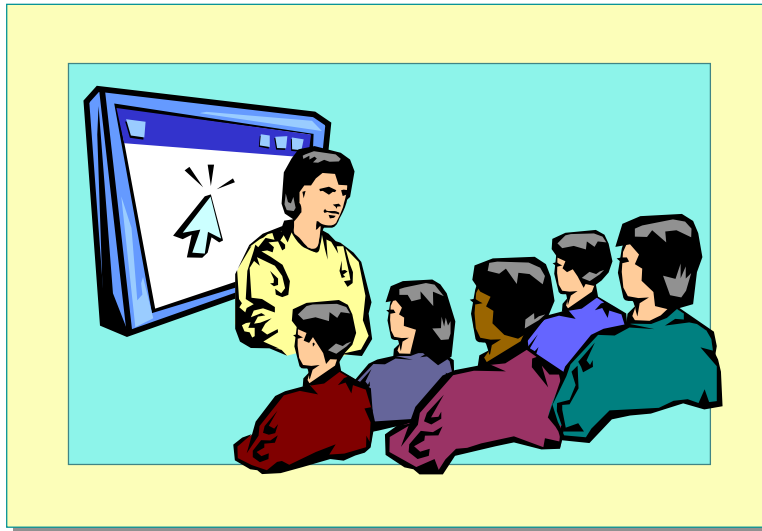
Demonstration: Using an XML Schema

Topic Objective

To demonstrate how an XML Schema can be used to define an XML document.

Lead-in

In this demonstration, you will see how an XML Schema can be used to define an XML document.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

1. Move to the folder `<install folder>\DemoCode\Mod01\` Schema and double-click `OrderSchema.xml` to open it in Internet Explorer.
2. Point out the element and attribute definitions. This schema defines an `<Order>` element that contains one `CustomerID` element and one or more `OrderDetail` elements.
3. Double-click `Order.xml` to open it in Internet Explorer. Then right-click in the document and choose `Validate XML`. This proves that the document adheres to the schema.
4. Double-click `InvalidOrder.xml` to open it in Internet Explorer. Then right-click in the document and choose `Validate XML`. This proves that the document does not adhere to the schema. (The `Quantity` element is missing in the second `OrderDetail` element.)

In this demonstration, you will first see an XML Schema that defines a purchase order document. You will then see both a valid and invalid instance of the purchase order.

Document validity is checked by using the XML Validation Tools add-in for Internet Explorer, which is available free from the MSDN® Web site.

It is important to understand that the schema provides the rules that can be used by either party in a business document exchange, to determine whether or not a specific instance of the document is valid.

Navigating an XML Document Using XPath

Topic Objective

To introduce XPath query language syntax.

Lead-in

Extracting data programmatically from an XML document requires a query syntax that can be used against hierarchical data

- XPath is a standard navigation language
- Nodes from an XML hierarchy can be returned based on XPath queries, for example:

- All OrderDetail elements `Order/OrderDetail`

- OrderDetail elements with a ProductID attribute of 23

`Order/OrderDetail[@ProductID='23']`

- OrderDetail elements with a Quantity element greater than 1

`Order/OrderDetail[Quantity > 1]`

*****ILLEGAL FOR NON-TRAINER USE*****

In order to process an XML document programmatically, there must be a mechanism for extracting data values from the document. The hierarchical nature of XML documents requires a query language that can navigate through the hierarchy.

For Your Information

Not all features of XPath are supported by SQL Server. SQL Server Books Online contains a list of both supported and unsupported features.

Using XPath Queries

The XPath query language is the standard navigation language used to query XML documents. When using XPath, you can think of the elements in a document as a tree of nodes, and specify a path through the nodes. XPath queries return node-sets. You can specify attributes by prefixing the attribute name with an “@” symbol. You can use conditional expressions to retrieve data by enclosing the expression in square brackets.

XPath syntax includes the following features.

Feature	Description	Example
Axes	Used to specify the path to a particular node, set of nodes, or attribute.	catalog/product Returns all <product> elements immediately beneath a <catalog> element.
Relational operators	Used to test values by using standard relational operators.	catalog/product[@price > 10] Returns all <product> elements with a price attribute greater than 10.
Arithmetic operators	Used to evaluate arithmetic expressions.	catalog/product/[@price * 0.9 > 1] Returns all <product> elements where the price attribute multiplied by 0.9 results in a value greater than 1.
Conversion functions	Used to convert data to a different data type.	catalog/product[string(@PID)='4'] Returns all <product> elements where the PID attribute can be resolved to the string "4".

By using these and other features of XPath, you can create programmatic solutions that use complex logic to extract and manipulate data in an XML document.

Note The preceding table shows only a subset of the functionality available through XPath. For example, XPath includes functions to calculate the sum of values in a node-set, or to test for the existence of a substring within a node value. For more information about XPath, view the W3C specification at <http://www.w3.org/TR/xpath>

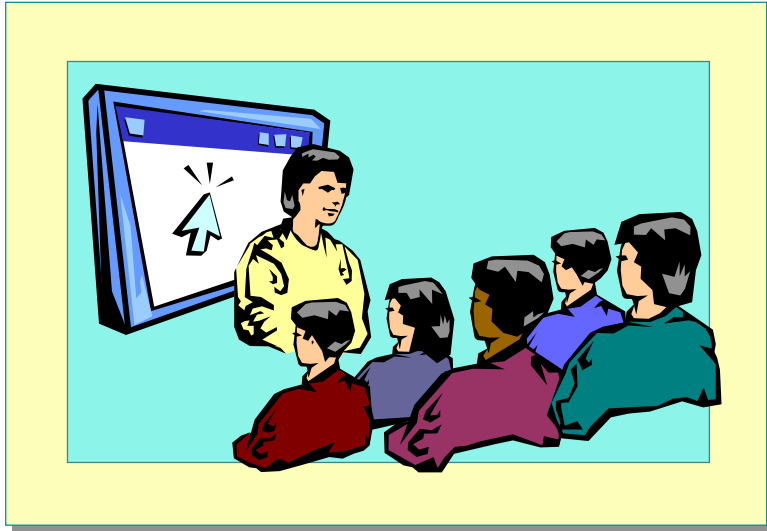
Demonstration: Using XPath Expressions

Topic Objective

To demonstrate how XPath can be used to navigate through an XML document and select data.

Lead-in

In this demonstration, you will see how XPath expressions can be used to retrieve data from an XML document.



*****ILLEGAL FOR NON-TRAINER USE*****

In this demonstration, you will see how XPath expressions can be used to retrieve data from an XML document.

The following XML document was used in this demonstration:

Delivery Tip

1. View the contents of `<install Folder>\DemoCode\Mod01\XPath`.
2. Double-Click `OrderData.vbs` and point out that the default XPath expression(`Order`) is the root element of the document. Click OK to view the results (the whole document).
3. Run the script again, this time entering the following XPath:
`Order/OrderDetail`
This returns the data in the `<OrderDetail>` elements.
4. Enter
`Order/OrderDetail [Quantity>10]`
This returns the `<OrderDetail>` elements with a `Quantity` child element with a value greater than 10.
5. Enter
`Order/OrderDetail [@ProductID=11]`
This returns `<OrderDetail>` elements with a `ProductID` attribute of 11.

```
<?xml version='1.0' ?>
<Order OrderID='10248' OrderDate='04/07/1996'>
  <CustomerID>VINET</CustomerID>
  <OrderDetail ProductID='11'>
    <UnitPrice>14.00</UnitPrice>
    <Quantity>12</Quantity>
  </OrderDetail>
  <OrderDetail ProductID='42'>
    <UnitPrice>9.80</UnitPrice>
    <Quantity>10</Quantity>
  </OrderDetail>
</Order>
```

Transforming Documents Using XSL

Topic Objective

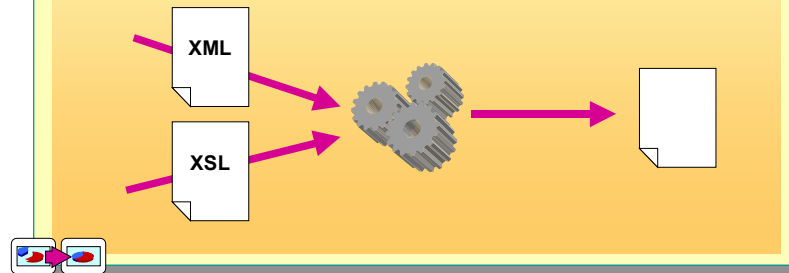
To introduce XSL style sheets.

Lead-in

XML documents commonly need to be transformed into different formats.

■ Apply XSL style sheets to:

- Transform XML into presentation format (e.g., HTML)
- Transform XML into an alternative XML grammar



*****ILLEGAL FOR NON-TRAINER USE*****

XML is a suitable format for documents that you will process programmatically. However, you will often need to transform XML into a format that is more appropriate for display.

Delivery Tip

This slide builds automatically to show an XML document and a style sheet being combined to create a new document. To restart the build, press the left arrow key on the keyboard.

Extensible Stylesheet Language

You use XSL to process and apply formatting to XML data. The most common use of XSL is to display XML data in HTML format. XSL style sheets are XML documents that can contain HTML elements, XPath expressions to extract specific nodes in an XML document, and XSL functions to process the nodes in the XML document. An XSL style sheet often contains one or more templates that can be applied to different sets of data within the XML document.

The following example shows a style sheet that generates HTML.

Example Style Sheet

Delivery Tip

Discuss this sample style sheet, paying particular attention to the `xsl:for-each` loop and the `xsl:value-of` function.

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:template match='/ '>
  <HTML>
  <Body>
  <Table border='0'>
    <xsl:for-each select='Order/OrderDetail'>
      <TR>
        <TD><xsl:value-of select = '@ProductID' /></TD>
        <TD><xsl:value-of select='Quantity' /></TD>
      </TR>
    </xsl:for-each>
  </Table>
  </Body>
  </HTML>
</xsl:template>
</xsl:stylesheet>
```

The style sheet described in the preceding example could be applied to the following XML document.

Example XML Document

```
<?xml version='1.0'?>
<?xml-stylesheet type='text/xsl' href='RenderPO.xsl'?>
<Order OrderID='10248' OrderDate='04/07/1996'>
  <CustomerID>VINET</CustomerID>
  <OrderDetail ProductID='11'>
    <UnitPrice>14.00</UnitPrice>
    <Quantity>12</Quantity>
  </OrderDetail>
  <OrderDetail ProductID='42'>
    <UnitPrice>9.80</UnitPrice>
    <Quantity>10</Quantity>
  </OrderDetail>
</Order>
```

The following example shows the HTML result of applying the style sheet to the order document.

Example HTML Output

```
<HTML>
  <Body>
    <Table border='0'>
      <TR>
        <TD>11</TD>
        <TD>42</TD>
      </TR>
      <TR>
        <TD>12</TD>
        <TD>10</TD>
      </TR>
    </Table>
  </Body>
</HTML>
```

To apply the style sheet to an XML document, you can include an `<xml-stylesheet>` element that references the XSL in the XML document itself, as shown in the following example.

Example

```
<? xml-stylesheet type='text/xsl' href='RenderPO.xsl' ?>
```

Alternatively, you can apply the style sheet programmatically by using the Microsoft XML (MSXML) component, as shown in the following example.

Example

```
Set xmlDoc = CreateObject("Microsoft.XMLDom")
xmlDoc.load "c:\Order.xml"
Set xslDoc = CreateObject("Microsoft.XMLDom")
xslDoc.Load "c:\RenderPO.xsl"
strHTML = xmlDoc.transformNode(xslDoc)
```

Note XSL style sheets can be used to transform XML documents into many formats, including the Website META Language (WML) and alternative XML representations, as well as HTML.

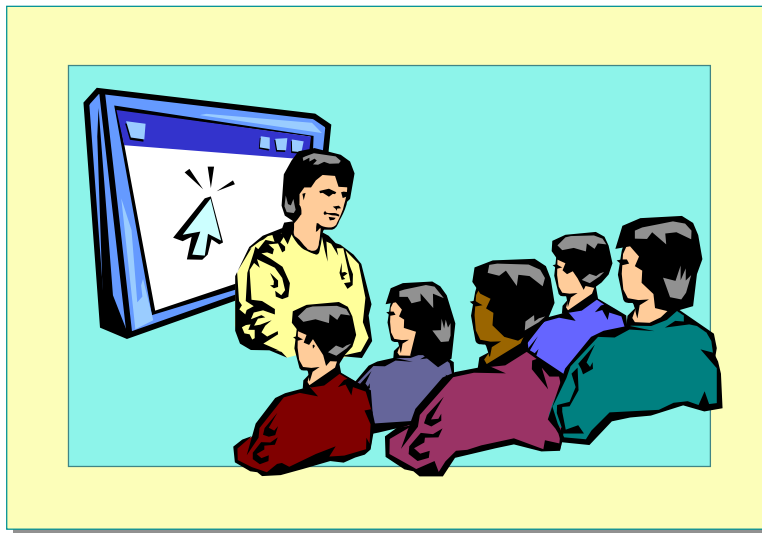
Demonstration: Applying XSL Style Sheets

Topic Objective

To demonstrate XSL style sheets.

Lead-in

In this demonstration, you will see how an XSL style sheet can be used to transform an XML document into HTML format.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

1. View the contents of `<install folder>\DemoCode\Mod01\XSL` and double-click `Order.xml` to open it in Internet Explorer.
2. Allow the students to view the XML document, then close it and double-click `Order.xsl` to view the style sheet.
3. Talk the students through the code in the style sheet, describing what each instruction will produce.
4. Open `Order.xml` with Microsoft Notepad and remove the `<!--` and `-->` comments around the style sheet Processing Instruction. Save the file and close Notepad.
5. Double-click `Order.xml` to open it in Internet Explorer. The style sheet should be applied to the XML data to produce an HTML document.

In this demonstration, you will see how an XML document can be transformed into HTML format by using an XSL style sheet.

◆ XML Support in SQL Server 2000

Topic Objective

To introduce the topics covered in this section.

Lead-in

SQL Server 2000 includes many features that make it easier for you to work with XML.

- Accessing SQL Server Data as XML
- Transact-SQL Enhancements
- Modifying SQL Data Using XML
- SQL Server 2000 XML Architecture
- Demonstration: SQLOLEDB Services

*****ILLEGAL FOR NON-TRAINER USE*****

SQL Server 2000 provides a platform for extremely scalable, high-performance database solutions. Developers can use traditional relational database techniques to store, retrieve, analyze, and manipulate all kinds of business data.

SQL Server 2000 also includes many features that can help developers build XML-based business-integration solutions.

This section describes these features and provides a high-level overview of the SQL Server XML architecture.

Accessing SQL Server Data as XML

Topic Objective

To describe two access techniques for retrieving XML from SQL Server 2000.

Lead-in

There are two main approaches to retrieving XML data from SQL Server 2000.

■ ActiveX Data Objects 2.6

- Familiar programming model
- Enhanced support for XML data

■ HTTP Database Publishing

- Access through ISAPI application
- Enables database integration over the Internet

■ XML is retrieved by using FOR...XML queries, templates, or mapping schemas

*****ILLEGAL FOR NON-TRAINER USE*****

There are two main approaches to retrieving XML data from SQL Server 2000:

- COM-based solutions can use the Microsoft ActiveX® Data Objects (ADO) library to connect to SQL Server and retrieve XML data.
- SQL Server supports database publishing over HTTP, allowing access to data through an IIS Internet Server Application Programming Interface (ISAPI) application.

ActiveX Data Objects 2.6

Most developers on the Microsoft Windows® operating system are familiar with the ADO object library. ADO is the standard object model used to access data in relational and non-relational data stores.

ADO 2.6 provides the following features to enable XML data access:

- A **Stream** object, which you use to submit XML queries and receive XML results.
- Access to extended properties of the **Command** object when using the SQL Server OLE DB (SQLOLEDB) Provider.

Note **Stream** objects are an efficient way to handle XML data because they allow the application to start processing the data as it is being read. There is no need to wait until the data is fully loaded into memory.

You can use ADO to submit queries in the form of XML streams, and to retrieve XML or HTML streams. You can also use ADO to access XML data by using an annotated schema and XPath syntax.

HTTP Database Publishing

SQL Server 2000 includes support for publishing a database through an IIS virtual directory. This support enables you to create Internet or intranet applications that can access data directly through an HTTP connection. This makes it easier to configure data access through a firewall and integrate business solutions across applications, networks, and organizations.

SQL Server databases are published in IIS virtual directories by using the SQLISAPI application that ships with SQL Server 2000. This allows an OLE DB connection to a SQL server database to be associated with an IIS application. Each HTTP request to the IIS application is handled by the SQLISAPI application, which accesses the database over the OLE DB connection.

Retrieving XML Data

When using either of these data access approaches, you can retrieve XML data from SQL Server by using the following techniques:

- *FOR...XML queries*. These are Transact-SQL statements that return data as XML fragments.
- *Templates*. These are XML documents containing one or more queries. They result in well-formed XML results.
- *Mapping schemas*. These are XML Schema documents that map data in the database to elements or attributes in the resulting XML document.

Transact-SQL Enhancements

Topic Objective

To introduce the two XML-related enhancements to the Transact-SQL language.

Lead-in

There are two important XML-related statements in the Transact-SQL language in SQL Server 2000.

■ SELECT ... FOR XML

- Specifies that the result set should be returned as an XML stream
- Used to retrieve relational data as XML

■ OpenXML

- Function to return a rowset from an XML stream
- Used to insert XML data into relational tables

*****ILLEGAL FOR NON-TRAINER USE*****

There are two important XML-related statements in the Transact-SQL language in SQL Server 2000. The first is an additional clause, **SELECT ... FOR XML**, which can be specified in a **SELECT** statement in order to retrieve XML data. The other is a function, **OpenXML**, which generates an internal rowset from an XML stream.

SELECT ... FOR XML

You can add the **FOR XML** clause to a **SELECT** statement in order to return the results as an XML stream rather than a rowset. This clause can be used with one of three modifiers (**RAW**, **AUTO**, and **EXPLICIT**) to control the format of the returned XML. The **FOR XML** clause is discussed in Module 2, “Retrieving XML Data.”

SELECT ... FOR XML statements are useful when you need to generate an XML document from relational data stored in SQL Server tables.

OpenXML

You can use the **OpenXML** function to generate an in-memory rowset from an XML stream. This function is used together with the **sp_xml_preparedocument** and **sp_xml_releasedocument** system stored procedures. The **OpenXML** function is discussed in Module 3, “Manipulating XML with Transact-SQL.”

OpenXML is most commonly used in stored procedures, where it can provide a useful way to insert data from an XML document into relational tables.

Modifying SQL Data Using XML

Topic Objective

To introduce Updategrams and the Bulk Load facility.

Lead-in

Other XML-related features of SQL Server 2000 include Updategrams and the XML Bulk Load facility.

■ XML Updategrams

- XML documents used to perform inserts, updates, and deletes

■ XML Bulk Loader Component

- Scriptable COM component for bulk-loading XML data

*****ILLEGAL FOR NON-TRAINER USE*****

XML-related functionality also includes the ability to perform data modification by submitting XML documents called *Updategrams*, and to import large quantities of XML data by using the XML Bulk Load facility. This functionality is discussed in Module 7, “XML for SQL Tools.”

Note Updategrams and the XML Bulk Load facility are not shipped with SQL Server 2000. They can be downloaded from the MSDN site at <http://msdn.microsoft.com/downloads> as part of an additional tool set named XML for SQL Server 2000 Web Release.

XML Updategrams

XML Updategrams are XML documents that contain “before” and “after” images of data in tables. You submit the Updategram to SQL Server through the SQLOLEDB Provider, and the relevant data is modified in the database.

Updategrams are useful for providing batch update services from XML-aware client applications. For example, data could be downloaded to an XML island in a browser, and updates to that data could be sent back to the server as an Updategram.

XML Bulk Load Facility

The Bulk Load facility can be used from any COM-aware programming language to read data from an XML data source and import it into a SQL Server database.

The ability to bulk-load XML data is useful in situations where large volumes of data might be received as XML. For example, an online retailer might receive catalog data from a supplier as XML. The Bulk Load facility could be used to load the catalog into a relational database.

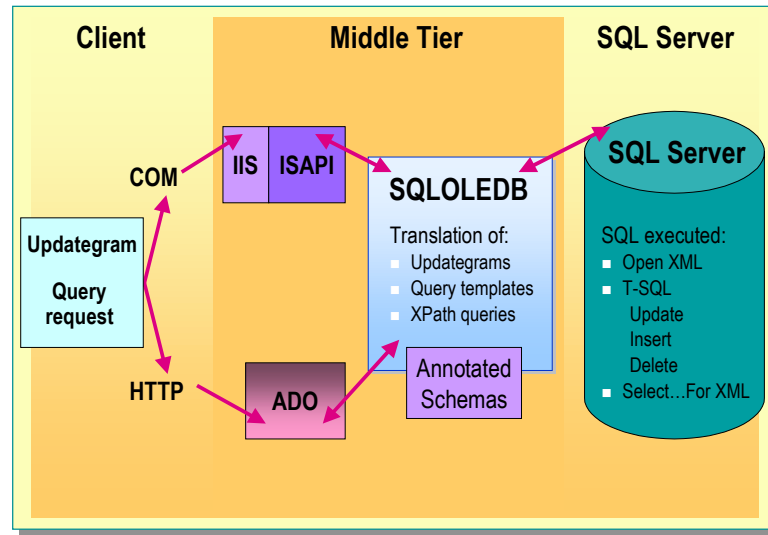
SQL Server 2000 XML Architecture

Topic Objective

To show the architecture of SQL Server's XML functionality.

Lead-in

The heart of the SQL Server 2000 XML architecture is in the SQLOLEDB Provider.



*****ILLEGAL FOR NON-TRAINER USE*****

The architecture for XML support in SQL Server promotes the development of three-tier solutions. Middle-tier data access services are provided by either IIS, through the SQLISAPI.dll application, or by business components using ADO to connect to the database.

Key Points

The SQLOLEDB Provider performs all of the necessary translation between requests for XML data and the Transact-SQL statement used to retrieve or modify the data.

In either case, data access is always accomplished by using the SQLOLEDB Provider. This provider supplies all XML data-retrieval functionality by parsing XML templates, annotated schemas, and queries, generating appropriate **SQL SELECT** statements for the query processor, and applying any specified XSL style sheets to result streams.

The SQLOLEDB Provider communicates with the SQL Server query processor, which uses enhanced Transact-SQL support to retrieve or manipulate XML data. Transact-SQL enhancements to SQL Server 2000 are discussed in the following section.

SQLOLEDB Services

The following list describes some of the SQLOLEDB services:

- *Template Translation.* When an XML template is used to query SQL Server over HTTP, the SQLOLEDB Provider parses the template and generates the appropriate **SELECT ... FOR XML** statement to submit to the query processor.
- *XPath Translation.* When an XPath query is performed against an annotated schema, the SQLOLEDB provider parses the annotated schema and the XPath query to generate the appropriate **SELECT ... FOR XML** statement to submit to the query processor.
- *Updategram Translation.* When an Updategram is submitted to SQL Server, the SQLOLEDB Provider parses the Updategram and generates the appropriate **INSERT**, **UPDATE**, or **DELETE** statement to submit to the query processor.

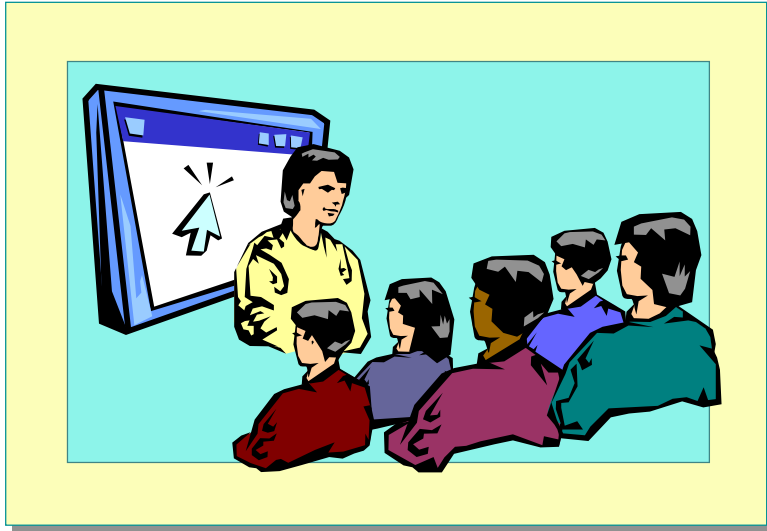
Demonstration: SQLOLEDB Services

Topic Objective

To demonstrate how the SQLOLEDB Provider generates SELECT...FOR XML statements from annotated schemas.

Lead-in

This demonstration shows how the SQLOLEDB Provider translates data requests to the appropriate SQL statements.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

1. Start Windows Explorer and move to <install folder>\DemoCode\Mod01\SQLOLEDB.
2. Open ProductSchema.xml to show the schema. (Explain that annotated schemas will be discussed later.)
3. Edit GetProducts.vbs to show the XPath that will be used when a category ID is specified. (Explain that ADO will be discussed later.)
4. Start Profiler from the SQL Server program group, and create a new trace by using the default settings.
5. Run GetProducts.vbs from the SQLOLEDB folder, entering 2 when prompted for a category ID.
6. Stop the trace and select the SQL:BatchCompleted statement with the TextData 'SELECT 1 as TAG..'. (Explain that FOR XML queries will be discussed later.)
7. Explain that this is the SQL generated by SQLOLEDB based on the schema and the XPath.

In this demonstration, you will see how the SQLOLEDB Provider translates a request for data by using an XPath query and an annotated schema, to the appropriate Transact-SQL **SELECT...FOR XML** statement.

Lab 1: Working with XML

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will create and manipulate XML documents.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Represent business data by using XML.
- Reference an existing XSL style sheet to format or transform an XML document.
- Use XPath expressions in XSL style sheets.

Prerequisites

Before working on this lab, you must have:

- Familiarity with XML syntax, including attributes and elements.
- Familiarity with XPath query syntax.

Scenario

In this lab, you will implement XML-based solutions for a fictitious e-commerce organization. These solutions include representing business data as XML, and applying style sheets to transform the XML data into HTML format.

Starter and Solution Files

There are starter and solution files associated with this lab. The starter files are in the folder *<install folder>\Labs\Lab01\Starter*, and the solution files are in the folder *<install folder>\Labs\Lab01\Solution*.

Lab Setup

To complete this lab, you need a computer running Microsoft Windows with Microsoft Internet Explorer version 5.5 or later.

Estimated time to complete this lab: 30 minutes

Exercise 1

Using XML to Represent Business Data

In this exercise, you will examine a table of business data and create an XML representation of it. You will then view your XML document by using Internet Explorer.

Scenario

An e-commerce organization stores product data in a relational database. The organization wants to be able to represent its product data as XML in order to make it easier to present the data in multiple presentation formats.

You must create an XML representation of the product data.

► To view the relational product data

- Examine the following table of data.

ProductID	ProductName	UnitPrice	UnitsInStock
1	Chai	18.00	39
2	Chang	19.00	17
3	Aniseed Syrup	10.00	13

► To create an XML representation of the data

The file Catalog.xml in the folder <install folder>\Labs\Lab01\Solution contains a solution to this procedure.

- Start Microsoft Notepad and type the following processing instruction in the new Notepad document:

```
<?xml version='1.0' ?>
```

- On the next line, create an opening tag for an element named Catalog. This will be the root element in the document.
- On the following lines, create an attribute-centric XML representation of the customer data in the table shown in the previous procedure. Each row in the table should be represented by an XML element named Product. Each column should be represented by an attribute with the same name as the corresponding column in the table.
- On the final line, create the closing tag for the Catalog element.
- Save the file as Catalog.xml in the folder <install folder>\Labs\Lab01\Starter.
- Close Notepad.

► **To view the XML document**

1. Start Windows Explorer and view the contents of the folder *<install folder>\Labs\Lab01\Starter*.
2. Double-click *Catalog.xml* to open it in Internet Explorer.
3. Examine the document and ensure that all data in the relational table is represented in the XML document.
4. Right-click in the document and choose **Validate XML**. Verify that the XML document is valid.
5. Close Internet Explorer.

Exercise 2

Formatting XML with XSL

In this exercise, you will examine an XSL style sheet that can be used to format an XML document as HTML. You will then add a reference to an XML document so that the style sheet will be applied when the document is opened in Internet Explorer.

Scenario

An e-commerce organization wants to be able to view product data on a Web page. To achieve this, the organization needs a style sheet that will format the XML data as HTML.

The style sheet has been partially created for you. You need to complete the style sheet and add a reference to it in the `Catalog.xml` document in order to apply the style sheet.

► To complete the `Catalog.xsl` style sheet

The file `Catalog.xsl` in the folder `<install folder>\Labs\Lab01\Solution` contains a solution to this procedure.

1. Start Windows Explorer and view the contents of the folder `<install folder>\Labs\Lab01\Starter`.
2. Right-click `Catalog.xsl` and then click **Open With....** Choose Notepad from the list of available programs.
3. Examine the style sheet, noting that it creates an HTML document containing a table.
4. Find the following comment:

```
<!-- Add your code below this line -->
```
5. Under this line, add an opening tag for an `<xsl:for-each>` element with a **select** attribute of `'Catalog/Product'`.
6. On the next line, add the matching closing tag for the element. Everything between these tags will be applied to each of the `Product` elements in the XML document.
7. Between the opening and closing tags for the `<xsl:for-each>` element, create a new line containing a `<TR>` tag in order to create a new table row.
8. On the next line, add the following code to create a table cell containing the value of the **ProductID** attribute:

```
<TD><xsl:value-of select='@ProductID' /></TD>
```
9. Under this line, add three lines to create table cells containing the values of the **ProductName**, **UnitPrice**, and **UnitsInStock** attributes.
10. Between the `<xsl:for-each>` tags, add a final line to close the `<TR>` element and end the table row.
11. Save the file and close Notepad.

► **To add a reference to the style sheet in the XML document**

This procedure requires the Catalog.xml file created in Exercise 1. If you did not complete Exercise 1, copy the Catalog.xml file from the folder *<install folder>\Labs\Lab01\Solution* to the folder *<install folder>\Labs\Lab01\Starter* before you begin.

The file Catalog(XSLRef).xml in the folder *<install folder>\Labs\Lab01\Solution* contains a solution to this procedure.

1. In the folder Lab01\Starter, right-click Catalog.xml and click **Open With...**. Choose Notepad from the list of available programs.
2. Under the processing instruction that indicates the version of XML in use, add the following processing instruction:

```
<?xml:stylesheet type='text/xsl' href='Catalog.xsl'?>
```
3. Save the file and close Notepad.

Note You can view the solution to this procedure by examining the file Catalog(XSLRef).xsl in the folder Lab01\Solution.

► **To test the style sheet**

1. In the folder Lab01\Starter, double-click Catalog.xml to open it in Internet Explorer.
2. Verify that the Catalog.xsl style sheet is applied. The list of products should be displayed as a table in an HTML document.
3. Close Internet Explorer.

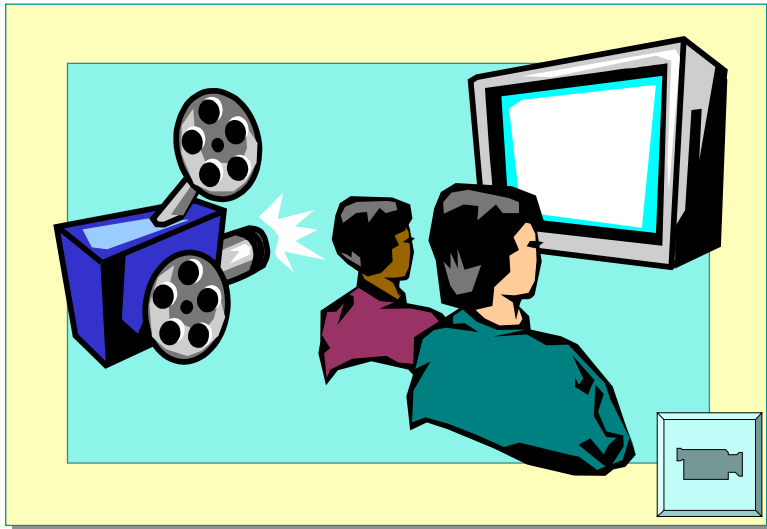
Multimedia: Microsoft BizTalk Server 2000 and XML (Optional)

Topic Objective

To introduce Microsoft BizTalk Server 2000.

Lead-in

In this animation, you will see how Microsoft BizTalk Server 2000 uses XML to transform documents between different formats, and to orchestrate business processes.



*****ILLEGAL FOR NON-TRAINER USE*****

In this animation, you will see how Microsoft® BizTalk™ Server 2000 uses XML to transform documents into different formats, and how BizTalk Server helps orchestrate business processes.

Delivery Tip

This animation is 3 animations in one: a main animation that is about 9 minutes long that provides an overview of how BizTalk Server 2000 transforms business documents, and two linked animations that are each about 5 minutes long. The first linked animation shows how developers create and modify XDR Schemas. The second linked animation shows how BizTalk Orchestration brings long-running processes together. You access the linked animations by clicking the "For more information" buttons when they appear in the main animation, more than halfway through, and near the end of the main animation.

If your SQL Server applications are part of long-running business processes, you might find that using BizTalk Server can save you time and effort in orchestrating the flow of these processes. In addition, BizTalk Server can save you time when creating and maintaining XML Data-Reduced (XDR) Schemas for use with documents and applications. Course 2379A, *Developing and Deploying Microsoft BizTalk Server 2000 Solutions*, covers BizTalk Server 2000 in detail.

Review

Topic Objective

To reinforce module objectives by reviewing key points.

Lead-in

The review questions cover some of the key concepts taught in the module.

- Building Solutions with XML and SQL Server 2000
- XML Technology Review
- XML Support in SQL Server 2000

*****ILLEGAL FOR NON-TRAINER USE*****

1. List as many reasons as you can why XML is a suitable language for building business-integration solutions.

XML is:

- platform-neutral
- protocol-neutral
- language-neutral
- self-describing
- extensible
- flexible

2. Two businesses need to agree on the format of an invoice document. What XML-related technology can help them to do this?

An XML Schema.

3. An e-commerce site generates a product catalog in an XML document. The data needs to be transformed into HTML format for presentation on the Web site. How can you convert the XML into HTML format?

Use an XSL style sheet.

4. In an XML document with a <catalog> root element, a <category> element contains <product> elements. Each <product> element has a **price** attribute. What XPath query would be used to return all products with a price greater than \$10?

catalog/category/product[@price > 10]

5. When using SQL Server's XML-related functionality to access XML data from SQL Server, what task does the SQLOLEDB Provider perform in order to allow the query processor to access the appropriate data?

The SQLOLEDB Provider performs all of the necessary translation between requests for data and the Transact-SQL statement to access that data.

Module 2: Retrieving XML Data

Contents

Overview	1
Using the FOR XML Clause to Retrieve Data	2
Controlling XML Output	13
Lab 2.1: Retrieving XML Data	26
Generating Custom XML Formats	32
Lab 2.2: EXPLICIT Queries	42
Best Practices	49
Review	50



Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BackOffice, BizTalk, MSDN, MS-DOS, SQL Server, Visual Basic, Visual C++, Visual InterDev, Visual J++, Visual Studio, Windows, Windows Media, Windows NT, and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Instructor Notes

Presentation:
120 Minutes

This module provides students with an introduction to the FOR XML clause.

After completing this module, students will be able to:

Lab:
60 Minutes

- Generate Extensible Markup Language (XML) from Microsoft® SQL Server™ data by using the **RAW** and **AUTO** modifiers.
- Use the **Elements** option to return an element-centric XML document.
- Use Table and Column aliases to name the returned elements and attributes.
- Create XML documents containing data from multiple tables by using **JOIN** statements.
- Generate an XML document with a custom structure by using the **EXPLICIT** modifier.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the following materials:

- Microsoft PowerPoint® file 2091A_02.ppt
- The query tool provided on the Instructor compact disc

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the labs and practices.
- Practice the demonstration.

Instructor Setup for a Lab

This section provides setup instructions that are required to prepare the instructor computer or classroom configuration for a lab.

Labs 2.1 and 2.2

► To prepare for the lab

1. Ensure that each student machine is running Windows 2000 with COM+ services.
2. Install the lab files from the Student Materials compact disc.

Demonstration

This section provides demonstration procedures that will not fit in the margin notes or are not appropriate for the student notes.

Universal Tables and XML

► To prepare for the demonstration

1. Install the lab files from the Instructor compact disc.
2. Run Setup.bat from the *<install folder>*.

Module Strategy

Use the following strategies to present this module:

- Using the FOR XML Clause to Retrieve Data

Introduce the FOR XML syntax, and explain that each part of it will be explained in this module. In this section, concentrate on the differences in default behavior when using RAW or AUTO mode.

- Controlling XML Output

Explain that developers might need more control over the format of the resulting XML; for example, they might need to adhere to a particular schema. Using aliases can often be a simple technique for retrieving XML in the format you need. Emphasize the importance of column order and sorting when using **JOIN** statements in AUTO mode queries—failure to sort the results properly can produce incorrect XML.

- Generating Custom XML Formats

Emphasize the importance of universal tables when constructing EXPLICIT mode queries. Encourage students to examine the required XML output format in order to work out the necessary universal table. Once the table structure has been identified, the Transact-SQL to retrieve it can be created more easily.

Overview

Topic Objective

To provide an overview of the module topics and objectives.

Lead-in

In this module, you will learn about the FOR XML Transact-SQL clause, and how it can be used to retrieve data from SQL Server in XML format.

- Using the FOR XML Clause to Retrieve Data
- Controlling XML Output
- Generating Custom XML Formats

*****ILLEGAL FOR NON-TRAINER USE*****

After completing this module, you will be able to:

- Generate Extensible Markup Language (XML) documents from Microsoft® SQL Server™ data by using the RAW and AUTO modifiers.
- Use the **Elements** option to return an element-centric XML document.
- Use Table and Column aliases to name the returned elements and attributes.
- Create XML documents containing data from multiple tables by using **JOIN** statements.
- Generate an XML document with a custom structure by using the **EXPLICIT** modifier.

◆ Using the FOR XML Clause to Retrieve Data

Topic Objective

To introduce the topics covered in this section.

Lead-in

The FOR XML clause is central to XML data retrieval in Microsoft SQL Server 2000.

- Retrieving Data in XML Format
- How SQL Server Generates XML
- Using the FOR XML Clause
- Using RAW Mode to Retrieve XML
- Using AUTO Mode to Retrieve XML
- Practice: Using RAW and AUTO

*****ILLEGAL FOR NON-TRAINER USE*****

The FOR XML clause is central to XML data retrieval in SQL Server 2000. This clause instructs the SQL Server query engine to return the data as an XML stream rather than a rowset. Application developers can build solutions that extract XML business documents, such as orders, invoices, or catalogs, directly from the database.

This section describes the syntax for the FOR XML clause, and how SQL Server uses the RAW and AUTO modifiers to generate the XML output in a predefined format.

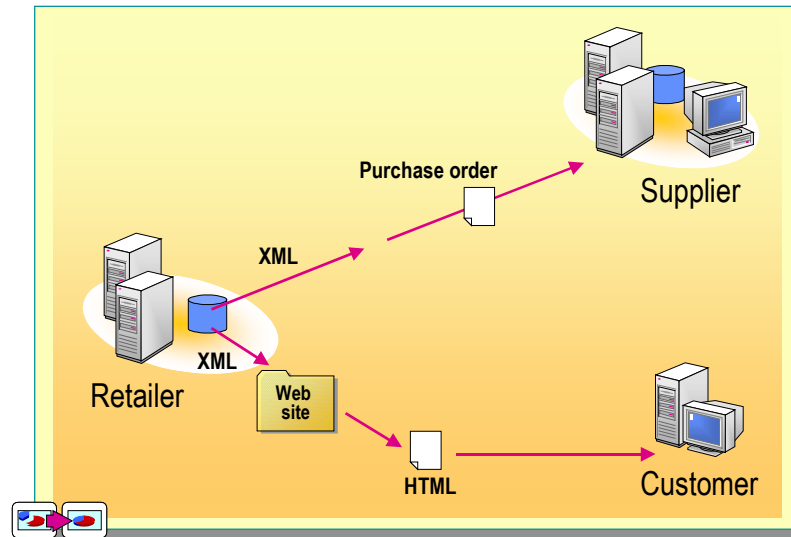
Retrieving Data in XML Format

Topic Objective

To provide a rationale for retrieving data as XML.

Lead-in

There are several scenarios where it is beneficial to retrieve data as XML.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

This slide builds automatically to show XML data being retrieved and published on a Web site, and XML data being retrieved and then sent to a trading partner. To restart the build, press the left arrow key on the keyboard.

There are several situations where you may want to retrieve data in XML format, rather than as a rowset. For example, consider the following data access scenarios:

- Retrieving data to publish it on a Web site.

You can write code in an Active Server Pages (ASP) script to retrieve data as a recordset, and then iterate through the data to generate Hypertext Markup Language (HTML). However, this may not be the most efficient way to publish data on a Web site.

If you can retrieve the data as XML, you can apply an Extensible Stylesheet Language (XSL) style sheet to render the data as HTML. In addition, you can apply a different style sheet to the same XML data to generate an alternative presentation format, such as Wireless Markup Language (WML), and thus support different client devices without rewriting any data access logic.

- Retrieving data to exchange with a trading partner

XML is a natural format for data that must be sent to a trading partner. By retrieving business data in XML format, you can easily integrate your systems with those of external organizations, no matter what data technologies the organizations use internally.

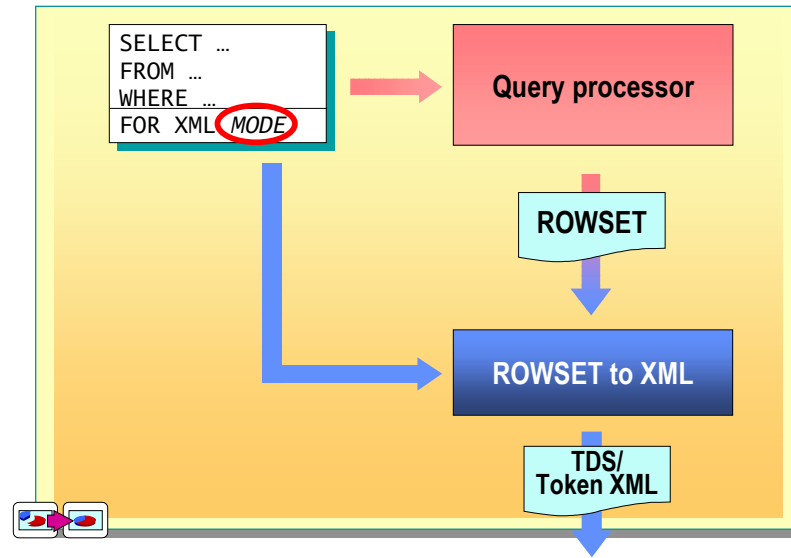
How SQL Server Generates XML

Topic Objective

To describe how SQL Server generates XML.

Lead-in

SQL Server uses certain internal components to return data as XML.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

This slide builds automatically to show the steps listed in the student manual. To restart the build, press the left arrow key on the keyboard.

SQL Server 2000 contains new internal components and functionality to retrieve data as XML.

FOR XML Query Execution Process

When you execute a FOR XML query, SQL Server performs the following process:

1. The SQL Server query processor processes the **SELECT** statement without the FOR XML clause in order to produce a rowset.
2. The rowset is passed to an internal component that transforms the rowset into XML. The XML mode specified in the query determines which component is used.
3. The resulting XML data stream is sent to the client application.

XML and Data Caching

One important aspect of the SQL Server 2000 XML query architecture is that the data streams out directly to the client application—no XML is cached. The query processor caches the rowset data in the normal way, but regenerates the XML for each FOR XML query it executes.

Using the FOR XML Clause

Topic Objective

To introduce the FOR XML syntax.

Lead-in

Use the FOR XML clause in a SELECT statement to retrieve an XML document fragment.

■ The FOR XML Clause Syntax

```
SELECT select_list
FROM table_source
WHERE search_condition
FOR XML RAW | AUTO | EXPLICIT
[, XMLDATA] [, ELEMENTS] [, BINARY BASE64]
```

■ XML Document Fragments

```
<Order OrderID="10248" OrderDate="07/04/1996"/>
<Order OrderID="10249" OrderDate="07/05/1996"/>
```

*****ILLEGAL FOR NON-TRAINER USE*****

You use the FOR XML clause in a Transact-SQL **SELECT** statement to retrieve data as XML. You determine the format of the XML by specifying either the RAW, AUTO, or EXPLICIT modifier. In addition, you can specify three options to control the output.

The For XML Clause Syntax

The modifiers and options are described in the following table.

Modifier / option	Description
RAW modifier	A predefined generic XML grammar formats the resulting XML data. Each row returned by the query produces a <row> element in the resulting XML.
AUTO modifier	A predefined XML grammar based on the name(s) of the table(s) in the query formats the resulting XML data.
EXPLICIT modifier	A custom format specified in the query formats the resulting XML data.
XMLDATA option	The resulting XML data includes an inline XML Schema.
ELEMENTS option	When used with the AUTO modifier, an element-centric grammar formats the resulting XML data.
BINARY BASE64 option	Binary data fields, such as images, are returned as Base64 encoded binary.

Note When you use the SQL Server Query Analyzer to execute a FOR XML query, a single column containing the XML output is returned, and the results might be truncated. You can control the result size by changing the **Maximum characters per column** option in the Query Analyzer.

XML Document Fragments

Delivery Tip

Tell the students that in later modules they will be shown how a root element can be added when retrieving the data through ADO or HTTP.

FOR XML queries do not return well-formed XML documents. Each element in the returned XML represents a row returned by the query; no root element is generated. The partial XML documents returned by FOR XML queries are referred to as *XML document fragments*.

In the following example, a FOR XML query retrieves the following product data from the Northwind Traders SQL Server database.

Fragment Example

```
<Order OrderID="10248" OrderDate="1996-07-04T00:00:00"/>
<Order OrderID="10249" OrderDate="1996-07-05T00:00:00"/>
```

However, the client application must add a root element in order to generate a well-formed XML invoice document, as shown in the following example, which contains the `<OrderList>` root element.

Well-Formed Example

```
<OrderList>
  <Order OrderID="10248" OrderDate="1996-07-04T00:00:00"/>
  <Order OrderID="10249" OrderDate="1996-07-05T00:00:00"/>
</OrderList>
```

Using RAW Mode to Retrieve XML

Topic Objective

To describe how to retrieve XML data by using RAW mode.

Lead-in

You can use the RAW modifier to retrieve data in generic XML <row> elements.

- Generic <row> element produced for each row returned by the query

```
SELECT OrderID, OrderDate
FROM Orders
FOR XML RAW
```

```
<row OrderID="10248" OrderDate="07/04/1996"/>
<row OrderID="10249" OrderDate="07/05/1996"/>
```

*****ILLEGAL FOR NON-TRAINER USE*****

You use a RAW mode query to retrieve an XML fragment in a generic XML format. Applications can then process the XML in the generic format, or apply an Extensible Stylesheet Language Transformations (XSLT) style sheet to transform the XML into the required business document format. Consider the following features of RAW mode:

- A <row> element represents each row in the result set that the query returns.
- An attribute with the same name as the column name or alias that you used in the query represents each column in the result set.
- RAW mode queries can include aggregated columns and GROUP BY clauses.

Retrieving Data in Generic Row Elements

The following example shows how you can retrieve an XML fragment that contains order data by using a FOR XML query in RAW mode.

Example

For Your Information

This query is saved as RAW.txt in the folder DemoCode\Mod02. You can execute it with the HTML Query Tool by navigating to <http://localhost/querytool/query.htm>

```
SELECT OrderID,
       OrderDate
FROM Orders
FOR XML RAW
```

This query produces an XML fragment in the format that contains generic <row> elements, as shown in the following example:

```
<row OrderID="10248" OrderDate="1996-07-04T00:00:00"/>
<row OrderID="10249" OrderDate="1996-07-05T00:00:00"/>
```

Using Aliases in RAW Mode

You can use aliases in RAW mode queries to change the names of the attributes that represent the columns returned by the query. You cannot change table names in RAW mode; all elements will be named <row>.

The following example shows how to use aliases in a RAW mode query that returns a list of orders.

Example

For Your Information

This query is saved as RAWAlias.txt in the folder DemoCode\Mod02. You can execute the query with the HTML Query Tool by moving to <http://localhost/querytool/query.htm>

```
SELECT    OrderID OrderNo,
          OrderDate Date
FROM Orders
FOR XML RAW
```

This query produces the following XML fragment:

```
<row OrderNo="10248" Date="1996-07-04T00:00:00"/>
<row OrderNo="10249" Date="1996-07-05T00:00:00"/>
```

Because you can use RAW mode queries to return aggregate values, you must use an alias to name any aggregate columns in the result set. The following example shows how to produce an XML customer order summary with the alias “TotalOrders”.

Example

For Your Information

This query is saved as RAWGroupBy.txt in the folder DemoCode\Mod02. You can execute it with the HTML Query Tool by navigating to <http://localhost/querytool/query.htm>

```
SELECT    CustomerID,
          Count(OrderID) TotalOrders
FROM Orders
GROUP BY CustomerID
FOR XML RAW
```

This query produces the following XML fragment:

```
<row CustomerID="ALFKI" TotalOrders="6"/>
<row CustomerID="ANATR" TotalOrders="4"/>
```

Using AUTO Mode to Retrieve XML

Topic Objective

To describe how to retrieve XML data by using AUTO mode.

Lead-in

AUTO mode queries retrieve XML data with elements that are named after the tables in the query.

- Table name(s) are used to name elements

```
SELECT OrderID, OrderDate
FROM Orders
FOR XML AUTO
```

↳

```
<Orders OrderID="10248" OrderDate="07/04/1996"/>
<Orders OrderID="10249" OrderDate="07/05/1996"/>
```

*****ILLEGAL FOR NON-TRAINER USE*****

AUTO mode queries produce XML fragments that are either attribute-centric or element-centric. Consider the following features of AUTO mode:

- Each row returned by the query is represented by an XML element with the same name as the table from which it was retrieved (or the alias used in the query).
- Each **JOIN** operator in the query results in a nested XML element.
- Each column in the result set is represented by an attribute, unless the **ELEMENTS** option is specified, in which case each column is represented by a child element.
- Aggregated columns and GROUP BY clauses are not supported in AUTO mode queries.

AUTO mode is useful when business documents must be generated from SQL Server data by using an exclusively attribute-centric or element-centric mapping.

Retrieving Elements Named Using Table Names

The following example shows how you can use an AUTO mode query to return an XML fragment that contains a list of orders.

Example

For Your Information

This query is saved as AUTO.txt in the folderDemoCode\Mod02. You can execute it with the HTML Query Tool by moving to <http://localhost/querytool/query.htm>

```
SELECT    OrderID,  
          OrderDate  
FROM Orders  
FOR XML AUTO
```

This query produces an XML fragment in the following format:

```
<Orders OrderID="10248" OrderDate="1996-07-04T00:00:00"/>  
<Orders OrderID="10249" OrderDate ="1996-07-05T00:00:00"/>
```

Using Aliases in AUTO Mode

You can use aliases in AUTO mode queries to change the names of columns and tables. This approach is useful when the table names do not match the required element names in the business document. In the following example, the Orders table is queried to produce an <Order> element.

Example

For Your Information

This query is saved as AUTOAlias.txt in the folder DemoCode\Mod02. You can execute it with the HTML Query Tool by moving to <http://localhost/querytool/query.htm>

```
SELECT    OrderID OrderNo,  
          OrderDate Date  
FROM Orders Order  
FOR XML RAW
```

This query produces the following XML fragment:

```
<Order OrderNo="10248" Date="1996-07-04T00:00:00"/>  
<Order OrderNo="10249" Date="1996-07-05T00:00:00"/>
```

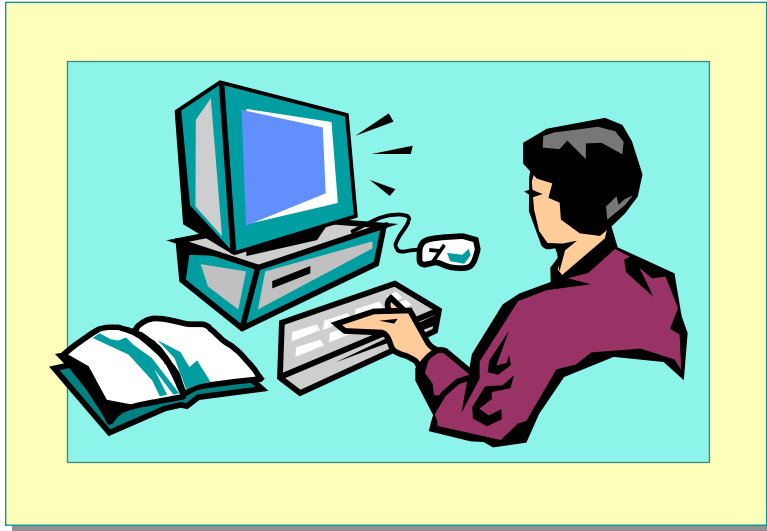

Practice: Using RAW and AUTO

Topic Objective

To give the students an opportunity to practice using the RAW and AUTO modes.

Lead-in

In this practice, you will use the Query Analyzer to execute RAW and AUTO mode queries.



*****ILLEGAL FOR NON-TRAINER USE*****

In this practice, you will use the Query Analyzer to execute RAW and AUTO mode queries in the Northwind Traders database. After completing this practice, you will be able to generate XML from SQL Server data by using the RAW and AUTO modifiers.

► **To view the data from the Titles table by using the Query Analyzer**

1. Start the Query Analyzer and connect to your local server by using Microsoft Windows® authentication.
2. On the **Query** menu, click **Results in Text**.
3. On the **Tools** menu, click **Options**.
4. On the **Results** tab, change the **Maximum characters per column** value to **2000**, and then click **OK**.
5. In the Query window, type the following query:

```
USE Northwind
SELECT OrderID, OrderDate, CustomerID
FROM Orders
```
6. On the **Query** menu, click **Execute**.
7. Examine the rows returned by the query.

► To retrieve data by using a RAW mode query

1. Type the following query, which adds a FOR XML clause with the RAW modifier:

```
USE Northwind
SELECT OrderID, OrderDate, CustomerID
FROM Orders
FOR XML RAW
```
2. On the **Query** menu, click **Execute**.
3. Examine the XML fragment returned by the query. It should contain <row> elements.

► To retrieve data by using an AUTO mode query

1. Change the RAW modifier in the preceding query to AUTO.

```
USE Northwind
SELECT OrderID, OrderDate, CustomerID
FROM Orders
FOR XML AUTO
```
2. On the **Query** menu, click **Execute**.
3. Examine the XML fragment returned by the query. It should contain <Order> elements.

◆ Controlling XML Output

Topic Objective

To introduce the topics in this section.

Lead-in

There are many ways to control the format of the returned XML.

- Specifying the **ELEMENTS** Option
- Working with Multiple Tables
- Practice: Retrieving Data from Multiple Tables
- Specifying the **XMLDATA** Option
- Retrieving Binary Data

*****ILLEGAL FOR NON-TRAINER USE*****

There are many ways to control the format of the XML returned by a RAW or AUTO mode query. For example, you can specify the **ELEMENTS** option to return element-centric XML when using AUTO mode, and use the **JOIN** operator to return data from multiple tables. This type of control is an important aspect of generating business documents from SQL Server data.

This section describes techniques for formatting the XML results of a FOR XML query, and discusses the use of the **ELEMENTS**, **XMLDATA**, and **BINARY BASE64** options.

Specifying the ELEMENTS Option

Topic Objective
To describe the ELEMENTS option.

Lead-in

The ELEMENTS option can be used with AUTO mode to retrieve element-centric XML.

- Columns are returned as child elements

```
SELECT OrderID, OrderDate
FROM Orders
FOR XML AUTO, ELEMENTS
```

```
<Orders>
  <OrderID>10248</OrderID>

  <OrderDate>07/04/1996</OrderDate>
</Orders>
<Orders>
  <OrderID>10249</OrderID>

  <OrderDate>07/05/1996</OrderDate>
</Orders>
```

*****ILLEGAL FOR NON-TRAINER USE*****

By default, queries in both RAW and AUTO mode return attribute-centric XML document fragments. However, you might need to generate an XML document that uses an element-centric grammar to describe business data. For example, a marketing system might require you to send business documents in which you use the **ELEMENTS** option to describe a list of customers.

Using ELEMENTS to Return Columns as Child Elements

You can use the **ELEMENTS** option to produce an element-centric XML fragment. Consider the following facts about the **ELEMENTS** option:

- You can use the **ELEMENTS** option only in AUTO mode queries.
- The **ELEMENTS** option results in a child element for each column in the rows that the query returns.

Note You cannot use the **ELEMENTS** option to produce XML fragments that contain a mixture of attribute-centric and element-centric mappings for the columns in a row returned by the query.

The following example shows how to use the **ELEMENTS** option to return an element-centric XML fragment that contains a list of product details.

Example**For Your Information**

This query is saved as AUTOElements.txt in the folder DemoCode\Mod02. You can execute it with the HTML Query Tool by moving to <http://localhost/querytool/query.htm>

```
SELECT    OrderID,  
          OrderDate  
FROM Products  
FOR XML AUTO, ELEMENTS
```

This query produces an XML fragment in the following format:

```
<Orders>  
  <OrderID>10248</OrderID>  
  <OrderDate>1996-07-04T00:00:00</OrderDate>  
</Orders>  
<Orders>  
  <OrderID>10249</OrderID>  
  <OrderDate>1996-07-05T00:00:00</OrderDate>  
</Orders>
```

Working with Multiple Tables

Topic Objective

To describe how to use JOIN operators in FOR XML queries.

Lead-in

It is often necessary to return data from more than one table.

```
SELECT OrderForm.OrderID, Item.ProductID Item.Quantity
FROM Orders OrderForm JOIN [Order Details] Item
ON OrderForm.OrderID = Item.OrderID
ORDER BY OrderForm.OrderID
FOR XML RAW
```

↳

```
<row OrderID="10248" ProductID="1" Quantity="12"/>
<row OrderID="10248" ProductID="42"
Quantity="10"/>
```

```
SELECT OrderForm.OrderID, Item.ProductID Item.Quantity
FROM Orders OrderForm JOIN [Order Details] Item
ON OrderForm.OrderID = Item.OrderID
ORDER BY OrderForm.OrderID
FOR XML AUTO
```

↳

```
<OrderForm OrderID="10248">
  <Item ProductID="1" Quantity="12"/>
  <Item ProductID="42" Quantity="10"/>
</OrderForm>
```

*****ILLEGAL FOR NON-TRAINER USE*****

It is often necessary to retrieve data for a business document from multiple tables. For example, you might need to generate an invoice that lists orders and their details, using information from separate Order and Order Details tables. In the RAW and AUTO modes, you can use the Transact-SQL **JOIN** operator to produce a result set that contains data from more than one table.

Using the JOIN Operator in RAW Mode

When you use the **JOIN** operator in RAW mode, each row returned by the query returns a <row> element. The <row> element contains attributes that represents columns from the tables in the FROM clause, as shown in the following example.

Example

For Your Information

This query is saved as RAWJoin.txt in the folder DemoCode\Mod02. You can execute it with the HTML Query Tool by moving to <http://localhost/querytool/query.htm>

```
SELECT OrderForm.OrderID,
       Item.ProductID,
       Item.Quantity
FROM Orders OrderForm JOIN [Order Details] Item
ON OrderForm.OrderID = Item.OrderID
ORDER BY OrderForm.OrderID
FOR XML RAW
```

This query produces an XML fragment similar to the following:

```
<row OrderID="10248" ProductID="1" Quantity="12"/>
<row OrderID="10248" ProductID="42" Quantity="10"/>
<row OrderID="10248" ProductID="72" Quantity="5"/>
<row OrderID="10249" ProductID="14" Quantity="9"/>
<row OrderID="10249" ProductID="51" Quantity="40"/>
```

Using the JOIN Operator in AUTO Mode

Delivery Tip

Emphasize that an ORDER BY clause should be used to order the results in the correct hierarchy.

In AUTO mode queries, using the **JOIN** operator produces nested XML elements for each table that is joined. This reduces the duplication of data in the resulting XML fragment. The XML hierarchy is based on the order in which tables are identified from the columns in the SELECT clause. To ensure that the child elements are collated correctly with their parent elements, use an ORDER BY clause to return the data in the correct hierarchical order, as shown in the following example.

Example

For Your Information

This query is saved as AUTOJoin.txt in the folderDemoCode\Mod02. You can execute it with the HTML Query Tool by moving to <http://localhost/querytool/query.htm>

```
SELECT    OrderForm.OrderID,
          Item.ProductID,
          Item.Quantity
FROM Orders OrderForm JOIN [Order Details] Item
ON OrderForm.OrderID = Item.OrderID
ORDER BY OrderForm.OrderID
FOR XML AUTO
```

This query produces an XML fragment similar to the following:

```
<OrderForm OrderID="10248">
  <Item ProductID="1" Quantity="12"/>
  <Item ProductID="42" Quantity="10"/>
  <Item ProductID="72" Quantity="5"/>
</OrderForm>
<OrderForm OrderID="10249">
  <Item ProductID="14" Quantity="9"/>
  <Item ProductID="51" Quantity="50"/>
</OrderForm>
```

Using the JOIN Operator with the ELEMENTS Option

When you use a **JOIN** operator with the **ELEMENTS** option in AUTO mode queries, the results contain an element for each table in the query that contains a child element for the columns returned from that table. The following example shows this type of query.

Example

For Your Information

This query is saved as AUTOElementsJoin.txt in the folderDemoCode\Mod02. You can execute it with the HTML Query Tool by moving to <http://localhost/querytool/query.htm>

```
SELECT    OrderForm.OrderID,
          Item.ProductID,
          Item.Quantity
FROM Orders OrderForm JOIN [Order Details] Item
ON OrderForm.OrderID = Item.OrderID
ORDER BY OrderForm.OrderID
FOR XML AUTO, ELEMENTS
```

This query produces the following XML fragment:

```
<OrderForm>
  <OrderID>10248</OrderID>
  <Item>
    <ProductID>1</ProductID>
    <Quantity>12</Quantity>
  </Item>
  <Item>
    <ProductID>42</ProductID>
    <Quantity>10</Quantity>
  </Item>
  <Item>
    <ProductID>72</ProductID>
    <Quantity>5</Quantity>
  </Item>
</OrderForm>
```

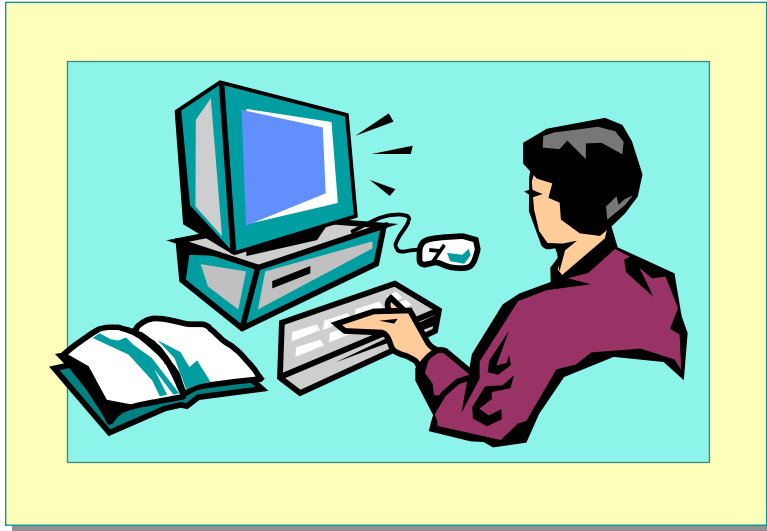

Practice: Retrieving Data from Multiple Tables

Topic Objective

To introduce the practice.

Lead-in

In this practice, you will use RAW and AUTO mode queries to retrieve data from multiple tables.



*****ILLEGAL FOR NON-TRAINER USE*****

In this practice, you will use the Query Analyzer to execute FOR XML queries that retrieve data from multiple tables. After completing this practice, you will be able use **JOIN** operators to create XML documents that contain data from multiple tables.

► To view data from the Orders and Order Details tables

1. Start the Query Analyzer and connect to your local server by using Windows authentication.
2. On the **Query** menu, click **Results in Text**.
3. On the **Tools** menu, click **Options**.
4. On the **Results** tab, change the **Maximum characters per column** value to **2000**, and then click **OK**.
5. In the Query window, type the following query:

```
USE Northwind
SELECT Invoice.OrderID,
       Invoice.OrderDate,
       LineItem.ProductID,
       LineItem.UnitPrice,
       LineItem.Quantity
FROM Orders Invoice JOIN [Order Details] LineItem
ON Invoice.OrderID = LineItem.OrderID
WHERE Invoice.OrderID= 10248
ORDER BY Invoice.OrderID, LineItem.ProductID
```

6. On the **Query** menu, click **Execute**.
7. Examine the rows returned by the query.

► **To use a JOIN operator in a RAW mode query**

1. Add a FOR XML clause with the RAW modifier to the previous query as follows:

```
USE Northwind
SELECT Invoice.OrderID,
       Invoice.OrderDate,
       LineItem.ProductID,
       LineItem.UnitPrice,
       LineItem.Quantity
FROM Orders Invoice JOIN [Order Details] LineItem
ON Invoice.OrderID = LineItem.OrderID
WHERE Invoice.OrderID= 10248
ORDER BY Invoice.OrderID, LineItem.ProductID
FOR XML RAW
```

2. On the **Query** menu, click **Execute**.
3. Examine the XML fragment returned by the query. It should contain a <row> element for each item in order 10248. (The Query Analyzer might truncate the results.)

► **To use a JOIN operator in an AUTO mode query**

1. Change the RAW modifier to the AUTO mode query as follows:

```
USE Northwind
SELECT Invoice.OrderID,
       Invoice.OrderDate,
       LineItem.ProductID,
       LineItem.UnitPrice,
       LineItem.Quantity
FROM Orders Invoice JOIN [Order Details] LineItem
ON Invoice.OrderID = LineItem.OrderID
WHERE Invoice.OrderID= 10248
ORDER BY Invoice.OrderID, LineItem.ProductID
FOR XML AUTO
```

2. On the **Query** menu, click **Execute**.
3. Examine the XML fragment returned by the query. It should contain an <Invoice> element for order 10248 containing a <LineItem> element for each item in the order. (The Query Analyzer might truncate the results.)

► **To use the ELEMENTS option**

1. Add the **ELEMENTS** option to the FOR XML query as follows:

```
USE Northwind
SELECT Invoice.OrderID,
       Invoice.OrderDate,
       LineItem.ProductID,
       LineItem.UnitPrice,
       LineItem.Quantity
FROM Orders Invoice JOIN [Order Details] LineItem
ON Invoice.OrderID = LineItem.OrderID
WHERE Invoice.OrderID= 10248
ORDER BY Invoice.OrderID, LineItem.ProductID
FOR XML AUTO, ELEMENTS
```

2. Execute the query.
3. Examine the XML fragment returned by the query. It should contain the Invoice data in an element-centric format.

Specifying the XMLDATA Option

Topic Objective

To describe the XMLDATA option.

Lead-in

The XMLDATA option can be used to retrieve an XDR Schema in addition to the data.

- XML-Data Reduced (XDR) Schema pre-pended to results

```
SELECT OrderID, OrderDate
FROM Orders
FOR XML AUTO, XMLDATA
```

```
<Schema name="Schema1"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="Orders" content="empty"
    model="closed">
    <AttributeType name="OrderID" dt:type="14"/>
    <AttributeType name="OrderDate" dt:type="dateTime"/>
    <attribute type="OrderID"/>
    <attribute type="OrderDate"/>
  </ElementType>
</Schema>
```

*****ILLEGAL FOR NON-TRAINER USE*****

There are situations where it is useful to include an inline XML-Data Reduced (XDR) Schema in an XML business document to define the data types of the elements and attributes in the document. When you include an inline XDR Schema, you allow trading partners to validate your document when they process it. To retrieve an XML document pre-pended with an XDR Schema, you can specify the **XMLDATA** option in both RAW and AUTO mode queries.

Using XMLDATA to Retrieve a Schema

When using the **XMLDATA** option, consider the following issues:

- Schemas are useful when data types in an XML document must be explicitly defined.
- Generating a schema places a significant load on the server, and should only be done when necessary.
- Queries that produce more than one alias of the same name may result in an invalid schema, because a different data type may be used for each element or attribute that is generated.

Example

The following example shows a FOR XML query that uses the **XMLDATA** option:

For Your Information

This query is saved as XMLDATA.txt in the folder DemoCode\Mod02. You can execute it with the HTML Query Tool by moving to <http://localhost/querytool/query.htm>.

```
SELECT    OrderID,  
          OrderDate  
FROM Orders  
FOR XML AUTO, XMLDATA
```

This query produces an XML fragment similar to the following:

```
<Schema name="Schema1"  
  xmlns="urn:schemas-microsoft-com:xml-data"  
  xmlns:dt="urn:schemas-microsoft-com:datatypes">  
  <ElementType name="Orders" content="empty"  
    model="closed">  
    <AttributeType name="OrderID" dt:type="i4"/>  
    <AttributeType name="OrderDate" dt:type="dateTime"/>  
    <attribute type="OrderID"/>  
    <attribute type="OrderDate"/>  
  </ElementType>  
</Schema>  
<Orders xmlns="x-schema:#Schema1" OrderID="10248"  
  OrderDate="1996-07-04T00:00:00"/>  
<Orders xmlns="x-schema:#Schema1" OrderID="10249"  
  OrderDate="1996-07-05T00:00:00"/>
```

Retrieving Binary Data


Topic Objective

To discuss how binary data can be retrieved.


Lead-in

Binary data can be retrieved as an XPath reference or in base64 encoded format.

```
SELECT EmployeeID, Photo
FROM Employees
WHERE EmployeeID = 10
FOR XML AUTO
```

 `<Employees EmployeeID="1" Photo="dbObject/Employees[@EmpID='1']/@Photo"/>`

```
SELECT EmployeeID, Photo
FROM Employees
WHERE EmployeeID = 1
FOR XML AUTO, BINARY BASE64
```

 `<Employees EmpID="1" Photo="FRwvAAIAAANA4AFAAhAP///9Ca ..."/>`

*****ILLEGAL FOR NON-TRAINER USE*****

You can represent binary data, such as images, in an XML document in two ways:

- Use an XML Path Language (XPath) reference to retrieve the data through a Uniform Resource Locator (URL), which is useful when building Web-based applications that must display images.
- Encode the data as a BASE64 string, which is useful when you must transmit binary data to a trading partner.

Retrieving Binary Data as an XPath Reference

To retrieve an XPath reference to binary data, follow these rules:

- You must use an AUTO mode query.
- You must include the primary key column in the SELECT clause.

The following example shows a FOR XML query that returns a binary field as an XPath reference.

Example

For Your Information

This query is saved as BinaryURL.txt in the folder DemoCode\Mod02. You can execute it with the HTML Query Tool by moving to <http://localhost/querytool/query.htm>

```
SELECT EmployeeID,
       Photo
FROM Employees
WHERE EmployeeID=1
FOR XML AUTO
```

This query produces the following XML fragment:

```
<Employees EmployeeID="1"  
Photo="dbObject/Employees[@EmployeeID='1']/@Photo"/>
```

Delivery Tip

Mention that URL access is discussed fully in a later module.

You can use the **Photo** attribute to retrieve the data from a SQL Server Database Object virtual name by specifying a URL. This technique is particularly useful when data is being published on a Web page.

Retrieving Binary Data in Base64 Encoding

Base 64 is a standard encoding for binary data that is being transferred across the Internet. You can retrieve binary data in a FOR XML query by specifying the **BINARY BASE64** option, as shown in the following example.

Example

```
SELECT EmployeeID,  
       Photo  
FROM Employees  
WHERE EmployeeID=1  
FOR XML AUTO, BINARY BASE64
```

This query produces the following XML fragment:

```
<Employees EmployeeID="1"  
Photo="FRwvAAIAAAANAAP////9CaXtYXAgSW1hZ2UAUG..." />
```

For Your Information

This query is saved as BinaryBASE64.txt in the folder DemoCode\Mod02. You can execute it with the HTML Query Tool by moving to <http://localhost/querytool/query.htm>

The **Photo** attribute has been truncated in this example for clarity. Using the **BINARY BASE64** option is particularly useful when you need to exchange binary data between trading partners.

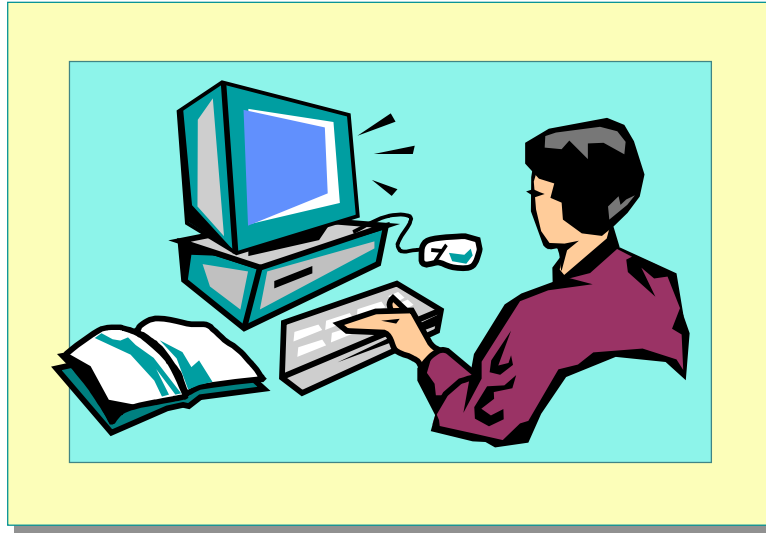
Lab 2.1: Retrieving XML Data

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will submit FOR XML queries to SQL Server.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Generate XML from SQL Server data by using the RAW and AUTO modifiers.
- Use the **Elements** option to return an element-centric XML document.
- Use Table and Column aliases to name the returned elements and attributes.
- Create XML documents that contain data from multiple tables by using **JOIN** statements.

Prerequisites

Before working on this lab, you must know how to:

- Retrieve data from a database by using a **SQL SELECT** statement.
- Use Column and Table aliases in a **SQL SELECT** statement.
- Retrieve aggregate values and use the **GROUP BY** clause in a **SQL SELECT** statement.

Lab Setup

To complete this lab, you need the following:

- A computer running Windows 2000 Server
- SQL Server 2000

For More Information

Read the section “Retrieving XML Documents Using FOR XML” under “Reading and Writing XML Data” in the topic “XML and Internet Support” in SQL Server Books Online.

Scenario

Northwind Traders stores details of products, customers, employees, and orders in a SQL Server database. Data must be extracted from this database in XML format to be exchanged between applications and trading partners.

Estimated time to complete this lab: 30 minutes

Exercise 1

Retrieving XML Data from a Single Table

In this exercise, you will use the FOR XML clause of the **SELECT** statement to retrieve XML data from tables in the Northwind Traders database.

Scenario

You will generate XML documents from the information in individual tables in the Northwind Traders database. In each procedure, you will examine the XML document formats you want to generate, evaluate the appropriate FOR XML mode, and write a query to extract the data. You will use an HTML-based query tool to execute queries and view the results.

Note When the query tool returns XML fragments, it encloses the results in a <queryresults> root element.

► To install and test the query tool

1. Open Windows Explorer and view the contents of the folder <install folder>\Labs\Lab02.
2. Double-click **SetupQueryTool.bat** to install the query tool. Do not run this batch file more than once, because it will produce an error if you do so.
3. Read the message that appears, and then press any key to close the batch file window.
4. Close Windows Explorer.

Note When returning XML fragments, the query tool encloses the results in a <queryresults> root element.

► To retrieve a category list document using a RAW mode query

The file 21-1aCategories.txt in the folder <install folder>\Labs\Lab02\Solution contains a solution to this procedure.

1. Open <http://localhost/QueryTool/Query.htm> with Microsoft Internet Explorer to start the query tool.
2. Confirm that the **Database** text box specifies the **Northwind** database.
3. Type the following query, click **Submit**, and then view the table returned by the query.

```
SELECT CategoryID, CategoryName FROM Categories
```

4. In the browser, click **Back**. Modify the query and add a FOR XML clause so that it retrieves a list of categories in the following XML format:

```
<row CategoryID='1' CategoryName='Beverages' />
```

► To retrieve a product list document by using an AUTO mode query

The file 21-1bProducts.txt in the folder <install folder>\Labs\Lab02\Solution contains a solution to this procedure.

1. Use the query tool to execute the following query and view the contents of the Products table:

```
SELECT ProductID, ProductName, UnitPrice FROM Products
```

2. In the browser, click **Back**. Add a FOR XML clause to the query so that it retrieves a list of products in the following XML format:

```
<Products ProductID='1' ProductName='Chai' UnitPrice='18'/>
```

► To retrieve an element-centric product list document

The file 21-1cProducts.txt in the folder <install folder>\Labs\Lab02\Solution contains a solution to this procedure.

1. Examine the format of the following XML fragment:

```
<Products>
  <ProductID>1</ProductID>
  <ProductName>Chai</ProductName>
  <UnitPrice>18</UnitPrice>
</Products>
```

2. Use the query tool to create a Transact-SQL statement to retrieve a list of products in the XML format in step 1.

► To retrieve a product list document by using aliases

The file 21-1dProducts.txt in the folder <install folder>\Labs\Lab02\Solution contains a solution to this procedure.

1. Examine the format of the following XML fragment:

```
<Product ProductNo='1' Name='Chai' Price='18'/>
```

2. Use the query tool to create a Transact-SQL statement to retrieve a list of products in the preceding XML format.

Exercise 2

Retrieving XML Data from Multiple Tables

In this exercise, you will use the FOR XML clause in queries that contain **JOIN** statements to retrieve XML data from tables in the Northwind Traders database.

Scenario

You need to generate XML documents from multiple tables in the Northwind Traders database. You must examine the XML document formats that need to be returned, evaluate the appropriate FOR XML mode, and identify the queries that are required in order to extract the necessary data.

► To retrieve category summary information by using a RAW mode query

The file 21-2aProductCount.txt in the folder *<install folder>\Labs\Lab02\Solution* contains a solution to this procedure.

1. Open <http://localhost/querytool/query.htm> with Microsoft Internet Explorer to start the query tool.
2. Confirm that the **Database** text box specifies the **Northwind** database.
3. Type the following query, click **Submit**, and then view the table returned by the query.

```
SELECT Categories.CategoryName,  
COUNT(*) ProductCount  
FROM Products JOIN Categories  
ON Products.CategoryID = Categories.CategoryID  
GROUP BY Categories.CategoryName
```

4. Examine the following XML element describing a category and its product count:

`<row CategoryName='Beverages' ProductCount='12' />`
5. Use the query tool to create a Transact-SQL statement to retrieve category and product details in the preceding XML format.

► **To retrieve categories and their products by using an AUTO mode query**

The file 21-2bCatalog.txt in the folder *<install folder>\Labs\Lab02\Solution* contains a solution to this procedure.

1. Open <http://localhost/querytool/query.htm> with Microsoft Internet Explorer to start the query tool.
2. Confirm that the **Database** text box specifies the **Northwind** database.
3. Type the following query, click **Submit**, and then view the table returned by the query.

```
SELECT Category.CategoryID,
       Category.CategoryName,
       Product.ProductID,
       Product.ProductName,
       Product.UnitPrice
FROM Categories Category JOIN Products Product
ON Category.CategoryID = Product.CategoryID
ORDER BY Category.CategoryID
```

4. Examine the following XML fragment describing a category and its products:

```
<Category CategoryID='1' CategoryName='Beverages'>
  <Product ProductID='1' ProductName='Chai' UnitPrice='18'/>
  ...
</Category>
```

5. Use the query tool to create a Transact-SQL statement to retrieve category and product details in the preceding XML format.

◆ Generating Custom XML Formats

Topic Objective

To introduce the topics covered in this section.

Lead-in

In many scenarios, RAW or AUTO mode will provide you with the XML format you need. However, you often might need more control over the output format.

- Custom XML Formats
- Universal Tables
- Using EXPLICIT Mode
- Demonstration: Universal Tables and XML
- Using EXPLICIT Mode with Multiple Tables

*****ILLEGAL FOR NON-TRAINER USE*****

In many scenarios, you can use RAW or AUTO mode queries to produce the required XML format. However, depending on the document definitions you have agreed upon with your trading partners, many business documents might require a more complex XML grammar, perhaps including a mixture of attribute-centric and element-centric mappings, or XML CDATA sections.

This section discusses the use of EXPLICIT mode queries for retrieving custom XML document formats.

Custom XML Formats

Topic Objective

To describe the possible custom XML formats that may be required.

Lead-in

Sometimes the exchange of business documents requires an XML format that cannot be retrieved by using RAW or AUTO mode queries.

■ Columns in a table that are mapped to an XML element can be represented as:

- Element values
- Attributes
- Child elements

```
<Invoice InvoiceNo="10248">
  <Date>1996-07-04T00:00:00</Date>
  <LineItem ProductID="11">Queso Cabrales</LineItem>
  <LineItem ProductID="42">Singaporean Fried Mee</LineItem>
</Invoice>
```

*****ILLEGAL FOR NON-TRAINER USE*****

Sometimes the business documents you must exchange with trading partners require a custom XML format that you cannot retrieve by using RAW or AUTO mode queries.

When you map data from a table to an XML element, the columns in the table can be represented as:

- The value of the element.
- An attribute.
- A child-element.

For example, consider the following XML Invoice document:

```
<Invoice InvoiceNo="10248">
  <Date>1996-07-04T00:00:00</Date>
  <LineItem ProductID="11">Queso Cabrales</LineItem>
  <LineItem ProductID="42">Singaporean Fried Mee</LineItem>
</Invoice>
```

Delivery Tip

Point out the attributes, child elements, and element values in this example. Later in this module, students will be shown how to generate this Invoice document from the Northwind Traders database.

This XML Invoice document contains data from three tables in the Northwind Traders database: Orders, Order Details, and Products. The Invoice document contains data that is mapped by using all three of the mapping methods in the preceding list.

For example, data for the <Invoice> element comes from the Orders table, and the **InvoiceNo** attribute maps to the Order ID column in the Orders table. The value of the <Date> child element of the <Invoice> element comes from the Order Date column in the Orders table.

The <LineItem> child element of the <Invoice> element contains data from the OrderDetails and Products tables that is also mapped by using a combination of mapping methods. For example, the **ProductID** attribute of the <LineItem> element is mapped to the ProductID column in the OrderDetails table, and the <LineItem> element is mapped to the ProductName column in the Products table.

The following list shows a number of document formats that you cannot retrieve by using RAW or AUTO mode queries:

For Your Information
ID, IDREF, and IDREFS data types are XDR Schema-specific data types that are used to create relationships between XML elements.

- Documents that contain element values
- Documents that contain a mix of attribute-centric and element-centric mappings
- Documents that contain CDATA sections or unencoded XML
- Documents that require **ID**, **IDREF**, or **IDREFS** data types

To retrieve documents that require one or more of these formats, you must take one of the following approaches:

- Retrieve the data by using RAW or AUTO mode and apply an XSL style sheet to transform the results,
—or—
- Explicitly define the XML structure in the query.

Universal Tables

Topic Objective

To introduce the concept of a universal table.

Lead-in

The key to generating custom XML is to understand universal tables.

■ Tabular representation of an XML document

- Tag and Parent columns determine hierarchy
- Column names determine element / attribute mapping

Tag	Parent	Invoice!1!InvoiceNo	Invoice!1!Date!Element	LineItem!2!ProductID	LineItem!2
1	NULL	10248	1996-07-04T00:00:00	NULL	NULL
2	1	10248	NULL	11	Queso Cabrales
2	1	10248	NULL	42	Singaporean ...

*****ILLEGAL FOR NON-TRAINER USE*****

The concept of a universal table is the key to understanding the retrieval of custom XML documents. A universal table is a tabular representation of an XML document and contains both the data required in an XML document, and the metadata that defines the structure of the document.

Representing XML with a Universal Table

Each row in a universal table represents data that will be represented as an element in the resulting XML document.

Defining XML Document Hierarchy in a Universal Table

The first two columns in a universal table define the hierarchical position in the resulting XML document of the element that contains the data for the row. The following table describes these columns.

Key Points

When discussing universal tables, it is important to make clear the distinction between a tag and an element. An *element* is an instance of a data entity described in an XML document. A *tag* is an XML markup notation that is used to mark the beginning and end of an element.

Column	Description
Tag	A numerical value that uniquely identifies the tag for the element that contains the data in this row
Parent	A numerical value that identifies the immediate parent tag for this element

Each different XML tag in the resulting document that maps to a table or view in the database must be represented by a different Tag value in the universal table. The Parent value determines the hierarchical position of the tag in the resulting document. Tags at the top level of the XML fragment (and therefore having no immediate parent element) have a parent value of NULL.

For example, the XML document at the end of this section contains an `<Invoice>` with no parent, and two instances of a `<LineItem>`, which is a child of the `<Invoice>` tag. In the universal table, each of the two tags is assigned an arbitrary Tag number to identify it, and the Parent column is used to define how the elements are nested. Notice that the `<LineItem>` element appears twice in the XML document, and the Tag number 2 appears in two rows in the universal table.

Defining Column Mappings in a Universal Table

The remaining columns in a universal table contain the data that will be represented in the document. The column name determines whether the data will be represented as an element value, an attribute, or a child element. Columns for data in a universal table have a name with up to four parts, in the following format.

Syntax

ElementName!TagNumber!AttributeName!Directive

The following table describes the parts of a column name.

Name part	Description
ElementName	The name of the element that contains the data in this row.
TagNumber	The unique number that identifies the tag (as specified in the Tag column). The same ElementName must be used consistently with a given TagNumber.
AttributeName	(Optional) The name of the attribute or child element that represents the data in this column. If this column is unspecified, the data is represented as an element value.
Directive	(Optional) Additional formatting instructions that represent the data as a child element or other XML-specific format.

For example, the following universal table defines an XML invoice document.

Tag	Parent	Invoice!1!InvoiceNo	Invoice!1!Date!Element	LineItem!2!ProductID	LineItem!2
1	NULL	10248	1996-07-04T00:00:00	NULL	NULL
2	1	10248	NULL	11	Queso ...
2	1	10248	NULL	42	Singapore ..

The XML represented by this table looks like the following:

```
<Invoice InvoiceNo='10248'>
  <Date>1996-07-04T00:00:00</Date>
  <LineItem ProductID='11'>Queso Cabrales</LineItem>
  <LineItem ProductID='42'>Singaporean Fried Mee</LineItem>
</Invoice>
```

Using EXPLICIT Mode

Topic Objective

To describe how to use EXPLICIT mode.

Lead-in

After you have determined which universal table you require, you can construct the EXPLICIT mode query.

- Construct the Transact-SQL to generate the universal table
- Add the FOR XML EXPLICIT clause

```
SELECT 1 AS Tag, NULL AS Parent,
       OrderID AS [Invoice!1!InvoiceNo],
       OrderDate AS [Invoice!1!Date!Element]
FROM Orders
WHERE OrderID = 10248
FOR XML EXPLICIT
```

```
<Invoice InvoiceNo="10248">
  <Date>1996-07-04T00:00:00</Date>
</Invoice>
```

*****ILLEGAL FOR NON-TRAINER USE*****

After you have determined the universal table you require in order to represent the desired XML document, you can construct the Transact-SQL query necessary to generate the table by using aliases to name the columns. You can assign the values for the Tag and Parent columns explicitly.

Constructing a Query to Retrieve the Universal Table

The following example shows the necessary Transact-SQL query to retrieve the universal table representing an Invoice document.

Example

For Your Information

This query is saved as ExplicitInvoice.txt in the folder DemoCode\Mod02. You can execute it with the HTML Query Tool by navigating to <http://localhost/querytool/query.htm>

```
SELECT 1 AS Tag,
       NULL AS Parent,
       OrderID AS [Invoice!1!InvoiceNo],
       OrderDate AS [Invoice!1!Date!Element]
FROM Orders
WHERE OrderID=10248
```

Note The “AS” keyword is optional when assigning an alias.

The universal table produced by this query looks like the following:

Tag	Parent	Invoice!1!InvoiceNo	Invoice!1!Date
1	NULL	10248	1996-07-04T00:00:00

Adding the FOR XML EXPLICIT Clause

When you have identified the necessary Transact-SQL query to generate the universal table, add the FOR XML clause with the EXPLICIT modifier to retrieve the XML fragment, as shown in the following example.

Example

```
SELECT 1 AS Tag,  
        NULL AS Parent,  
        OrderID AS [Invoice!1!InvoiceNo],  
        OrderDate AS [Invoice!1!Date!Element]  
FROM Orders  
WHERE OrderID=10248  
FOR XML EXPLICIT
```

This query produces the following XML fragment:

```
<Invoice InvoiceNo='10248'>  
  <Date>1996-07-04T:00:00:00</Date>  
</Invoice>
```

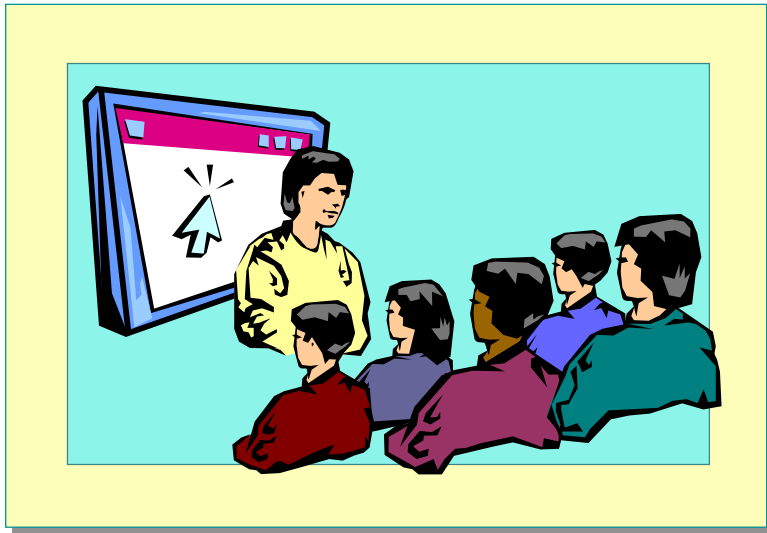
Demonstration: Universal Tables and XML

Topic Objective

To demonstrate how a universal table can be used to represent an XML document.

Lead-in

In this demonstration, you will see how a universal table can be used to represent an XML document.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

1. Open [HTTP://localhost/QueryTool/query.htm](http://localhost/QueryTool/query.htm) with Internet Explorer.
2. Ensure that the Database is set to Northwind.
3. Enter the query shown in the student workbook, and click Submit.
4. Point out that NULL values in the resulting table are represented by empty cells.
5. Point out that all of the rows have a Tag value of 1 and a Parent value of NULL. In other words, each row will result in a top-level element.
6. Point out the names of the columns. The top-level element will be called "Item", the ProductName will be used as the value of the element, and the ProductID, Quantity, and UnitPrice columns will be attributes.
7. Click the browser's Back button to return to the query.
8. Add FOR XML EXPLICIT to the query, and click Submit.
9. View the XML fragment that is returned.
10. Close Internet Explorer.

In this demonstration, you will see how to use a universal table to represent an XML document.

Use the following query to generate the universal table:

```
SELECT 1 AS Tag,
       NULL AS Parent,
       OD.ProductID AS [Item!1!ProductID],
       OD.Quantity AS [Item!1!Quantity],
       OD.UnitPrice AS [Item!1!Price],
       P.ProductName AS [Item!1]
FROM [Order Details] OD
JOIN Products P ON OD.ProductID = P.ProductID
WHERE OrderID=10248
```

Note This query is provided in a text file named ExplicitItem.txt in the folder *<install folder>\DemoCode\Mod02*.

Using EXPLICIT Mode with Multiple Tables

Topic Objective

To describe how data is retrieved from multiple tables in an EXPLICIT mode query.

Lead-in

To create hierarchical XML document fragments, the universal table must contain results from multiple queries.

- Use the UNION ALL operator to retrieve multiple Tag values

```
SELECT 1 AS Tag, NULL AS Parent,
       OrderID AS [Invoice!1!InvoiceNo],
       OrderDate AS [Invoice!1!Date!Element],
       NULL AS [LineItem!2!ProductID],
       NULL AS [LineItem!2]
FROM Orders
WHERE OrderID=10248
UNION ALL
SELECT 2 AS Tag, 1 AS Parent,
       OD.OrderID,
       NULL,
       OD.ProductID,
       P.ProductName
FROM [Order Details] OD JOIN Orders O ON OD.OrderID=O.OrderID
JOIN Products P ON OD.ProductID = P.ProductID
WHERE OD.OrderID=10248
ORDER BY [Invoice!1!InvoiceNo], [LineItem!2!ProductID]
FOR XML EXPLICIT
```

*****ILLEGAL FOR NON-TRAINER USE*****

In order to use EXPLICIT mode to generate an XML document containing multiple tags, you must write individual queries for each tag and then merge the queries by using a **UNION ALL** operator.

Using the UNION ALL Operator to Retrieve Multiple Tags

Consider the following facts when using the **UNION ALL** operator in **EXPLICIT** mode queries:

- Each query must return a consistent set of columns so that the results can be merged successfully. You can assign **NULL** entries to columns that are not used by the current query.
- You use an **ORDER BY** clause to merge the results into the correct XML hierarchy.

For example, an XML invoice document might require the following format.

Example

```
<Invoice InvoiceNo=OrderID>
  <Date>InvoiceDate</Date>
  <LineItem ProductID=ProductID Qty=Quantity Price=UnitPrice>
    ProductName
  </LineItem>
  <LineItem ProductID=ProductID Qty=Quantity Price=UnitPrice>
    ProductName
  </LineItem>
  etc.
</Invoice>
```

Because this format contains two XML tags that are mapped to tables or views (Invoice and LineItem), you must use two queries to retrieve the data.

The first query must produce an <Invoice> element that contains an **InvoiceNo** attribute and a <Date> child element. Because the query will be combined with another query by using the **UNION ALL** operator, you must also specify columns as placeholders for the ProductID, Qty, and Price fields that are returned in the other query.

The second query must retrieve <LineItem> elements under the Invoice element. These contain the **ProductID**, **Qty**, and **Price** attributes of each item, as well as the ProductName assigned to the value of the <LineItem> element. You must also retrieve the OrderID in order to join items to their orders.

The following example shows the two queries that are required to retrieve the <Invoice> and <LineItem> elements, combined by using a **UNION** operator, to retrieve an XML document with multiple tags.

Example

For Your Information

This query is saved as ExplicitUnion.txt in the folder DemoCode\Mod02. You can execute it with the HTML Query Tool by navigating to <http://localhost/querytool/query.htm>

```
SELECT    1 AS Tag, NULL AS Parent,
          OrderID AS [Invoice!1!InvoiceNo],
          OrderDate AS [Invoice!1!Date!Element],
          NULL AS [LineItem!2!ProductID],
          NULL AS [LineItem!2!Qty],
          NULL AS [LineItem!2!Price],
          NULL AS [LineItem!2]
FROM Orders
WHERE OrderID=10248
UNION ALL
SELECT    2 AS Tag, 1 AS Parent,
          OD.OrderID,
          NULL,
          OD.ProductID,
          OD.Quantity,
          OD.UnitPrice,
          P.ProductName
FROM [Order Details] OD JOIN Orders O ON OD.OrderID=O.OrderID
JOIN Products P ON OD.ProductID = P.ProductID
WHERE OD.OrderID=10248
FOR XML EXPLICIT
```

This query produces the following XML fragment:

```
<Invoice InvoiceNo='10248'>
  <Date>1996-07-04T00:00:00</Date>
  <LineItem ProductID='1' Qty='12' Price='14'>
    Queso Cabrales
  </LineItem>
  <LineItem ProductID='42' Qty='10' Price='9.8'>
    Singaporean Fried Mee
  </LineItem>
</Invoice>
```

Lab 2.2: EXPLICIT Queries

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will use EXPLICIT mode queries to retrieve data in custom XML formats.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Retrieve a universal table.
- Generate an XML document with a custom structure by using the **EXPLICIT** option.

Prerequisites

Before working on this lab, you must know how to:

- Retrieve data from a database by using a SQL **SELECT** statement.
- Use column and table aliases in a SQL **SELECT** statement.

Lab Setup

To complete this lab, you need the following:

- A computer running Windows 2000 Server
- SQL Server 2000 installed

For More Information

Read the section “Retrieving XML Documents Using FOR XML” under “Reading and Writing XML Data” in the topic “XML and Internet Support” in SQL Server Books Online.

Scenario

Northwind Traders stores details about its products and customers in a SQL Server database. You want to generate XML documents in custom formats from the database. The documents will be used to exchange information among applications and trading partners.

Estimated time to complete this lab: 30 minutes

Exercise 1

Using EXPLICIT Mode to Retrieve Data

In this exercise, you will use EXPLICIT mode in FOR XML queries to extract data from the Northwind Traders database into custom XML documents.

Scenario

You want to generate custom XML documents from the tables in the Northwind Traders database. You will examine the XML document formats you want to generate, create a universal table, and write the queries required to extract the necessary data.

► To retrieve an order with an EXPLICIT mode query

The file 22-1aOrder.txt in the folder *<install folder>\Labs\Lab02\Solution* contains a solution to this procedure.

1. Open <http://localhost/QueryTool/Query.htm> with Internet Explorer.
2. Confirm that the **Database** text box specifies the **Northwind** database.
3. Type and execute the following Transact-SQL query:

```
SELECT 1 AS Tag, NULL AS Parent,
       OrderID AS [Order!1!OrderNo],
       CustomerID AS [Order!1!CustomerID],
       OrderDate AS [Order!1!OrderDate!ELEMENT]
FROM Orders
WHERE Orders.OrderID=10248
```

4. Examine the universal table produced by this query.
5. In the browser, click **Back**, add a FOR XML EXPLICIT clause to the query, and then execute the query. Examine the XML fragment that is produced, and note that it contains a mixture of element values, attributes, and child elements.

```
<Order OrderNo="10248" CustomerID="VINET">
  <OrderDate>1996-07-04T00:00:00</OrderDate>
</Order>
```

► To generate an invoice with an EXPLICIT query

The file 22-1bInvoice.txt in the folder *<install folder>\Labs\Lab02\Solution* contains a solution to this procedure.

1. Modify the query you created in the last procedure so that it retrieves data from the Orders table in the following XML format:

```
<Invoice InvoiceNo="10248" CustomerID="VINET">
  <OrderDate>1996-07-04T00:00:00</OrderDate>
</Invoice>
```

2. Execute the query and examine the results.

► **To retrieve order details with an EXPLICIT mode query**

The file 22-1cOrderDetails.txt in the folder *<install folder>*\Labs\Lab02\Solution contains a solution to this procedure.

1. Write a Transact-SQL query that retrieves data from the [Order Details] table for the order with an OrderID of 10248 in the following XML format:

```
<LineItem ProductID="11" Quantity="12">  
  <UnitPrice>14</UnitPrice>  
</LineItem>  
<LineItem ProductID="42" Quantity="10">  
  <UnitPrice>9.8</UnitPrice>  
</LineItem>  
<LineItem ProductID="72" Quantity="5">  
  <UnitPrice>34.8</UnitPrice>  
</LineItem>>
```

2. Execute the query and examine the results.

Exercise 2

Using EXPLICIT Mode with Multiple Tables

In this exercise, you will use EXPLICIT mode queries to retrieve XML data from multiple tables.

Scenario

You need to generate custom XML documents from multiple tables in the Northwind Traders database. You must examine the XML document formats that need to be returned, identify the appropriate universal table, and write the queries required to extract the necessary data.

► To retrieve an invoice and its line items

The file 22-2aInvoice.txt in the folder *<install folder>\Labs\Lab02\Solution* contains a solution to this procedure.

1. Open <http://localhost/QueryTool/Query.htm> with Internet Explorer.
2. Confirm that the **Database** text box specifies the **Northwind** database.
3. Type and execute the following Transact-SQL query:

```
SELECT 1 AS Tag,
        NULL AS Parent,
        OrderID AS [Invoice!1!InvoiceNo],
        OrderDate AS [Invoice!1!OrderDate],
        NULL AS [LineItem!2!ProductID],
        NULL AS [LineItem!2!UnitPrice],
        NULL AS [LineItem!2!Quantity!Element]
FROM Orders
WHERE OrderID=10248
UNION ALL
SELECT 2,
        1,
        Orders.OrderID,
        NULL,
        [Order Details].ProductID,
        [Order Details].UnitPrice,
        [Order Details].Quantity
FROM [Order Details] JOIN Orders
ON [Order Details].OrderID = Orders.OrderID
WHERE [Order Details].OrderID=10248
ORDER BY [Invoice!1!InvoiceNo], [LineItem!2!ProductID]
```

4. Examine the universal table produced by this query. Note that the universal table represents two levels of elements, indicated by the Tag and Parent column values.

5. In the browser, click **Back**, add a FOR XML EXPLICIT clause to the query, and then click **Submit**. Examine the XML fragment that is produced.
6. Click **Back** and rewrite the query to produce an invoice in the following XML format:

```
<Invoice InvoiceNo="10248">
  <OrderDate>1996-07-04T00:00:00</OrderDate>
  <LineItem ProductID="11">
    <UnitPrice>14</UnitPrice>
    <Quantity>12</Quantity>
  </LineItem>
  <LineItem ProductID="42">
    <UnitPrice>9.8</UnitPrice>
    <Quantity>10</Quantity>
  </LineItem>
  <LineItem ProductID="72">
    <UnitPrice>34.8</UnitPrice>
    <Quantity>5</Quantity>
  </LineItem>
</Invoice>
```

If Time Permits

Generating an XML Document with Three Elements

In this exercise, you will use an EXPLICIT mode query to generate an XML invoice for an order.

► **To generate an XML invoice containing three elements**

The file 22ExtraInvoice.txt in the folder *<install folder>\Labs\Lab02\Solution* contains a solution to this procedure.

1. Write a Transact-SQL query that generates an invoice document for order 10248 in the following XML format:

```
<Invoice InvoiceNo="10248" OrderDate="1996-07-04T00:00:00">
  <Customer CustomerID="VINET">
    <CompanyName>Vins et alcools Chevalier</CompanyName>
    <ContactName>Paul Henriot</ContactName>
  </Customer>
  <LineItem ProductID="11">
    <UnitPrice>14</UnitPrice>
    <Quantity>12</Quantity>
  </LineItem>
  <LineItem ProductID="42">
    <UnitPrice>9.8</UnitPrice>
    <Quantity>10</Quantity>
  </LineItem>
  <LineItem ProductID="72">
    <UnitPrice>34.8</UnitPrice>
    <Quantity>5</Quantity>
  </LineItem>
</Invoice>
```

2. Execute the query and examine the results.

Best Practices

Topic Objective

To describe best practices for retrieving XML.

Lead-in

This slide lists some best practices for retrieving XML data from SQL Server by using the FOR XML clause.

- **Use RAW Mode for Aggregated Data**
- **Use AUTO Mode for Attribute-Centric or Element-Centric XML**
- **Use Aliases with AUTO and RAW to Name XML Elements and Attributes**
- **Use an ORDER BY Clause when Querying Multiple Tables**
- **Use EXPLICIT Mode for XML Documents Containing a Mix of Attribute-Centric and Element-Centric Mappings**

*****ILLEGAL FOR NON-TRAINER USE*****

Consider the following best practices when using the FOR XML clause to retrieve XML data from SQL Server.

Use RAW Mode for Aggregated Data

You must use RAW mode when you need to retrieve an aggregate value (such as an average or a sum), or you need to group the results by using a GROUP BY clause.

Use AUTO Mode for Attribute-Centric or Element-Centric XML

Use AUTO mode to retrieve the data for XML documents that use completely attribute-centric or element-centric mappings.

Use Aliases with AUTO and RAW to Name XML Elements and Attributes

You can exercise control over a business document by using FOR XML queries that use aliases to name the elements and attributes.

Use an ORDER BY Clause when Querying Multiple Tables

When you retrieve XML data from more than one table, you should use an ORDER BY clause to ensure that the XML hierarchy is correctly nested.

Use Explicit Mode for XML Documents Containing a Mix of Attribute-Centric and Element-Centric Mappings

Use EXPLICIT mode to generate the XML when the required document contains both attribute-centric and element-centric mappings.

Review

Topic Objective

To reinforce module objectives by reviewing key points.

Lead-in

The review questions cover some of the key concepts taught in the module.

- Using the FOR XML Clause to Retrieve Data
- Controlling XML Output
- Generating Custom XML Formats

*******ILLEGAL FOR NON-TRAINER USE*******

1. You need to retrieve data from a SQL Server database in an attribute-centric XML format. The elements representing the data should be given the names(s) of the tables(s) they came from. Which mode should you use?

AUTO mode.

2. You need to generate an XML document from a Customers table and an Orders table. The resulting document should list each customer and the total number of orders he or she has placed. Which mode should you use?

RAW mode.

3. You need to retrieve data from a SQL Server database in an element-centric XML format. Which mode should you use?

AUTO mode.

4. You need to produce the following XML document. Which mode should you use?

```
<Invoice InvoiceNo="10012">
  <Customer CustomerID="1234">
    <Email>chor@NorthWindTraders.com</Email>
  </Customer>
</Invoice>
```

EXPLICIT Mode.

5. In a universal table, what name must you give to a column that contains data for an attribute named **InvoiceNo** of an element named Invoice with a Tag value of 1?

[Invoice!1!InvoiceNo]

6. In a universal table, what name must you give to a column that contains data for an encoded child element named InvoiceNo of an element named Invoice with a Tag value of 1?

[Invoice!1!InvoiceNo!ELEMENT]

Module 3: Manipulating XML with Transact-SQL

Contents

Overview	1
Creating a Rowset from an XML Document	2
Specifying the Structure of a Rowset	13
Lab 3: Using OPENXML	26
Best Practices	33
Review	34



Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BackOffice, BizTalk, MSDN, MS-DOS, SQL Server, Visual Basic, Visual C++, Visual InterDev, Visual J++, Visual Studio, Windows, Windows Media, Windows NT, and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Instructor Notes

Presentation:
90 Minutes

This module provides students with an understanding of how to insert XML data into SQL Server tables by using the **OPENXML** statement.

Lab:
30 Minutes

After completing this module, students will be able to:

- Use the **OPENXML** statement to create a rowset from a single-level Extensible Markup Language (XML) document.
- Use the **OPENXML** statement to process rowsets from complex XML documents.
- Retrieve attributes, elements, or both from an XML document by specifying the appropriate flags parameter with the **OPENXML** statement.
- Use XML Path Language (XPath) expressions in rowpattern and colpattern parameters to specify rowset structure.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need Microsoft® PowerPoint® file 2091A_03.ppt.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the lab and practice.
- Practice the demonstration.
- Review the multimedia animation.

Module Strategy

Use the following strategies to present this module:

- Creating a Rowset from an XML Document

Emphasize the importance of using the **sp_xml_removedocument** system stored procedure to remove from memory the tree generated by the **sp_xml_preparedocument** system stored procedure.

Emphasize that the main purpose of using **OPENXML** is to shred a document into one or more tables.

Do not go into details about the rowpattern, flags, and Table Name parameters, and do not discuss colpattern parameters. These will be introduced in the next section.

- Specifying the Structure of a Rowset

Emphasize that combining flags parameters by using a logical **OR** can introduce performance issues. A better approach is to specify either an attribute-centric or element-centric mapping and use column patterns for non-default-mapped data.

Overview

Topic Objective

To provide an overview of the module topics and objectives.

Lead-in

In this module, you will learn about the **OPENXML** statement and how you can use it to manipulate XML data in Transact-SQL.

- **Creating a Rowset from an XML Document**
- **Specifying the Structure of a Rowset**

*******ILLEGAL FOR NON-TRAINER USE*******

After completing this module, you will be able to:

- Use the **OPENXML** statement to create a rowset from a single-level Extensible Markup Language (XML) document.
- Use the **OPENXML** statement to process rowsets from complex XML documents.
- Retrieve attributes, elements, or both from an XML document by specifying the appropriate *flags* parameter with the **OPENXML** statement.
- Use XML Path Language (XPath) expressions in *rowpattern* and *colpattern* parameters to specify rowset structure.

◆ Creating a Rowset from an XML Document

Topic Objective

To introduce the topics in this section.

Lead-in

In an application integration scenario, you often must create a rowset from XML data in order to process it.

- Process Overview
- Multimedia: Parsing and Shredding XML
- Creating an Internal Tree
- Retrieving a Rowset from an XML Tree
- Inserting Data from XML Documents into Tables
- Practice: Retrieving Rowsets with OPENXML

*****ILLEGAL FOR NON-TRAINER USE*****

A *rowset* is an OLE DB object that contains the result set for a query. In a trading partner integration scenario, you might need to generate a rowset from an XML document. For example, a retailer sends orders to a supplier as XML documents. The supplier must then generate rowsets from the XML in order to insert the data into one or more tables in the database.

This section discusses the use of the **OPENXML** statement to generate rowsets from data in XML documents.

Process Overview

Topic Objective

To describe the process used to generate a rowset from XML.

Lead-in

There are five steps involved in processing XML data as a rowset.

1. **Receive an XML document**
 - Usually a parameter to a stored procedure
2. **Generate an internal tree representation**
 - Use `sp_xml_preparedocument` to parse the XML
3. **Retrieve a rowset from the tree**
 - Use `OPENXML`
4. **Process the data from the rowset**
 - Usually “shredding” into permanent tables
5. **Destroy the internal tree when it is no longer required**
 - Use `sp_xml_removedocument`

*****ILLEGAL FOR NON-TRAINER USE*****

Processing XML data as a rowset involves the following five steps:

1. Receive an XML document.

When an application receives an XML document, it can process the document by using Transact-SQL code. For example, when a supplier receives an XML order from a retailer, the supplier logs the order in a Microsoft® SQL Server™ database. Usually, the Transact-SQL code to process the XML data is implemented in the form of a stored procedure, and the XML string is passed as a parameter.

2. Generate an internal tree representation.

Before processing the document, use the **sp_xml_preparedocument** system stored procedure to parse the XML document and transform it into an in-memory tree structure. The tree is conceptually similar to a Document Object Model (DOM) representation of an XML document. You can use only a valid, well-formed XML document to generate the internal tree.

3. Retrieve a rowset from the tree.

You use the **OPENXML** statement to generate an in-memory rowset from the data in the tree. Use XPath query syntax to specify the nodes in the tree to be returned in the rowset.

4. Process the data from the rowset.

Use the rowset created by **OPENXML** to process the data, in the same way that you would use any other rowset. You can select, update, or delete the data by using Transact-SQL statements. The most common use of **OPENXML** is to insert rowset data into permanent tables in a database. For example, an XML order received by a supplier might contain data that must be inserted into OrderHeader and OrderDetails tables.

5. Destroy the internal tree when it is no longer required.

Because the tree structure is held in memory, use the **sp_xml_removedocument** system stored procedure to free the memory when the tree is no longer required.

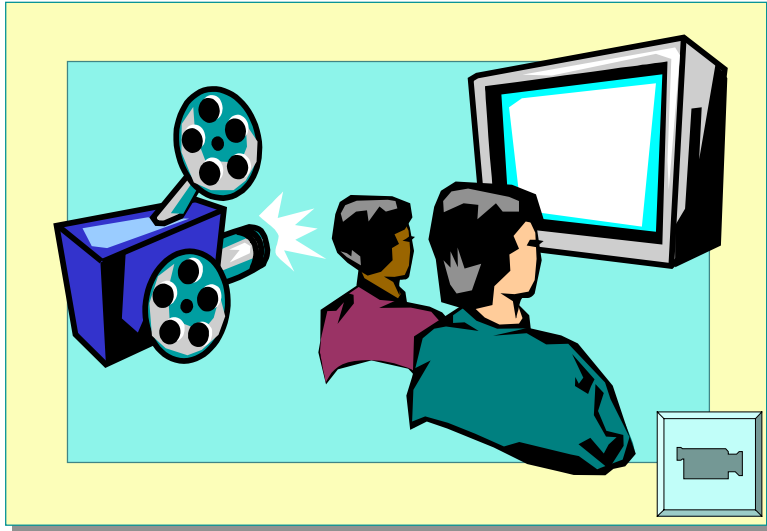
Multimedia: Parsing and Shredding XML

Topic Objective

To introduce the animation, which is about 2 ½ minutes long.

Lead-in

In this animation, you will observe the process of shredding an XML document into a SQL Server database.



*****ILLEGAL FOR NON-TRAINER USE*****

In this animation, you will observe the process used to receive an XML document and insert its data into tables in a SQL Server database.

Creating an Internal Tree

Topic Objective

To describe the use of the `sp_xml_preparedocument` stored procedure.

Lead-in

Use the `sp_xml_preparedocument` stored procedure to parse an XML document and generate the internal tree.

- Create the tree with `sp_xml_preparedocument`
- Free memory with `sp_xml_removedocument`

```
CREATE PROC ProcessOrder @doc NText
AS
DECLARE @idoc integer
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc

-- Process Document

EXEC sp_xml_removedocument @idoc
```

*****ILLEGAL FOR NON-TRAINER USE*****

Before you can process an XML document by using Transact-SQL statements, you must parse the document and transform it into a tree structure.

Parsing XML with `sp_xml_preparedocument`

The `sp_xml_preparedocument` system stored procedure parses an XML document and generates an internal tree representation of the document, and has the following syntax.

Syntax

```
sp_xml_preparedocument hdoc OUTPUT [, xmltext]
[, xpath_namespaces]
```

The following table describes the parameters for the `sp_xml_preparedocument` system stored procedure.

Parameter	Description
xmltext	The original XML document to be processed
hdoc	A handle to the parsed XML tree
xpath_namespaces (optional)	XML namespace declarations that are used in row and column XPath expressions in OPENXML statements

The following example shows how to use the **sp_xml_preparedocument** system stored procedure to parse an XML document that has been passed to a new custom stored procedure.

Example

```
CREATE PROC ProcessOrder @doc NText
AS
DECLARE @idoc integer
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
```

Freeing Memory with **sp_xml_removedocument**

You store parsed documents in the internal cache. To avoid running out of memory, use the **sp_xml_removedocument** system stored procedure to release the document handle and destroy the tree structure when it is no longer required.

You must call **sp_xml_removedocument** in the same query batch as the **sp_xml_preparedocument** system stored procedure that is used to generate the node tree. This is because the **hdoc** parameter that is used to reference the tree is a local variable, and if it goes out of scope, there is no way to remove the tree from memory.

Note You implement the **sp_xml_preparedocument** and **sp_xml_removedocument** system stored procedures as extended stored procedures. This means that you can call them from within your own custom stored procedures, but not from within a user-defined function.

Retrieving a Rowset from an XML Tree

Topic Objective

To describe the OPENXML statement.

Lead-in

After you have parsed the XML document, you can use the OPENXML statement.

■ OPENXML syntax

- The rowpattern parameter determines rows
- The WITH clause determines columns

Use the Flags parameter to determine attribute centricity or element centricity

■ Using OPENXML

- Valid anywhere you can use a rowset
- Commonly used in SELECT statements

```
SELECT * FROM OpenXML (@idoc, 'order', 1)
WITH (orderno integer,
      orderdate datetime)
```

*****ILLEGAL FOR NON-TRAINER USE*****

After you have parsed an XML document by using the **sp_xml_preparedocument** system stored procedure, and a handle to the internal tree has been returned, you can generate a rowset from the parsed tree.

Delivery Tip

Although the rowpattern, flags, and Table Name syntax elements are only briefly mentioned here, they will be discussed in detail in the next section.

Using the OPENXML Statement

You use the **OPENXML** statement to retrieve a rowset from the tree. You can then write Transact-SQL **SELECT**, **UPDATE**, or **INSERT** statements that modify a database.

The **OPENXML** statement has the following syntax.

Syntax

```
OPENXML(idoc, rowpattern [, flags])
[WITH (SchemaDeclaration) | TableName]
```

The following table describes parameters of the **OPENXML** statement.

Parameter	Description
rowpattern	XPath query defining the nodes that should be returned
idoc	Handle to the internal tree representation of the XML document
flags	Optional bit-mask determining attribute centricity or element centricity
SchemaDeclaration	Rowset schema declaration for the columns to be returned
TableName	Name of an existing table, the schema of which should be used to define the columns that are returned

You can use an **OPENXML** statement anywhere you can use a rowset provider, such as a table, view, or the **OPENROWSET** function. You use the **OPENXML** statement primarily in **SELECT** statements, as shown in the following example.

Example

```
SELECT * FROM OPENXML(@idoc, 'order/lineitem', 1)
WITH (productid integer,
      price money)
```

In the preceding example, “@idoc” is a handle to the internal tree representation of the following XML order document:

```
<order orderno="1001" orderdate="01/01/2001" custid="1235">
  <lineitem productid="14" quantity="2" price="15.99"/>
  <lineitem productid="17" quantity="1" price="5.49"/>
  <lineitem productid="21" quantity="2" price="14.99"/>
</order>
```

The following table shows the rowset that the preceding **OPENXML** statement returns.

productid	price
14	15.99
17	5.49
21	14.99

Formatting the Rowset

You use the **WITH** clause of the **OPENXML** statement to define the structure of the rowset returned by the query. The rowset must be compatible with the table into which the data is inserted. If you do not include a **WITH** clause with an **OPENXML** statement, the query returns an edge table, which is discussed later in this module.

A **WITH** clause must contain a description of the mapping between the XML nodes and the columns in the rowset. This description is in the form of a schema declaration, which can be a table name or a table schema. The following table contains guidelines for when to use a name or a schema.

Use	When
<i>Table Name</i>	You have an existing table with the exact structure that you want, so that the table schema is the same as the rowset schema.
<i>Table Schema</i>	You need to map the attributes and elements in the XML document to columns in a table that have different names from the columns in the rowset.

Inserting Data from XML Documents into Tables

Topic Objective

To describe how OPENXML can be used to insert XML data into a table.

Lead-in

The primary use of OPENXML is to insert XML data into tables.

■ Use an INSERT statement for an existing table

```
INSERT orders
SELECT * FROM OpenXML (@idoc, 'order', 1)
WITH (orderno integer,
      orderdate datetime)
```

■ Use a SELECT INTO statement to create a new table

```
SELECT * INTO neworders
FROM OpenXML (@idoc, 'order', 1)
WITH (orderno integer,
      orderdate datetime)
```

*****ILLEGAL FOR NON-TRAINER USE*****

For Your Information

Previous versions of SQL Server required that the SELECT INTO / BULKCOPY option be set before using SELECT INTO to create a permanent table. SQL Server 2000 has no requirement to set this option.

The primary use of the **OPENXML** statement is to insert XML data into tables, which is a common task when integrating two applications that use XML to represent business data. You can achieve this integration by using an **OPENXML** query within an **INSERT** statement, or by using a **SELECT INTO** statement.

The following table contrasts the **INSERT** and **SELECT INTO** statements.

Statement	Usage	Permissions
INSERT	Inserts data only into existing tables.	Requires INSERT permission on the table.
SELECT INTO	Creates a new table and populates it with the result set of a SELECT statement.	Requires CREATE TABLE permission in the destination database.

Using the INSERT Statement with Existing Tables

You use the **INSERT** statement to insert data from an XML document into an existing SQL Server table. For example, use the following Transact-SQL statement to insert data from an XML document into an existing orders table.

Example

```
INSERT orders
SELECT * FROM OPENXML(@idoc, 'order', 1)
WITH (orderno integer,
      [date] datetime,
      customer integer)
```

This Transact-SQL statement inserts the following data into the orders table.

orderno	date	customer
1001	01/01/2001	1235

Using the SELECT INTO Statement to Create a Table

You use the **SELECT INTO** statement to create a new table containing data from an XML document. For example, use the following Transact-SQL statement to create a new table named neworders that contains data from an XML document.

Example

```
SELECT * INTO neworders
FROM OPENXML(@idoc, 'order', 1)
WITH (orderno integer,
      [date] datetime,
      customer integer)
```

This statement creates the following neworders table.

orderno	date	customer
1001	01/01/2001	1235

Practice: Retrieving Rowsets with OPENXML

Topic Objective

To introduce the practice.

Lead-in

In this practice, you will use the OPENXML statement to retrieve a rowset from an XML document.



*****ILLEGAL FOR NON-TRAINER USE*****

In this practice, you will use the Microsoft SQL Server Query Analyzer to execute an **OPENXML** statement and retrieve a rowset from an XML document.

► To execute an OPENXML statement

1. Start the SQL Server Query Analyzer.
2. Connect to your local server by using Microsoft Windows® authentication.
3. Open the file **openxml.sql** in the folder *<install folder>\Practices\Mod03*.
4. Under the comment **Add your code here**, type the following Transact-SQL statement:

```
EXEC sp_xml_preparedocument @idoc OUTPUT, @xmlidoc
SELECT * FROM
OPENXML (@idoc, 'order', 1)
WITH (orderno integer,
      orderdate datetime,
      custid integer)
EXEC sp_xml_removedocument @idoc
```

5. Execute the query and view the resulting rowset. It should look similar to the following table.

orderno	orderdate	custid
1001	2001-01-01 00:00:00.000	1235

◆ Specifying the Structure of a Rowset

Topic Objective

To introduce the topics in this section.

Lead-in

You can specify the structure of the rowset to be retrieved by using **OPENXML** in many different ways.

- Specifying Rows Using the **rowpattern** Parameter
- Specifying Attributes or Elements Using Flags
- Specifying Columns Using a Table Name
- Specifying a Column Pattern
- Using Column Patterns Instead of Combining Flags
- Retrieving Data from XML Documents with Complex Hierarchies
- Demonstration: Controlling a Rowset Structure

*****ILLEGAL FOR NON-TRAINER USE*****

You can use the **OPENXML** statement to create a rowset from a nested XML document and shred the document into multiple tables. In this section, you will see how to control the format of the rowset produced by an **OPENXML** statement.

Specifying Rows Using the rowpattern Parameter

Topic Objective

To describe how to use the rowpattern parameter.

Lead-in

The rowpattern parameter specifies the rows returned in the rowset.

- Use an XPath query to specify the hierarchy level to search

```
SELECT * FROM OpenXML (@idoc, 'order/lineitem', 1)
WITH (productid integer,
      quantity integer,
      price money)
```

- Add a condition to the XPath query to limit the rows returned

```
SELECT * FROM
OpenXML (@idoc, 'order/lineitem[@quantity>2]', 1)
WITH (productid integer,
      quantity integer,
      price money)
```

*****ILLEGAL FOR NON-TRAINER USE*****

The rowpattern parameter specifies the rows that the rowset returns. This parameter is an XPath pattern that relates to an element in the XML document hierarchy.

Using an XPath Query to Specify the Hierarchy Level to Search

You use the rowpattern parameter to specify the level of the XML hierarchy to be searched. For example, consider the following XML order document.

Example

```
<order orderno="1001" orderdate="01/01/2001" custid="1235">
  <lineitem productid="14" quantity="2" price="15.99"/>
  <lineitem productid="17" quantity="1" price="5.49"/>
  <lineitem productid="21" quantity="2" price="14.99"/>
</order>
```

This document has two levels in its hierarchy: “order” and “lineitem”. You can use the following Transact-SQL statement to retrieve data from the order hierarchy level:

```
SELECT * FROM OPENXML(@idoc, 'order', 1)
WITH (orderno integer,
      [date] datetime,
      customer integer)
```

Use the following Transact-SQL statement to search the lineitem level:

```
SELECT * FROM OPENXML(@idoc, 'order/lineitem', 1)
WITH (productid integer,
      quantity integer,
      price money)
```

Using an XPath Condition to Limit the Rows Returned

You use an XPath condition to limit the rows that the query returns. For example, you can use the following Transact-SQL statement to return only items with a quantity attribute of greater than 2.

Example

```
SELECT *
FROM OPENXML(@idoc, 'order/lineitem[@quantity>2]', 1)
WITH (productid integer,
      quantity integer,
      price money)
```

Specifying Attributes or Elements Using Flags

Topic Objective

To discuss how to control the centrality of the rowset.

Lead-in

You can choose to retrieve attributes or elements from the XML document.

- **Use the flags parameter to specify attributes or elements**
 - 0 = default (attributes)
 - 1 = attributes
 - 2 = elements
- **Combine flags to retrieve both**
 - 3 = attributes and elements (1 + 2)

*****ILLEGAL FOR NON-TRAINER USE*****

You can map the columns in the rowset produced by the **OPENXML** statement to attributes or element values in the XML document.

Using the flags Parameter to Specify Attributes or Elements

You can use the *flags* parameter to specify that the **OPENXML** statement should retrieve values from attributes or elements. You can set the *flags* parameter to one of the values in the following table in order to retrieve attributes or elements.

flags Value	Description
0	Use the default mapping (which is attributes).
1	Retrieve attribute values.
2	Retrieve element values.
3	Retrieve both attribute and element values.

For example, you can set the flags attribute to retrieve attributes or elements from the following XML document:

```
<order orderno="1001" orderdate="01/01/2001" custid="1235">
  <lineitem productid="14" quantity="2" price="15.99">
    <color>red</color>
  </lineitem>
  <lineitem productid="17" quantity="1" price="5.49">
    <color>yellow</color>
  </lineitem>
  <lineitem productid="21" quantity="2" price="14.99">
    <color>green</color>
  </lineitem>
</order>
```

You can use the following code to retrieve attributes from the XML data:

```
SELECT * FROM OPENXML(@idoc, 'order/lineitem', 1)
WITH (productid integer,
      quantity integer,
      price money)
```

The preceding **SELECT** statement produces the following rowset.

orderno	quantity	price
14	2	15.99
17	1	5.49
21	2	14.99

Use the following code to retrieve elements from the XML document.

Example

```
SELECT * FROM OPENXML(@idoc, 'order/lineitem', 2)
WITH (color varchar(10))
```

The preceding **SELECT** statement produces the following rowset.

color
red
yellow
green

Combining Flags

You can combine flags to retrieve both attributes and elements. For example, use the following code to retrieve both attributes and elements from the XML document.

Example

```
SELECT * FROM OPENXML(@idoc, 'order/lineitem', 3)
WITH (productid integer,
      quantity integer,
      price money,)
      color varchar(10))
```

The preceding statement produces the following rowset.

productid	quantity	price	color
14	2	15.99	red
17	1	5.49	yellow
21	2	14.99	green

Caution Combining flags can introduce a performance overhead.

Specifying Columns Using a Table Name

Topic Objective

To describe how a rowset can be based on an existing table.

Lead-in

The structure of a rowset can be based on an existing table.

■ Specify the table name in the WITH Clause

- Column names must match
- Data types must be compatible

```
INSERT lineitems
SELECT * FROM OpenXML (@idoc, 'order/lineitem', 3)
WITH lineitems
```

*****ILLEGAL FOR NON-TRAINER USE*****

You can base the structure of a rowset that you retrieve by using the **OPENXML** statement on an existing table. This is particularly useful when you use the **OPENXML** statement to insert data from an XML document into the table.

To retrieve a rowset based on an existing table, specify the table name in the **WITH** clause instead of a rowset schema definition.

Specifying the Table Name in the WITH Clause

The table that you specify in the **WITH** clause determines the structure of the rowset produced by the **OPENXML** statement. Consider the following issues when you use a table name in the **WITH** clause:

- The column names in the table must match the attribute or element names in the XML document.
- The data types of the values in the XML document must be compatible with the column data types in the table definition.

For example, you can use the following Transact-SQL statement to retrieve a rowset based on an existing lineitems table.

Example

```
SELECT * FROM OPENXML (@idoc, 'order/lineitem', 3)
WITH
lineitems
```


Specifying a Column Pattern

Topic Objective

To describe how you can use column patterns to retrieve data from anywhere in the hierarchy.

Lead-in

Sometimes the required data must be retrieved from various locations within the document hierarchy.

- Relative XPath used to retrieve data from anywhere in the hierarchy
- Use relative XPath instead of combining flags

```
INSERT lineitems
SELECT * FROM OpenXML (@idoc, 'order/lineitem', 1)
WITH (orderno integer '@orderno',
      productid integer, --Default Mapping
      quantity integer './quantity',
      price money --Default Mapping)
```

*****ILLEGAL FOR NON-TRAINER USE*****

Sometimes you must retrieve the data you require from multiple levels in the hierarchy of the XML document. For example, you can represent a customer order document as shown in the following example.

Example

```
<order orderno="1001" orderdate="01/01/2001" custid="1235">
  <lineitem productid="14" price="15.99">
    <quantity>1</quantity>
  </lineitem>
  <lineitem productid="17" price="5.49">
    <quantity>2</quantity>
  </lineitem>
</order>
```

Delivery Tip

Use this XML sample to explain the OPENXML statement on the slide. The rowpattern in the OPENXML statement returns all of the <lineitem> elements. The XPath './@orderno' retrieves the orderno attribute from the parent element (<order>), and the './quantity' XPath retrieves the quantity child element. The other columns use the flag value (1 = attributes) to retrieve the productid and price attributes.

In this example, the lineitem element does not contain the "orderno" value required by the lineitems table. You must retrieve the "orderno" value from the order element, which is the parent of the lineitem element.

Using a Relative XPath to Retrieve Data

In situations where you must retrieve data from throughout the hierarchy, you can use a column pattern that contains an XPath pattern. The XPath pattern in a colpattern parameter is relative to the XPath pattern specified in the rowpattern parameter.

For example, you can use the following stored procedure to insert the order data into the orders and lineitems tables. This procedure retrieves the “orderno” value for the lineitems table from the <order> element.

Example

```
CREATE PROC insertorder @xmlOrder NText
AS
INSERT orders
SELECT * FROM OPENXML (@idoc, 'order', 1)
WITH orders
INSERT lineitems
SELECT * FROM OPENXML (@idoc, 'order/lineitem', 3)
WITH (orderno integer '@orderno',
      productid integer,      price money,
      quantity integer)
```

This stored procedure inserts the following sets of data into the orders and lineitems tables.

orderno	orderdate	custid
1001	01/01/2001	1235

orderno	productid	price	quantity
1001	14	15.99	1
1001	17	5.49	2

Using Column Patterns Instead of Combining Flags

Topic Objective

To describe how column patterns can be used instead of flags to retrieve data from throughout the hierarchy.

Lead-in

Sometimes it is more efficient to use column patterns to retrieve the required data from various locations within the document hierarchy.

■ Use column patterns instead of combining flags to:

- Retrieve child elements
- Retrieve attributes

*****ILLEGAL FOR NON-TRAINER USE*****

You can use column patterns to retrieve data from elements that use both attribute-centric and element-centric mappings. This may be more efficient than using a combined flags value. Consider the following approaches to using column patterns to retrieve data from elements that have mixed centricity:

- Specify attribute centricity (1) as the default flag, and use a column pattern in the format “*./element_name*” to retrieve child elements.
- Specify element centricity (2) as the default flag, and use a column pattern in the format “*@attribute_name*” to retrieve attributes.

The following example shows the use of column patterns to retrieve attributes and elements.

Example

```
CREATE PROC insertorder @xmlOrder NText
AS
INSERT orders
SELECT * FROM OPENXML (@idoc, 'order', 1)
WITH orders
INSERT lineitems
SELECT * FROM OPENXML (@idoc, 'order/lineitem', 2)
WITH (orderno integer '../@orderno',
      productid integer '@productid',    price money
      '@price',
      quantity integer, --default mapping)
```

The preceding stored procedure inserts the following sets of data into the orders and lineitems tables.

orderno	orderdate	custid
1001	01/01/2001	1235

orderno	productid	price	quantity
1001	14	15.99	1
1001	17	5.49	2

Retrieving Data from XML Documents with Complex Hierarchies

Topic Objective

To describe how to retrieve data in nested tables.

Lead-in

XML documents often contain a nested hierarchy of elements.

- **Use multiple OPENXML statements for parent child documents**
 - Use different rowpatterns to locate data
 - Data types must be compatible
- **Combine flags to retrieve attributes *and* elements**
 - May introduce a performance overhead

```
INSERT orders
SELECT * FROM OpenXML (@idoc, 'order', 1)
WITH orders
INSERT lineitems
SELECT * FROM OpenXML (@idoc, 'order/lineitem', 3)
WITH lineitems
```

*****ILLEGAL FOR NON-TRAINER USE*****

XML documents often contain a nested hierarchy of elements that relate to multiple tables. For example, an XML order document is likely to contain order header data such as the order number and date, and order detail data such as the product IDs, quantities, and prices of the items ordered.

You can use an **OPENXML** statement to create a rowset from nested XML documents and shred the document into multiple tables. This section describes the use of the **flags** option to retrieve both attributes and elements, and the use of column patterns to retrieve data from throughout the hierarchy.

It is common in many business data integration scenarios to exchange hierarchical XML documents. For example, a customer order could be represented by using an order element containing multiple lineitem elements.

Using Multiple OPENXML Statements

You can use the following XML document to represent a customer order that must be logged in two SQL Server tables. Use an orders table to store the order data, and use a lineitems table for the lineitem data.

Example

```
<order orderno="1001" orderdate="01/01/2001" custid="1235">
  <lineitem orderno="1001" productid="14" price="15.99"/>
  <lineitem orderno="1001" productid="17" price="5.49"/>
</order>
```

Notice that the <lineitem> elements contain an **orderno** attribute that is used to match the line items to the particular order they belong to. To insert the data from this hierarchical document into the appropriate tables, use two **OPENXML** statements: one for the order table, the other for the lineitems table, as shown in the following stored procedure.

Example

```
CREATE PROC insertorder @xmlOrder NText
AS
INSERT orders
SELECT * FROM OPENXML (@idoc, 'order', 1)
WITH orders
INSERT lineitems
SELECT * FROM OPENXML (@idoc, 'order/lineitem', 1)
WITH lineitems
```

The preceding stored procedure inserts the following sets of data into the orders and lineitems tables.

orderno	orderdate	custid
1001	01/01/2001	1235

orderno	productid	price
1001	14	15.99
1001	17	5.49

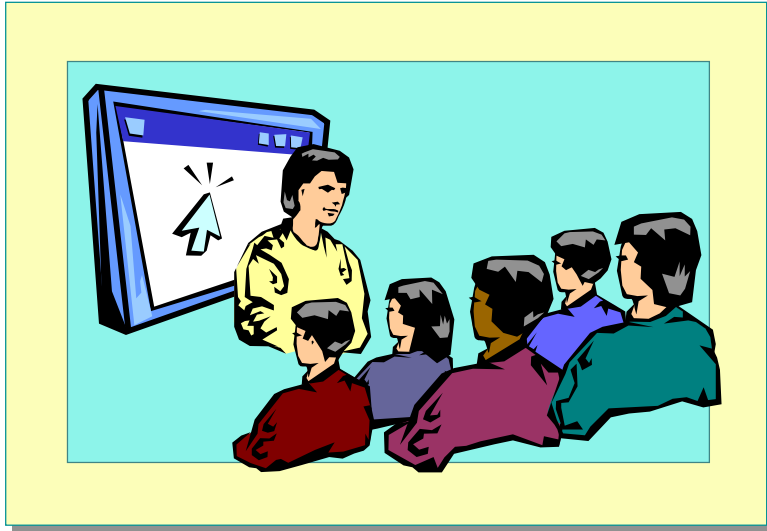
Demonstration: Controlling a Rowset Structure

Topic Objective

To demonstrate how to use an OPENXML statement to retrieve data from nested XML documents.

Lead-in

In this demonstration, you will see how an OPENXML statement can be used to process nested XML documents.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

1. Start the SQL Server Query Analyzer.
2. Open the file nested.sql from the folder `<install folder>\democode\mod03`.
3. Point out that the code contains two OPENXML statements: one for the order header data, and one for the line items data.
4. Highlight the OPENXML statement for the line items, and explain the XPaths used for the orderno and quantity columns:
orderno = the orderno attribute of the parent element;
quantity= the value of the quantity child element.
5. Execute the query and show the retrieved rowsets in the results grid.

In this demonstration, you will see how to use the **OPENXML** statement to create a rowset from data in a nested XML document.

The following XML order document will be processed in the demonstration:

```
<order orderno="1001" orderdate="01/01/2001" custid="1235">
  <lineitem productid="14" price="15.99">
    <quantity>1</quantity>
  </lineitem>
  <lineitem productid="17" price="5.49">
    <quantity>2</quantity>
  </lineitem>
  <lineitem productid="21" price="14.99">
    <quantity>1</quantity>
  </lineitem>
</order>
```

Lab 3: Using OPENXML

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will use the OPENXML statement to generate rowsets from XML documents and insert data into tables.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Use an **OPENXML** statement to create a rowset from a single-level XML document.
- Process rowsets from complex XML documents by using **OPENXML** statements.
- Retrieve attributes, elements, or both from an XML document by specifying the appropriate flags parameter.
- Use XPath expressions in rowpattern and colpattern parameters to specify rowset structure.

Prerequisites

Before working on this lab, you must:

- Know how to use XPath expressions to retrieve nodes from an XML document.
- Know how to write a Transact-SQL script containing an **OPENXML** statement.

Lab Setup

To complete this lab, you need a computer running Windows 2000 Server with SQL Server 2000 installed.

For More Information

Review the section “Writing XML Using OpenXML” in SQL Server Books Online.

Scenario

You are a developer for an online supplier organization. Your organization often receives XML documents from its trading partners, and data must be extracted from the documents and inserted into tables in a SQL Server database. You must write the necessary **OPENXML** statements to extract the data.

Estimated time to complete this lab: 30 minutes

Exercise 1

Retrieving Rowsets from XML Data with OPENXML

In this exercise, you will write the necessary **OPENXML** statements to insert data from an XML document into a SQL Server table.

Scenario

XML documents often contain data that must be inserted into a single SQL Server table. The XML documents can use attribute-centric mappings, element-centric mappings, or both. You must write appropriate **OPENXML** statements to retrieve the required data for the table.

► To extract a rowset by using OPENXML

The file 3-1aOpenXML.sql in the folder *<install folder>\Labs\Lab03\Solution* contains a solution to this procedure.

1. Start the SQL Server Query Analyzer and connect to your server by using Windows authentication.
2. Open the SQL script file 3-1aOpenXML.sql from the folder *<install folder>\Labs\Lab03\Starter*.
3. Find the comment **Type your code here**.
4. Under the comment, type the following query:

```
SELECT * FROM
OpenXML(@idoc, 'invoice', 1)
WITH
(invoiceno integer,
 invoicedate datetime)
```

5. Execute the query and view the rowset that is returned.
6. Modify the query to return the customer element as an **integer** column and the total element as a **money** column.
7. Execute the query again and view the rowset that is returned. It should contain a single row containing the **customer** and **total** fields.
8. Modify the query again to return the **invoiceno**, **invoicedate**, **customer**, and **total** fields.
9. Execute the query again and view the rowset that is returned. It should contain a single row containing the **invoiceno**, **invoicedate**, **customer**, and **total** fields.

► **To insert XML data into a table with a matching structure**

The file 3-1bInsertXML.sql in the folder <install folder>\Labs\Lab03\Solution contains a solution to this procedure.

1. Open the SQL script file 3-1bInsertXML.sql from the folder <install folder>\Labs\Lab03\Starter.
2. Find the comment **Table Definition**. Note the structure of the **#invoices** table.
3. Find the comment **Type your code here**. Enter an **INSERT** statement to insert the **invoiceno**, **invoicedate**, **customer**, and **total** values into the **#invoices** table. Use the table name to define the rowset structure.
4. Execute the query and view the rowset that is returned. This represents the contents of the **#invoices** table, and should contain a single row containing the **invoiceno**, **invoicedate**, **customer**, and **total** fields.

► **To insert XML data into a table with a non-matching structure**

The file 3-1cInsertXML2.sql in the folder <install folder>\Labs\Lab03\Solution contains a solution to this procedure.

1. Open the SQL script file 3-1cInsertXML2.sql from the folder <install folder>\Labs\Lab03\Starter.
2. Find the comment **Table Definition**. Note the structure of the **#invoices** table.
3. Find the comment **Type your code here**. Enter an **INSERT** statement to insert the **invoiceno**, **invoicedate**, **customer**, and **total** values into the **#invoices** table. Define the necessary table schema in the **WITH** clause of the **OPENXML** statement.
4. Execute the query and view the rowset that is returned. This represents the contents of the **#invoices** table, and should contain a single row containing the **invoiceno**, **invoicedate**, **customer**, and **total** fields.

Exercise 2

Inserting XML Data into Multiple Tables

In this exercise, you will write the necessary **OPENXML** statements to insert data from an XML document into a related SQL Server tables.

Scenario

XML documents often contain data that must be inserted into multiple SQL Server tables. The XML documents may use attribute-centric mappings, element-centric mappings, or both. You must write appropriate **OPENXML** statements to retrieve the required data for the tables.

► To retrieve multiple rowsets from an XML document

The file 3-2aOpenXMLMulti.sql in the folder *<install folder>\Labs\Lab03\Solution* contains a solution to this procedure.

1. Open the SQL script file 3-2aOpenXMLMulti.sql from the folder *<install folder>\Labs\Lab03\Starter*.
2. Find the comment **Type your code here**.
3. Under the comment, type two queries:
 - The first query should return a rowset containing the **invoiceno**, **invoicedate**, **customer**, and **total** fields.
 - The second query should return a rowset containing the **productid**, **price**, and **quantity** fields.
4. Execute the query and view the two rowsets that are returned.

► To insert XML data into multiple tables

The file 3-2bInsertMultiXML.sql in the folder *<install folder>\Labs\Lab03\Solution* contains a solution to this procedure.

1. Open the SQL script file 3-2bInsertMultiXML.sql from the folder *<install folder>\Labs\Lab03\Starter*.
2. Find the comment **Table Definitions**. Note the structure of the **#invoices** and **#lineitems** tables.
3. Find the comment **Type your code here**.
4. Under the comment, type two queries:
 - The first query should insert the **invoiceno**, **invoicedate**, **customer**, and **total** fields into the **#invoices** table.
 - The second query should insert the **invoiceno**, **productid**, **price**, and **quantity** fields into the **#lineitems** table.
5. Execute the query and view the two rowsets that are returned. These represent the contents of the **#invoices** and **#lineitems** tables.

Exercise 3

Creating a Stored Procedure to Insert XML Data

In this exercise, you will create a stored procedure to insert an XML document into tables in a SQL Server database.

Scenario

Stored procedures are commonly used to insert XML data into SQL Server database tables. You must create a stored procedure that can accept an order document as a parameter and insert the data into the appropriate tables. The order document has the following structure:

```
<order orderid="10359" customerid="CONTO"
      employeeid="5" shipvia="3" freight="288.43">
  <dates orderdate="01/01/2001"
        requireddate="02/02/2001"
        shippeddate="01/01/2001"/>
  <shippingdetails shipname="Contoso Ltd."
                  shippingaddress="Maple St."
                  city="London" region="Europe"
                  postcode="OX15 4NB" country="UK"/>
  <orderdetails>
    <lineitem productid="16" unitprice="13.9"
              quantity="56" discount="0.00"/>
    <lineitem productid="31" unitprice="10"
              quantity="70" discount="0.00"/>
    <lineitem productid="60" unitprice="27.2"
              quantity="80" discount="0.00"/>
  </orderdetails>
</order>
```

► **To create a stored procedure**

The SQL script file 3-3CreateSProc.sql in the folder *<install folder>\Labs\Lab03\Solution* contains a solution to this procedure.

1. Open the SQL script file 3-3CreateSProc.sql from the folder *<install folder>\Labs\Lab03\Starter*.
2. Find the comment **Table Definitions**. Note the structure of the **Neworders** and **[New Order details]** tables.
3. Find the comment **Type your code here**.
4. Complete the stored procedure so that it accomplishes the following tasks:
 - Create an internal tree representation of the XML document passed in the xmldoc parameter.
 - Insert data from the XML document into the **Neworders** table.
 - Insert data from the XML document into the **[New Order Details]** table.
 - Release the memory used by the internal tree.
5. Execute the query to create the tables and the stored procedure.

► **To test the stored procedure**

1. Open the SQL script file 3-3TestSProc.sql from the folder *<install folder>\Labs\Lab03\Starter*.
2. Execute the query and view the rowsets that are returned. The details from the XML order should have been inserted into the **Neworders** and **[New Order Details]** tables by your stored procedure.

Best Practices

Topic Objective

To describe best practices for using OPENXML statements.

Lead-in

The following best practices should be considered when using OPENXML statements.

- **Create stored procedures to insert incoming XML data**
- **Do not use OPENXML for large volumes of data**
- **Always call sp_xml_removedocument**
- **Use column patterns instead of combining flags to handle XML with mixed centricity**

*****ILLEGAL FOR NON-TRAINER USE*****

Consider the following best practices when using the **OPENXML** statement.

Create Stored Procedures to Insert Incoming XML Data.

A stored procedure can encapsulate the logic necessary to shred an XML document, insert it into multiple tables, and remove the node tree from memory. The XML document should be passed as a parameter that is sufficiently large for the maximum size of document you expect to pass.

Do Not Use OPENXML for Large Volumes of Data.

The **OPENXML** statement is designed to shred data from an XML document into a small number of rows (for example a single row into an Orders table, and a few rows into an Order Details table). It does not perform efficiently when bulk-loading large volumes of data (for example, importing a product catalog or loading an entire table). If you need to bulk-load XML data into SQL Server, consider using the Bulk Load component shipped with the XML for SQL Web Release.

Always Call sp_xml_removedocument

You should always call the sp_xml_removedocument system stored procedure to reclaim the memory used by a node tree when you are finished processing the XML data. If trees are not cleared from memory, performance of the server as a whole may begin to suffer.

Use Column Patterns Instead of Combining Flags to Handle XML with Mixed Centricity.

Combining flags may introduce a performance overhead, so you should specify a flag for either attributes or elements and use column patterns to retrieve non-default XML data mappings.

Review

Topic Objective

To reinforce module objectives by reviewing key points.

Lead-in

The review questions cover some of the key concepts taught in the module.

- **Creating a Rowset from an XML Document**
- **Specifying the Structure of a Rowset**

*******ILLEGAL FOR NON-TRAINER USE*******

1. You need to reduce the amount of memory used by your SQL Server application. How can you ensure efficient use of memory when using an **OPENXML** statement to process XML documents?

Execute the `sp_xml_removedocument` system stored procedure to free memory that is used by the internal tree representation.

2. In what ways can you define the structure of the rowset that is produced by an **OPENXML** statement?

By specifying an existing table name (if the XML structure matches) or a table schema in the **WITH clause.**

3. You need to retrieve data from an XML document with mixed attribute centricity and element centricity. When producing a rowset from an XML document, how can data from both attributes and child elements be retrieved?

By specifying a flags parameter of 3 (1 + 2), or by using column patterns.

Module 4: Using ADO to Access XML Data

Contents

Overview	1
Retrieving XML Data with ADO	2
Lab 4.1: Creating an ADO Client Application	16
Using ADO and XML on an ASP-Based Web Site	22
Lab 4.2: Using ADO on a Web Site	27
Best Practices	31
Review	32



Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BackOffice, BizTalk, MSDN, MS-DOS, SQL Server, Visual Basic, Visual C++, Visual InterDev, Visual J++, Visual Studio, Windows, Windows Media, Windows NT, and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Instructor Notes

Presentation:
60 Minutes

This module provides students with the information required to retrieve Extensible Markup Language (XML) data from Microsoft® SQL Server™ by using Microsoft ActiveX® Data Objects (ADO).

Lab:
30 Minutes

After completing this module, students will be able to:

- Construct an XML query template.
- Use a **Command** object and **Stream** objects to submit an XML query.
- Apply a style sheet to the results of an XML query that uses ADO.
- Return an XML query result to the Active Server Pages (ASP) **Response** object.
- Specify the output encoding of an XML query result.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need Microsoft PowerPoint® file 2091a_04.ppt.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the labs.

Instructor Setup for a Lab

This section provides setup instructions that are required to prepare the instructor computer or classroom configuration for a lab.

Lab 4.1: Creating an ADO Client Application

► **To prepare for the lab**

- Ensure that the students perform the setup procedure on the introductory page of the lab.

Module Strategy

Use the following strategies to present this module:

- Retrieving XML Data with ADO

Emphasize that the ability to save recordsets as XML is not specific to SQL Server—this approach can be used with any OLE DB provider to generate XML data from any source (for example, Microsoft Access or Microsoft Active Directory). However, if you want to be able to take advantage of native support for XML in SQL Server, you must use the SQL Server OLE DB Provider (SQLOLEDB Provider).

Use the slide “Submitting an XML Query with ADO” as a high-level introduction to the process. Then cover each step in detail on the appropriate slide.

- Using ADO and XML on an ASP-Based Web Site

Discourage students from writing ADO code directly in an ASP page. Emphasize that components are a much more scalable and flexible solution.

When building a Web site with ADO and XML, the most important property is the **XSL** property. Emphasize that you can use the **XSL** property to transform the retrieved XML into Hypertext Markup Language (HTML), Wireless Markup Language (WML), or any other presentation format required.

The use of the **SS Stream Flags** property is relatively specialized. Explain that you can use this property as a way to limit access to files in client applications where the user can specify the location of the style sheet to be used. Because this is not a common scenario, students will rarely use this property with ADO.

Emphasize that XML data islands can only be used with XML-aware browsers such as Microsoft Internet Explorer.

Overview

Topic Objective

To provide an overview of the module topics and objectives.

Lead-in

In this module, you will learn how ADO can be used to retrieve XML data from Microsoft SQL Server.

- Retrieving XML Data with ADO
- Using ADO and XML on an ASP-Based Web Site

*****ILLEGAL FOR NON-TRAINER USE*****

After completing this module, you will be able to:

- Construct an Extensible Markup Language (XML) query template.
- Use a **Command** object and **Stream** objects to submit an XML query.
- Apply a style sheet to the results of an XML query that uses Microsoft® ActiveX® Data Objects (ADO).
- Return an XML query result to the Active Server Page (ASP) **Response** object.
- Specify the **Output Encoding** property of an XML query result.

◆ Retrieving XML Data with ADO

Topic Objective

To introduce the topics in this section.

Lead-in

ADO is the preferred data access technology for many developers.

- Using ADO to Retrieve XML
- ADO Support for XML
- Persisting ADO Recordsets as XML
- Submitting an XML Query with ADO
- Using a Command Object
- Building XML Query Templates for ADO Commands
- Assigning a Query to a Command Object
- Creating a Result Stream
- Executing a Query

*****ILLEGAL FOR NON-TRAINER USE*****

Most developers working with the Microsoft Windows® operating system use ADO as the preferred data access technology for multiple diverse data sources. If you have already created a recordset, you can use ADO to transform it into XML. Alternatively, you can use Microsoft SQL Server™ to generate the XML, and then use ADO to enter queries and retrieve results.

The ADO 2.6 library supplied with SQL Server 2000 provides the **Connection**, **Command**, **Recordset**, **Record**, and **Stream** objects, and enables access to many different data sources by using OLE DB providers.

ADO provides native support for XML so that you can persist recordsets in XML format. You can also use the XML-related functionality of SQL Server with ADO when you use the SQL OLE DB Provider (SQLOLEDB).

This section discusses the use of ADO to transform recordsets into XML and retrieve data in XML format, and also describes the XML-related properties of the SQLOLEDB Provider.

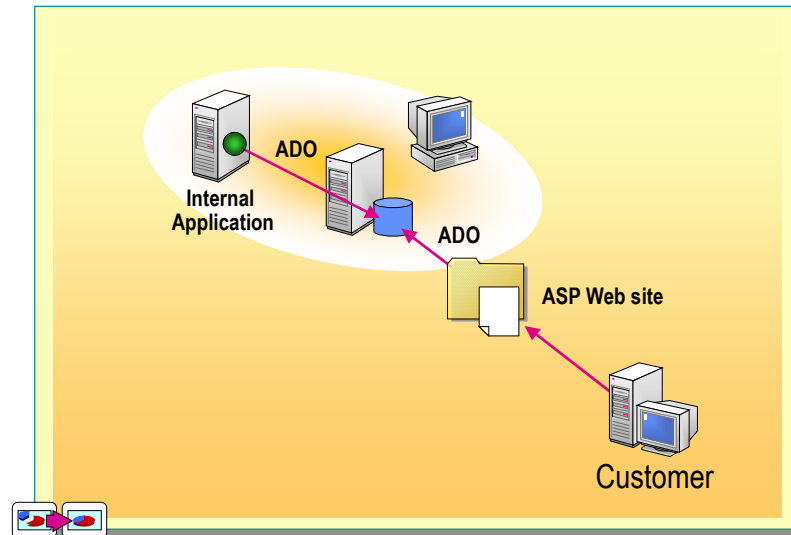
Using ADO to Retrieve Data

Topic Objective

To describe scenarios where ADO can be used to retrieve XML data.

Lead-in

ADO is the standard data access API for Windows.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

This slide builds automatically to show XML data being retrieved from an ASP Web site, and XML data being retrieved by a line-of-business application. To restart the build, press the LEFT ARROW key on the keyboard.

ADO is the standard data access API for Windows development. Most developers are familiar with the objects in the ADO object model.

Using ADO to Access XML on a Web Site

When you are developing an ASP-based Web application that needs to access XML data from SQL Server, consider using ADO. ADO 2.6 provides extensive support for accessing XML data, especially when used with the SQLOLEDB Provider. It can be more efficient to retrieve XML and transform it into Hypertext Markup Language (HTML) format by using a style sheet than to retrieve data as a recordset and iterate through the data in order to render it as HTML.

Using ADO to Access XML in a Line-of-Business Application

ADO is often used for data access in internal applications and COM components. If you use XML in a line-of-business (LOB) application, you can retrieve it from SQL Server by using ADO.

ADO Support for XML

Topic Objective

To describe the XML support in ADO 2.6.

Lead-in

The ADO library provides extensive support for XML.

- **Saving Recordsets as XML**
- **Support for Executable Streams**
- **SQLOLEDB Provider XML-Related Properties**

*******ILLEGAL FOR NON-TRAINER USE*******

The ADO library provides extensive support for XML. You can retrieve data from any data source into a recordset, and then persist the recordset as XML. Extensions to the SQLOLE DB Provider make it possible to retrieve XML data from SQL Server by using the FOR XML syntax.

Saving Recordsets as XML

An ADO recordset provides a programmable object that represents a specific set of data. You can open recordsets from any OLE DB data source or create them dynamically, and then persist them as XML. Consider the following advantages of saving recordsets as XML:

- You can represent data from any data source by using XML.
- You can open XML recordsets that you have persisted, and process them by using any ADO-enabled or XML-enabled application.

Support for Executable Streams

The ADO library provides **Stream** objects to allow ADO to transfer and manipulate stream data, such as the contents of a file. Additionally, you can use ADO to write a query into a **Stream** object and execute it by using a **Command** object. This is a particularly useful way of executing XML queries that are defined in a text file.

Note Streams are an efficient way to process large quantities of text data such as XML documents. An application can begin processing data read from a stream, even if the whole stream has not yet been read. Consequently, less memory needs to be set aside for the data in the stream.

SQLOLEDB XML-Related Properties

SQLOLEDB, the SQL Server native OLE DB provider, exposes a number of XML-related properties that you can use when executing XML queries. These properties enable you to apply style sheets to XML results, and to specify a different character encoding and other options.

Provider-specific properties are accessed through the Properties collection of ADO objects.

Persisting ADO Recordsets as XML

Topic Objective

To explain how recordsets can be saved as XML.

Lead-in

You can represent data from any data source as XML by persisting a recordset.

■ Persisting a Recordset in a Generic XML Format

```
objRS.Save "c:\products.xml", adPersistXML
```

```
<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
      xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
      xmlns:rs='urn:schemas-microsoft-com:rowset'
      xmlns:z='#RowsetSchema'>
  <s:Schema id='RowsetSchema'>
    ...
  </s:Schema>
  <rs:data>
    <z:row ProductID='1' ProductName='Chai' />
    <z:row ProductID='2' ProductName='Chang' />
  </rs:data>
</xml>
```

*****ILLEGAL FOR NON-TRAINER USE*****

You can represent data from any data source as XML by persisting an ADO recordset. You can then open and process the persisted recordset by using any XML-enabled or ADO-enabled application.

Note Persisting recordsets as XML is a function of the ADO library, and is not specific to SQL Server 2000.

Persisting a Recordset in a Generic XML Format

To persist an open recordset as XML, call the recordset's **Save** method and specify the **adPersistXML** constant for the options parameter. You can save the recordset to a file path or to the ASP **Response** object that sends the XML recordset directly to the browser. You can use this approach to transform non-SQL Server data into XML.

Consider the following features of the XML document produced by the **Save** method:

- The document contains a generic XML representation of a recordset. Each row in the recordset is represented as a row element, and each column is represented as an attribute with the column name.
- An inline XML Schema is used to define the elements and attributes. This includes data type and nullability metadata for each column.

The following example shows how to persist a recordset containing **custid** and **name** columns as XML.

Example

For Your Information

Code similar to this is saved as RSasXML.vbs in the folder DemoCode\Mod04. You can run this script if you want to demonstrate an ADO recordset being persisted as XML.

```
Set objRS = CreateObject("ADODB.Recordset")
sqlstr = "SELECT ProductID, ProductName "
sqlstr = sqlstr & "FROM Products"
objRS.Open sqlstr, CONN_STRING
objRS.Save "c:\Products.xml", adPersistXML
```

The XML document saved by this code contains an XML Schema that defines the **ProductID** and **ProductName** columns and the XML representation of the recordset, as shown in the following XML document:

```
<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
  xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
  xmlns:rs='urn:schemas-microsoft-com:rowset'
  xmlns:z='#RowsetSchema'>
<s:Schema id='RowsetSchema'>
  <s:ElementType name='row' content='eltOnly'
    rs:CommandTimeout='30'>
    <s:AttributeType name='ProductID' rs:number='1'>
      <s:datatype dt:type='int' dt:maxLength='4'
        rs:precision='10' rs:fixedlength='true'
        rs:maybenull='false'/>
    </s:AttributeType>
    <s:AttributeType name='ProductName' rs:number='2'
      rs:nullable='true'>
      <s:datatype dt:type='string'
        dt:maxLength='40'/>
    </s:AttributeType>
    <s:extends type='rs:rowbase'/>
  </s:ElementType>
</s:Schema>
<rs:data>
  <z:row ProductID='1' ProductName='Chai'/>
  <z:row ProductID='2' ProductName='Chang'/>
</rs:data>
</xml>
```

Submitting an XML Query with ADO

Topic Objective

To describe the process of submitting an XML query by using ADO.

Lead-in

The process of retrieving XML data by using ADO relies on the **Command** object.

1. **Create a Command object**
2. **Construct an XML query**
3. **Assign the query to the Command object**
4. **Assign an output stream to the Command object**
5. **Execute the command**

*****ILLEGAL FOR NON-TRAINER USE*****

The process of retrieving XML data from a SQL Server database by using ADO relies on the **Command** object. There are five steps involved in retrieving XML from SQL Server by using ADO:

Delivery Tip

Do not discuss the steps in this overview in too much detail. They are described in depth on the following slides.

1. Create a **Command** object.
Use the **Command** object to execute the query.
2. Construct an XML query.
Submit XML queries as XML documents called templates.
3. Assign the query to the **Command** object.
Assign the XML template to the **CommandText** or **CommandStream** property of the **Command** object.
4. Assign an output stream to the **Command** object.
The output stream is a **Stream** object that receives the query results.
5. Execute the command.
Call the **Execute** method on the **Command** object, and process the results.

Using a Command Object

Topic Objective

To describe how to create an ADO Command object for use with an XML query.

Lead-in

An ADO Command object is required for an XML query.

■ ActiveConnection property must reference a SQLOLEDB connection

- An open Connection object
 - or –
- A SQLOLEDB connection string

```
Dim objCmd As ADODB.Command
Set objCmd = CreateObject("ADODB.Command")
objCmd.ActiveConnection = "PROVIDER=sqloledb;
    ↳DATA SOURCE=deptserver1;INITIAL CATALOG=Northwind;
    ↳INTEGRATED SECURITY=SSPI;"
```

*****ILLEGAL FOR NON-TRAINER USE*****

For Your Information

The folder DemoCode\Mod04 contains two scripts—(ADOCmdText.vbs and ADOCmdStream.vbs)—that you may wish to refer to when discussing the following slides.

Before you can execute a FOR XML query that uses ADO, you must create the **Command** object and connect to the data source. To use the SQLOLEDB XML-related properties, you must assign the **ActiveConnection** property of the **Command** object to a valid OLE DB connection to SQL Server 2000.

Referencing a SQLOLEDB Connection

You can use an explicitly created **Connection** object to connect to the data source. You can also create a **Connection** object implicitly by assigning an OLE DB connection string to the **ActiveConnection** property. The following example shows how to create a **Command** object and use an implicit **Connection** object to connect to the sales database on the server deptserver1.

Example

```
Dim objCmd As ADODB.Command
Set objCmd = CreateObject("ADODB.Command")
objCmd.ActiveConnection = "PROVIDER=sqloledb;
    ↳DATA SOURCE=deptserver1;INITIAL CATALOG=Northwind;
    ↳INTEGRATED SECURITY=SSPI;"
```

Building XML Query Templates for ADO Commands

Topic Objective

To describe the syntax for an XML query template.

Lead-in

You submit XML queries in a query template document.

- An XML document references the XML-SQL namespace
- A template can include one or more queries
 - Can be a FOR XML statement or a stored procedure

```
<Products xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:query>
    SELECT ProductID, ProductName
    FROM Products Product
    FOR XML AUTO
  </sql:query>
</Products>
```

*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

The XML templates used with ADO are generally simple (for example, with no parameters). More complex templates that are designed for use over HTTP are discussed in a later module.

You submit XML queries to SQL Server in a query template document. The query template is an XML document that defines the root element for the resulting XML document and one or more queries.

Referencing the XML-SQL Namespace

The Microsoft XML-SQL schema defines a number of elements and attributes that you can include in a query template document. Use the XML-SQL namespace both for query templates that you submit by using ADO, and for XML template files that you access by using HTTP.

To reference the XML-SQL schema, you must specify a namespace declaration in the root element of the query template document. The reference must use the Uniform Resource Identifier (URI) `urn:schemas-microsoft-com:xml-sql`

Note The prefix for the XML-SQL namespace can be any valid XML prefix. The documentation supplied with SQL Server uses “sql” as the prefix in all code examples.

Including Queries in a Template

A template can contain one or more queries. You declare queries by using the query tag that is defined in the XML-SQL namespace. The queries can contain Transact-SQL FOR XML queries, or calls to stored procedures that return XML. The resulting XML fragment in the output document will replace each query element in the template.

For example, you could use the following template to retrieve a list of products.

Example

```
<Products xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:query>
    SELECT ProductID, ProductName
    FROM Products Product
    FOR XML AUTO
  </sql:query>
</Products>
```

Assigning a Query to a Command Object

Topic Objective

To discuss how query templates can be assigned to ADO Command objects.

Lead-in

There are two ways to assign a query template to a Command object.

■ Assign the query to the **CommandText** property

- Specify the MSSQLXML dialect

```
objCmd.Dialect = "{5D531CB2-E6Ed-11D2-B252-00C04F681B71}"
objCmd.CommandText = strQry
```

OR ■ Assign the query to the **CommandStream** property

- Write the query to a stream object

```
Set objStrmQry = CreateObject("ADODB.Stream")
With objStrmQry
    .Open
    .WriteText strQry, adWriteChar
    .Position = 0 'Move back to the start
End With
Set objCmd.CommandStream = objStrmQry
```

*****ILLEGAL FOR NON-TRAINER USE*****

There are two ways to assign a query template to a **Command** object. You can write a string to the **CommandText** property, or a stream to the **CommandStream** property.

Assigning the Query to the CommandText Property

You can assign a query template to the **CommandText** property of the **Command** object. Consider the following facts when using the **CommandText** property for XML queries:

- You must set the **Dialect** property of the **Command** object to the Globally Unique Identifier (GUID) representing the MSSQLXML format. This GUID is {5D531CB2-E6Ed-11D2-B252-00C04F681B71}.
- You must pass the query template as a Unicode encoded string.

Use the **CommandText** property approach for small query templates that have been built dynamically in code (and are therefore encoded as Unicode). The following example shows a query template being assigned to the **CommandText** property.

For Your Information

The SQLOLEDB Provider supports three dialects: SQL (the default, Transact-SQL), MSSQLXML (XML templates), and XPATH (XPath queries used with annotated schemas).

Example

```
strQry = _
"<Products xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
strQry = strQry & "<sql:query>"
strQry = strQry & "SELECT * FROM Products FOR XML AUTO"
strQry = strQry & "</sql:query></Products>"
objCmd.Dialect = "{5D531CB2-E6Ed-11D2-B252-00C04F681B71}"
objCmd.CommandText = strQry
```


Assigning the Query to the CommandStream Object

You can also write query templates to a stream and assign them to the **CommandStream** property of the **Command** object. Consider the following facts about using the **CommandStream** property for XML queries:

- The stream may contain a query template encoded by using any encoding supported by the XML parser.
- Streams are a more efficient way to handle large text values, especially when loaded from a file.
- You must reset the current position marker to the beginning of the stream before you assign the **CommandStream** property. This is because the OLE DB provider begins reading the query from the current position marker.

The **CommandStream** property approach requires slightly more coding effort than using the **CommandText** property. However, using the **CommandStream** property offers greater performance and flexibility for large query templates. This is because the data is streamed rather than written into a string variable in memory. The following example shows how to use the **CommandStream** property for an XML query.

Example

```
strQry = _
"<customers xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
strQry = strQry & "<sql:query>"
strQry = strQry & "SELECT * FROM Products FOR XML AUTO"
strQry = strQry & "</sql:query></Products>"
Set objStrmQry = CreateObject("ADODB.Stream")
With objStrmQry
    .Open
    .WriteText strQry, adWriteChar
    .Position = 0 'Move back to the start
End With
Set objCmd.CommandStream = objStrmQry
```

Creating a Result Stream

Topic Objective

To discuss the creation of an output stream.

Lead-in

The results of the query are returned as a stream.

- **Create a Stream object to receive the results**
 - Assign it to the **Output Stream** property

```
Set objStrmResults = CreateObject("ADODB.Stream")
objStrmResults.Open
objCmd.Properties("Output Stream") = objStrmResults
```

*****ILLEGAL FOR NON-TRAINER USE*****

XML query results are returned as a stream. This ensures efficient use of memory in the client application. An ADO **Stream** object receives the query results for processing by the client application.

Creating a Stream Object to Receive Query Results

Consider the following facts when using a **Stream** object to receive the results of an XML query:

- You must open the **Stream** object before assigning it to the **Output Stream** property.
- **Output Stream** is an OLE DB provider–specific property supported by the SQLOLEDB Provider. The **Command** object must have an **ActiveConnection** property defining a SQLOLEDB connection before the **Output Stream** property can be assigned.
- Although it is an object, the **Set** keyword is not used when assigning the **Output Stream** property.

The following example shows how to assign a **Stream** object to the **Output Stream** property.

Example

```
Dim objStrmResults As ADODB.Stream
Set objStrmResults = CreateObject("ADODB.Stream")
objStrmResults.Open
objCmd.Properties("Output Stream") = objStrmResults
```

Executing a Query

Topic Objective

To describe how to execute an XML query.

Lead-in

After all properties have been set, the query can be executed.

■ Execute the command with the **adExecuteStream** option

- Use 1024 in scripts with no ADO constants

```
objCmd.Execute, , adExecuteStream
```

■ Read the XML results from the output stream

```
Dim strXML As String
strXML = objStrmResults.ReadText
```

*****ILLEGAL FOR NON-TRAINER USE*****

After you have assigned the query template and created an output stream, you can execute the **Command** object.

Executing the Command

To receive the query results as a stream, you must call the **Execute** method of the **Command** object with the **adExecuteStream** option. When you use ADO in a script with no ADO constant declarations, use the value 1024.

The following example shows the **Execute** method of the **Command** object that is being called.

Example

```
objCmd.Execute, , adExecuteStream
```

When the **Command** object has been executed, the query results can be read from the **Output Stream** property and then processed by the client application.

Reading Query Results from an Output Stream

Use the **ReadText** method of the **Stream** object to read the contents of the output stream. This method returns a string.

The following example shows how to read the contents of a **Stream** object.

Example

```
Dim strXML as String
strXML = objStrmResults.ReadText
```

Lab 4.1: Creating an ADO Client Application

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will create a client application that uses ADO to retrieve XML from SQL Server.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Construct an XML query template.
- Use a **Command** object and **Stream** objects to submit an XML query.

Prerequisites

Before working on this lab, you must know how to:

- Create ADO automation objects in Microsoft Visual Basic® Scripting Edition (VBScript).
- Construct XML query templates.

Lab Setup

To complete this lab, you need the following:

- A computer running Windows 2000 Server and SQL Server 2000
- A virtual root named Lab04Web that contains the lab files

► To create the Lab04Web virtual root

1. In Windows Explorer, move to the folder *<install folder>\Labs\Lab04\Lab04Web* and view the contents.
2. Right-click the folder **Lab04Web**, and then click **Properties**.
3. On the **Web Sharing** tab, select **Share this folder**.
4. In the **Edit Alias** dialog box, accept the default settings, and then click **OK**.
5. Click **OK** to close the **Lab04Web Properties** dialog box.
6. Run *<install folder>\Labs\Lab04\LabSetupFiles\EnableAccess.bat* to enable anonymous access to the **Northwind** database.

For More Information

Read the section “ADO Support for SQL Server XML Features” in SQL Server Books Online.

Scenario

In this lab, you will create ADO-based data access solutions to retrieve data from the Northwind Traders database.

Estimated time to complete this lab: 15 minutes

Exercise 1

Retrieving XML Data Using the CommandText Property

In this exercise, you will write VBScript code to retrieve XML data from a SQL Server database. You will use the **CommandText** property of the **Command** object to submit a query template for execution.

Scenario

You need to create an Active Server Pages (ASP) page script that returns a list of categories from the Northwind Traders database as XML.

► To edit the CategoriesXML.asp page

The file Categories.xml in the folder *<install folder>\Labs\Lab04\Solution* contains a solution to this procedure.

1. View the contents of the folder *<install folder>\Labs\Lab04\Lab04Web\Starter*.
2. Right-click **CategoriesXML.asp**, and click **Open with Notepad**.
3. Examine the declarations in the script. These are the variables and constants that you will use in your solution.
4. Find the comment **Add your code here**. Under this comment, add code to perform the following tasks:
 - Set the **objCmd** variable to an instance of an **ADODB.Command** object, and assign the **strConn** constant to the **ActiveConnection** property.
 - Use the **strQry** variable to build a query template. The template should return an XML document with a root element named **<CategoryList>**. The document should include a query that returns an XML fragment containing the **CategoryID** and **CategoryName** fields from the **Categories** table in **AUTO** mode.

Note Because XML is case-sensitive, you should match the case specified above in your query.

- Assign the query template to the command by setting the **Dialect** and **CommandText** properties of the **objCmd** object.
- Set the **objStrmOut** variable to an instance of an **ADODB.Stream** object. Then open the stream and assign it to the **Output Stream** property of the **objCmd Command** object.

- Execute the command by using the **adExecuteStream** option, and read the results from the objStrmOut variable into the strResults variable. Your code should look similar to the following example:

```
'Create Command object and assign connection string
Set objCmd = Server.CreateObject("ADODB.Command")
objCmd.ActiveConnection = strConn

'Build Query Template
strQry = _
"<CategoryList xmlns:sql='urn:schemas-microsoft-com:xml-
  ↳sql'>"
strQry = strQry & "<sql:query>SELECT CategoryID,
  ↳CategoryName FROM Categories FOR XML AUTO</sql:query>"
strQry = strQry & "</CategoryList>"

'Assign query template to Command
objCmd.Dialect = DBGUID_MSSQLXML
objCmd.CommandText = strQry

'Assign the Output Stream
Set objStrmOut = Server.CreateObject("ADODB.Stream")
objStrmOut.Open
objCmd.Properties("Output Stream") = objStrmOut

'Execute the Command and retrieve the results
objCmd.Execute, , adExecuteStream
strResults = objStrmOut.ReadText
```

5. Save the ASP file and close Notepad.

Note The solution code for this procedure can be viewed in the file
<install folder>\Labs\Lab04\Lab04Web\Solution\CATEGORIESXML.asp.

► To test your solution

1. Open <http://localhost/Lab04Web/Starter/CategoriesXML.asp> with Microsoft Internet Explorer.
2. Verify that an XML document containing a list of categories is returned.
3. Close Internet Explorer.

Exercise 2

Retrieving XML Data Using the CommandStream Property

In this exercise, you will write VBScript code to retrieve XML data from a SQL Server database. You will use the **CommandStream** property of the **Command** object to submit a query template for execution.

Scenario

You need to create an ASP page script that will return a list of products from the Northwind Traders database as XML.

► To edit the ProductsXML.asp page

The file ProductsXML.asp in the folder *<install folder>\Labs\Lab04\Solution* contains a solution to this procedure.

1. View the contents of the folder **Lab04Web**.
2. Right-click **ProductsXML.asp**, and then click **Open with Notepad**.
3. Examine the declarations in the script. These are the variables and constants that you will use in your solution.
4. Find the comment **Add your code here**. Under this comment, add code to perform the following tasks:
 - Set the objCmd variable to an instance of an **ADODB.Command** object, and assign the strConn constant to the **ActiveConnection** property.
 - Use the strQry variable to build a query template. The template should return an XML document with a root element named <ProductList>. The document should include a query that returns an XML fragment containing the **ProductID**, **ProductName**, **UnitPrice**, and **UnitsInStock** fields from the **Products** table in **AUTO** mode.
 - Assign the query template to the command by setting the objStrmIn variable to an instance of an **ADODB.Stream** object and writing the **strQry** string to the opened stream. Remember to reset the current position marker to the beginning of the stream.
 - Set the objStrmOut variable to an instance of an **ADODB.Stream** object. Then open the stream and assign it to the **Output Stream** property of the **objCmd** object.

- Execute the **Command** object by using the **adExecuteStream** option, and read the results from the objStrmOut variable into the strResults variable.

Your code should look similar to the following example:

```
'Create Command object and assign connection string
Set objCmd = Server.CreateObject("ADODB.Command")
objCmd.ActiveConnection = strConn

'Build Query Template
strQry = _
"<ProductList xmlns:sql='urn:schemas-microsoft-com:xml-
  ↳sql'>"
strQry = strQry & "<sql:query>SELECT ProductID,
  ↳ProductName, UnitPrice, UnitsInStock FROM Products FOR
  ↳XML AUTO</sql:query>"
strQry = strQry & "</ProductList>"

'Assign query template to Command
Set strmIn = Server.CreateObject("ADODB.Stream")
With strmIn
    .Open
    .WriteText strQry ,adWriteChar
    .Position = 0
End With
Set objCmd.CommandStream = strmIn

'Assign the Output Stream
Set objStrmOut = Server.CreateObject("ADODB.Stream")
objStrmOut.Open
objCmd.Properties("Output Stream") = objStrmOut

'Execute the Command and retrieve the results
objCmd.Execute, , adExecuteStream
strResults = objStrmOut.ReadText
```

5. Save the ASP file and close Notepad.

Note The solution code for this procedure can be viewed in the file
 <install folder>\Labs\Lab04\Lab04Web\Solution\ProductsXML.asp.

► To test your solution

1. Open <http://localhost/Lab04Web/Starter/ProductsXML.asp> with Internet Explorer.
2. Verify that an XML document containing a list of products is returned.
3. Close Internet Explorer.

◆ Using ADO and XML on an ASP-Based Web Site

Topic Objective

To introduce the topics in this section.

Lead-in

ADO can be used to retrieve XML data for a Web Site.

- Applying an XSL Style Sheet
- Using the Output Encoding Property
- Using ADO and ASP to Publish Data on a Web Page

*****ILLEGAL FOR NON-TRAINER USE*****

A common way to abstract the data from its presentation is to use XML to represent the data. This is a particularly effective way to handle data on a Web site where the presentation of the data may depend on the type of client that is accessing the site. For example, a browser and a Wireless Application Protocol (WAP)-enabled cellular phone, which requires two different presentations of the same data, could both access a single site.

You can use ADO to retrieve XML data for a Web site, either in script on an ASP page or in a component called from the Web page. For maximum scalability and performance, encapsulate ADO data access code in components that are called by ASP scripts.

This section discusses how to use ADO and XML to build Web sites.

Applying an XSL Style Sheet

Topic Objective

To describe how to apply a style sheet when using ADO.

Lead-in

An XSL style sheet can be applied to the XML data that is retrieved.

- **Specify a style sheet with the XSL property**
- **Other properties used to control XSL access**
 - **Base Path**
 - **SS Stream Flags**

```
With objCmd
    .Properties("Base Path") = "c:\stylesheets"
    .Properties("XSL") = "products.xsl"
End With
```

*****ILLEGAL FOR NON-TRAINER USE*****

You can use an Extensible Stylesheet Language (XSL) style sheet to transform the XML that you have retrieved by using an ADO **Command** object. You can transform the XML into HTML, Wireless Markup Language (WML), another XML grammar, or any other text format by using an XSL style sheet.

Specifying a Style Sheet with the XSL Property

The SQLOLEDB Provider supports several properties that you can use to manage XSL transformations when you retrieve XML data. The main property used to apply a style sheet is the **XSL** property. By default, you can use the following types of values for the **XSL** property:

- *Relative File Path*. The relative file path of the XSL style sheet to be applied.
- *Absolute File Path*. The complete path to the XSL style sheet to be applied.
- *Relative URL*. The relative Uniform Resource Locator (URL) to the XSL style sheet to be applied.
- *Absolute URL*. The complete URL of the XSL style sheet to be applied.

The following example shows how to use Microsoft Visual Basic® code to apply a style sheet. You apply a different style sheet depending on the type of client.

Example

```
Select Case intFmt
    Case FMT_HTML
        objCmd.Properties("XSL") = "c:\stylesheets\products.xsl"
    Case FMT_WML
        objCmd.Properties("XSL") = "http://wapserver/xsl/prod.xsl"
End Select
```

Other Properties Used to Control XSL Access

In addition to the **XSL** property, you can use the **Base Path** and **SS Stream Flags** properties to control access to style sheet files.

The Base Path Property

Use the **Base Path** property to specify a base folder or URL to which all other references are relative. Specifying a base path allows a relative path to be used for the **XSL** property.

The following example shows how to assign a base path, which allows a relative path to be used for the **XSL** property.

Example

```
objCmd.Properties("Base Path") = "c:\stylesheets"
objCmd.Properties("XSL") = "Products.xsl"
```

Delivery Tip

Explain that annotated schemas are discussed in Module 6, "Using Mapping Schemas."

The SS Stream Flags Property

Use the **SS Stream Flags** property to restrict the possible paths used to access files, such as style sheets and annotated schemas. This restriction is useful in applications where a user can specify the path to the style sheet.

You can set the **SS Stream Flags** property to one of the following values:

- **STREAM_FLAGS_DISALLOW_URL** This constant represents the value 1 and prevents a URL from being used as an XSL property.
- **STREAM_FLAGS_DISALLOW_ABSOLUTE_PATH** This constant represents the value 2 and ensures that the XSL path must be relative to the base path.
- **STREAM_FLAGS_DISALLOW_QUERY** This constant represents the value 4 and prevents queries being submitted to SQL Server in a template.
- **STREAM_FLAGS_DONTCACHEMAPPINGSHEMA** This constant represents the value 8 and is used to prevent caching of annotated schemas.
- **STREAM_FLAGS_DONTCACHETEMPLATE** This constant represents the value 16 and is used to prevent the caching of query templates.
- **STREAM_FLAGS_DONTCACHEXSL** This constant represents the value 32 and is used to prevent the caching of style sheets.

The following example shows how to set the **SS Stream Flags** property.

Example

```
objCmd.Properties("SS STREAM FLAGS") = _
    STREAM_FLAGS_DISALLOW_URL
```

Using the Output Encoding Property

Topic Objective

To explain how the encoding of the result stream can be controlled.

Lead-in

You can set the encoding used in the result stream by using the Output Encoding property.

- Control the encoding used for the result stream

```
objCmd.Properties("Output Encoding") = "Windows-1250"
```

*****ILLEGAL FOR NON-TRAINER USE*****

You can control the encoding used in the result stream by using the **Output Encoding** property. The ability to specify how the data should be encoded is useful when the data includes special characters that must be rendered by using a particular encoding.

Controlling the Encoding Used in the Result Stream

You can encode the result stream by using any recognized encoding, such as:

- UTF-8
- Unicode
- ANSI
- Windows-1250
- ISO-8859-1

If no encoding is specified, the result stream is encoded as UTF-8. To specify an alternate encoding, set the **Output Encoding** property.

The following example shows how to set the **Output Encoding** property.

Example

```
objCmd.Properties("Output Encoding") = "Windows-1250"
```

Using ADO and ASP to Publish Data on a Web Page

Topic Objective

To describe some strategies for using ADO and ASP pages to build a Web page.

Lead-in

ADO can be used from an ASP page to generate a Web page.

■ Assign the Response object as the output stream

- Send results directly to the browser

```
<%
objCmd.Properties("XSL") =
Server.MapPath("products.xml")
objCmd.Properties("Output Stream") = Response
objCmd.Execute, , 1024
%>
```

■ Create an XML data island

```
<%
strXML = objStrmResults.ReadText
Response.Write "<XML id='productdata'>" & strXML & "</XML>"
%>
```

*****ILLEGAL FOR NON-TRAINER USE*****

You can use ADO and ASP pages together to build Web pages. You can implement the ADO data access code in the ASP page itself, or in a component.

Assigning the Response Object as the Output Stream

Assign the data retrieved from SQL Server directly to the **Response** object. This object is particularly useful when you apply a style sheet to the XML data to transform it into HTML, and no further processing is required.

The following example shows data being retrieved and sent directly to the browser as HTML.

Example

Delivery Tip

Explain that the `Server.MapPath` function returns the complete file path of the `products.xml` file in the same virtual root as the ASP page.

```
<%
objCmd.Properties("XSL") = Server.MapPath("products.xml")
objCmd.Properties("Output Stream") = Response
objCmd.Execute, , 1024
%>
```

Creating an XML Data Island

Use XML-aware browsers to process XML data on the client. A common technique is to store the data in an XML data island on the Web page, and bind data controls to the XML data by using client-side script.

You can create an XML data island by using an ASP page and writing an XML tag with an **id** attribute. You can populate the data island by writing the contents of the output stream. The following example shows how to use an ASP page and ADO to create a client-side XML data island.

Example

```
<%
strXML = objStrmResults.ReadText
Response.Write "<HTML><HEAD>"
Response.Write "<XML id='productdata'>" & strXML & "</XML>"
Response.Write "</HEAD><BODY>Product Catalog</BODY></HTML>"
%>
```

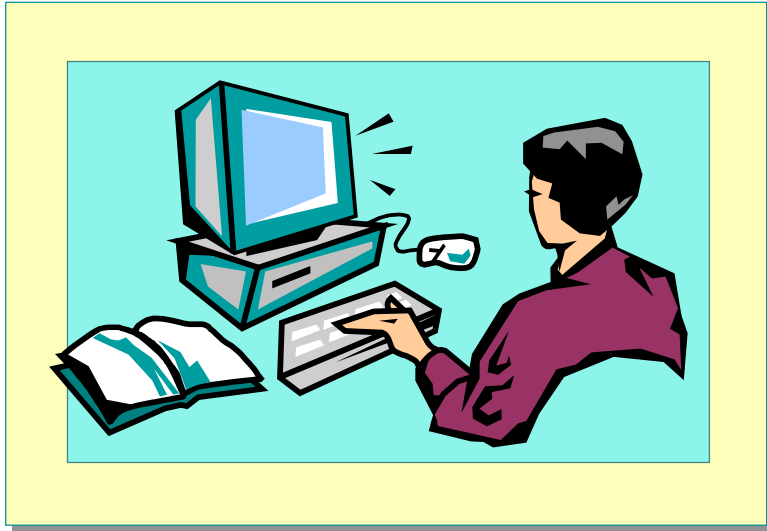
Lab 4.2: Using ADO on a Web Site

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will use ADO to retrieve XML data from SQL Server.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Apply a style sheet to the results of an XML query by using ADO.
- Return an XML query result to the ASP page **Response** object.
- Specify the output encoding of an XML query result.

Prerequisites

Before working on this lab, you must know how to retrieve XML from SQL Server by using ADO.

Lab Setup

To complete this lab, you need the following:

- A computer running Windows 2000 Server and SQL Server 2000
- A virtual root named Lab04 containing the lab files

Note If you did not complete Lab 4.1, complete the following procedure.

► To create the Lab04Web virtual root

1. In Windows Explorer, move to the folder *<install folder>\Labs\Lab04\Lab04Web* and view the contents.
2. Right-click the folder **Lab04Web**, and then click **Properties**.
3. On the **Web Sharing** tab, select **Share this folder**.
4. In the **Edit Alias** dialog box, accept the default settings, and then click **OK**.
5. Click **OK** to close the **Lab04Web Properties** dialog box.
6. Run *<install folder>\Labs\Lab04\LabSetupFiles\EnableAccess.bat* to enable anonymous access to the **Northwind** database.

For More Information

Read the section “ADO Support for SQL Server XML Features” in SQL Server Books Online.

Scenario

In this lab, you will create ADO-based data access solutions to retrieve data from the Northwind Traders database and publish it on an ASP Web site.

Estimated time to complete this lab: 15 minutes

Exercise 1

Applying a Style Sheet

In this exercise, you will retrieve data from the Northwind database and apply an XSL style sheet to transform the XML.

Scenario

You need to develop a Web page that retrieves the list of categories in the Northwind Traders database and displays the categories as an HTML table.

► To edit the CategoriesHTML.asp page

The file CategoriesHTML.asp in the folder *<install folder>\Labs\Lab04\Lab04Web\Solution* contains a solution to this procedure.

1. Right-click **CategoriesHTML.asp** in the folder Lab04Web\Starter, and then click **Open With....**
2. In the list of applications, select **Notepad**.
3. Examine the declarations in the script. These are the variables and constants that you will use in your solution.
4. Find the comment **Add your code here**. Under this comment, add code to perform the following tasks:
 - a. Assign the style sheet to be used by setting the **XSL** property of the **objCmd** Command object to **Server.MapPath("Categories.xml")**.
 - b. Execute the Command by using the **adExecuteStream** option, and read the results from the objStrmOut variable into the strResults variable.

Your code should look similar to the following example:

```
'Specify the Style Sheet
objCmd.Properties("XSL") = Server.MapPath("Categories.xml")

'Execute the Command and retrieve the results
objCmd.Execute, , adExecuteStream
strResults = objStrmOut.ReadText
```

5. Save the ASP file, and close Notepad.

► To test your solution

1. Open <http://localhost/Lab04Web/Starter/CategoriesHTML.asp> with Internet Explorer.
2. Verify that a list of categories is returned.
3. Close Internet Explorer.

Exercise 2

Using the Response Stream as the Output Stream

In this exercise, you will retrieve data from the Northwind database and apply a style sheet to it. The resulting data will be encoded as Windows-1250 and sent straight to the browser through the **Response** object.

Scenario

You need to develop a Web page that retrieves the list of products in the Northwind database and displays them as an HTML table.

► To edit the ProductsHTML.asp page

The file ProductsHTML.asp in the folder *<install folder>\Labs\Lab04\Lab04Web\Solution* contains a solution to this procedure.

1. Right-click **ProductsHTML.asp** in the folder Lab04Web\Starter, and then click **Open With...**
2. In the list of applications, select **Notepad**.
3. Examine the declarations in the script. These are the variables and constants that you will use in your solution.
4. Find the comment **Add your code here**. Under this comment, add code to perform the following tasks:
 - Assign the **Response** object to the **Output Stream** property.
 - Assign **"Windows-1250"** to the **Output Encoding** property.
 - Assign the style sheet to be used by setting the **XSL** property of the **objCmd** Command object to **Server.MapPath("Products.xml")**.
 - Execute the Command by using the **adExecuteStream** option.

Your code should look similar to the following example:

```
'Assign the Output Stream
objCmd.Properties("Output Stream") = Response

'Assign the Output Encoding
objCmd.Properties("Output Encoding")="Windows-1250"

'Specify the Style Sheet
objCmd.Properties("XSL") =
Server.MapPath("Products.xml")

'Execute the Command
objCmd.Execute, , adExecuteStream
```

5. Save the ASP file and close Notepad.

► To test your solution

1. Open <http://localhost/Lab04Web/Starter/ProductsHTML.asp> with Internet Explorer
2. Verify that an XML document containing a list of products is returned.
3. Close Internet Explorer.

Best Practices

Topic Objective

To identify best practices for using ADO to retrieve XML data.

Lead-in

Keep in mind the following best practices when using ADO to retrieve XML data.

- Use the SQLOLEDB Provider to Connect to SQL Server
- Use the CommandStream Property for Large Queries
- Use Components for Data Access Code
- Use XSL Style Sheets for Presentation Logic

*****ILLEGAL FOR NON-TRAINER USE*****

Follow these practices when using ADO to retrieve XML data.

Use the SQLOLEDB Provider to Connect to SQL Server

Most of the XML-related functionality provided by SQL Server is specific to the SQLOLEDB Provider. Applications that were built by using the ODBC provider for SQL Server will not take advantage of XML-related properties.

Use the CommandStream Property for Large Queries

Use a **Stream** object for large query templates because it is more efficient than assigning a string value to the **CommandText** property. This is especially true if the template must be read from a file.

Additionally, the **CommandStream** property supports any encoding understood by the XML parser. In contrast, the **CommandText** property supports Unicode only.

Use Components for Data Access Code

Encapsulate ADO code in a component to access data instead of writing the code in an ASP page script. Using components allows greater performance and scalability, and allows the same data access logic to be reused by multiple client applications.

For example, you could use a component to retrieve XML data and apply a specific style sheet, depending on a parameter passed to the component method. This component could then be used from both a Web page for HTML clients and another Web page for WML clients.

Use XSL Style Sheets for Presentation Logic

Use a style sheet to abstract the data access logic from the presentation logic. You can change the presentation by editing the style sheet.

Review

Topic Objective

To reinforce module objectives by reviewing key points.

Lead-in

The review questions cover some of the key concepts taught in the module.

- Retrieving XML Data with ADO
- Using ADO and XML on an ASP-Based Web Site

*****ILLEGAL FOR NON-TRAINER USE*****

1. You need to render data retrieved from a Microsoft Access database as XML. How can you use ADO to do this?

Open an ADO recordset that contains the required data, and save it as XML.

2. You want to retrieve three distinct XML fragments from a SQL Server database. What is the minimum number of query templates you must create and execute?

One. Query templates can contain multiple queries.

3. You are building an application that executes a query template loaded from a file. How should you assign the template to the **Command** object?

Read the file into a stream and assign it to the CommandStream property of the Command object.

4. You need to retrieve an XML stream in a Unicode encoding. What provider-specific property must you set?

Output Encoding.

5. You are creating an ASP page script to retrieve data from SQL Server. An XSL style sheet is applied to the data so that an HTML stream is returned. What is the most efficient way to send the resulting HTML to the browser?

Specify the Response object as the Output Stream.

Module 5: Using HTTP to Access XML Data

Contents

Overview	1
Enabling HTTP Access to SQL Databases	2
Publishing XML Templates on the Server	14
Lab 5.1: Publishing a SQL Server Database	23
Posting Query Templates from the Client	30
Publishing a Database Object	37
Lab 5.2: Accessing a Database Object	42
Best Practices	45
Review	47



Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BackOffice, BizTalk, MSDN, MS-DOS, SQL Server, Visual Basic, Visual C++, Visual InterDev, Visual J++, Visual Studio, Windows, Windows Media, Windows NT, and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Instructor Notes

Presentation:
90 Minutes

This module provides students with the knowledge necessary to publish and access Extensible Markup Language (XML) data in a Microsoft® SQL Server™ database over the Hypertext Transport Protocol (HTTP).

Labs:
90 Minutes

After completing this module, students will be able to:

- Create an IIS virtual directory that references a SQL Server 2000 database.
- Test functionality of a SQL Server 2000 database through HTTP.
- Create an XML template to retrieve XML data over HTTP.
- Create a client that uses the HTTP **POST** method to publish templates to SQL Server 2000.
- Use HTTP to enable access to database objects.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need Microsoft PowerPoint® file 2091a_05.ppt.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the labs.
- Practice the demonstrations.

Module Strategy

Use the following strategies to present this module:

- Enabling HTTP Access to SQL Databases

Emphasize the importance of configuring an appropriate security model, especially when publishing a database on the Internet.

Although Uniform Resource Locator (URL) queries are a powerful way to retrieve data from SQL Server, emphasize that they should not be allowed in most production environments because of the security issues they raise.

- Publishing XML Templates on the Server

Point out that the template files published over HTTP on the server are the same as the templates submitted by using the Microsoft ActiveX® Data Objects (ADO) **Command** object.

Emphasize that in most Web-based applications, Extensible Stylesheet Language (XSL) style sheets should be processed on the server.

- Posting Query Templates from the Client

Again, students should carefully consider the security implications of using this approach in a production environment.

- Publishing a Database Object

The primary use of this technique is to retrieve image data for display on a Web page. Emphasize that naming the dboject virtual name “dbobject” makes it easier to retrieve image URLs from AUTO mode queries.

Overview

Topic Objective

To provide an overview of the module topics and objectives.

Lead-in

In this module, you will learn how SQL Server enables you to use the Hypertext Transfer Protocol to publish databases.

- Enabling HTTP Access to SQL Databases
- Publishing XML Templates on the Server
- Posting Query Templates from the Client
- Publishing a Database Object

*****ILLEGAL FOR NON-TRAINER USE*****

You can build an integrated solution that uses a Hypertext Transport Protocol (HTTP) connection to Microsoft® Internet Information Services (IIS) in order to access data stored in a Microsoft SQL Server™ 2000 database.

After completing this module, you will be able to:

- Create an IIS virtual directory that references a SQL Server 2000 database.
- Test connectivity to a SQL Server 2000 database via HTTP.
- Create an Extensible Markup Language (XML) template to retrieve XML data over HTTP.
- Create a client application that uses the HTTP **POST** method to publish templates to SQL Server 2000.
- Use HTTP to enable access to database objects.

Note The terms “virtual directory” and “virtual root” both refer to an IIS virtual directory.

◆ Enabling HTTP Access to SQL Databases

Topic Objective

To introduce the topics in this section.

Lead-in

In this section, you will see how to use an IIS virtual directory publish a SQL Server database.

- Publishing SQL Server Data over HTTP
- SQL Server 2000 Virtual Directory Architecture
- Creating a Virtual Directory for SQL Server
- Securing a SQL Server Virtual Directory
- Demonstration: Creating a Virtual Directory for SQL Server
- Testing HTTP Connectivity to SQL Server
- Encoding URL Characters

*****ILLEGAL FOR NON-TRAINER USE*****

The easiest way to publish SQL Server 2000 databases over HTTP is with Microsoft Internet Information Services. In this section, you will learn how to create a secure IIS virtual directory for your SQL Server 2000 data sources. You will also learn how to use a Uniform Resource Locator (URL) query to test a virtual directory.

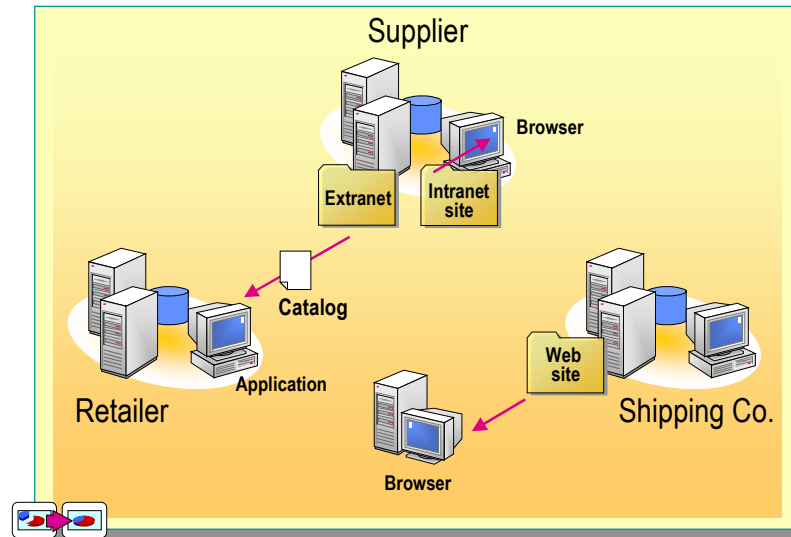
Publishing SQL Server Data over HTTP

Topic Objective

To describe scenarios where data can be published in a virtual directory and accessed by using HTTP.

Lead-in

The ability to publish data in a SQL Server database through a virtual directory for access over HTTP can be useful in a number of scenarios.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

This slide builds automatically to show data being published on an intranet site, an extranet site, and a Web site. To restart the build, press the LEFT ARROW key on the keyboard.

Publishing SQL Server data by using a virtual directory is useful in the following data access scenarios:

■ Intranet applications

In most business organizations, users need to access data in a corporate database. By publishing a SQL Server database through an IIS virtual directory, users can access the data by using a browser such as Microsoft Internet Explorer. This reduces the need to build and deploy a custom data access client application. For example, a supplier might make customer order data available to warehouse employees through an intranet site that publishes data from a SQL Server database.

■ Extranet applications

In a business-to-business e-commerce scenario, an organization might need to access data from another organization programmatically over the Internet. By publishing the data in a virtual directory, the data can be retrieved by a client application that is capable of making an HTTP request. For example, a retailer might download catalog information from a supplier through an HTTP connection over the Internet.

- Web sites

Many Web sites are used primarily to publish data. Allowing access to a SQL Server database through a virtual directory makes it possible to publish data on a Web site with minimal development effort. For example, a shipping company could make parcel shipping status information available to customers over the Internet.

Note Publishing a database through a virtual directory requires less coding effort than building a complete Active Server Pages (ASP)–based site. However, if your Web application must include complex business logic, such as a custom authentication model, it may be more sensible to build an ASP-based solution that accesses the data by using Microsoft ActiveX® Data Objects (ADO).

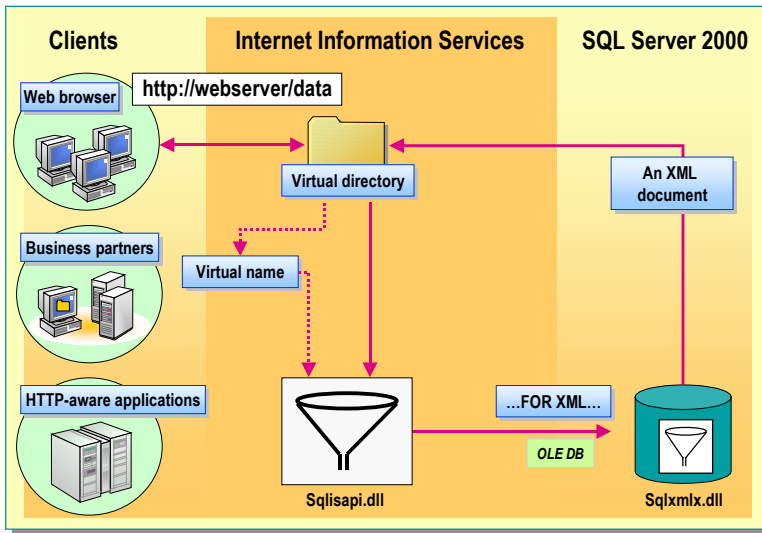
SQL Server 2000 Virtual Directory Architecture

Topic Objective

To review the functional components of a solution that uses HTTP, SQL, and IIS.

Lead-in

The main reason to use HTTP is to integrate clients whose only common connection method is HTTP. IIS works with SQL Server to accomplish this goal.



*****ILLEGAL FOR NON-TRAINER USE*****

SQL Server 2000 databases can be published in an IIS virtual directory.

Publishing Data in a Virtual Directory

Microsoft Windows® 2000 and Windows NT® 4.0 provide IIS to allow you to create a Web site that publishes information in a folder, making it accessible through an HTTP request. The folder containing a Web application is known as a *virtual directory* (or sometimes a *virtual root*).

Virtual directories are accessed over HTTP by specifying a URL of the form *http://servername/virtualdirectoryname*. The *servername* portion of the URL can be simply the name of a server running IIS (for example, Webserver), or a fully qualified domain name of an Internet host.

Using the SQLISAPI DLL

Most virtual directories are used to publish HTML pages, or dynamic ASP-based applications. However, a virtual directory can also be used to publish an Internet Services Application Programming Interface (ISAPI) application. This is a dynamic-link library (DLL) that routes any HTTP requests to the appropriate virtual directory.

SQL Server 2000 provides an ISAPI application (SQLISAPI) that is implemented in a file called SQLISAPI.dll. This application intercepts requests for data and retrieves it from a specified SQL Server database by using an OLE DB connection. Requests for XML data are processed on the SQL Server by the sqlxml.dll component, which retrieves the data and sends the XML results back to the SQLISAPI application. The XML data is then returned to the client over HTTP.

Using Virtual Names

Although it is possible to allow access to data through SQL queries submitted directly to the virtual directory, it is more common to create predefined query templates or schemas that encapsulate data access logic. These are published in a *virtual name*. A virtual name is conceptually a subfolder of the virtual directory, although it does not necessarily need to represent a physical folder (and even if it does, the folder can be in a different physical path from the virtual directory). A virtual name can be one of the following three types:

- *Template*. A folder in which XML template files are published.
- *Schema*. A folder in which mapping schemas are published.
- *DBObject*. A virtual name used to access binary objects in the database.

Using the Configure SQL XML Support in IIS Tool

SQLISAPI-based virtual directories and virtual names are created by using the Configure SQL XML Support in IIS tool. This tool is implemented as a Microsoft Management Console (MMC) snap-in and can be used to manage all aspects of SQLISAPI virtual directory management.

Creating a Virtual Directory for SQL Server

Topic Objective

To explain the decisions you need to make when you create a virtual directory for a SQL Server database.

Lead-in

SQL Server virtual directories are IIS applications that use the ISAPI DLL Extension provided with SQL Server.

1. Choose a name and a physical path
2. Connect to a SQL Server database
3. Configure security
4. Choose an HTTP request option
5. Create a virtual name for any template file, mapping schema, or database object
6. Specify any connection string parameters

*****ILLEGAL FOR NON-TRAINER USE*****

To enable IIS to communicate directly with SQL Server, you first need to create a virtual directory by using the Configure SQL XML Support in IIS tool. This utility instructs IIS to create an association between the new virtual directory and a SQL Server database.

Creating a SQL Server Virtual Directory

You need to make the following decisions when creating a virtual directory:

1. Choose a name and a physical path for the virtual directory.
2. Connect the virtual directory to a specific database on a SQL Server.
3. Configure security for the virtual directory.
4. Specify the kinds of requests that will be allowed to this virtual directory.
5. If you want to allow URLs to reference a template file, mapping schema, or database object, create a virtual name for any of these you plan to use.
6. When necessary, specify any additional connection string parameters and caching options.

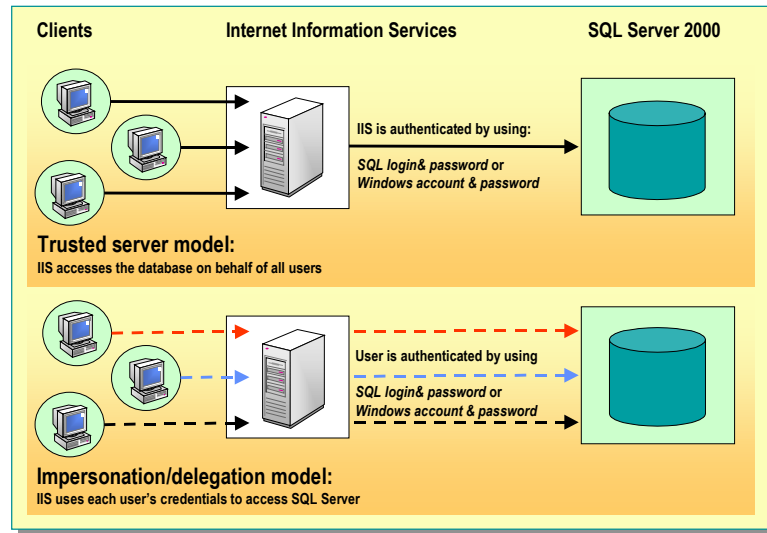
Securing a SQL Server Virtual Directory

Topic Objective

To describe how to secure a SQL Server virtual directory.

Lead-in

There are two general security models for distributed applications: the trusted server model and the impersonation/delegation model.



*****ILLEGAL FOR NON-TRAINER USE*****

There are two general security models for distributed applications such as a Web application. These are often referred to as the *trusted server* model and the *impersonation/delegation* model. In the trusted server model, the user account belonging to the IIS server is used to access the database, and the end users are considered anonymous. In the impersonation/delegation model, IIS passes the user's credentials (in other words, it *impersonates* the user) when making a request to SQL Server.

Using the Trusted Server Model

To configure a SQL Server virtual directory to use the trusted server model, on the **Security** tab, select **Always log on as:**, and then provide the username and password for a SQL Server or Windows login. All data requests, regardless of the end user, will be made by using the user credentials specified here. You must ensure that the account you specify has sufficient permissions to access data from the SQL Server database.

Using the Impersonation/Delegation Model

In the impersonation/delegation model, each user provides separate security credentials to IIS. IIS then accesses SQL Server by using the security credentials provided by the user.

You can configure your virtual directory to require Windows Integrated Authentication, in which case the user must provide Windows network login credentials, or Basic Authentication (Clear Text) to a SQL Server Account, in which case the user must provide a SQL Server login name and password.

Note The term "impersonation" is used when SQL Server is running on the same computer as IIS. The term "delegation" is used when IIS and SQL Server are running on separate computers. Windows NT 4.0 does not support delegation of Windows accounts.

Choosing an Authentication Method

Whether you choose the trusted server or impersonation/delegation security model, you still must decide how SQL Server will authenticate user requests. SQL Server can authenticate users based on a Windows user account, or a SQL Server login.

Using a Windows Account for SQL Server Authentication

By default, SQL Server 2000 authenticates user requests based on the user's Windows account. If the user (or a group that the user has membership in) has been granted access to the database, the request succeeds and the data is returned; otherwise the request fails and an error is returned.

When using the trusted server model in a SQL Server virtual directory, the Windows account specified in the Security tab of the **New Virtual Directory Properties** dialog box is used to access SQL Server. If this user account has been granted appropriate permissions, the data is returned to IIS and then sent to the client that made the original request.

When using the impersonation/delegation model, the user must provide a valid Windows account name and password for access to SQL Server. If the user is already logged on to a Windows network, his or her credentials are forwarded with the request; otherwise the user is prompted to enter a Windows username and password. The user must access the virtual directory by using an application that supports Windows authentication (such as Internet Explorer), so that the credentials can be passed securely over the network.

Using a SQL Server Login

You can configure SQL Server to allow users to access the server by using a SQL Server login account. This is a user account maintained in the SQL Server master database, and is separate from the Windows security system. Generally, you should use this configuration only for backward compatibility, as it allows users to bypass Windows security.

When using a SQL Server login with the trusted server model, the SQL Server login name and password specified in the Security tab of the **New Virtual Directory Properties** dialog box is used to access SQL Server for every user request.

When using a SQL Server login with the impersonation/delegation model, the user is prompted to enter a SQL Server username and password. This information is then sent unencrypted across the network to IIS, and passed to SQL Server for authentication. The lack of encryption makes this an unsuitable approach for Internet-based applications.

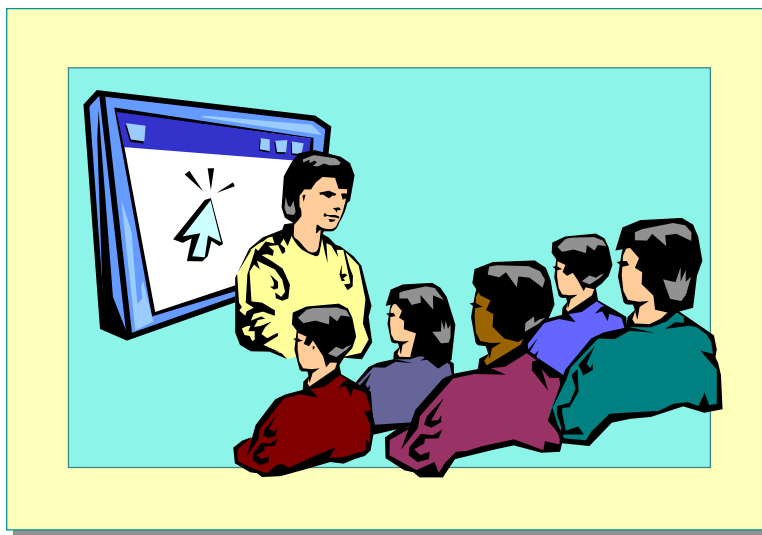
Demonstration: Creating a Virtual Directory for SQL Server

Topic Objective

To demonstrate how to create a virtual directory for SQL Server.

Lead-in

In this demonstration, you will see how to create a virtual directory for SQL Server.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

1. Start the Configure SQL XML Support in IIS tool in MMC.
2. Expand your server and select Default Web Site.
3. Click Action, point to New, and then click Virtual Directory.
4. On the General tab, name the virtual directory "Northwind", and then set the path to the My Documents folder.
5. Click Security, and then click Use Windows Integrated Authentication.
6. Click Data Source, and select the Northwind database on the (local) server.
7. Click Settings, and then select Allow sql=...or template= ... URL queries. Point out to students that Allow template queries is already selected.
8. Show the remaining tabs without changing them, and then click OK.
9. Close the MMC window and test the virtual directory by using the URL provided in the student notes on this page.

In this demonstration, you will see how to use the SQL XML Support in IIS tool to create a virtual directory that publishes Northwind Traders, a SQL Server sample database.

You will then see how you can test IIS connectivity to the database by opening the following URL in Internet Explorer:

`http://localhost/northwind?sql=SELECT * FROM products FOR XML AUTO&root=catalog`

Testing HTTP Connectivity to SQL Server

Topic Objective

To describe how to create a URL query to SQL Server.

Lead-in

The easiest way to test access to SQL Server over HTTP is to embed a Transact-SQL query in a URL.

■ Embedding SQL queries in the URL

- Enable the virtual directory to accept URL queries
- Include the sql parameter within the URL and specify the XML root element

■ Specifying the root element that SQL will return

- Explicitly concatenate the query with literal tags

```
http://webserver/data?sql=SELECT
'<catalog>';SELECT productname,unitprice FROM products
FOR XML AUTO;SELECT '</catalog>'
```

- Or, specify the XML root parameter

```
http://webserver/data?
sql=SELECT productname, unitprice FROM products
FOR XML AUTO;&root=catalog
```

*****ILLEGAL FOR NON-TRAINER USE*****

The easiest way to test access to SQL Server over HTTP is to embed a Transact-SQL query in a URL that references a virtual directory. To enable IIS to accept Transact-SQL queries embedded in URLs, you must select the **Allow URL queries** option in the Configure SQL XML Support in IIS tool.

In production systems, the option to permit URL queries is usually disabled because it presents a potential security risk for SQL Server. This is because the SQL Administrator must apply a strict security model so that dangerous or unauthorized queries will not be executed against the database. For production systems, it is better to use a template file to run a specific query than to allow URL queries.

Embedding SQL Queries in the URL

You can embed a SQL query in a URL in two ways. First, the URL can contain any valid Transact-SQL statement. Typically, these queries will make use of a **SELECT...FOR XML** statement. Second, the URL may contain a call to a stored procedure to SQL Server or Open Database Connectivity (ODBC). In either case, you must precede the query with the **sql** parameter within the URL.

In the following example, a URL contains a **SELECT...FOR XML** statement:

Example

```
http://localhost/northwind?sql=SELECT * FROM products FOR XML AUTO&root=catalog
```

In the next example, a URL calls a stored procedure by using a Transact-SQL statement or an ODBC Call syntax:

Example

```
http://localhost/northwind?sql=EXEC CustOrdersOrders 'ALKFI'
```

```
http://localhost/northwind?sql={Call CustOrdersOrders } 'ALKFI'
```

Specifying that the XML Result Is Well-Formed

To ensure that SQL Server returns a well-formed XML document, you specify an XML root element in the URL. You can specify the XML root element in one of two ways:

1. Explicitly concatenate the literal opening and closing tags to the SQL query.

The following URL could be used to retrieve product name and price data within a catalog root element in a document in the data virtual directory of a server named “webserver”:

```
http://webserver/data?sql=SELECT '<catalog>';SELECT productname, unitprice FROM products FOR XML AUTO;SELECT '</catalog>'
```

2. Use the **root** parameter when executing the query.

The following URL retrieves the same data as in the previous example. Because **root** is the second parameter in the URL, it is prefixed with an ampersand (&):

```
http://webserver/data?sql=SELECT productid, unitprice FROM products FOR XML AUTO&root=catalog
```

Encoding URL Characters

Topic Objective

To describe how to handle special characters in a URL query.

Lead-in

Some characters have special meaning in a URL.

■ Using Transact-SQL Characters in a URL

Original T-SQL that looks like this...

```
SELECT * FROM Orders WHERE CustomerID LIKE 'G%' FOR XML AUTO
```

...must be encoded like this when it appears within a URL:

```
?sql=
SELECT+*+FROM+Orders+WHERE+CustomerID+LIKE+'G%25'+
FOR+XML+AUTO
&root=myresults
```

*****ILLEGAL FOR NON-TRAINER USE*****

Some Transact-SQL characters are not compatible with HTTP and must be encoded when they are embedded in a URL.

Using Transact-SQL Characters in a URL

The following table shows characters that have special meaning in a URL, as well as the appropriate encoding for these characters.

Transact-SQL character	Meaning in Transact-SQL	Encode in a URL as:
+	Space	%20
/	Directory separator	%2F
?	First parameter separator	%3F
%	Encoding escape character	%25
#	Bookmark indicator	%23
&	Additional parameter separator	%26

Note Spaces are not permitted in a URL and should be indicated by using the “+” symbol or encoded as “%20”. Internet Explorer automatically converts spaces in a URL to %20.

◆ Publishing XML Templates on the Server

Topic Objective

To introduce the topics in this section.

Lead-in

XML templates provide a way to publish data without providing direct access to the database.

- Creating XML Template Files
- Publishing a Template Virtual Name
- Demonstration: Publishing an XML Template
- Using Parameters in Template Files
- Applying a Style Sheet to a Template File

*****ILLEGAL FOR NON-TRAINER USE*****

In a production environment, one of the main problems with URL queries is that the database schema is exposed to the client application. To avoid this, you can create XML templates on the server that retrieve specific data and return it as XML or HTML to the client. In this solution, the client only receives the query results, and the developer has much greater control over the data that is published.

This section discusses the creation and use of XML template files.

Creating XML Template Files

Topic Objective

To describe how an XML template file is created.

Lead-in

A template file consists of a valid XML document that contains one or more SQL statements or XPath queries.

■ Compared to URL queries, template files are:

- More secure and easier to maintain because they are stored on the server
- Flexible because they can accept parameters

■ A template file consists of a valid XML document that contains one or more SQL statements or XPath queries

```
<?xml version="1.0"?>
<catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:query>
    SELECT productid, productname, unitprice
    FROM products product
    FOR XML AUTO
  </sql:query>
</catalog>
```

*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

Point out that the XML documents in template files are the same as the XML templates submitted by using ADO. The main difference is that template files are typically stored on the server instead of being defined on the client.

A template file is a valid XML document stored on the Web server that contains one or more SQL statements and XPath queries. It is better to store your queries in template files rather than encode them in URLs, primarily because it is easier to maintain and secure queries when you store them on the server. Another good reason to use template files is that you can include one or more query parameters in the URL that calls the template file, thus increasing the template file's flexibility.

Creating an XML Template File

You can create an XML template by using Microsoft Notepad or any other text editor.

In the XML document, you represent each query in a separate <query> element. Each <query> element may contain either a FOR XML query or a call to a stored procedure.

When an HTTP client requests a URL that references a template file, SQL Server processes each of the queries in the template file and returns the results as XML fragments. The resulting XML document is then sent to the client.

The following text could be saved as a template file named catalog.xml and used to retrieve product information:

Example

```
<?xml version="1.0"?>
<catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:query>
    SELECT productid, productname, unitprice
    FROM products product
    FOR XML AUTO
  </sql:query>
</catalog>
```

Publishing a Template Virtual Name

Topic Objective

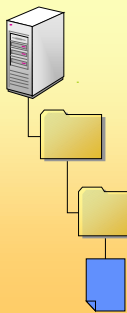
To describe how template virtual names are created and used.

Lead-in

Template files are saved in a folder that is configured as a virtual name.

■ Assign virtual names to folders containing template files

■ Users include the virtual name in a URL



	<u>Object</u>	<u>Physical path</u>	<u>Name</u>
	Web server		webserver
	Virtual directory	C:\SQLSite	data
	Virtual name	C:\Templates	templates
	Template file	C:\Templates\catalog.xml	catalog.xml

```
http://webserver/data/templates/catalog.xml
```

*****ILLEGAL FOR NON-TRAINER USE*****

Template files can be saved in any folder on the Web server. They do not need to be in the same directory tree as the virtual directory. To allow a template file to be accessed via a URL, the folder that contains the template file must be configured in IIS as a virtual name of the Template type.

Configuring a Template Virtual Name

You use the Configure SQL XML Support tool in IIS to assign a virtual name to a folder that contains a template file. Consider the following facts and guidelines when assigning a virtual name to the folder that contains a template file:

- Each virtual name must be unique within the virtual directory application.
- A virtual name may contain a hierarchy of subfolders.
- You must assign appropriate file system permissions to the folder referenced by the virtual name, and to its subfolders and files, so that the template files can be read. The account that needs to be granted permissions will depend on the security model that you are using on the virtual directory.

Specifying a Virtual Name in a URL

A URL that uses a template file follows this syntax:

```
http://domain/virtualdirectory/virtualname/[subfolder/..]template
```

In this example, the URL accesses a template named catalog.xml in a virtual name called templates, located in a virtual directory named data on a Web server named webserver:

Example

```
http://webserver/data/templates/catalog.xml
```

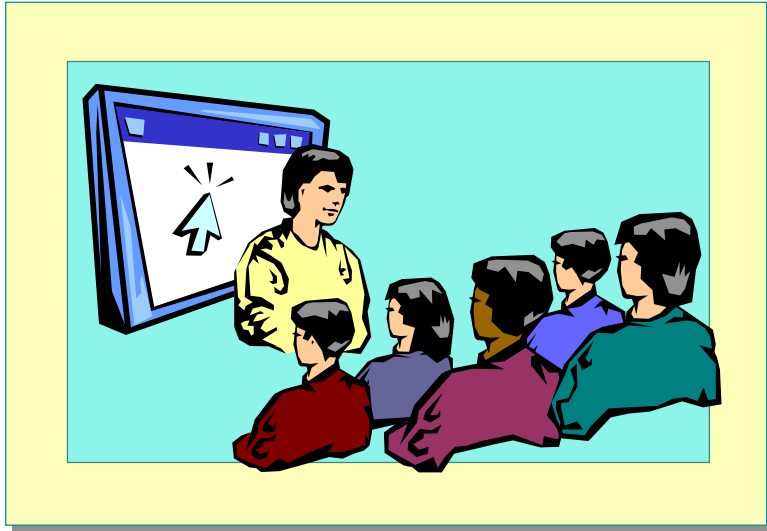
Demonstration: Publishing an XML Template

Topic Objective

To demonstrate how to publish an XML template on IIS, and how the XML template can be called from a URL.

Lead-in

In this demonstration, you will see how an XML template is published and used.



*****ILLEGAL FOR NON-TRAINER USE*****

This demonstration requires an existing virtual directory named Northwind. If you did not complete the previous demonstration, run CreateNWind.vbs in the folder DemoCode\Mod05.

1. Copy the Templates folder from DemoCode\Mod05 into the My Documents folder.
2. Start the Configure SQL XML Support in IIS tool.
3. Expand your server and select Default Web Site.
4. Double-click the Northwind virtual directory, and then click Settings.
5. Point out that template queries are allowed.
6. Click Virtual Names, and then click New.
7. Create a virtual name called "templates" with a type of "template" and a path to the Templates folder you copied into My Documents. Click Save, and then click OK.
8. Close the Microsoft Management Console window and test the template with the URL provided in the student notes on this page.

In this demonstration, you will see how a template can be used to retrieve data from the Northwind Traders database.

The following template will be saved as catalog.xml in a virtual name called Templates, and will be accessed by using a browser:

```
<?xml version="1.0"?>
<catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:query>
    SELECT productid, productname, unitprice
    FROM products product
    FOR XML AUTO
  </sql:query>
</catalog>
```

The template will be accessed by using the following URL:

<http://localhost/northwind/templates/catalog.xml>

Using Parameters in Template Files

Topic Objective

To describe how to add one or more parameters to a template.

Lead-in

You make your templates more flexible if you add one or more parameters.

- Specify any parameters in the template header

- Prefix the parameter with @ in the query

```
<?xml version="1.0"?>
<catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:header>
    <sql:param name="category">1</sql:param>
  </sql:header>
  <sql:query>
    SELECT productid, productname, unitprice
    FROM products product
    WHERE categoryid = @category
    FOR XML AUTO
  </sql:query>
</catalog>
```

*****ILLEGAL FOR NON-TRAINER USE*****

In a template file, a parameter is a variable within the query string whose value is provided by the user. Using parameters makes template files more flexible.

Specifying Parameters in a Template Header

All parameters must be specified within the <sql:header> element. Consider the following facts about template parameters:

- All parameters for a template must be contained within a single <sql:header> element.
- Use a separate <sql:param> element to specify each parameter.
- For each <sql:param> element you create, you must supply a **name** attribute that is unique within the template.
- Assign each parameter a default value by using the element value.

Referencing Parameters in a Query

You reference parameters in queries by prefixing the name of the parameter with the @ symbol. For example, a parameter named category could be referenced in a query as @category.

Using Comparison Operators in a Parameter

You can make a parameter more flexible by including comparison operators, such as “greater than” or “less than”. When you use comparison operators, you must encode any characters that have specific meanings in XML. For example, the “greater than” operator must be encoded as “>”, and the “less than” operator as “<”.

Passing Parameters in a URL

Parameters are passed in a URL as a query string. The first parameter is indicated by using a question mark (?), and subsequent parameters are separated by using an ampersand (&).

The following URL could be used to pass category and price parameters to a template called catalog.xml:

Example

`http://webserver/data/templates/catalog.xml?category=1&price=11.99`

The next example shows an XML template that accepts category and price parameters. Note that the “greater than” operator > has been encoded in the **SELECT** statement:

```
<?xml version="1.0"?>
<catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:header>
    <sql:param name="category">1</sql:param>
    <sql:param name="price">0</sql:param>
  </sql:header>
  <sql:query>
    SELECT productid, productname, unitprice
    FROM products product
    WHERE categoryid = @category
    AND unitprice &gt; @price
    FOR XML AUTO
  </sql:query>
</catalog>
```

Note To pass NULL as a parameter, you must install the XML for SQL Web Release and create a **nullvalue** attribute in the header element. This allows you to specify a string constant that will be interpreted as NULL when passed to a template.

Applying a Style Sheet to a Template File

Topic Objective

To describe how a style sheet can be applied to a template.

Lead-in

You can apply XSL style sheets to XML templates.

■ Server-side XSL processing

```
<?xml version="1.0"?>
<catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'
          sql:xsl="catalog.xsl">
  <sql:query>
    ...
  </sql:query>
</catalog>
```

■ Client-side XSL processing

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="catalog.xsl"?>
<catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:query>
    ...
  </sql:query>
</catalog>
```

*****ILLEGAL FOR NON-TRAINER USE*****

By default, executing a query by using a template file returns XML. However, it is also possible to render the data in a different format such as HTML or the Website META Language. To do this, you can apply one or more Extensible Stylesheet Language (XSL) style sheets to a template file.

You can apply a style sheet at the server, the client, or both. When style sheets are applied at both the client and the server, the server-side style sheet is applied first, and then the client processes the results.

Choosing Whether to Process at the Server or the Client

The advantage to server-side processing is that your XSL style sheet will be run regardless of the type of browser making the request. The disadvantage to server-side processing is the additional load that it places on the server.

Client-side XSL processing should be reserved for use in intranet environments where the Web browser is known to support XSL processing, as in Internet Explorer. Relying on client-side XSL processing in an Internet-based solution will limit the number of users who can successfully browse your site.

Applying a Style Sheet on the Server

To apply a style sheet on the server, add a **sql:xsl** attribute to the template's root element. This attribute should reference a style sheet in a relative path to the virtual name that contains the template.

The following template includes a **sql:xsl** attribute.

Example

```
<?xml version="1.0"?>
<catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'
  sql:xsl="catalog.xsl">
  <sql:header>
    <sql:param name="category">1</sql:param>
  </sql:header>
  <sql:query>
    SELECT productname, unitprice
    FROM products product
    WHERE categoryid = @category
    FOR XML AUTO
  </sql:query>
</catalog>
```

Specifying the Content Type of Returned Data

By default, SQL Server returns data requests from virtual directories in text or XML format. If you use a server-side style sheet to transform the document into a different content type, you must specify the appropriate contenttype parameter in the URL.

The following URL retrieves data from a template named catalog.xml that has been transformed into HTML by a server-side style sheet:

Example

```
http://webserver/data/templates/catalog.xml?category=1&contentt
ype=text/html
```

Specifying a Client-Side Style Sheet

When you use an XML-aware client such as Internet Explorer, you can add a processing instruction to the template that specifies a style sheet to be applied by the client. XML-aware clients will parse the XML document that contains the processing instruction and download the specified style sheet before applying it to the XML. Client applications that are not XML-aware will simply receive the XML results, and might need custom code in order to process the data.

The following template includes a processing instruction for the client to apply a style sheet. Because the client expects to receive an XML document, there is no requirement to specify the content type in the URL.

Example

```
<?xml version="1.0"?>
<?xml-stylesheet type='text/xsl' href='catalog.xsl'?>
<catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:header>
    <sql:param name="category">1</sql:param>
  </sql:header>
  <sql:query>
    SELECT productname, unitprice
    FROM products product
    WHERE categoryid = @category
    FOR XML AUTO
  </sql:query>
</catalog>
```


Lab 5.1: Publishing a SQL Server Database

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will create and use XML templates in a virtual directory that references a SQL Server 2000 database.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to:

- Create an IIS virtual directory for a SQL Server database.
- Test a virtual directory by using URL queries.
- Create and use XML templates to retrieve XML data over HTTP.

Prerequisites

Before working on this lab, you must have:

- Knowledge of IIS virtual directories.
- Knowledge of how to use the MMC tool Configure SQL XML Support in IIS.

For More Information

Read the section “Accessing SQL Server Using HTTP” in SQL Server Books Online.

Scenario

Northwind Traders wants to publish its catalog on the Internet. In this lab, you will create a virtual directory for the Northwind database and test it by using URL queries.

Estimated time to complete this lab: 45 minutes

Exercise 1

Creating a Virtual Directory

In this exercise, you will use the Configure SQL XML Support in IIS tool to create a virtual directory that allows HTTP access to the Northwind database.

Scenario

The first step in publishing the Northwind Traders catalog on the Internet is to create an IIS virtual directory that can be used to access the Northwind database.

► To create a new SQL Server virtual directory

1. In the **Microsoft SQL Server** program group, open **Configure SQL XML Support in IIS**.
2. Expand your server and select **Default Web Site**.
3. Click **Action**, point to **New**, and then click **Virtual Directory**.
4. On the **General** tab, configure the following settings.

Setting	Value
Virtual Directory Name	Lab05
Local Path	<install folder>\Labs\Lab05\ Northwindweb

5. Click **Security**, and then click **Use Windows Integrated Authentication**.
6. Click **Data Source** and configure the following settings.

Setting	Value
SQL Server	(local)
Database	Northwind

7. Click **Settings** and select the necessary options to allow URL (sql=...) queries and template queries.
8. Click **OK** to close the dialog box and create the virtual directory.
9. Close the IIS Virtual Directory Management for SQL Server window.

► To view the virtual directory with Internet Services Manager

1. In the **Administrative Tools** program group, start **Internet Services Manager**.
2. In the left pane, expand your server, and then open **Default Web Site**. Note that **Lab05** is listed among the Web applications.
3. Close Internet Services Manager.

Exercise 2

Testing the Virtual Directory Using URL Queries

In this exercise, you will test the Lab05 virtual directory by submitting URL queries.

Scenario

To test HTTP connectivity, you can execute queries by submitting them in a URL.

► To execute a URL query including an explicit root element

1. In Internet Explorer, type the following URL and examine the resulting document:

```
http://localhost/lab05?sql=SELECT '<pricelist>';  
SELECT productname, unitprice FROM products  
FOR XML AUTO; SELECT '</pricelist>'
```

2. Edit the URL to return a document with a top-level element named <catalog>.

► To execute a URL query by using the root parameter

- Enter a new URL query that retrieves the **productid**, **productname**, and **unitprice** fields from the **products** table. Use a **root** parameter to create a directory element named <catalog>.

► To execute a query containing encoded characters

- Enter a URL query that returns a document with a top-level element named <catalog> by using the following SQL query. Encode any characters that have special meaning in a URL.

```
SELECT productid, productname, unitprice  
FROM products product  
WHERE productname LIKE 'C%'  
FOR XML AUTO
```

Exercise 3

Creating a Template Virtual Name

In this exercise, you will create a template virtual name that will be used to store templates.

Scenario

Northwind Traders wants users to be able to browse its catalog by using templates. To make this possible, a template virtual name must be created to host the template files.

► To create a virtual name for the templates folder

1. Start the **Configure SQL XML Support in IIS** tool.
2. Expand your server and select **Default Web Site**.
3. Open **Lab05** to view its properties.
4. Click **Settings** and ensure that the option to allow template queries is selected.
5. Click **Virtual Names**, and then click **New**.
6. Create a new virtual name with the following configuration.

Setting	Value
Virtual name:	templates
Type:	template
Path:	<install folder> \\Labs\\Lab05\\Northwindweb\\Templates

7. Click **Save** to create the virtual name, and then click **OK** to close the dialog box.
8. Close the **IIS Virtual Directory Management for SQL Server** window.

Exercise 4

Creating Templates

In this exercise, you will create templates to retrieve data from the Northwind Traders database.

Scenario

Users of the Northwind Traders Web site must be able to browse products by category. This requires the creation of two templates: one to list the categories, and another to list the products in a specified category.

► To create a template containing a Transact-SQL query

The file Categories.xml in the folder *<install folder>\Labs\Lab05\Solution* contains a solution to this procedure.

1. Open Notepad and create an XML template with the following characteristics:
 - The root element should be named `<categorylist>`.
 - The root element should contain the following attribute declaration:
`xmlns:sql="urn:schemas-microsoft-com:xml-sql"`
 - The template should contain a `<sql:query>` element.
 - The `<sql:query>` element should contain the following query:

```
SELECT categoryid, categoryname, picture
FROM categories category
FOR XML AUTO
```
2. Save the template as categories.xml in the folder *<install folder>\Lab05\Northwindweb\Templates*.
3. Test your template by opening <http://localhost/lab05/templates/categories.xml> in Internet Explorer. Internet Explorer should be able to open an XML document that contains `<category>` elements.

► **To create a parameterized template**

The file Products.xml in the folder *<install folder>\Labs\Lab05\Solution* contains a solution to this procedure.

1. In Notepad, create an XML template with the following characteristics:
 - Name the root element `<productlist>`. It should contain the **xml-sql** namespace attribute.
 - The template should contain a parameter named “category” with a default value of 1.
 - The template should contain a query that returns the **productid**, **productname**, and **unitprice** of each row in the **products** table where the categoryid matches the template parameter. Each row returned by the query should result in a `<product>` element.
2. Save the template as products.xml in the folder *<install folder>\Lab05\Northwindweb\Templates*.
3. Test your template by opening <http://localhost/lab05/templates/products.xml?category=4> with Internet Explorer. Internet Explorer should be able to open an XML document containing `<product>` elements.

Exercise 5

Applying Style Sheets

In this exercise, you will edit the categories.xml and products.xml templates so that a style sheet is applied to the results.

Scenario

To make the Northwind Traders product data available to users through a browser, XSL style sheets must be applied to the templates to render the data as HTML.

► To add a style sheet reference to categories.xml

The file CategoriesEX5.xml in the folder *<install folder>\Labs\Lab05\Solution* contains a solution to this procedure.

1. Open the file categories.xml with Notepad from the folder Lab05\Starter.
2. Modify the template so that categories.xsl will be applied on the server.
3. Save the template to *<install folder>\Lab05\Northwindweb\Templates*.

► To add a style sheet reference to products.xml

The file ProductsEX5.xml in the folder *<install folder>\Labs\Lab05\Solution* contains a solution to this procedure.

- Repeat the preceding procedure to configure the products.xml file to reference the products.xsl style sheet.

► To test your templates

1. Open the template categories.xml in Internet Explorer. You will need to include an appropriate *contenttype* parameter in the URL.
2. When the category list is displayed, click any link on the page to view the products in that category.

◆ Posting Query Templates from the Client

Topic Objective

To introduce the topics in this section.

Lead-in

Templates can be posted from a client application over HTTP.

- Enabling POST Queries
- Using an HTML Form to Post a Template
- Posting Templates Directly to the Virtual Directory
- Demonstration: Posting a Template

*****ILLEGAL FOR NON-TRAINER USE*****

POST is an HTTP method that allows you to send data to a server with an HTTP request. The **POST** method is commonly used on Web pages that contain forms in which a user enters data and then clicks a Submit button to send the data to the server. SQL Server allows you to send XML templates to the server in this way.

In some circumstances, you might want to allow templates to be sent by a client application by using the HTTP **POST** method. This section describes the necessary configuration and development tasks for using queries that incorporate the **POST** method. Such queries will be referred to as *POST queries*.

Enabling POST Queries

Topic Objective

To describe the configuration necessary to allow POST queries.

Lead-in

The Configure SQLXML Support in IIS tool can be used to allow POST queries.

- **Configuring application settings**
 - Allow URL queries *and* POST queries
 - Specify a maximum POST length
- **Consider security issues carefully**

*****ILLEGAL FOR NON-TRAINER USE*****

You can use the MMC tool Configure SQL XML Support in IIS to allow POST queries.

Configuring Application Settings

POST templates are configured on the **Settings** tab of the **New Virtual Directory Properties** dialog box. You can configure POST queries by setting the following options:

- Allow SQL= Queries
- Allow POST Queries
- Set the maximum size in bytes of a POST query

To allow templates to be posted, both URL queries and POST queries must be enabled.

Security Considerations

You should consider potential security risks when deciding whether to allow POST queries. Keep in mind that any user who can make an HTTP connection to the virtual directory can send a template containing any valid SQL query. Also consider that, because you must enable URL queries, users can execute any SQL statement by adding it to the URL.

Any database object permissions that you have set will be enforced, allowing you to prevent access to sensitive data. However, in most Internet-based solutions, you should restrict data access to the maximum extent possible. For this reason, allowing POST templates is not recommended in production Internet sites.

Delivery Tip

The option to allow SQL= queries appears in the user interface as **Allow URL Queries** unless the XML for SQL Web Release has been installed.

Using an HTML Form to Post a Template

Topic Objective

To describe how an HTML form can be used to post a template.

Lead-in

You can post XML templates from a Web page by using an HTML form.

- **Add a FORM tag containing a template to a Web page**
 - Set **Action** attribute to the virtual directory URL
 - Set **Method** attribute to POST
 - Specify template in a hidden INPUT field named *template*
- **Use INPUT fields for parameters**
 - Field names and parameter names must match
- **Default content type is HTML**
 - Use a hidden INPUT field named *xml* to specify a style sheet
 - Use a hidden INPUT field named *contenttype* to specify text/xml

*****ILLEGAL FOR NON-TRAINER USE*****

You can post a template from a Web page by using an HTML form. Forms are a common way to post data from a Web page to a Web server.

Adding a FORM Containing a Template to a Web Page

You specify a form on a Web page by using the <FORM> HTML tag. To post the form to a virtual root SQL Server, you must include the following attributes.

FORM attribute	Value
Action	URL of the virtual directory
Method	'POST'

HTML forms generally contain tags such as <INPUT>, which create fields to allow users to enter data. The data is then appended as a query string to the URL specified in the **Action** attribute, and sent to the server when the user clicks a Submit button.

The template containing the query to be executed should be specified in an INPUT field with the following attributes.

INPUT attribute	Value
Type	'hidden'
Name	'template'
Value	The XML template as a string

Because the template field is hidden, users will not see the template text on the Web page. However, users will be able to read the template definition by viewing the page's HTML source.

Using INPUT Fields for Query Parameters

If the template contains parameters, you can use INPUT fields to allow users to set the values to be passed to the template. The **name** attribute for each INPUT field must match the corresponding parameter name in the template.

INPUT fields for parameter values should be assigned a **type** attribute that allows user input, such as "text". Additionally, the **value** attribute may be set to a default parameter value to allow users to accept a default choice.

Managing the Content Type

The default content type for templates posted from a Web form is text/html. Typically, you will want the data returned to the browser by the query to be formatted as HTML by using a server-side style sheet. You can specify which style sheet to apply to the data by including a hidden INPUT field with a **name** attribute of **xsl** and a **value** attribute indicating the relative path of the style sheet. The style sheet path must be relative to the virtual directory on the Web server.

Alternatively, you might want the results to be sent to the browser as XML. To do so, you can include a hidden INPUT field with a **name** attribute of **contenttype** and a **value** attribute of **text/xml**.

The following example shows an HTML page that contains a template in a FORM tag. The template contains a parameter, and the results will be returned as HTML by applying a style sheet named catalog.xsl.

Example

```
<HTML><HEAD><TITLE>Request Catalog</TITLE></HEAD>
<BODY>
Enter a Category ID.
<FORM action="http://webserver/data" method="POST">
<B>Category ID</B>
<INPUT type=text name=category value='1'>
<INPUT type=hidden name=xsl value='catalog.xsl'>
<INPUT type=hidden name=template value='

<catalog xmlns:sql="urn:schemas-microsoft-com:xml-sql" >
<sql:header>
  <sql:param name="category">1</sql:param>
</sql:header>

<sql:query>
  SELECT productid, productname, unitprice
  FROM   products product
  WHERE  categoryID=@category
  FOR XML AUTO
</sql:query>

</catalog>
'>
<p><INPUT type="submit">
</FORM>
</BODY></HTML>
```

Posting Templates Directly to the Virtual Directory

Topic Objective

To describe how templates can be posted directly to the virtual directory.

Lead-in

Templates can also be posted to a virtual directory programmatically.

- Load the template into a DOM document
- Post the DOM document to the virtual directory

```
Set objDoc = CreateObject("Microsoft.XMLDom")
Set objhttp = CreateObject("Microsoft.xmlhttp")
objDoc.LoadXML strTemplate
objhttp.Open "POST", "http://webserver/data", False
objhttp.Send objDoc
Msgbox objhttp.ResponseText
```

*****ILLEGAL FOR NON-TRAINER USE*****

There might be situations where a code component needs to post an XML query template to a SQL Server virtual directory. For example, a business component written in Microsoft Visual Basic® might need to retrieve product data in XML format.

Templates can be posted programmatically by using the Microsoft XML 3.0 component library (MSXML3.dll).

Loading a Template into a DOM Document

The first step in posting a template by using the MSXML components is to load the template into a Document Object Model (DOM) document object.

The template can be loaded from a string variable by using the **LoadXML** method, or from a file by using the **Load** method. When using the **Load** method, it is important to first set the **Async** property to **False**; otherwise your code might begin processing the XML before the entire XML document has been fully loaded and parsed.

When the document is loaded, you should use the MSXML parser to validate the XML before further processing. You can do this by checking the **parseError.errorCode** property for any value other than 0.

Posting the DOM Document

After the template is loaded into a DOM document, the object can be posted to the virtual directory by using the **XMLHTTP** object. There are three steps required to post an XML template by using the **XMLHTTP** object:

1. Use the **Open** method to open a connection to the virtual directory, specifying **POST** as the HTTP method you want to use.
2. Use the **Send** method to submit the template to the virtual directory.
3. Use the **ResponseText** property to retrieve the query results.

This example shows a Visual Basic script that could be used to post a template to SQL Server:

Example

```
Dim objDoc
Dim objHTTP
Dim strTemplate

Set objDoc = CreateObject("Microsoft.xmlDOM")
Set objHTTP = CreateObject("Microsoft.xmlhttp")

' create the template
strTemplate = "<?xml version='1.0'?>" & _
"<catalog xmlns:sql='urn:schemas-microsoft-com:xml-sql'>" & _
"<sql:query >" & _
" SELECT productid, productname FROM products FOR XML RAW" & _
"</sql:query>" & _
"</catalog>"

' Validate the document using the MSXML parser.
objDoc.loadXML strTemplate
If doc.parseError.errorCode Then
    MsgBox "The template is invalid"

' Post the template.
Else
    objHTTP.Open "POST", "http://webserver/data", False
    objHTTP.send objDoc

' Retrieve the results.
MsgBox objHTTP.responseText
End If
```

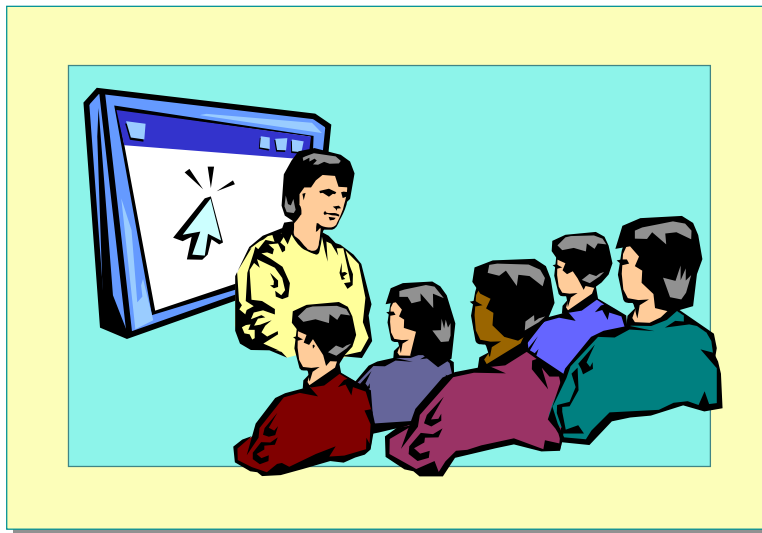
Demonstration: Posting a Template

Topic Objective

To demonstrate how to post a template.

Lead-in

In this demonstration, you will see how a template can be posted to a SQL Server virtual directory.



*****ILLEGAL FOR NON-TRAINER USE*****

This demonstration requires an existing virtual directory named Northwind. If you did not complete the previous demonstration, run CreateNWind.vbs from the folder DemoCode\Mod05.

1. Start the Configure SQL XML Support in IIS tool.
2. Expand your server and select Default Web Site.
3. Double-click Northwind and select the Settings tab.
4. Point out that, Allow SQL=...or template=...URL queries, is already selected. Select Allow POST, click OK, and then close the MMC.
5. Use Notepad to show the contents of the post.htm file in the folder DemoCode\Mod05.
6. Copy catalog.xsl to My Documents.
7. Open post.htm in Internet Explorer. Enter 4 as the category ID, and click Submit Query to view the results.
8. In Notepad, show the contents of post.vbs.
9. Close Notepad and double-click post.vbs to execute it.

In this demonstration, you will see how to configure a virtual directory to allow POST templates. You will see an example of a template posted from an HTML form, and an example of a template posted programmatically by using the Microsoft XML components.

◆ Publishing a Database Object

Topic Objective

To introduce the topics in this section.

Lead-in

Binary objects such as images can be retrieved by using a dbobject virtual name.

- Creating a dbobject Virtual Name
- Retrieving an Object over HTTP

*****ILLEGAL FOR NON-TRAINER USE*****

Databases are often used to store binary data such as images. This binary data can be retrieved through a SQL Server virtual directory by using a dbobject virtual name.

This section discusses the creation and use of a dbobject virtual name.

Creating a dbobject Virtual Name

Topic Objective

To describe how to create a dbobject virtual name.

Lead-in

To publish binary objects over HTTP, you must create a dbobject virtual name.

- **Create a dbobject virtual name to retrieve BLOB data (for example, an image)**
 - Name the virtual name “dbobject” to match the default for binary objects returned in AUTO mode
 - No path is specified
 - Only one dbobject virtual name is required for each virtual directory
- **Allow XPath queries**

*****ILLEGAL FOR NON-TRAINER USE*****

To publish binary large object (BLOB) data in a SQL Server virtual directory, you must create a dbobject virtual name.

Creating a dbobject Virtual Name for BLOB Data

Similar to template virtual names, you create dbobject virtual names by using the Configure SQL XML Support in IIS tool. Consider the following configuration issues when creating a dbobject virtual name:

- If you name the virtual name “dbobject”, the paths used to retrieve binary data items will be consistent with the paths returned by AUTO mode queries that include binary fields.
- No file path is specified for dbobject virtual names.
- A single dbobject virtual name allows access to any binary data in the published database. You only need to create one dbobject virtual name in a virtual directory to allow access to all dbobjects.

Allowing XPath Queries

To access objects by using a dbobject virtual name, XPath queries must be allowed in the virtual directory. To enable XPath queries, select the **Allow XPath** check box on the **Settings** tab of the **Properties** dialog box for the virtual directory.

Retrieving an Object over HTTP

Topic Objective

To describe how to retrieve binary data by using a dbobject virtual name.

Lead-in

You must specify a single binary field when retrieving data by using a dbobject virtual name.

- Specify an XPath to identify a single column / single value

```
dbobject/categories[@categoryid='1']/@picture
```

- In style sheets that use AUTO mode templates, use the dbobject path as the source for IMAGE fields

```
<IMG>
  <xsl:attribute name='SRC'>
    http://webserver/data/
    <xsl:value-of select='@picture' />
  </xsl:attribute>
</IMG>
```

*****ILLEGAL FOR NON-TRAINER USE*****

You can only use a dbobject virtual name to retrieve a single binary value, such as a single image.

Specifying an XPath for a Single Column/Single Value

A primary key value is usually specified in an XPath to retrieve a single column/single value result set through a dbobject virtual name. The XPath must specify the table containing the data, a unique identifier for the row, and the name of the binary field to be retrieved, in an attribute-centric format.

The following XPath could be used to retrieve the picture field for the row in the categories table with a categoryid value of 1:

Example

```
categories[@categoryid='1']/@picture
```

To retrieve the data over HTTP, the XPath is appended to the URL for the dbobject virtual name as follows:

```
http://webserver/data/dbobject/categories[@categoryid='1']/@picture
```

Using a dbobject Path in an XSL Style Sheet

You can use a dbobject path in an XSL style sheet to retrieve images and display them in an HTML tag.

You use an AUTO mode query to retrieve data from a table that includes a dbobject path. This technique requires you to include the primary key field in the query. You can then apply a style sheet to this data.

The style sheet that you apply should create an tag for each row returned by the query that has a **SRC** attribute constructed from the URL of the virtual directory and the dbobject path returned by the query. When the page is rendered, the dbobject path is used to retrieve the individual images and display them on the HTML page.

The following template could be used to retrieve a list of category IDs, category names, and pictures:

Example

```
<?xml version="1.0"?>
<catalog xmlns:sql="urn:schemas-microsoft-com:xml-sql"
  sql:xsl="catalog.xsl">
  <sql:query>
    SELECT categoryid, categoryname, picture
    FROM categories category
    FOR XML AUTO
  </sql:query>
</catalog>
```

The query in this template would return an XML document similar to the following:

```
<catalog xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <category categoryid="1"
    categoryname="Beverages"
    picture="dbobject/categories[@categoryid='1']/@picture"/>
  <category categoryid="2"
    categoryname="Condiments"
    picture="dbobject/categories[@categoryid='2']/@picture"/>
</catalog>
```

The catalog.xml style sheet could be used to create an tag for each category with a **SRC** attribute specifying the appropriate dbobject path to the image. The **SRC** attribute is created by using the <xsl:attribute> tag, as shown in the following example:

Example

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML><TITLE>Catalog</TITLE><BODY>

  <xsl:for-each select="catalog/category">
    <IMG>
      <xsl:attribute name="SRC">
        http://webserver/data/<xsl:value-of select="@picture"/>
      </xsl:attribute>
    </IMG>
    <BR/>
    <B><xsl:value-of select="@categoryname"/></B>
    <BR/>
    <BR/>
  </xsl:for-each>

</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Lab 5.2: Accessing a Database Object

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will access binary data by using a dbobject virtual name.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to access database objects over HTTP.

Prerequisites

Before working on this lab, you must have:

- Knowledge of how to create a virtual name in a virtual directory for a SQL Server database.
- Knowledge of how to retrieve a database object by using a dbobject virtual name and an XPath query.

For More Information

Read the section “Accessing SQL Server using HTTP” in SQL Server Books Online.

Scenario

Northwind Traders wants to use images on its Web site. These images are stored as binary objects in the database.

Estimated time to complete this lab: 15 minutes

Exercise 1

Creating a Virtual Name for a dbobject

In this exercise, you will create a dbobject virtual name to allow access to images in the Northwind database over HTTP.

Scenario

Each category in the Northwind Traders catalog has an associated image. The images are stored in the Northwind database. You must create a dbobject virtual name so that the images can be retrieved by using HTTP.

► To create a dbobject virtual name

1. If you did not complete Lab 5.1, run the script SkipLab51.vbs from the folder *<install folder>\Labs\Lab05*.
2. Start the **Configure SQL XML Support in IIS** tool from the **Microsoft SQL Server** program group.
3. Expand your server and select **Default Web Site**.
4. Open **Lab05** to view its properties.
5. Click **Settings**, and then select **Allow XPath**.
6. Click **Virtual Names**, and then click **New**.
7. Create a new virtual name with the following configuration.

Setting	Value
Virtual name:	dbobject
Type:	dbobject

8. Click **Save** to create the virtual name, and then click **OK**.
9. Close IIS Virtual Directory Management for SQL Server.

Exercise 2

Retrieving Image Data over HTTP

In this exercise, you will test your dbobject virtual name by retrieving image data over HTTP.

Scenario

The images associated with each category in the Northwind Traders catalog must be displayed on the Web page. To achieve this, you must retrieve the images by using the dbobject virtual name.

► To retrieve image data over HTTP

1. Open Internet Explorer and enter a URL that includes the following information:
 - The path to the dbobject virtual name in the Lab05 virtual directory
 - An XPath specifying the **picture** columns for the row in the **categories** table where the **categoryid** column has the value **3**
2. View the image.
3. Retrieve the image where the **categoryid** column has the value **4**.
4. Close Internet Explorer.

► To reference image data in a style sheet

The file Categories(with IMG).xsl in the folder *<install folder>*\Labs\Lab05\Solution contains a solution to this procedure.

1. Open the file categories.xsl with Notepad from the folder *<install folder>*\Labs\Lab05\Starter.
2. Find the comment **<!-- Add code to display picture here -->**.
3. Under this comment, add the necessary XSL code to create an **** tag with a **SRC** attribute referencing the current picture field.
4. Save the style sheet to *<install folder>*\Lab05\Northwindweb\Templates and close Notepad.
5. In Internet Explorer, test your style sheet by navigating to:
<http://localhost/lab05/templates/categories.xml?contenttype=text/html>
The image for each category should be displayed above its name.

Best Practices

Topic Objective

To describe best practices for HTTP database publishing.

Lead-in

Consider these best practices when using the HTTP publishing capabilities of SQL Server.

- **Avoid using URL queries except for testing**
- **Choose an appropriate security model**
- **Use server-side templates with parameters**
- **Use server-side XSL processing on the Web**

*****ILLEGAL FOR NON-TRAINER USE*****

Consider the following best practices when using the HTTP publishing capabilities of SQL Server.

Avoid Using URL Queries Except for Testing

URL queries are useful for testing SQL Server virtual directories. However, in a production environment, they allow users to submit any SQL query to the server. Although you can use database object permissions to prevent access to sensitive data, you should generally avoid allowing users to specify their own queries. This is particularly true when publishing a database on the Internet.

Choose an Appropriate Security Model

Ensure that the security model you implement will provide adequate protection of your data while minimizing the impact on manageability and scalability. In an intranet environment, you may want to use Windows Integrated Authentication to allow different database access rights to different groups of users. In an Internet solution, specifying that the virtual directory should always log on by using a particular Windows account will create a solution that is more scalable, because all users access the database by using the same connection string, making connection pooling more efficient. This solution is also more manageable, because only one Windows account must be granted access to the database.

Generally, you should avoid using SQL Server logins, and configure the server to accept Windows logins only.

Use Server-Side Templates with Parameters

Using server-side templates abstracts the database schema from users, ensuring a more secure solution. You should use parameters whenever users need to specify selection criteria, instead of allowing users to post their own templates from the client.

Use Server-Side XSL Processing on the Web

Use the **sql:xsl** attribute in templates that will be accessed from the Web to apply style sheets on the server. This allows your data to be accessed by any browser.

Review

Topic Objective

To reinforce module objectives by reviewing key points.

Lead-in

The review questions cover some of the key concepts taught in the module.

- Enabling HTTP Access to SQL Databases
- Publishing XML Templates on the Server
- Posting Query Templates from the Client
- Publishing a Database Object

*****ILLEGAL FOR NON-TRAINER USE*****

1. Which security model allows users on the Web to access your site anonymously and retrieve data from the database?

The trusted server model, using either a Windows or SQL Server login.

2. What URL should be used to execute the following query as a URL query, returning a document with a directory element named “<customerdata>” from a virtual directory named “data” on a server called “webserver”?

```
SELECT customerid, firstname, lastname
FROM customers
WHERE lastname LIKE 'M%'
FOR XML AUTO
```

**http://webserver/data?sql=SELECT customerid, firstname, lastname
FROM customers WHERE lastname LIKE 'M%25' FOR XML
AUTO&root=customerdata**

3. A server named “webserver” hosts a virtual directory named “data”, which contains the template virtual name “HR”. What URL would be used to pass an employeeid parameter with a value of 2 to a template named “employees.xml”?

http://webserver/data/HR/employees.xml?employeeid=2

4. The following template is saved as products.xml in the virtual name “catalog” in the virtual directory “ecommerce” on the Web server “www.contoso.msft”. What URL would you use to retrieve the products in category 3 as HTML?

```
<?xml version="1.0"?>
<products xmlns:sql="urn:schemas-microsoft-com:xml-sql"
  sql:xsl="products.xsl">
  <sql:header>
    <sql:param name="categoryid">1</sql:param>
  </sql:header>
  <sql:query>
    SELECT productid, productname, price
    FROM products product
    WHERE categoryid = @categoryid
    FOR XML AUTO
  </sql:query>
</products>
```

**http://www.contoso.msft/ecommerce/catalog/products.xml
?categoryid=3&contenttype=text/html**

5. You want to allow templates to be posted from a client. What configuration options should you set?

Allow URL Queries

Allow POST Queries

Maximum POST size

6. Assuming that you have the dbobject virtual name “dbobject” in the virtual directory “data” on the server “webserver”, what URL is required to retrieve the photo field for the record in the employees table with an employeeid value of 1?

http://webserver/data/dbobject/employees[@employeeid='1']/@photo

Module 6: Using Mapping Schemas

Contents

Overview	1
Using Mapping Schemas	2
Using Schemas to Retrieve Data	11
Lab 6.1: Using Mapping Schemas	21
Mapping Multiple Tables	30
Lab 6.2: Mapping Multiple Tables	37
Annotating an Existing Schema	42
Lab 6.3: Annotating an Existing Schema	52
Best Practices	56
Review	57



Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BackOffice, BizTalk, MSDN, MS-DOS, SQL Server, Visual Basic, Visual C++, Visual InterDev, Visual J++, Visual Studio, Windows, Windows Media, Windows NT, and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Instructor Notes

Presentation:
90 Minutes

Lab:
90 Minutes

This module introduces students to the use of XML-Data Reduced (XDR) Schemas as mapping schemas, and discusses the main annotations the students will need in order to map an Extensible Markup Language (XML) document to data in a Microsoft® SQL Server™ database.

After completing this module, students will be able to:

- Describe the uses and structure of annotated schemas.
- Create annotated schemas that map business documents to database entities.
- Use the XML Path Language (XPath) to access data through an annotated schema.
- Use Microsoft ActiveX® Data Objects (ADO) to request data by using an annotated schema.
- Access data by using a schema over the Hypertext Transport Protocol (HTTP).

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need the following materials:

- Microsoft PowerPoint® file 2091a_06.ppt
- The XDR.xml example schema (for the demonstration)

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the labs.
- Practice the demonstration.

Demonstration

This section provides demonstration procedures that will not fit in the margin notes or are not appropriate for the student notes.

Creating an Annotated Schema

► **To prepare for the demonstration**

1. Follow the demonstration steps to create the schema.
2. Because the schema is based on data in the Northwind Traders database, you might want to extend the demonstration by publishing the schema in a virtual name and accessing data through it with Microsoft Internet Explorer.

Module Strategy

Use the following strategies to present this module:

- Using Mapping Schemas

This section sets the scene and explains why schemas are a good way to retrieve data. First, introduce schemas as a way of defining business documents to be exchanged between trading partners or applications. Then, describe how to create a default mapping between database entities and XML documents by using a mapping schema. Finally, emphasize that in most production environments, you will need to add annotations in order to map your business documents to database entities.

- Using Schemas to Retrieve Data

Explain that there are three basic ways to retrieve data with a schema: by using ADO, by referencing the schema in an XML template, and by publishing the schema in a virtual name. Regardless of which approach you take, you must use an XPath expression to define the rows you want to return.

- Mapping Multiple Tables

Explain that this section presents a much more realistic use of mapping schemas, because most business data is stored in a relational format and normalized into multiple tables. Emphasize the importance of the `sql:key-fields` annotation, which ensures that the results are grouped correctly by identifying the primary key. Although not always required, you should encourage students to use the `sql:key-fields` annotation whenever they create a schema with a relationship annotation.

- Annotating an Existing Schema

It is very likely that most mapping schemas in a business solution will be based on existing schemas rather than newly created schemas designed explicitly to map data. This section discusses the annotations that can be used to include non-mapped elements (such as the root element or collection elements) in a mapping schema. It is also worth pointing out that if the schema contains a root element annotated with the `sql:is-constant` annotation, you do not have to specify a root element when retrieving data because a well-formed document will be returned.

Overview

Topic Objective

To provide an overview of the module topics and objectives.

Lead-in

In this module, you will learn how to use mapping schemas to retrieve XML data and map business document definitions to data in a SQL Server database.

- Using Mapping Schemas
- Using Schemas to Retrieve Data
- Mapping Multiple Tables
- Annotating an Existing Schema

*****ILLEGAL FOR NON-TRAINER USE*****

Mapping schemas are XML-Data Reduced (XDR) Schemas that specify the structure of an Extensible Markup Language (XML) document and map the elements and attributes it contains to tables and columns in a Microsoft® SQL Server™ database.

After completing this module, you will be able to:

- Describe the uses and structure of annotated schemas.
- Create annotated schemas that map business documents to database entities.
- Use the XML Path Language (XPath) to access data through an annotated schema.
- Use Microsoft ActiveX® Data Objects (ADO) to request data by using an annotated schema.
- Access data by using a schema over the Hypertext Transport Protocol (HTTP).

◆ Using Mapping Schemas

Topic Objective

To introduce the topics in this section.

Lead-in

In this section, we will examine how to map XDR Schemas to SQL data.

- Using Schemas for Data Exchange
- Defining Business Documents Using Schemas
- Defining a Default Table-to-Schema Mapping
- Adding Annotations to a Schema
- Demonstration: Creating an Annotated Schema

*****ILLEGAL FOR NON-TRAINER USE*****

You can use XDR Schemas to define an XML representation of a specific business document. This type of schema is particularly useful when multiple organizations must exchange business documents by using XML, because an XDR Schema provides a way to agree on a document's XML format. Later in this module, you will see how to use schemas to access data in SQL Server tables. For example, a retailer and a supplier could agree to use a schema to define a purchase order document that they will exchange.

You use a *mapping schema* to associate a business document's representation in XML with the underlying data in the database. You define the mapping by using table and column names to define XML elements and attributes. You can also use an *annotated schema* that contains annotations that map XML data to tables and columns in the database. You use an annotated schema to specify exactly which elements and attributes map to database tables and columns.

In addition, you can specify XPath queries against the XML view by using the mapping schema to query the database. The result of this type of query is an XML document.

This section discusses XDR Schemas and the creation of mapping schemas.

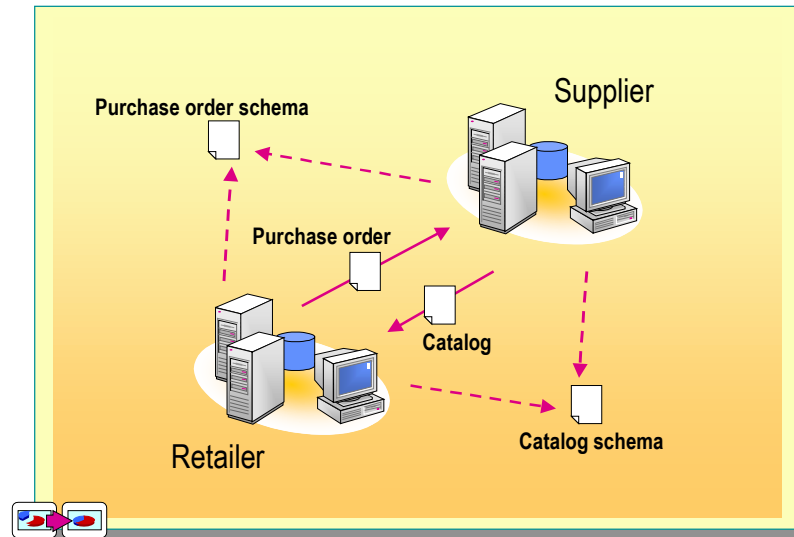
Using Schemas for Data Exchange

Topic Objective

To explain why schemas are useful for exchanging data.

Lead-in

Schemas are useful when two organizations need to agree on the XML representation of a business document.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

This slide builds automatically. First, the slide shows a purchase order schema being used as an agreed definition for XML purchase orders that a retailer will send to a supplier. Then, it shows a catalog schema being used as an agreed definition for a catalog document that will be downloaded from the supplier.

You can restart the build by pressing the LEFT ARROW key on the keyboard.

When two organizations need to agree on the XML representation of a particular type of business document, they can define a schema.

Defining Business Documents with Schemas

Schemas are XML documents that define the elements and attributes that can be legally used in a particular class of XML document. Schemas are useful when two business partners need to agree on a common definition for a document they will exchange.

For example, a retailer and a supplier might create a schema that defines a purchase order. Because there is an agreed definition of what a purchase order should look like, the two organizations can implement business processing logic to handle the purchase order documents that they will exchange. Similarly, a second schema could be used to define a catalog document, allowing catalog data to be exchanged in a consistent format.

Mapping Data in Tables to XML with Schemas

You can use XML schemas with SQL Server 2000 to map the data in XML business documents to data in the tables of a SQL Server database. This provides a way of generating XML business documents that contain data from SQL Server.

Defining Business Documents Using Schemas

Topic Objective

To describe how to use an XDR Schema to define the XML representation of a business document.

Lead-in

XDR provides a way to define the structure of an XML document: its elements and attributes and their types.

- **Agree on an XML representation of the document**
- **Create an XML-Data Reduced (XDR) schema**
 - Reference namespaces for element, attribute, and data types
 - Define valid elements and attributes

*****ILLEGAL FOR NON-TRAINER USE*****

You can use schemas to define an XML document's structure. The ability to define a valid XML representation of a business document is particularly important when instances of the document must be exchanged between systems and organizations.

Agreeing on an XML Representation of the Document

When two organizations or applications must exchange a business document such as a purchase order or invoice, they must agree on a common definition of a valid instance of that document. Consider the following issues when agreeing on the definition of an XML representation of a business document:

- Which elements and attributes can be included in the document?
- Which elements or attributes are mandatory, and which are optional?
- How many instances of each element in the document are allowed?
- What data types must the values in the elements and attributes be?

You can define the agreement in an XDR Schema, which is an XML document that contains the definition for a particular type of business document.

Creating an XDR Schema

XDR Schemas commonly use two namespaces, as described in the following table.

Namespace	Description
urn:schemas-microsoft-com:xml-data	Defines XDR elements and attributes.
urn:schemas-microsoft-com:datatypes	Defines data types for elements and attributes.

Delivery Tip

It is assumed that the students are already familiar with XDR syntax. Therefore, the XDR elements are not explained in detail at this time.

The xml-data namespace is necessary to define the elements and attributes within the XDR Schema. Use the optional datatypes namespace to define types of the elements and attributes.

Delivery Tip

You might want to draw attention to the precise capitalization of these XDR elements.

Consider the following important elements in an XDR Schema.

Element	Description
<Schema>	Defines the root element for an XDR Schema.
<ElementType>	Defines a type of element that can be placed in a document.
<element>	Defines an instance of an element.
<AttributeType>	Defines a type of attribute that can be placed within an element.
<attribute>	Defines an instance of the attribute.

For example, consider an XML document that represents a purchase order with an <Order> element that contains multiple <Item> elements. Each purchase order has **OrderID** and **CustomerID** attributes. Each item within the purchase order contains a ProductID and a Quantity attribute.

The following example shows an XDR Schema that defines a purchase order document.

Example

```
<?xml version="1.0" ?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data">

  <ElementType name="Item">
    <AttributeType name="ProductID"/>
    <AttributeType name="Quantity"/>
  </ElementType>

  <ElementType name="Order">
    <AttributeType name="OrderID"/>
    <AttributeType name="CustomerID"/>
    <attribute type="OrderID"/>
    <attribute type="CustomerID"/>
    <element type="Item"/>
  </ElementType>

</Schema>
```

Tip You can find a reference for XDR Schemas at <http://msdn.microsoft.com/xml/reference/schema/start.asp>

Defining a Default Table-to-Schema Mapping

Topic Objective

To explain the default mappings between the database and XML elements in the schema.

Lead-in

XDR Schemas can be mapped to data in a SQL Server database. The SQLOLEDB Provider recognizes a default mapping between a schema and a single table.

■ Creating an Attribute-Centric Mapping

- Elements map to tables or views
- Attributes map to columns

■ Creating an Element-Centric Mapping

- Use textOnly to map columns to child elements

*****ILLEGAL FOR NON-TRAINER USE*****

Key Points

The SQLOLEDB provider identifies the mapping between the schema and the database table. XDR provides no database mapping functionality of its own.

You can use an XDR Schema to map XML data to data in a SQL Server database. The SQL Server OLEDB (SQLOLEDB) Provider recognizes a default mapping between a schema and a single table when the element and attribute names in the schema precisely match table and column names in the database. This correspondence allows data to be returned to a client application by using the mapping schema.

Creating an Attribute-Centric Mapping

In an XDR Schema, an XML element by default maps to the table or view of the same name, and an XML attribute maps to the column of the same name within that table or view. This kind of mapping is convenient when there is a single underlying database table and you require attribute-centric documents.

In the following example, the annotated schema contains an <ElementType> element with a name attribute of **Orders**, and therefore is associated with the Orders table. The **OrderID** and **OrderDate** attributes defined within the <Orders> element are associated with corresponding columns of the Orders table in the database.

Example

```
<?xml version="1.0"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data">
  <ElementType name="Orders">
    <AttributeType name="OrderID"/>
    <AttributeType name="OrderDate"/>

    <attribute type="OrderID"/>
    <attribute type="OrderDate"/>
  </ElementType>
</Schema>
```

The results of a query against the Northwind Traders database that uses this schema appear as a set of <Orders> elements. Each element has **OrderID** and **OrderDate** attributes as follows:

```
<Orders OrderID="10248" OrderDate="1996-07-04T00:00:00" />
<Orders OrderID="10249" OrderDate="1996-07-05T00:00:00" />
<Orders OrderID="10250" OrderDate="1996-07-08T00:00:00" />
```

Creating an Element-Centric Mapping

The default mapping produces attribute-centric output. If you require element-centric output, the schema must contain an `ElementType` declaration with a “textOnly” value for its **content** attribute. This maps database columns to elements instead of attributes.

The following example shows a mapping schema that uses element-centric mapping with the Orders table from the Northwind Traders database.

Example

```
<?xml version="1.0"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data">
  <ElementType name="OrderID" content="textOnly"/>
  <ElementType name="OrderDate" content="textOnly"/>
  <ElementType name="Orders">
    <element type="OrderID"/>
    <element type="OrderDate"/>
  </ElementType>
</Schema>
```

Running results from this schema against the Northwind database produces an XML fragment with the following format:

```
<Orders>
  <OrderID>10248</OrderID>
  <OrderDate>1996-07-04T00:00:00</OrderDate>
</Orders>
<Orders>
  <OrderID>10249</OrderID>
  <OrderDate>1996-07-05T00:00:00</OrderDate>
</Orders>
<Orders>
  <OrderID>10250</OrderID>
  <OrderDate>1996-07-08T00:00:00</OrderDate>
</Orders>
```

Caution Element and attribute names in XML are always case-sensitive. Names of tables and columns in SQL Server may be case-sensitive, depending on the configuration of the database. To avoid problems, annotate the database as if the database names were case-sensitive.

Adding Annotations to a Schema

Topic Objective

To show how to create an alias for table names and column names by adding annotations to an existing schema.

Lead-in

When a schema's elements and attributes do not have the same names as tables and columns, you can use annotations to map the data.

- Reference the XML-SQL namespace
- Use the `sql:relation` annotation to create an alias for a table or view
- Use the `sql:field` annotation to create an alias for a column

```
<?xml version="1.0"?>
<Schema
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <ElementType name="Invoice" sql:relation="Orders">
    <AttributeType name="InvoiceID"/>
    <attribute type="InvoiceID" sql:field="OrderID"/>
  </ElementType>
</Schema>
```

*****ILLEGAL FOR NON-TRAINER USE*****

The default mapping is convenient in simple cases, but assumes that element and attribute names in the XML output correspond exactly to table and column names in the underlying database.

It is more common to annotate an existing schema than to create a new schema. You will often have an existing schema and database where the names of elements and attributes do not match the names in the database tables and columns. In this case, you should use a mapping schema that contains annotations.

Referencing the XML-SQL Namespace

To add annotations to a schema, you must reference the schema's XML-SQL namespace. The XML-SQL namespace defines the annotations that can be added to a schema in order to map XML data to SQL Server data.

The following table shows the annotations that you can add to a schema.

Delivery Tip

Explain that most of these annotations will be discussed in more detail later in the module.

Annotation	Purpose
relation	Map an element to a table.
field	Map an element or attribute to a column.
relationship	Define a primary/foreign key relationship.
datatype	Define the SQL data type of a column.
key-fields	Define the primary key field(s) for a table.
is-constant	Specify an element or attribute that appears in the XML document but does not map to a column.
map-field	Use to omit non-mapped schema elements or attributes from the resulting XML document.
limit-field & field-value	Use to filter the results based on a field value in a related table.

(continued)

Annotation	Purpose
target-namespace	Use to reference an alternative XML namespace (can be used to validate resulting documents against the schema). Note that this annotation can only be used when a schema is referenced in a template. For more information, see SQL Server Books Online.
id-prefix	Prepend XML ID, IDREF, or IDREFS fields with a unique identifier.
use-cdata	Generate XML data in a CDATA structure.
url-encode	Encode binary large objects (BLOB) data as a URL reference.
overflow-field	Define a column that contains overflow data.

Using the sql:relation Annotation to Map a Table

Use the sql:relation annotation to map an XML node in the XDR Schema to a database table or view in the following circumstances:

- When an XML element represents a table
- When an XML element or attribute represents a column in a different table from the parent element

Using the sql:field Annotation to Map a Column

You can add the sql:field annotation to map an XML node in the XDR Schema to a database column.

In the following example, the annotated schema defines an XML document containing a set of <Item> elements, each having **InvoiceID**, **ProductID**, and **Qty** attributes. The <Item> element is mapped to the [Order Details] table by using the sql:relation annotation. The **InvoiceID**, **ProductID**, and **Qty** attributes are mapped to the OrderID, ProductID, and Quantity columns, respectively, by using the sql:field annotation.

Example

```
<?xml version="1.0"?>
<Schema
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <ElementType name="Item" sql:relation="[Order Details]">
    <AttributeType name="InvoiceID"/>
    <AttributeType name="ProductID"/>
    <AttributeType name="Qty"/>

    <attribute type="InvoiceID" sql:field="OrderID"/>
    <attribute type="ProductID" sql:field="ProductID"/>
    <attribute type="Qty" sql:field="Quantity"/>
  </ElementType>
</Schema>
```

The following example shows the output from the preceding schema:

```
<Item InvoiceID="10278" ProductID="44" Qty="16" />
<Item InvoiceID="10278" ProductID="59" Qty="15" />
<Item InvoiceID="10278" ProductID="63" Qty="8" />
<Item InvoiceID="10278" ProductID="73" Qty="25" />
```

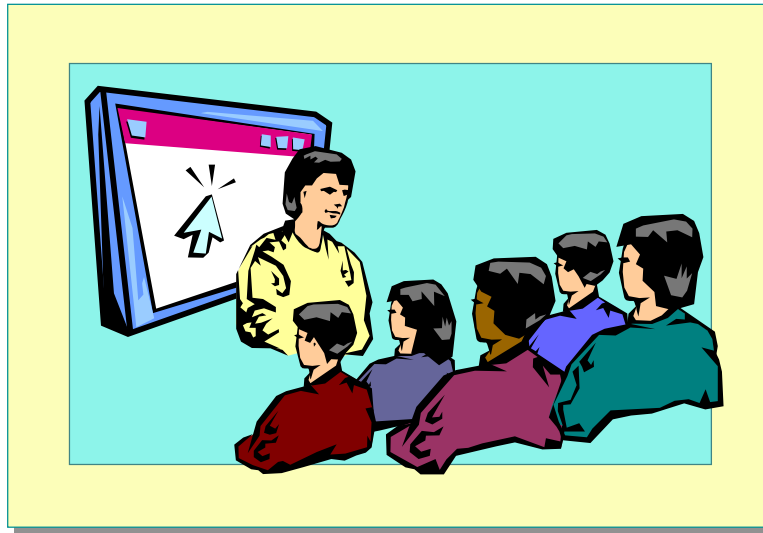
Demonstration: Creating an Annotated Schema

Topic Objective

To demonstrate how to create a simple annotated schema.

Lead-in

In this demonstration, you will see how to create a simple annotated schema.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

1. Open the file `<install folder>\DemoCode\Mod06\Annotated\XDR.xml` with Notepad.
2. Add the annotation `xmlns:sql="urn:schemas-microsoft-com:xml-sql"` to the Schema attribute.
3. Add the annotation `sql:relation="Products"` to the ElementType element.
4. Add the annotation `sql:field="ProductID"` to the first attribute element.
5. Add the annotation `sql:field="ProductName"` to the second attribute.

Alternatively, you may edit the file `<install folder>\DemoCode\Mod06\Annotated\Schema.xml` to show the complete schema.

In this demonstration, you will see how to create a simple annotated schema.

◆ Using Schemas to Retrieve Data

Topic Objective

To introduce the topics in this section.

Lead-in

In this section, we will examine how client applications can retrieve XML data by using a schema.

- Specifying Data with an XPath Expression
- Using a Schema with an ADO Command Object
- Querying a Schema from a Template
- Publishing a Schema Virtual Name
- Controlling Schema Caching

*****ILLEGAL FOR NON-TRAINER USE*****

You can develop client applications that use mapping schemas to retrieve XML data from a SQL Server database. An XPath expression determines the data that is retrieved.

You can reference mapping schemas by ADO 2.6–based applications, and by XML templates. You can also publish mapping schemas in a virtual name and access them directly by using a Uniform Resource Locator (URL).

This section discusses how to create client applications that use mapping schemas.

Specifying Data with an XPath Expression

Topic Objective

To show how XPath expressions are used to specify the data returned from a schema.

Lead-in

To generate an XML document, an XPath expression must be supplied with the schema and database.

■ XPath query selects the elements to generate

- Start element
- Selection criteria

■ SQLOLEDB Provider supports a subset of Xpath

```
/Item
```

```
/Item[@InvoiceID='10278']
```

■ Filtering the Result

*****ILLEGAL FOR NON-TRAINER USE*****

By itself, an annotated schema describes the structure of an XML document and how it relates to the underlying database. To perform a query, you need a way to specify which elements will be generated, as well as any selection criteria. An XPath query performs this role.

Using an XPath to Query an Annotated Schema

You can specify XPath expressions in several ways, such as by using a URL, a template, or programmatically through ADO. These techniques will be discussed later in this module. This section illustrates the syntax and semantics of XPath queries.

Delivery Tip

The fundamentals of XPath queries were discussed in Module 1, "Using XML and Microsoft SQL Server 2000." For more information, refer to Course 1913, *Exchanging and Transforming Data Using XML and XSLT*, Module 2, "Selecting and Navigating Nodes Using XPath."

For example, assume that you have the following annotated schema, which describes invoice item information from the Northwind Traders database.

Example

```
<?xml version="1.0"?>
<Schema
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <ElementType name="Item" sql:relation="[Order Details]">
    <AttributeType name="InvoiceID"/>
    <AttributeType name="ProductID"/>
    <AttributeType name="Qty"/>

    <attribute type="InvoiceID" sql:field="OrderID"/>
    <attribute type="ProductID" sql:field="ProductID"/>
    <attribute type="Qty" sql:field="Quantity"/>
  </ElementType>
</Schema>
```

To select all <Item> elements, you can use the following XPath expression:

```
/Item
```

Filtering the Result

You can use the element selection features of XPath expressions to filter results. For example, to select only items belonging to invoice 10278, use the following XPath expression:

```
/Item[@InvoiceID='10278']
```

Delivery Tip

Point out that the @ symbol denotes an attribute name here.

Note that in the preceding examples, where sql:relation and sql:field annotations have been added to create an alias for the table and column names of the database, the aliased names are used in the XPath expression.

Caution The SQLOLEDB Provider implements a subset of the XPath specification. The full XPath specification is at <http://www.w3.org/TR/xpath> ; See the SQL Server 2000 documentation for details of the supported subset.

Using a Schema with an ADO Command Object

Topic Objective

To show how to use an ADO Command object for a schema-based query.

Lead-in

You can use an ADO Command object to retrieve XML data through a schema.

■ Specifying the Schema

- Properties("Mapping Schema")
 May be relative to Properties("Base Path")

■ Specifying the XPath Expression

- Dialect = "{ec2a4293-e898-11d2-b1b7-00c04f680c56}"
- CommandText = *XPath*

■ Specifying the Root Element

- Properties("XML Root")

*****ILLEGAL FOR NON-TRAINER USE*****

For Your Information

In the Active Server Pages (ASP) environment, the Response object may also be used as an output stream.

Version 2.6 of the ADO libraries (supplied with SQL Server 2000) supports access to XML data through annotated schemas. You use the ADO **Command** object to reference the schema, and to specify the XPath expression to be used. The resulting XML document is written to the ADO **Stream** object assigned to the **Command** object's **Output Stream** property.

Specifying the Schema

You assign the file name of the schema to the **Mapping Schema** provider-specific property in the Properties collection of the command. You can specify the absolute path to the schema file, or set the **Base Path** property and use a relative path. The following code sets the **Base Path** and **Mapping Schema** properties of a **Command** object named **objCmd**:

```
objCmd.Properties("Base Path") = "c:\Schemas"
objCmd.Properties("Mapping Schema") = "InvoiceSchema.xml"
```

Specifying the XPath Expression

By default, The SQLOLEDB Provider works with **Command** objects that use SQL as the query language. To use XPath as the query language, you must set the **Dialect** property of the **Command** object to DBGUID_XPATH GUID "{ec2a4293-e898-11d2-b1b7-00c04f680c56}", which represents the XPath dialect. If you omit this step, the query string will be interpreted as SQL rather than as an XPath query, resulting in an error.

Assign the XPath query to the **CommandText** property of the **Command** object. The following code sets the **Dialect** and **CommandText** properties:

```
objCmd.Dialect = "{ec2a4293-e898-11d2-b1b7-00c04f680c56}"
objCmd.CommandText = "Invoice[@InvoiceNo='1923']"
```

Specifying the Root Element

To specify the name of a root element for the generated XML, set the **XML Root** member of the **Command** object's Properties collection.

The **XML Root** property is optional. If it is supplied, a complete XML document will be generated with a root element and `<?xml ... ?>` processing instruction. If the root element is omitted, an XML fragment will be returned.

The following code sets the xml root property:

```
ObjCmd.Properties("xml root") = "Invoice"
```

The following example shows how to use an ADO **Command** object to retrieve XML data by specifying a schema, an XPath expression, and a root element.

Example

```
Dim objConnect
Dim objCmd
Dim objStrm
Dim strXMLResults

Set objConnect=CreateObject("ADODB.Connection")
objConnect.Open "Provider=SQLOLEDB;Data Source=DBServer;
↳Initial catalog=Northwind;Integrated Security=SSPI;"

Set objStrm=CreateObject("ADODB.Stream")
objStrm.Open

Set objCmd=CreateObject("ADODB.Command")

With objCmd
    Set .ActiveConnection = objConnect
    .Properties("output stream")=objStrm
    .Properties("mapping schema")="c:\schemas\invoiceschema.xml"
    .Properties("xml root")="Invoice"
    .Dialect="{ec2a4293-e898-11d2-b1b7-00c04f680c56}"
    .CommandText="Invoice[@InvoiceNo='1923']"
    .Execute, , 1024 ' value for adExecuteStream
End With
StrXMLResults=objStrm.ReadText
```

Delivery Tip

This caution is necessary for the lab work, because Microsoft Internet Explorer version 5.5 has difficulty rendering XML documents containing UTF-8 escape sequences. The lab uses ISO-8859-1 (Latin 1) encoding.

Caution By default, an XML document is created with UTF-8 encoding, which represents accented and non-Latin characters by using multibyte escape sequences. If your character data falls entirely within the 7-bit character set, this will not cause any problems. Otherwise, ensure that your other XML-processing tools can correctly handle UTF-8 escape sequences. You can specify alternative encodings by setting the **Properties("Output Encoding")** property.

Tip You can apply an XSL style sheet before sending the XML results to the output stream. To specify which style sheet to use, set the **Properties("xsl")** property.

Querying a Schema from a Template

Topic Objective

To show how an annotated schema can be used from a template.

Lead-in

A template file can contain an annotated schema.

- **Using an <xpath-query> Element in a Template**
 - **mapping-schema** attribute defines the schema
 - Element contains the XPath expression
- **Passing Parameters to a Template**
 - The param annotation defines parameter and default
 - Use \$ as parameter placeholder

*****ILLEGAL FOR NON-TRAINER USE*****

XML template files can contain queries and can be published in a SQL Server virtual directory application. Although template files usually contain a SQL query, a template file alternatively can contain an XPath expression and a reference to an annotated schema.

Using an <xpath-query> Element in a Template

To reference a mapping schema, place an <xpath-query> element within the root element. This element must contain a **mapping-schema** attribute that identifies the annotated schema to be used. The text within the element is the XPath for the query.

The xml-sql namespace must appear in the template, usually by specifying the namespace urn:schemas-microsoft-com:xml-sql, with an alias of sql, in the root element of the template.

The following is an example of a template that contains the XPath expression “/Invoice” and a reference to the annotated schema “orderschema.xml”.

Example

```
<?xml version="1.0" ?>
<purchaseorder xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:xpath-query mapping-schema="orderschema.xml">
    /Invoice
  </sql:xpath-query>
</purchaseorder>
```


Delivery Tip

Compare this process to the process of creating SQL-based templates, which was covered in Module 5, "Using HTTP to Access XML Data."

You can then publish the template in a template virtual name, and execute queries by using a URL. For example, you could create a template called `order.xml` and save it in the virtual name "templates". If you publish this virtual name in the virtual directory "retaildata" on the webserver server, you can access the template by using the following URL:

`http://webserver/retaildata/templates/order.xml`

Delivery Tip

The parameter-passing mechanism is similar to SQL-based templates, described in Module 5, "Using HTTP to Access XML Data." However, when a schema is used, the @ and \$ characters perform different roles.

Passing Parameters to a Template

You can declare parameters in a template file by using the `<param>` element inside the header element. Each parameter has a name and a default value that is used if parameters are not supplied. In the XPath query, the dollar sign (\$) signifies that a parameter value will be used.

In the following example, the template has an `OrderID` parameter with the default value 0. The XPath query selects the `<publishers>` elements with a state value equal to the parameter value.

Example

```
<?xml version="1.0" ?>
<root xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:header>
    <sql:param name='OrderID'>0</sql:param>
  </sql:header>
  <sql:xpath-query mapping-schema="orderschema.xml">
    /orderdetails[@orderid=$OrderID]
  </sql:xpath-query>
</root>
```

When you invoke this template from a browser, you can provide a value for `OrderID` as an HTTP query string in the URL, as follows:

`http://webserver/retaildata/templates/order.xml?OrderID=1907`

Publishing a Schema Virtual Name

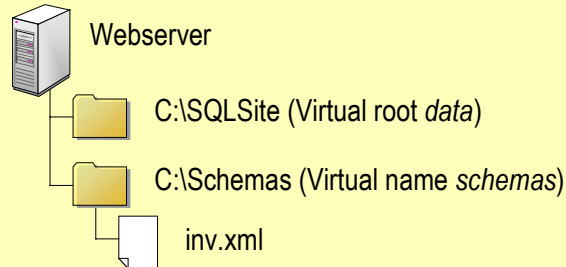
Topic Objective

To describe how to create and use virtual names for schemas.

Lead-in

You save schema files in a folder that is configured as a virtual name.

- Folders containing schemas are configured as virtual names



- Client applications access the schema by using a URL

`http://webserver/query/schemas/inv.xml/Item?root=Invoice`

*****ILLEGAL FOR NON-TRAINER USE*****

You can save schema files in any folder on the Web server. The schema files do not need to be in the same directory tree as the virtual root. Folders that contain schema files are configured as virtual names to allow them to be accessed in a URL.

Delivery Tip

This process is the same as configuring a schema virtual name, but with “schemas” substituted for “templates”.

Configuring a Schema Virtual Name

You can configure virtual names by using the Microsoft Management Console tool Configure SQL XML Support in IIS (where IIS represents Microsoft Internet Information Services). Consider the following issues when creating a virtual name for a schema:

- The virtual name must be unique within the virtual root application.
- The virtual name must be specified with the type *Schema*.
- The virtual name must relate to a physical folder containing XML schema files.
- A virtual name can contain a hierarchy of subfolders if desired.
- Appropriate file system permissions must be assigned to the physical folder referenced by the virtual name, and to its subfolders and files. The accounts that require access depend on the security model for the virtual root.

Accessing the Schema by Using a URL

You can access an annotated schema over HTTP by specifying the schema name, the XPath, and the root element in a URL.

To retrieve data from a Schema virtual name, create a URL that includes the following information:

- The path to the server, virtual root, and virtual name.
- The name of the schema file.
- An XPath expression defining the data to be returned.
- A root parameter to specify the root element name. (If you omit this, you will retrieve an XML fragment with no root element.)

The following example retrieves an XML document from the webserver server through the virtual root “query”, which has the virtual name “schemas” for schemas. The schema file name is `inv.xml`, the XPath expression is `Item`, and the name of the root element is `Invoice`.

Example

`http://webserver/query/schemas/inv.xml/Item?root=Invoice`

Controlling Schema Caching

Topic Objective

To describe the schema-caching mechanism and discuss when it is appropriate to use it.

Lead-in

IIS will cache schema documents in memory by default. You may want to change this in a development environment.

■ Disabling Caching for the Development Environment

- Set **disable schema caching** option for HTTP
- Use **SS STREAM FLAGS** property for ADO

*****ILLEGAL FOR NON-TRAINER USE*****

By default, IIS maintains an in-memory cache of the XML files that define the annotated schemas. When you execute an XPath query against an annotated XDR Schema, the schema is stored in memory, and the necessary data structures are built in memory. If you have set schema caching, the schema remains in memory, thus improving performance for subsequent XPath queries.

This method of caching is intended to promote efficiency by reducing the number of times a file system is accessed. However, if the schema files are changing (especially in a development environment where changes may be made very frequently), the caching will cause the server to use only old versions of the schema instead of later ones.

Disabling Caching for the Development Environment

You should not cache the schemas during development because changes to the schemas are not reflected in the cache for about two minutes. When you disable caching, a schema reloads each time a query is executed against it.

To disable caching for a virtual root, edit the properties of the virtual directory by using the Configure SQL XML Support in IIS tool. In the **Properties** dialog box, click the **Advanced** tab for the virtual directory, and then select **Disable Caching of Mapping Schemas**.

To disable caching when using ADO, set the **SS STREAM FLAGS** property to **STREAM_FLAGS_DONTCACHEMAPPINGCHEMA (8)** as shown in the following example:

```
objCmd.Properties("SS STREAM FLAGS") = 8
```

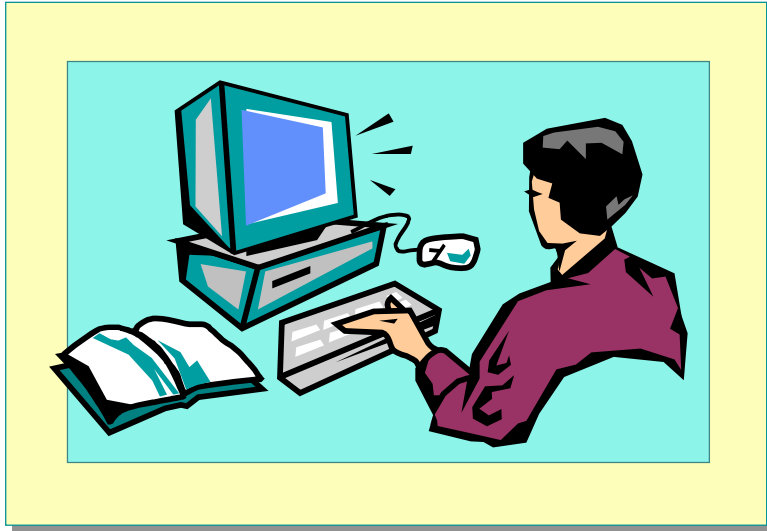
Lab 6.1: Using Mapping Schemas

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will create a simple mapping schema and generate XML documents from it in various ways.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

In this lab, you will create a simple mapping schema and generate XML documents from the schema in different ways.

After completing this lab, you will be able to:

- Create annotated schemas that map business documents to single database tables.
- Use XPath language to access data through a mapping schema.
- Use ADO 2.6 to request data by using a mapping schema.
- Access data by referencing a mapping schema in a template.
- Access data by accessing a schema published in a schema virtual name.

Prerequisites

Before working on this lab, you must understand:

- XDR Schema syntax.
- XML-SQL annotations for single tables.
- ADO development by using Microsoft Visual Basic® Scripting Edition.

Lab Setup

To complete this lab, you need a computer running Microsoft Windows® 2000 Server, with SQL Server 2000 and Microsoft Internet Explorer 5.5 or later installed.

Scenario

Northwind Traders has agreed to supply a copy of its product catalog to a business partner in the form of an XML document. The root element of this document is <Catalog>, which contains a set of <Product> elements. Each <Product> element has the following attributes.

Attribute	Description
ID	Unique ID
Name	Name of the product
QuantityPerUnit	Number of items in each ordered unit
UnitPrice	Price of each unit
InStock	Number in stock
OnOrder	Number on order
Discontinued	0 if available to purchase, 1 if not

The following is a sample catalog document:

```
<Catalog>
  <Product
    ID="1" Name="Chai"
    QuantityPerUnit="10 boxes x 20 bags"
    UnitPrice="18" InStock="39" OnOrder="0"
    Discontinued="0" />
  <Product
    ID="2" Name="Chang"
    QuantityPerUnit="24 - 12 oz bottles"
    UnitPrice="19" InStock="17" OnOrder="40"
    Discontinued="0" />
</Catalog>
```

In this lab, you will annotate an existing schema for this document and generate XML documents from the schema in various ways.

Estimated time to complete this lab: 30 minutes

Exercise 1

Creating a Simple Mapping Schema

In this exercise, you will add annotations to an existing XDR Schema in order to create a simple mapping schema. You will use a previously written test framework that will generate an XML document from the mapping schema.

Scenario

Two business partners have agreed to use an XDR Schema for a product catalog. You must annotate this schema to make it suitable for generating XML documents.

► To add annotations to the schema

The file ProductSchema1.xml in the folder *<install folder>\Labs\Lab06\Solution* contains a solution to this procedure.

1. Open the file ProductSchema1.xml with Microsoft Notepad from the directory *<install directory>\Labs\Lab06\Starter*.
2. Add the xml-sql namespace to the <Schema> element by adding the attribute **xmlns:sql="urn:schemas-microsoft-com:xml-sql"**.
3. Add the sql:relation annotation to the <ElementType> element that defines the <Product> element. This element must be mapped to the Products database table.
4. For each of the attributes defined in the schema for the <Product> element, create the following database column mappings by using the sql:field annotation.

Attribute	Database column
ID	ProductID
Name	ProductName
QuantityPerUnit	QuantityPerUnit
UnitPrice	UnitPrice
InStock	UnitsInStock
OnOrder	UnitsOnOrder
Discontinued	Discontinued

5. Save ProductSchema1.xml and close Notepad.

► **To test the annotated schema**

1. In Windows Explorer, double-click the file TestSchema1.htm in the folder *<install directory>\Labs\Lab06\Starter*. This is an application that will generate an XML document from the schema ProductSchema1.xml and the Northwind Traders database.
2. Click **Generate**.

Note Depending on the Internet Explorer security settings, you might see the message “An ActiveX control on this page might be unsafe to interact with other parts of the page. Do you want to allow this interaction?” If this question appears, click **Yes**.

3. If the document is successfully generated, the XML document source appears in the text area at the bottom of the screen. (You may paste the generated document into a new file and load it into Internet Explorer to view the results more clearly).

Exercise 2

Using ADO to Create XML Documents

In this exercise, you will write an application that uses ADO to create an XML document from a given mapping schema, database, and XPath expression.

Scenario

The business partners have agreed on an XDR Schema, which they have annotated. One of the partners requires an application that generates the product catalog document on demand.

This application should generate an XML document from the schema file ProductSchema1.xml. Save the generated document to disk with the file name ProductCatalog.xml.

You can find a template for this application in the file GenerateCatalog.vbs in the directory *<install directory>\Labs\Lab06\Starter*. This template contains some initialization code, and the necessary logic for the final save-to-disk operation. You must complete the ADO part of the application.

► To write the ADO application

The file GenerateCatalog.vbs in the folder *<install folder>\Labs\Lab06\Solution* contains a solution to this procedure.

1. Open GenerateCatalog.vbs with Notepad from the directory *<install directory>\Labs\Lab06\Starter*. Note the declarations that have been included.
2. Find the comment ' **SETUP ADO COMMAND**. The ADO **Command** object has been created, but is not yet initialized. After this comment you will find a **With .. End With** block in which you must set up the ADO command object **objCmd** by following the remaining steps.
3. Set the **ActiveConnection** property of **objCmd** to **objConn**.
4. Set the **Dialect** property of **objCmd** to **MSSQL_GUID_XPATH**. This is the constant for the globally unique identifier (GUID) that represents the XPath dialect.
5. Set the **Mapping Schema** entry in the Properties collection of **objCmd** to the value in the constant **sSchema**, which holds the filename of the schema file.
6. Set the **XML Root** entry to the value in the constant **sRoot**, which holds the root element name.
7. Set the **Output Stream** entry to the **objStream** variable, which holds the output stream.
8. Set the **Output Encoding** entry to **ISO-8859-1**. This is required because the database contains a number of accented characters, and Internet Explorer cannot handle this data in the default UTF-8 character encoding.

9. Set the **CommandText** property to the string in variable `sXPath`, which contains the XPath expression entered by the user. The complete code block for initialization of `objCmd` should resemble the following:

```
With objCmd
    Set .ActiveConnection = objConn
    .Dialect = MSSQL_GUID_XPATH
    .Properties("Mapping Schema") = sSchema
    .Properties("XML Root") = sRoot
    .Properties("Output Stream") = objStream
    .Properties("Output Encoding") = "ISO-8859-1"
    .CommandText = sXPath
End With
```

10. Locate the comment that begins with **EXECUTE COMMAND**. After this comment, write code that executes the command that sends the results to the output stream (remembering to specify the **adExecuteStream** option). The complete code block for execution of the command should read as follows:

```
objCmd.Execute , , adExecuteStream
```

11. Save the file and close Notepad.

► **To Test the ADO application**

1. Double-click the script file `GenerateCatalog.vbs` to run it.
2. The resulting XML is generated and saved to the file `ProductCatalog.xml`. Double-click this file to open it with Internet Explorer and check the results.

Exercise 3

Using a Schema with a Template

In this exercise, you will create a template that retrieves data from an annotated schema. You will install the template and schema files in Microsoft Internet Information Services (IIS), configure IIS, and generate an XML document.

Scenario

A business partner wants to access the Northwind Traders catalog over HTTP. To accomplish this, you will install an annotated schema and a template on a Web server running IIS.

You will need an annotated schema to complete this exercise. Use the file `ProductSchema1.xml` that you created in Exercise 1, or copy the solution file from the directory `<install directory>\Labs\Lab06\Solution` into the directory `<install directory>\Labs\Lab06\Starter`.

► To create a template virtual name

- Double-click the scripting host file `CreateRoot.vbs` from the directory `<install directory>\Labs\Lab06` directory to execute it. This script creates a virtual root called “Lab06” that allows queries based on templates and schemas, and provides two virtual names called “templates” and “schemas” for these purposes.

► **To create a template**

The file ProductTemplate1.xml in the folder *<install folder>\Labs\Lab06\Solution* contains a solution to this procedure.

1. Open Notepad to create a new XML query template.
2. Include the `<?xml version="1.0"?>` processing instruction at the head of the file.
3. Create a `<Catalog>` element by using the **urn:schemas-microsoft-com:xml-sql** namespace, aliased to the identifier **sql**.
4. In the `<Catalog>` element, place a `<sql:xpath-query>` element with a **mapping-schema** attribute set to the name of the schema file, ProductSchema1.xml.
5. In the `<sql:xpath-query>` element, place text containing the required XPath expression (**/Product**). The template file should look like the following:

```
<?xml version="1.0"?>
<Catalog xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:xpath-query mapping-schema="ProductSchema1.xml">
    /Product
  </sql:xpath-query>
</Catalog>
```

6. Save your work as ProductTemplate1.xml in the directory *<install directory>\Labs\Lab06\Starter*.

► **To test the template**

- Open the following URL with Internet Explorer:
<http://localhost/Lab06/templates/ProductTemplate1.xml>

Internet Explorer displays the XML document representing the product catalog.

Exercise 4

Using a Schema with a Virtual Name

In this exercise, you will configure IIS to allow queries to be made with an XPath expression and an annotated schema.

Scenario

An internal division of Northwind Traders wants the ability to request the company's product catalog. The division wants to use IIS and HTTP to execute arbitrary XPath expressions against the schema.

You will need an annotated schema to complete this exercise. Use the file `ProductSchema1.xml` that you created in Exercise 1, or copy the solution file from the directory `<install directory>\Labs\Lab06\Solution` into the directory `<install directory>\Labs\Lab06\Starter`.

To Enable XPath queries

1. If you completed Exercise 3, the virtual root **Lab06** already exists. If you did not complete Exercise 3, run the scripting host file `CreateRoot.vbs` in the `<install folder>\Labs\Lab06` folder to create the virtual root.
2. Start the **Configure SQL XML Support in IIS** tool from the **Microsoft SQL Server** program group and expand your server.
3. View the properties of the **Lab06** virtual root, and then click the **Settings** tab.
4. Select the **Allow XPath** checkbox, click **OK**, and then close the IIS Virtual Directory Management window.

► To test the schema

1. Open the following URL with Internet Explorer. The Web server will return the XML document representing the product catalog.

```
http://localhost/Lab06/schemas/ProductSchema1.xml  
↪/Product?root=Catalog
```

Caution The XPath expression component of the URL (`/Product`) is case-sensitive, so type it exactly as shown. The other components of the URL are not case-sensitive.

2. Modify the URL so that the XPath returns only rows where the `Discontinued` field has a value of 1. The URL used should resemble the following:

```
http://localhost/Lab06/schemas/ProductSchema1.xml  
↪/Product[@Discontinued=1]?root=Catalog
```

◆ Mapping Multiple Tables

Topic Objective

To introduce the topics in this section.

Lead-in

In this section, we will discuss how to map schemas to multiple database tables.

- Mapping Related Tables
- Controlling XML Document Nesting
- Filtering Data

*****ILLEGAL FOR NON-TRAINER USE*****

In most production solutions, you store the data required for a business document in more than one table. You use annotated schemas to map XML elements and attributes to multiple underlying tables, and to produce nested documents based on primary key and foreign key relationships in the database.

This section describes the annotations that you can use to reference multiple tables.

Mapping Related Tables

Topic Objective

To show how an annotated schema can contain references to multiple database tables.

Lead-in

In many cases, the annotated schema must base its data on more than one table.

- The `sql:relationship` annotation joins two database tables

```
<ElementType name="Item">
  ...
</ElementType>
<ElementType name="Invoice">
  ...
  <element type="Item">
    <sql:relationship
      key-relation="Orders"
      key="OrderID"
      foreign-relation="[Order Details]"
      foreign-key="OrderID"/>
  </element>
</ElementType>
```

*****ILLEGAL FOR NON-TRAINER USE*****

Until now, all of the schema examples in this module have mapped the schema to a single table or view. However, it is common for a schema to map to multiple tables in the underlying database. You achieve this type of mapping by using the `sql:relationship` annotation. You can map data from related tables by including the `sql:relationship` annotation and listing fields from a related table as one of the following:

- A nested element
- An attribute

Specifying Keys Using the `sql:relationship` Annotation

In the annotated XDR Schema, you use the `sql:relationship` annotation to nest the schema elements hierarchically. You can base the hierarchy on the primary key and foreign key relationships among the underlying tables to which the elements map. You must identify the following information when specifying the `sql:relationship` annotation:

- The primary table
- The primary key field in the primary table
- The foreign table
- The foreign key field in the foreign table

To provide this information, specify the following attributes with the relationship element.

Attribute	Description
Key-relation	Primary table
Key	Primary key column
foreign-relation	Foreign table
foreign-key	Foreign key column

In the following example, the annotated schema describes Invoice and Item elements. Each invoice can have a number of items, so each <Invoice> element can contain nested <Item> elements.

In the database, the Orders and Order Details tables are related by using the OrderID column. In the annotated schema, the join is expressed in the sql:relationship annotation.

Example

```
<?xml version="1.0"?>
<Schema
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">

  <ElementType name="Invoice" sql:relation="Orders">
    <AttributeType name="InvoiceID"/>
    <AttributeType name="InvoiceDate"/>
    <attribute type="InvoiceID"      sql:field="OrderID"/>
    <attribute type="InvoiceDate"   sql:field="OrderDate"/>
    <element type="Item">
      <sql:relationship
        key-relation="Orders"
        key="OrderID"
        foreign-relation="[Order Details]"
        foreign-key="OrderID"/>
    </element>
  </ElementType>

  <ElementType name="Item" sql:relation="[Order Details]">
    <AttributeType name="ProductID"/>
    <AttributeType name="Qty"/>
    <attribute type="ProductID"   sql:field="ProductID"/>
    <attribute type="Qty"         sql:field="Quantity"/>
  </ElementType>
</Schema>
```

Note The sql:relationship annotation uses the names of the underlying database tables and columns, not the XML element and attribute names.

XML documents generated by using this schema and an XPath of “Invoice” will have the following form:

```
<Invoice InvoiceID="10248" InvoiceDate="1996-07-04T00:00:00">
  <Item ProductID="11" Qty="12" />
  <Item ProductID="42" Qty="10" />
  <Item ProductID="72" Qty="5" />
</Invoice>
<Invoice InvoiceID="10249" InvoiceDate="1996-07-05T00:00:00">
  <Item ProductID="14" Qty="9" />
  <Item ProductID="51" Qty="40" />
</Invoice>
```

Note If the primary key is composed of multiple columns, specify values with a space between them. There is positional mapping between the values specified for the multicolumn key and the corresponding foreign key.

Controlling XML Document Nesting

Topic Objective

To explain the sql:key-fields annotation.

Lead-in

When two tables are related in a schema, the sql:relationship annotation might not contain all of the information necessary to uniquely identify rows in the source database.

- Use the sql:key-fields annotation when the relationship annotation does not provide primary keys

```
<ElementType name="InvoiceEmployee" sql:key-
fields="OrderID">
  <element type="Employee">
    <sql:relationship
      key-relation="Orders"
      key="EmployeeID"
      foreign-relation="Employees"
      foreign-key="EmployeeID"/>
    </element>
  ...
</ElementType>
<ElementType name="Employee">
  ...
</ElementType>
```

*****ILLEGAL FOR NON-TRAINER USE*****

When two tables are related in a schema, the sql:relationship annotation may not contain all of the information necessary to uniquely identify rows in the source database. This situation occurs if the required hierarchy in the XML document does not match a “one-many” relationship in the source database.

Using the sql:key-fields Annotation to Define a Primary Key

When the key for the primary key table is not specified in the sql:relationship annotation, use the sql:key-fields annotation to explicitly define it.

For example, consider a document that lists invoices and the employees who created them. Each <Invoice> element contains an <Employee> element. Examining the underlying database implies a join between the Orders.EmployeeID and Employees.EmployeeID fields, and this is correct. The corresponding sql:relationship annotation for this join is:

```
<sql:relationship
  key-relation="Orders" key="EmployeeID"
  foreign-relation="Employees" foreign-key="EmployeeID"/>
```

When the SQLOLEDB Provider creates the XML elements, it needs information about how to uniquely identify the records in the Orders table. Because the sql:relationship annotation does not provide this information in this case, incorrect XML will result—the Employee child elements will be incorrectly nested.

To correct this problem, use the sql:key-fields annotation after the sql:relationship annotation. This specifies the key field (or fields) for the underlying tables.

Tip Use the sql:key-fields annotation for all tables, even if you do not need it to correct the problem shown here. It makes the schema more self-documenting.

The following example shows a complete schema.

Example

```
<?xml version="1.0"?>
<Schema
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">

  <ElementType
    name="InvoiceEmployee"
    sql:relation="Orders"
    sql:key-fields="OrderID">

    <AttributeType name="InvoiceID"/>
    <attribute type="InvoiceID" sql:field="OrderID"/>

    <element type="Employee">
      <sql:relationship
        key-relation="Orders"
        key="EmployeeID"
        foreign-relation="Employees"
        foreign-key="EmployeeID"/>
    </element>
  </ElementType>

  <ElementType name="Employee" sql:relation="Employees">
    <AttributeType name="EmployeeID"/>
    <AttributeType name="LastName"/>
    <attribute type="EmployeeID" sql:field="EmployeeID"/>
    <attribute type="LastName" sql:field="LastName"/>
  </ElementType>
</Schema>
```

The following example shows output from the preceding schema.

Example

```
<InvoiceEmployee InvoiceID="10256">
  <Employee EmployeeID="3" LastName="Chor" />
</InvoiceEmployee>
<InvoiceEmployee InvoiceID="10251">
  <Employee EmployeeID="3" LastName="Chor" />
</InvoiceEmployee>
<InvoiceEmployee InvoiceID="10253">
  <Employee EmployeeID="3" LastName="Chor" />
</InvoiceEmployee>
```

Delivery Tip

Be sure to point out which output is correct and which is incorrect.

If data is extracted by using this schema without the `sql:key-fields` annotation, the following incorrectly nested XML fragment is produced:

```
<InvoiceEmployee InvoiceID="10256" />
<InvoiceEmployee InvoiceID="10251" />
<InvoiceEmployee InvoiceID="10253">
  <Employee EmployeeID="3" LastName="Chor" />
  <Employee EmployeeID="3" LastName="Chor" />
  <Employee EmployeeID="3" LastName="Chor" />
</InvoiceEmployee>
```

Filtering Data

Topic Objective

To show how the `sql:limit-field` and `sql:limit-value` annotations can be used to restrict data used in the query.

Lead-in

A developer may limit the rows returned from a query based on a limiting value, by using the `sql:limit-field` and `sql:limit-value` annotations.

- `sql:limit-field` and `sql:limit-value` annotations
- Apply to table defined in foreign-relation

```
<ElementType name="Item"> ... </ElementType>
<ElementType name="FullPriceItems">
  ...
  <element type="Item"
    sql:limit-field="Discount"
    sql:limit-value="0">
    <sql:relationship
      key-relation="Orders" key="OrderID"
      foreign-relation="[Order Details]"
      foreign-key="OrderID"/>
    </element>
  </ElementType>
```

*****ILLEGAL FOR NON-TRAINER USE*****

You can filter the rows returned from a query based on one or more values. Use `sql:limit-field` and `sql:limit-value` annotations to identify the database column that contains the limiting values, and to specify a specific limiting value to be used.

Using the `sql:limit-field` Annotation

Use the `sql:limit-field` annotation to filter information that is returned based on the value of a column in the database. You must use the `sql:limit-field` annotation on an element or attribute that has a relationship specified. Use the `sql:limit-value` annotation to specify the limited value in the column specified in a `sql:limit-field` annotation.

In effect, the `sql:limit-field` and `sql:limit-value` annotations act like a WHERE clause in SQL of the following form:

WHERE foreign-relation.limit-field = limit-value

The following example is a schema that uses the `sql:limit-field` and `sql:limit-value` annotations to restrict output to only those items that have a zero discount.

Example

```
<?xml version="1.0"?>
<Schema
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">

  <ElementType name="FullPriceItems" sql:relation="Orders">
    <AttributeType name="InvoiceID"/>
    <AttributeType name="InvoiceDate"/>
    <attribute type="InvoiceID"    sql:field="OrderID"/>
    <attribute type="InvoiceDate"  sql:field="OrderDate"/>

    <element type="Item"
      sql:limit-field="Discount"
      sql:limit-value="0">
      <sql:relationship
        key-relation="Orders" key="OrderID"
        foreign-relation="[Order Details]"
        foreign-key="OrderID"/>
    </element>
  </ElementType>

  <ElementType name="Item" sql:relation="[Order Details]">
    <AttributeType name="ProductID"/>
    <AttributeType name="Qty"/>
    <AttributeType name="Discount"/>

    <attribute type="ProductID"  sql:field="ProductID"/>
    <attribute type="Qty"         sql:field="Quantity"/>
    <attribute type="Discount"    sql:field="Discount"/>
  </ElementType>
</Schema>
```

Note You specify filtering by string equality. There are no relational operators such as `<` or `>`. This annotation is suitable for coding relationships between tables where the relationships are simple (based on column equality) and fixed. For example, there is no provision to pass a parameter to the schema above to change the `sql:limit-value` annotation to some other string value.

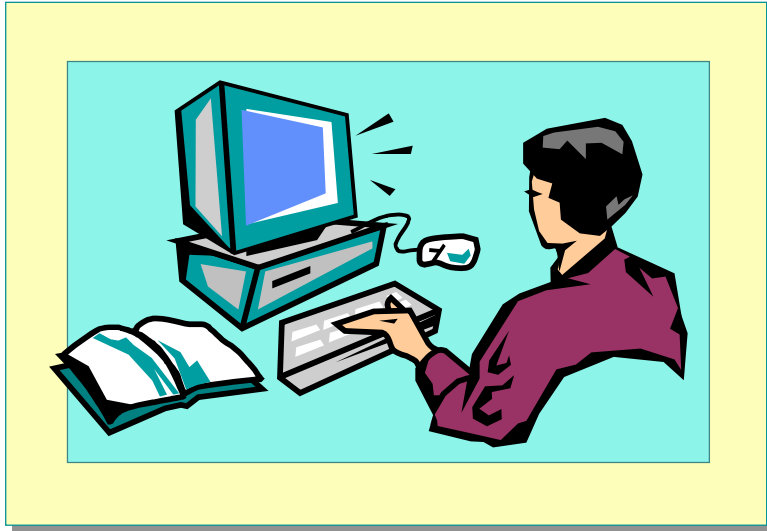
Lab 6.2: Mapping Multiple Tables

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will create more complex mapping schemas with multiple tables.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to create a mapping schema that maps XML data to multiple tables.

Prerequisites

Before working on this lab, you must understand:

- XDR Schema syntax.
- XML-SQL annotations for multiple tables.
- ADO development by using Microsoft Visual Basic® Scripting Edition.

Lab Setup

To complete this lab, you need a computer running Windows 2000 Server, with SQL Server 2000 and Internet Explorer 5.5 or later installed.

Scenario

The business partners of the Northwind Traders organization require a more descriptive format for catalog listings. This new format must include categories. You will modify the existing schemas and annotate them to include the product categories.

Estimated time to complete this lab: 30 minutes

Exercise 1

Creating a Mapping Schema with Multiple Tables

In this exercise, you will create a mapping schema that uses two tables. This will demonstrate how more complex schemas can be mapped.

Scenario

Northwind Traders wants to publish its product catalog with all of the <Product> elements grouped within <Category> elements.

The following is a sample catalog listing:

```
<Catalog>
  <Category ID="1" Name="Beverages">
    <Product ID="1" Name="Chai" ... />
    <Product ID="2" Name="Chang" ... />
  </Category>
  <Category ID="2" Name="Condiments">
    ...
  </Category>
</Catalog>
```

The <Category> element maps to the Categories table in the Northwind traders database, and the <Product> element maps to the Products table.

The attribute/column mappings for <Category> are listed in the following table.

Attribute	Database column
ID	CategoryID
Name	CategoryName

The attribute/column mappings for <Product> are listed in the following table.

Attribute	Database column
ID	ProductID
Name	ProductName
QuantityPerUnit	QuantityPerUnit
UnitPrice	UnitPrice
InStock	UnitsInStock
OnOrder	UnitsOnOrder
Discontinued	Discontinued

The column Products.CategoryID is a foreign key to Category.CategoryID.

► To edit the annotated schema

The file ProductSchema2.xml in the folder *<install folder>\Labs\Lab06\Solution* contains a solution to this procedure.

1. Open the file ProductSchema2.xml with Notepad from the folder *<install directory>\Labs\Lab06\Starter*. This file contains an XDR Schema for the *<Product>* and *<Category>* elements. This schema has not yet been annotated.

Add the **xml-sql** namespace to the *<Schema>* element by adding the attribute **xmlns:sql="urn:schemas-microsoft-com:xml-sql"**.

2. Add **sql:relation** annotations to map the *<Product>* element to the **Products** table, and the *<Category>* element to the **Categories** table.
3. Add **sql:field** annotations to map the attributes in the *<Product>* and *<Category>* elements to the appropriate database columns. For details, see the tables at the beginning of this exercise.
4. In the definition of the *<Category>* element, find the *<element>* element with a **type** attribute of **Product**. In this definition of the nested Product element, add a **sql:relationship** annotation with the attributes listed in the following table.

Attribute	Value
key-relation	Categories
key	CategoryID
foreign-relation	Products
foreign-key	CategoryID

5. Save the file.

► To test the mapping schema

1. If you have already completed Lab 6.1, "Using Mapping Schemas," the virtual root **Lab06** already exists. If you have not already done so, run the scripting host file CreateRoot.vbs in the folder *<install folder>\Labs\Lab06* to create the virtual root.
2. The template file ProductTemplate2.xml has been provided in the folder *<install folder>\Labs\Lab06\Solution* to test the schema. This template file generates a document with an XPath expression */Category* and root element of *<Catalog>*. To see the generated document, open the following URL with Internet Explorer:

<http://localhost/Lab06/templates/ProductTemplate2.xml>

Exercise 2

Annotating Key Fields

In this exercise, you will use the key-fields annotation.

Scenario

The schema for the product catalog document has changed. Although Northwind Traders still requires category information in its product document, the category information is to be held as a child element of the <Product> element.

The same element and attributes are present in the document, and have the same mapping to the database tables and columns as in Exercise 1.

The following is a sample of the catalog in the new format:

```
<Catalog>
  <Product ID="1" Name="Chai"
    QuantityPerUnit="10 boxes x 20 bags" UnitPrice="18"
    InStock="39" OnOrder="0" Discontinued="0" >
    <Category ID="1" Name="Beverages"/>
  </Product>
  <Product ID="2" Name="Chang"
    QuantityPerUnit="24 - 12 oz bottles" UnitPrice="19"
    InStock="17" OnOrder="40" Discontinued="0" >
    <Category ID="1" Name="Beverages"/>
  </Product>
</Catalog>
```

► To edit the annotated schema

The file ProductSchema3.xml in the folder <install folder>\Labs\Lab06\Solution contains a solution to this procedure.

1. Open the file ProductSchema3.xml with Notepad from the directory <install directory>\Labs\Lab06\Starter. This file contains an XDR Schema for the <Product> and <Category> elements. This schema is not yet annotated.

Add the **xml:sql** namespace to the <Schema> element by adding the attribute **xmlns:sql="urn:schemas-microsoft-com:xml-sql"**.

2. Add **sql:relation** annotations to map the <Product> element to the **Products** table, and the <Category> element to the **Categories** table.
3. Add **sql:field** annotations to map the attributes in the <Product> and <Category> elements to the appropriate database columns. For details, see the tables at the beginning of Exercise 1.

4. In the definition of the <Product> element, find the <element> element with a type attribute of **Category**. In this definition of the nested <Category>element, add a **sql:relationship** annotation with the following attributes.

Attribute	Value
key-relation	Products
key	CategoryID
foreign-relation	Categories
foreign-key	CategoryID

5. Add a **sql:key-fields** annotation to the <Product> element to identify the key field for this table as **ProductID**. This is required because this information cannot be inferred from the sql:relationship annotation.

► **To test the mapping schema**

1. If you have already completed Lab 6.1 or Exercise 1 of this lab, the virtual root Lab06 already exists. If you have not already done so, run the scripting host file CreateRoot.vbs from the folder <install folder>\Labs\Lab06 to create the virtual root.
2. The template file ProductTemplate3.xml has been provided in the folder <install folder>\Labs\Lab06\Solution to test the schema. This template file generates a document with an XPath expression “/Product” and a root element of <Catalog>. To see the generated document, open the following URL with Internet Explorer:

<http://localhost/Lab06/templates/ProductTemplate3.xml>

◆ Annotating an Existing Schema

Topic Objective

To introduce the topics in this section.

Lead-in

In this section, we will discuss some of the major issues facing developers who need to map existing XDR Schemas to SQL Server data.

- Using Constants
- Excluding Fields
- Mapping ID, IDREF, and IDREFS Attributes
- Specifying Data Types

*****ILLEGAL FOR NON-TRAINER USE*****

Until now, the schemas you have examined have been specifically designed to extract XML data from SQL Server. In reality, you might need to use an existing schema that defines a business document that you need to exchange with your trading partners.

This section discusses issues that arise when an existing schema document must be mapped to data in a SQL Server database.

Using Constants

Topic Objective

To explain the `sql:is-constant` annotation.

Lead-in

You might want to create an element in the schema that contains a constant that is not connected to any database table or column. This can be done with the `sql:is-constant` annotation.

- Use the `sql:is-constant` annotation to create constant elements

```
<ElementType name="Items" sql:is-constant="1">
  <element type="Item"/>
</ElementType>

<ElementType name="Item" sql:relation="[Order
Details]">
  ...
</ElementType>
```

*****ILLEGAL FOR NON-TRAINER USE*****

By default, every element and attribute in an annotated schema must map to a table or column in the database. However, schemas often contain an element or attribute that does not map to any database table or column, but still appears in the XML document.

For example, the root element of a document is usually defined in the schema, although it does not necessarily map to a field in the database. Similarly, a collection of elements (such as `Item`) may be contained within a container element (such as `Items`). These are called *constant elements*. To create a constant element, you can specify the `sql:is-constant` annotation to the `ElementType` element. This takes a Boolean value (0 meaning false, and 1 meaning true).

The following table describes the values that you can assign to the `sql:is-constant` annotation.

Boolean value	Result
0 (False)	The element must be mapped to a field in the database. This is the default setting.
1 (True)	The element appears in the resulting document as a constant. It is not mapped to a field in the database.

Using the sql:is-constant Annotation

You can use the sql:is-constant annotation to add a top-level element root element to the XML document, or to create container elements elsewhere in the XML document.

The following example shows a constant element <Items> around a set of <Item> elements.

Example

```
<?xml version="1.0"?>
<Schema
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">

  <ElementType name="Items" sql:is-constant="1">
    <element type="Item"/>
  </ElementType>

  <ElementType name="Item" sql:relation="[Order Details]">
    <AttributeType name="InvoiceID"/>
    <AttributeType name="ProductID"/>
    <AttributeType name="Qty"/>

    <attribute type="InvoiceID" sql:field="OrderID"/>
    <attribute type="ProductID" sql:field="ProductID"/>
    <attribute type="Qty" sql:field="Quantity"/>
  </ElementType>

</Schema>
```

The following example shows output from the preceding schema with an XPath expression of “Items”:

```
<Items>
  <Item InvoiceID="10248" ProductID="11" Qty="12" />
  <Item InvoiceID="10248" ProductID="42" Qty="10" />
  <Item InvoiceID="10248" ProductID="72" Qty="5" />
  <Item InvoiceID="10249" ProductID="14" Qty="9" />
</Items>
```

Excluding Fields

Topic Objective

To show how to use the `sql:map-field` annotation to exclude an XML element in the output document.

Lead-in

Elements in the schema can be excluded from the output document by setting the `sql:map-field` annotation to "0".

■ Using the `sql:map-field` Annotation

```
<ElementType name="Item" sql:relation="[Order Details]">
  <AttributeType name="InvoiceID"/>
  <AttributeType name="ProductID"/>
  <AttributeType name="Qty"/>
  <AttributeType name="TransferStatus"/>
  <attribute type="InvoiceID" sql:field="OrderID"/>
  <attribute type="ProductID" sql:field="ProductID"/>
  <attribute type="Qty" sql:field="Quantity"/>
  <attribute type="TransferStatus" sql:map-field="0"/>
</ElementType>
```

*****ILLEGAL FOR NON-TRAINER USE*****

The purpose of an annotated schema is not only to generate XML documents from SQL server, but also to define the elements and attributes you want to use in your XML documents. Some elements might be optional in the XDR Schema and do not map to any fields in the database.

You can use the `sql:map-field` annotation to cause SQL Server to ignore any optional fields in the schema that do not map to data in the database.

Using the `sql:map-field` Annotation

Similar to the `sql:is-constant` annotation, the `sql:map-field` annotation takes a Boolean value where 0 represents false and 1 represents true. This annotation is valid only on an `<attribute>`, `<element>`, or `<ElementType>` with text-only content (`content=textOnly`). The annotation is not valid on an `<element>` or `<ElementType>` that maps to a table.

Unlike the `sql:is-constant` annotation, in the `sql:map-field` annotation elements and attributes that are unmapped do not appear in the output XML document.

The following table describes the values that you can assign to the `sql:map-field` annotation.

Boolean value	Result
0 (False)	The element is not mapped to a field in the database, and will not be generated in the resulting XML document.
1 (True)	The element is mapped to a field in the database. This is the default setting.

In the following example schema, the **TransferStatus** attribute is declared as part of the annotated XML Schema but does not appear in the output XML and is not mapped to a database column.

Example

```
<?xml version="1.0"?>
<Schema
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">

  <ElementType name="Item" sql:relation="[Order Details]">
    <AttributeType name="InvoiceID"/>
    <AttributeType name="ProductID"/>
    <AttributeType name="Qty"/>
    <AttributeType name="TransferStatus"/>
    <attribute type="InvoiceID" sql:field="OrderID"/>
    <attribute type="ProductID" sql:field="ProductID"/>
    <attribute type="Qty" sql:field="Quantity"/>
    <attribute type="TransferStatus" sql:map-field="0"/>
  </ElementType>
</Schema>
```

Mapping ID, IDREF, and IDREFS Attributes

Topic Objective

To explain how ID, IDREF, and IDREFS attributes are introduced in an annotated schema.

Lead-in

ID, IDREF, and IDREFS values can be generated from the database. The sql:id-prefix annotation is used to ensure that ID values from each table are unique.

■ Using ID, IDREF, and IDREFS Attributes

■ Using sql:id-prefix for ID Prefixes

```
<ElementType name="Invoice">
  <AttributeType name="InvoiceID"
    dt:type="id"
    sql:id-prefix="in-"/>
  ...
```

```
<ElementType name="Item">
  <AttributeType name="InvoiceID"
    dt:type="idref"
    sql:id-prefix="in-"/>
  ...
```

*****ILLEGAL FOR NON-TRAINER USE*****

As an alternative to nesting elements within other elements, you can map the contents of several tables to elements that appear at the same level in the output document. To relate the information in the tables, you can use **ID**, **IDREF**, and **IDREFS** attributes, defined by the XML data types namespace (urn:schemas-microsoft-com:datatypes).

Using ID, IDREF, and IDREFS Attributes

Delivery Tip

This section assumes that students already have knowledge of these attribute types and the DOM programming model. This section should act as a refresher.

ID attributes correspond to primary keys. **IDREF** attributes correspond to foreign keys, representing the “many” reference in a one-many relationship. **IDREFS** attributes have no exact match in a relational database, but are used to define the “one” end of a one-many relationship.

You can use the Document Object Model (DOM) with applications to navigate from one set of elements to another by using the **nodeFromID()** method.

Note The value of an **ID**, **IDREF**, or **IDREFS** attribute must be a named token. (For example, integer value 101 cannot be an ID value.)

Using sql:id-prefix for ID Prefixes

In the XML document, the values of **ID** type attributes must be distinct within the whole document. However, in a database, **ID** attribute values are usually unique within a particular table.

You can use the sql:id-prefix annotation to automatically attach a different prefix to each set of **ID** attribute values, thus ensuring uniqueness within the document.

Example

```
<?xml version="1.0"?>
<Schema
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">

  <ElementType name="Invoice" sql:relation="Orders">
    <AttributeType name="InvoiceID"
      dt:type="id" sql:id-prefix="in_"/>
    <AttributeType name="InvoiceDate"/>
    <attribute type="InvoiceID" sql:field="OrderID"/>
    <attribute type="InvoiceDate" sql:field="OrderDate"/>
  </ElementType>

  <ElementType name="Item" sql:relation="[Order Details]">
    <AttributeType name="InvoiceID"
      dt:type="idref" sql:id-prefix="in_"/>
    <AttributeType name="ProductID"/>
    <attribute type="InvoiceID" sql:field="OrderID"/>
    <attribute type="ProductID" sql:field="ProductID"/>
  </ElementType>

  <ElementType name="InvoicesItems" sql:is-constant="1">
    <element type="Item"/>
    <element type="Invoice"/>
  </ElementType>
</Schema>
```


This schema creates mappings for two sets of elements. In the database, the corresponding tables have a one-many relationship through Orders.OrderID, and [Order Details].OrderID.

Delivery Tip

For simplicity, the example shows only one SQL prefix.

The **InvoiceID** attribute of the <Invoice> element is defined with a data type of **ID** because it is the primary key, and is given an ID prefix of “in_”. The <Item> element has an **OrderID** attribute, which is declared as an **IDREF** (foreign key).

To act as a container, the <InvoicesItems> element contains all of the titles and publishers elements. Because this is not mapped to any table in the database, it is marked as a constant.

The following is sample output from this schema:

```
<InvoicesItems>
  <Item InvoiceID="in_10285" ProductID="1" />
  <Item InvoiceID="in_10294" ProductID="1" />
  <Invoice InvoiceID="in_10248"
    InvoiceDate="1996-07-04T00:00:00" />
  <Invoice InvoiceID="in_10249"
    InvoiceDate="1996-07-05T00:00:00" />
</InvoicesItems>
```

Specifying Data Types

Topic Objective

To describe how to use the `sql:datatype` annotation.

Lead-in

The XML-DataTypes namespace provides the `type` attribute for XML data types. You can use the XML-SQL namespace `datatype` annotation to define SQL data types.

■ Using the `sql:datatype` Annotation

- Support for binary data included with SQL Server
- Support for all other SQL types added with the XML for SQL update

*****ILLEGAL FOR NON-TRAINER USE*****

The XML-datatypes namespace (`urn:schemas-microsoft-com:datatypes`) provides a **type** attribute that you can use in a schema to define the XML data type of an element or attribute. To match this to the appropriate SQL data type, the XML-SQL namespace (`urn:schemas-microsoft-com:xml-sql`) provides the `sql:datatype` annotation. You can use both attributes in a schema to ensure that the retrieved data is mapped to the correct data type.

Using the `sql:datatype` Annotation

By default, you can use the `sql:datatype` annotation to explicitly define the data type for binary fields, such as text or image columns. The XML-datatypes namespace defines all such fields as `bin.base64`, and therefore the `sql:datatype` annotation must be used to coerce the data to the correct data type. For example, the following element declaration could be used in a schema:

```
<ElementType name="Employee" sql:relation="Employees">
  <AttributeType name="EID"/>
  <AttributeType name="fname"/>
  <AttributeType name="lname"/>
  <AttributeType name="photo" dt:type="bin.base64"/>

  <attribute type="EID" sql:field="EmployeeID"/>
  <attribute type="fname" sql:field="FirstName"/>
  <attribute type="lname" sql:field="LastName"/>
  <attribute type="photo" sql:field="Photo"
    sql:datatype="image"/>
</ElementType>
```

Delivery Tip

The XML for SQL update is discussed fully in Module 7, "XML for SQL Tools."

Defining Other SQL Data Types

If you have installed the XML for SQL update, you can assign other built-in SQL data types (such as **varchar** and **integer**) to the `sql:datatype` annotation. For example, the following element definition could be used to explicitly define the data types of all fields.

Example

```
<ElementType name="Employee" sql:relation="Employees">
  <AttributeType name="EID"/>
  <AttributeType name="fname"/>
  <AttributeType name="lname"/>
  <AttributeType name="photo" dt:type="bin.base64"/>

  <attribute type="EID" sql:field="EmployeeID"
    sql:datatype="integer"/>
  <attribute type="fname" sql:field="FirstName"
    sql:datatype="varchar(20)"/>
  <attribute type="lname" sql:field="LastName"
    sql:datatype="varchar(30)"/>
  <attribute type="photo" sql:field="Photo"
    sql:datatype="image"/>
</ElementType>
```

Lab 6.3: Annotating an Existing Schema

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will annotate an existing XDR Schema.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to annotate an existing XDR Schema.

Prerequisites

Before working on this lab, you must understand:

- XDR Schema syntax.
- XML-SQL annotations for multiple tables.
- ADO development by using Visual Basic Scripting Edition.

Lab Setup

To complete this lab, you need a computer running Windows 2000 Server, with SQL Server 2000 and Internet Explorer 5.5 installed.

Scenario

Northwind Traders wants to publish information about the buying habits of each of its customers for internal use by its marketing department. An XML Schema has already been developed. Your task is to map this schema to elements of the database and generate a document.

The root element is <MarketingInfo>, which contains a single <Info> element. The <Info> element contains three independent sets of elements:

- The <Customer> element describes customers.
- The <Product> element describes products in the catalog.
- The <Invoice> element describes invoices raised by customers. Each <Invoice> element contains a set of <Item> elements. Each <Item> element describes one item in the invoice.

The element sets are related through ID fields. Each of these fields has a distinct prefix to ensure uniqueness across the document. The **Customer ID** attributes have a prefix of “cu_”. The **Product ID** attributes have a prefix of “pr_”, and the **invoice ID** attributes have a prefix of “in_”.

- The following is an abbreviated sample of the data:

```
<MarketingInfo>
  <Info>
    <Customer ID="cu_ALFKI"
      Company="Alfreds Futterkiste"
      Contact="Maria Anders"
      DirectMarketing="1"/>
    <Customer ID="cu_ANATR"
      Company="Ana Trujillo Emparedados y helados"
      Contact="Ana Trujillo"
      DirectMarketing="1"/>
    ...
    <Product ID="pr_1" Name="Chai" />
    <Product ID="pr_10" Name="Ikura" />
    ...
    <Invoice ID="in_10248" Date="1996-07-04T00:00:00">
      <Item ProductID="pr_11" Quantity="12" UnitPrice="14" />
    </Invoice>
    <Invoice ID="in_10249" Date="1996-07-05T00:00:00">
      <Item ProductID="pr_14" Quantity="9" UnitPrice="18.6" />
      <Item ProductID="pr_51" Quantity="40" UnitPrice="42.4"/>
    </Invoice>
    ...
  </Info>
</MarketingInfo>
```

The mapping between the elements and attributes of the XML document to the database is as shown in the following tables.

The <Customer> element maps to the Customers table as follows.

Attribute	Database column
ID	CustomerID
Company	CompanyName
Contact	ContactName
DirectMarketing	Not mapped to a database column. See below.

The **DirectMarketing** attribute is a flag used within the marketing department. It does not correspond to any column in the database, and in this exercise you will not retrieve data for this attribute.

The <Product> element is mapped to the Products table as follows.

Attribute	Database column
ID	ProductID
Name	ProductName

The <Invoice> element maps to the Orders table as follows.

Attribute	Database column
ID	OrderID
Date	OrderDate

The <Item> element maps to the [Order Details] table as follows.

Attribute	Database column
ProductID	ProductID
Quantity	Quantity
Unit Price	UnitPrice

Estimated time to complete this lab: 30 minutes

Exercise 1

Annotating the Existing Schema

In this exercise, you will take the existing XDR Schema and provide mappings to the Northwind Traders database. You will use the `sql:relation`, `sql:field`, and `sql:relationship` annotations to map the elements and attributes to the underlying database. You will also use `sql:is-constant` and `sql:map-field` annotations to provide constant elements and specify unmapped elements. In addition, you will use the `id-prefix` annotation to specify unique prefix values for IDs.

► To add annotations to the existing schema

The file `MarketingSchema1.xml` in the folder `<install folder>\Labs\Lab06\Solution` contains a solution to this procedure.

1. Open the schema file `MarketingSchema1.xml` with Notepad from the directory `<install directory>\Labs\Lab06\Starter`.
2. Add the **xml-sql** namespace to the `<Schema>` element.
3. Add the appropriate **sql:relation**, **sql:field**, and **sql:relationship** annotations as necessary.
4. Annotate the `<Info>` element as a constant by using **sql:is-constant**.
5. Annotate the **DirectMarketing** attribute of the `<Customer>` element with **sql:map-field** so that it is not mapped to a database column.
6. Add **sql:id-prefix** annotations to the **ID** attributes of the `<Customer>`, `<Invoice>`, and `<Products>` tables, specifying the appropriate prefix for each (**cu_**, **in_**, or **pr_**). Add a **sql:id-prefix** annotation to the **ProductID** attribute of `<Item>`, specifying the **pr_** prefix.

► To test the schema

- If you have already completed Lab 6.1, “Using Mapping Schemas,” or Lab 6.2, “Mapping Multiple Tables,” the virtual root `Lab06` already exists. If you have not already done so, run the scripting host file `CreateRoot.vbs` in the folder `<install folder>\Labs\Lab06` to create the virtual root. A template has been provided in the file `MarketingTemplate1.xml` in the directory `<install directory>\Labs\Lab06\Starter`. You can test the schema by accessing following URL:

<http://localhost/Lab06/templates/MarketingTemplate1.xml>

Best Practices

Topic Objective

To describe best practices for using annotated schemas.

Lead-in

You should consider the following best practices when using annotated schemas.

- Use schemas to represent business documents
- Explicitly specify `sql:key-fields` when a relationship is used
- Disable caching when developing and testing

*****ILLEGAL FOR NON-TRAINER USE*****

Consider the following best practices when using annotated schemas.

Use Schemas to Represent Business Documents

Schemas are important because they define the documents that your applications will generate. By adding annotations to a schema, you can encapsulate the mapping between these documents and the underlying database.

Explicitly Specify `sql:key-fields` when a Relationship is Used

The `sql:relationship` annotation, which specifies the relationship between two database tables, is not enough to identify the primary key column(s) in all cases. Get into the habit of specifying the `sql:key-fields` annotation.

Disable Caching when Developing and Testing

To ensure that the most up-to-date version of the schema is used in a development environment, make sure that caching is disabled.

Review

Topic Objective

To reinforce module objectives by reviewing key points.

Lead-in

The review questions cover some of the key concepts taught in the module.

- Using Mapping Schemas
- Using Schemas to Retrieve Data
- Mapping Multiple Tables
- Annotating an Existing Schema

*****ILLEGAL FOR NON-TRAINER USE*****

1. What are the purposes of the sql:relation and sql:field annotations?

The sql:relation annotation specifies the database table to which an element in the schema is bound. The sql:field annotation specifies a database column to which an element or attribute in the schema is bound.

2. Name the four attributes of the sql:relationship annotation and state the purpose of each attribute.

sql:key-relation	Primary table
sql:key	Primary key column(s)
sql:foreign-relation	Foreign table
sql:foreign-key	Foreign key columns(s)

3. List three ways of accessing data through a mapping schema.

An ADO application.

An IIS template query.

An IIS schema query.

4. Explain the purpose of the sql:id-prefix annotation.

The sql:id-prefix annotation is used to supply unique textual prefixes for key values that are derived from different database tables.

Module 7: XML for SQL Tools

Contents

Overview	1
Modifying Data with Updategrams	2
Lab 7.1: Using Updategrams	17
Inserting XML Data with the Bulk Load Facility	22
Lab 7.2: Bulk Loading XML Data	31
Best Practices	34
Review	36



Information in this document is subject to change without notice. The names of companies, products, people, characters, and/or data mentioned herein are fictitious and are in no way intended to represent any real individual, company, product, or event, unless otherwise noted. Complying with all applicable copyright laws is the responsibility of the user. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Microsoft Corporation. If, however, your only means of access is electronic, permission to print one copy is hereby granted.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, ActiveX, BackOffice, BizTalk, MSDN, MS-DOS, SQL Server, Visual Basic, Visual C++, Visual InterDev, Visual J++, Visual Studio, Windows, Windows Media, Windows NT, and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Instructor Notes

Presentation:
90 Minutes

This module provides students with an understanding of Updategrams and XML Bulk Load functionality provided in the XML for SQL Web Update.

Lab:
45 Minutes

After completing this module, students will be able to:

- Insert, delete, and update data by using Updategrams.
- Load Extensible Markup Language (XML) data into SQL Server by using the XML Bulk Load facility.

Materials and Preparation

This section provides the materials and preparation tasks that you need to teach this module.

Required Materials

To teach this module, you need Microsoft® PowerPoint® file 2091a_07.ppt.

Preparation Tasks

To prepare for this module, you should:

- Read all of the materials for this module.
- Complete the labs.

Module Strategy

Use the following strategy to present this module:

- **Modifying Data with Updategrams**

Point out that an Updategram is conceptually a specialized form of template. Emphasize the use of Updategrams to enable full data modification over HTTP in a SQLISAPI virtual directory-based application. This means that you can build a full data-enabled Web solution without any Active Server Pages (ASP) code.

- **Inserting XML Data with the Bulk Load Facility**

Emphasize that when inserting large volumes of data into SQL Server tables the Bulk Load facility is more efficient than OpenXML. This is because the Bulk Load facility reads data as a stream, allowing inserts to begin before the entire XML document has been read.

Overview

Topic Objective

To provide an overview of the module topics and objectives.

Lead-in

In this module, you will learn about the additional functionality available in the XML for SQL Web Update.

- **Modifying Data with Updategrams**
- **Inserting XML Data with the Bulk Load Facility**

*****ILLEGAL FOR NON-TRAINER USE*****

For Your Information

You can download the XML for SQL Web Update from the MSDN Web site at <http://msdn.microsoft.com>

In this module, you will learn how to use Updategrams to modify data, and how to use the XML Bulk Load facility.

After completing this module, you will be able to:

- Insert, delete, and update data by using Updategrams.
- Load Extensible Markup Language (XML) data into SQL Server by using the XML Bulk Load facility.

To enable support for Updategrams and XML bulk load functionality in SQL Server 2000, you must download and install XML for SQL Web Update from the MSDN® Web site at <http://msdn.microsoft.com>

◆ Modifying Data with Updategrams

Topic Objective

To introduce the topics in this section.

Lead-in

In this section, we will discuss the use of Updategrams to insert, delete, and update data.

- Using an Updategram to Modify Data
- Structure of an XML Updategram
- Mapping Updategram Values to SQL Data
- Inserting Data with an Updategram
- Deleting Data with an Updategram
- Updating Data with an Updategram
- Using Parameters with an Updategram
- Using Updategrams from a Client Application

*****ILLEGAL FOR NON-TRAINER USE*****

In this section, you will see how to use Updategrams to insert, update, and delete data in a Microsoft® SQL Server™ 2000 database.

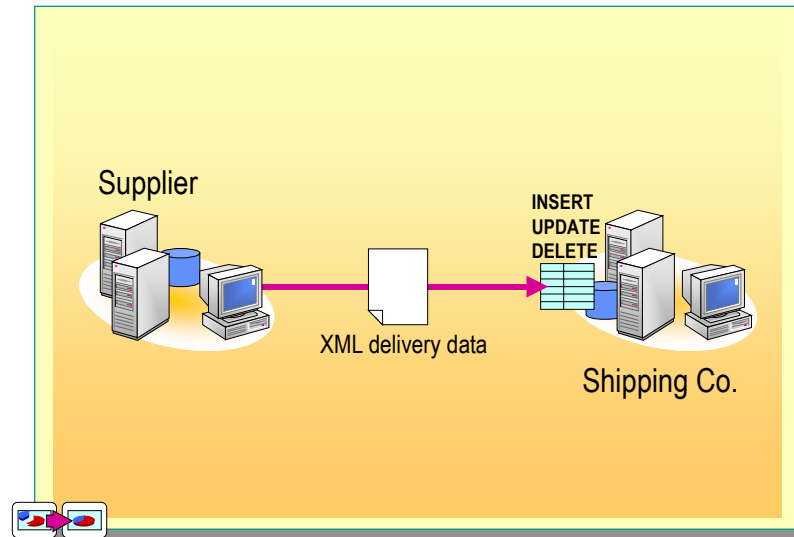
Using an Updategram to Modify Data

Topic Objective

To describe scenarios where you can use Updategrams to modify data.

Lead-in

You can use Updategrams for data modification in a number of different scenarios.



*****ILLEGAL FOR NON-TRAINER USE*****

Similar to template files, *Updategrams* are XML documents that you can submit as commands to SQL Server over a Microsoft ActiveX® Data Objects (ADO) or Hypertext Transfer Protocol (HTTP) connection. The main difference is that templates are used to *retrieve* data, while Updategrams are used to *modify* data.

Use an Updategram for data modifications that must be:

- Performed from a Web site that is based on a SQL Internet Server Application Programming Interface (SQLISAPI). In this case, templates or mapping schemas could be used to retrieve data, and Updategrams could be used to insert, update, or delete data.
- Submitted by an organization to a trading partner over the Internet.

For example, a supplier could send a delivery request to a shipping company by submitting an Updategram that inserts a record into a Deliveries table.

Structure of an XML Updategram

Topic Objective

To introduce Updategrams.

Lead-in

An Updategram is similar to a template, but is used for different purposes.

- Specify the xml-updategram namespace
- Mark transactions within the <sync> element
- Specify “before” and “after” images of data

```
<root xmlns:updg="urn:schemas-microsoft-com:xml-
updategram">
  <updg:sync [mapping-schema=mapping schema]>
    <updg:before>
      XML Fragment
    </updg:before>
    <updg:after>
      XML Fragment
    </updg:after>
  </updg:sync>
</root>
```

*****ILLEGAL FOR NON-TRAINER USE*****

An Updategram specifies what the XML data looks like now and what it will look like after the Updategram is executed.

Specifying the xml-Updategram Namespace

Updategrams must reference the Microsoft xml-Updategram namespace (schemas-microsoft-com:xml-Updategram). This namespace defines the elements and attributes that are used in an Updategram.

You can reference the xml-Updategram namespace by adding an **xmlns** attribute declaration in the root element of the Updategram, as shown in the following example.

Example

```
<root xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
```

Marking Transactions Within the <sync> Element

An Updategram contains one or more <sync> elements. Each <sync> element can contain multiple inserts, updates, or deletes. The failure of one operation in the sync element will cause all other modifications in the same <sync> element to fail.

However, in an Updategram that contains multiple <sync> elements, the failure of one <sync> element does not affect the other <sync> elements in the document.

Specifying “before” and “after” Images of the Data

Represent each data modification by using a <updg:before> and <updg:after> element pair. The <updg:before> element contains an image of the data before the update. The <updg:after> element contains an image of the data after the update.

Mapping Updategram Values to SQL Data

Topic Objective

To describe how to resolve XML data in an Updategram to SQL Server data.

Lead-in

You can resolve the XML data in an Updategram by using a default mapping, or by specifying a mapping schema.

■ Using Default Mapping

- Identical names assumed
- Can be attribute-centric, element-centric, or mixed

■ Specifying a Mapping Schema

- Specify a **mapping-schema** attribute for the <sync> element

*****ILLEGAL FOR NON-TRAINER USE*****

You can resolve XML elements and attributes in a before or after element of an Updategram to the corresponding SQL Server data in two ways.

Using a Default Mapping

If no schema is specified, SQL Server will apply a default (or *implicit*) mapping between the XML data in the Updategram and the data in the database. Consider the following default mapping rules:

- Elements that contain attributes or child elements map to table names.
- XML attributes and child elements map to database columns that have the same name as the attribute or element.

Example

The following Updategram modifies the ProductName column of the Products table:

```
<productupdate
  xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
<updg:sync >
  <updg:before>
    <Products ProductID="1">
      <ProductName>Chai</ProductName>
    </Products>
  </updg:before>

  <updg:after>
    <Products ProductID="1">
      <ProductName>Tea</ProductName>
    </Products>
  </updg:after>
</updg:sync>
</productupdate>
```

Specifying a Mapping Schema

Instead of relying on the default mapping, you can explicitly specify a mapping schema that maps XML data to SQL Server data.

To specify a mapping schema, add a **mapping-schema** attribute to the <sync> element in the Updategram.

Syntax

```
<updg:sync mapping-schema="schema path">
```

Productschema.xml is a schema file that maps XML data to SQL Server data.

Example

```
<productupdate
  xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
  <updg:sync mapping-schema="c:\productschema.xml">
  <updg:before>
    <Product ProductCode="1">
      <ProductName>Chai</ProductName>
    </Products>
  </updg:before>
  <updg:after>
    <Product ProductCode="1">
      <ProductName>Tea</ProductName>
    </Products>
  </updg:after>
  </updg:sync>
</productupdate>
```

Inserting Data with an Updategram

Topic Objective

To describe how to perform INSERT operations with an Updategram.

Lead-in

You can use an Updategram to insert rows into a database.

■ Inserting a Row

- Specify an *after* image of the data

■ Inserting IDENTITY Values

- Use the **at-identity** attribute for IDENTITY columns
- Use **returnid** to retrieve the value used

■ Generating a GUID

- Use the **guid** attribute

*****ILLEGAL FOR NON-TRAINER USE*****

You can use Updategrams to insert rows into a database. This is useful when you must send an XML document to a SQL Server virtual directory, or when you must insert an XML data island on a Web page into a database.

Inserting a Row

You use the `<updg:after>` processing instruction element to specify the data to be inserted. This element is paired with an empty `<updg:before>` element, as shown in the following example.

Example

```
<newproduct
  xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
  <updg:sync>
  <updg:before>
  </updg:before>
  <updg:after>
    <Products ProductID="101">
      <ProductName>Finest Scotch Whisky</ProductName>
    </Products>
  </updg:after>
  </updg:sync>
</newproduct>
```

If the table contains columns that are not specified in the Updategram, the default values for those columns are inserted. If no default values are defined, SQL Server attempts to insert NULL. If NULL values are not permitted in the column, an error occurs and the entire insertion is cancelled.

You can insert multiple rows in a single `<sync>` element by declaring an XML element for each row to be inserted.

Inserting IDENTITY Values

When a table contains an IDENTITY column, SQL Server automatically generates the value of the column for all new records. You can insert an IDENTITY value by using the **at-identity** attribute, and retrieve it by using the **returnid** attribute. This is shown in the following example.

Example

For Your Information

In the Updategram example, the space in Order Details is represented by the encoded value `_x0020_`. This is necessary because the space character is illegal in XML. The UnitPrice value is converted to the SQL Server data type "money" by prefixing it with a "\$" character. This is required for all money columns unless you use a schema declaring the column as XML data type "fixed.14.4".

```
<AddOrder
  xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
  <updg:sync>
    <updg:before>
    </updg:before>
    <updg:after updg:returnid="x">
      <Orders updg:at-identity="x"
        CustomerID="ALFKI"
        OrderDate="01/01/01"/>
      <Order_x0020_Details OrderID="x"
        ProductID="11"
        UnitPrice="$100.0"
        Quantity="1"
        Discount="0.0" />
    </updg:after>
  </updg:sync>
</AddOrder>
```

In this example, a row is inserted into the Orders and Order Details tables. The Order Details table contains an IDENTITY column, which is represented in the Updategram by the **at-identity** value "x". The value that SQL Server automatically inserts is returned in the Updategram response.

```
<AddOrder
  xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
  <returnid>
    <x>11078</x>
  </returnid>
</AddOrder>
```

Generating a GUID

You can generate globally unique identifier (GUID) by specifying a **guid** attribute. For example, the following Updategram inserts a new customer with a unique identifier.

Example

For Your Information

This example will not work with the Northwind Traders database.

```
<NewCustomer
  xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
  <updg:sync >
    <updg:before>
    </updg:before>
    <updg:after>
      <Customers updg:guid="x">
        <CustomerID>x</CustomerID>
        <CustomerName>Graeme</CustomerName>
      </Customers>
    </updg:after>
  </updg:sync>
</NewCustomer>
```

Deleting Data with an Updategram

Topic Objective

To describe how to perform DELETE operations with an Updategram.

Lead-in

You can use an Updategram to delete data from the database.

- Specify a *before* image of the data
- Each *before* child element must identify a single record

```
<DeleteProduct
  xmlns:updg="urn:schemas-microsoft-com:xml-
  updategram">
  <updg:sync>
    <updg:before>
      <Products ProductID="101"/>
    </updg:before>
    <updg:after>
    </updg:after>
  </updg:sync>
</DeleteProduct>
```

*****ILLEGAL FOR NON-TRAINER USE*****

You can use Updategrams to delete data.

Deleting a Single Record

Deleting a single record with an Updategram involves two easy steps:

1. First, specify a before image of the data. The contents of the before image act like a SQL WHERE clause. Each element in the before element must uniquely identify a single record to be deleted.
2. Specify an empty after image.

Example

The following Updategram deletes any record in the Products table matching a Product ID of 101:

```
<DeleteProduct
  xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
  <updg:sync>
    <updg:before>
      <Products ProductID="101"/>
    </updg:before>
    <updg:after>
    </updg:after>
  </updg:sync>
</DeleteProduct>
```

Deleting Multiple Records

You can use a single Updategram to delete multiple records. To do this, you must specify a separate <updg:before> element for each record you want to delete. If the <updg:before> element does not contain enough information to uniquely identify a single record, the deletion will create an error condition. For example, if more than one product has a Category value of “2”, the following element would cause an error:

```
<Products Category="2"/>
```


Updating Data with an Updategram

Topic Objective

To describe how to perform UPDATE operations with an Updategram.

Lead-in

You can use an Updategram to update data in a database.

- Specifying “before” and “after” Images of the Data
- Updating Multiple Rows
 - Use the **id** attribute to match before and after elements

```
<UpdateProduct
  xmlns:updg="urn:schemas-microsoft-com:xml-updategram">
  <updg:sync >
    <updg:before>
      <Products ProductID="101"/>
    </updg:before>
    <updg:after>
      <Products UnitPrice="$752.99"/>
    </updg:after>
  </updg:sync>
</UpdateProduct>
```

*****ILLEGAL FOR NON-TRAINER USE*****

You can use Updategrams to update data in the database.

Specifying “before” and “after” Images of the Data

When you update records, you use the elements identified in the <updg:before> element to specify which rows to update. You use the elements in the <updg:after> element to specify the changes you want to make to the data. Similar to DELETE operations, each element in the <updg:before> element must identify a single row in the database.

Example

The following Updategram could be used to change the price of product 101:

```
<UpdateProduct
  xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
  <updg:sync >
    <updg:before>
      <Products ProductID="101"/>
    </updg:before>
    <updg:after>
      <Products UnitPrice="$102.99"/>
    </updg:after>
  </updg:sync>
</UpdateProduct>
```

Updating Multiple Rows

When you update multiple rows, match the **id** attribute to the corresponding before and after elements.

Example

The following Updategram updates the CategoryID field for products 101 and 102:

```
<UpdateProducts
  xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
  <updg:sync >
    <updg:before>
      <Products updg:id="x" ProductID="101"/>
      <Products updg:id="y" ProductID="102"/>
    </updg:before>
    <updg:after>
      <Products updg:id="x" CategoryID="2"/>
      <Products updg:id="y" CategoryID="2"/>
    </updg:after>
  </updg:sync>
</UpdateProducts>
```

Using Parameters with an Updategram

Topic Objective

To describe how to use parameters in an Updategram.

Lead-in

You can use parameters in an Updategram.

■ Declare parameters in <updg:header>

■ Assign parameters using \$

```
<UpdateProduct
  xmlns:updg="urn:schemas-microsoft-com:xml-
updategram">
  <updg:header>
    <updg:param name="ProductID"/>
    <updg:param name="NewCategory"/>
  </updg:header>
  <updg:sync >
    <updg:before>
      <Products ProductID="$ProductID"/>
    </updg:before>
    <updg:after>
      <Products CategoryID="$NewCategory"/>
    </updg:after>
  </updg:sync>
</UpdateProduct>
```

*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

Point out that parameters in Updategrams are similar to parameters in templates. The main difference is that the "\$" character is used in Updategrams, while the "@" character is used in templates.

As with templates, you can supply an Updategram with one or more parameters to make it more flexible.

Declaring Parameters in the <header> Element

If you decide to use parameters, you must declare each parameter by using a separate <param> element. All of the <param> elements must be nested within the <header> element. The <header> and <param> elements are defined in the xml-Updategram namespace.

Assigning a Parameter

To assign a parameter, you use the dollar sign (\$) character within the <param> element.

The following Updategram could be used to change the CategoryID of a product:

Example

```
<UpdateProduct
  xmlns:updg="urn:schemas-microsoft-com:xml-Updategram">
  <updg:header>
    <updg:param name="ProductID"/>
    <updg:param name="NewCategory"/>
  </updg:header>
  <updg:sync >
    <updg:before>
      <Products ProductID="$ProductID"/>
    </updg:before>
    <updg:after>
      <Products CategoryID="$NewCategory"/>
    </updg:after>
  </updg:sync>
</UpdateProduct>
```

For Your Information

Because the "\$" character is used to indicate both parameters and money values, you cannot pass a parameter of the type "money" unless you use a mapping schema to define the field as "dt:fixed.14.4".

Using Updategrams from a Client Application

Topic Objective

To describe how to use Updategrams from a client.

Lead-in

Executing an Updategram is similar to executing a template.

- **Submitting an Updategram Using an ADO 2.6 Command**
 - Assign to CommandStream or CommandText
- **Publishing an Updategram in a Virtual Name**
 - Access over HTTP similar to a template
- **POSTing an Updategram over HTTP**
 - Must enable Updategram **POST** option

*****ILLEGAL FOR NON-TRAINER USE*****

Executing an Updategram is similar to executing a template.

Submitting an Updategram Using an ADO 2.6 Command

A client application can use an ADO 2.6 **Command** object to submit an Updategram for execution in exactly the same way as a template.

The following example shows how you can use ADO to execute an Updategram:

Example

```
Set objCmd = CreateObject("ADODB.Command")
ObjCmd.ActiveConnection = "Provider=SQLOLEDB; Data
➔Source=server1; Database=stock;Integrated Security=SSPI;"
SQLxml = "<PriceChange xmlns:updg='urn:schemas-microsoft-
➔com:xml-Updategram' >"
    SQLxml = SQLxml & "<updg:sync >"
    SQLxml = SQLxml & " <updg:before>"
    SQLxml = SQLxml & "    <Products ProductID='10' />"
    SQLxml = SQLxml & "</updg:before>"
    SQLxml = SQLxml & "<updg:after>"
    SQLxml = SQLxml & "<Products UnitPrice='$125.99' />"
    SQLxml = SQLxml & "</updg:after>"
    SQLxml = SQLxml & "</updg:sync>"
    SQLxml = SQLxml & "</PriceChange>"
Set objStrmUDgram = CreateObject("ADODB.Stream")
objStrmUDgram.Open
objStrmUDgram.WriteText SQLxml
objStrmUDgram.Position = 0
Set objCmd.CommandStream = objStrmUDgram
Set objStrmResponse = CreateObject("ADODB.Stream")
objStrmResponse.Open
objCmd.Properties("Output Stream").Value = objStrmResponse
objCmd.Execute , , 1024
msgbox objStrmResponse.ReadText
```

Publishing an Updategram in a Virtual Name

You can publish an Updategram in a template virtual name. The Updategram can then be accessed over HTTP like any other template.

Consider that you have created an Updategram and saved it as AddOrder.xml in a folder that is published as a virtual name called “templates”. If this virtual name is published in a virtual directory named “sales” on a server named “webserver1”, the URL used to execute the Updategram is:

Example

<http://webserver1/sales/templates/AddOrder.xml>

POSTing an Updategram over HTTP

You can submit an Updategram through an HTTP **POST**, either from a form on a Web page or from an HTTP-aware client application.

The following HTML form could be used to POST an Updategram:

Example

```
<body>
Enter the CustomerID and Name.
<form action="http://webserver1/sales" method="POST">
<B>Product ID:</B>
<input type="text" name="CustomerID" ><br><B>Price</B><input
type="text" name="CustomerName" >
<input type="hidden" name="contenttype" value="text/xml">
<input type="hidden" name="template" value='
<AddCust xmlns:updg="urn:schemas-microsoft-com:xml-
Updategram">
<updg:header>
  <updg:param name="CustomerID"/>
  <updg:param name="CustomerName" />
</updg:header>
<updg:sync>
  <updg:before>
  </updg:before>
  <updg:after>
    <Customers CustomerID="$CustomerID"
    CustomerName="$CustomerName" />
  </updg:after>
</updg:sync>
</AddCust>
'>
<p><input type="submit">
</form>
</body>
```

Enabling POSTed Updategrams

Before an Updategram can be POSTed, you must enable the option to allow POST queries in the Configure SQL XML Support in IIS tool (where IIS represents Microsoft Internet Information Services). In addition, you must enable the **Allow SQL= or Template=URL Queries** option or the **Allow Posted Updategrams** option. Using **Allow Posted Updategrams** limits the level of access available over HTTP so that clients cannot execute queries in a URL, thereby increasing security.

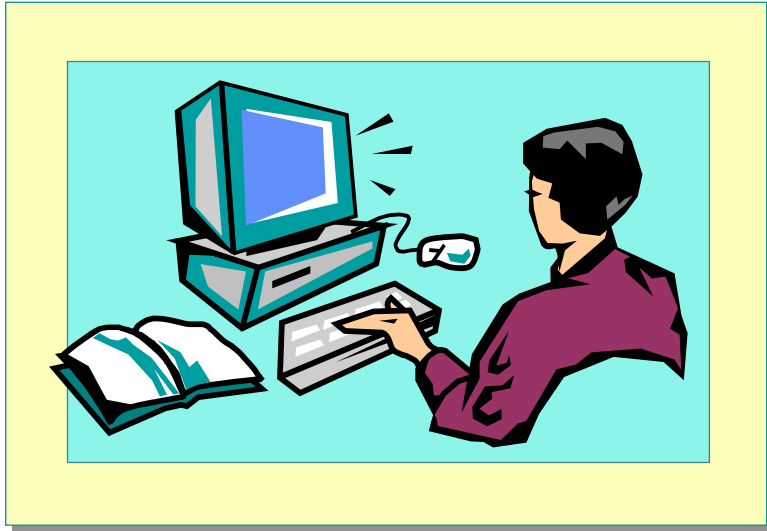
Lab 7.1: Using Updategrams

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will create and use Updategrams to modify data in the Northwind Traders database.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to insert, delete, and update data by using Updategrams.

Prerequisites

Before working on this lab, you must have:

- Knowledge of how to publish an XML template in a virtual name.
- Knowledge of how to submit data from an HTML form to a virtual root.

For More Information

Read the section “Using Updategrams to Modify Data” in the XML for SQL Web Update documentation.

Using the Solution Files

This lab uses an IIS virtual directory that points to the folder *<install folder>\Labs\Lab07\Starter* in order to process template files that contain Updategrams. To run a Solution file, you must copy the file from the folder *<install folder>\Labs\Lab07\Solution* to the folder *<install folder>\Labs\Lab07\Starter*.

Scenario

Data in the Northwind Traders database needs to be inserted, updated, and deleted by a variety of applications. In this lab, you will create solutions that use Updategrams to modify the data in the database.

Estimated time to complete this lab: 30 minutes

Exercise 1

Inserting Data Using an Updategram

In this exercise, you will create an Updategram to add data to the Northwind Traders database.

Scenario

New customers and suppliers who use the Northwind Traders Web site need to be able to register through their browsers. You will create an Updategram that inserts the new customer and supplier data into the database.

► To create a virtual root

- Double-click the script CreateRoot.vbs in the folder *<install folder>\Labs\Lab07*. This script creates a SQL Server virtual directory named Lab07, with the templates virtual name “updategrams”.

► To create an updategram to insert customer data

You can examine the solution for this procedure by viewing the file InsertCustomer.xml in the folder *<install folder>\Labs\Lab07\Solution*.

1. Open the file InsertCustomer.xml with Microsoft Notepad from the folder *<install folder>\Labs\Lab07\Starter*.
2. Between the `<newcustomer>` elements, add XML code to create an updategram with the following features:
 - A `<header>` element containing three parameters named **CustomerID**, **CompanyName**, and **ContactName**.
 - A **sync** element containing an empty `<before>` element and an `<after>` element that will insert a record into the Customers table by using the three parameters for the **CustomerID**, **CompanyName**, and **ContactName** columns.
3. Save the file and close Notepad.

► To use the customer updategram

1. Open the file AddCustomer.htm with Notepad from the folder *<install folder>\Labs\Lab07\Starter*. Note that this page sends three parameters to the InsertCustomer.xml Updategram.
2. Close Notepad and double-click **AddCustomer.htm** to open it with Microsoft Internet Explorer.
3. Confirm that the **Customer ID** and **Company Name** fields are filled in for you, and enter your own name in the **Contact Name** field.
4. Click **Add Customer**. An empty newcustomer XML document should be returned. (Subsequent insertions of the same record will return an error message because the CustomerID field is a primary key).
5. Close Internet Explorer, and use SQL Server Query Analyzer to confirm that the data has been inserted into the Customers table in the Northwind Traders database.

► **To create an updategram to insert supplier data with an IDENTITY column**

You can examine the solution for this procedure by viewing the file InsertSupplier.xml in the folder *<install folder>\Labs\Lab07\Solution*.

1. Open the file InsertSupplier.xml with Notepad from the folder *<install folder>\Labs\Lab07\Starter*.
2. Between the `<newsupplier>` elements, add XML code to create an updategram with the following features:
 - A **header** element containing two parameters named **CompanyName** and **ContactName**.
 - A **sync** element containing an empty `<before>` element and an `<after>` element that will insert a record into the Suppliers table. You should use the **at-identity** attribute to insert a value into the **SupplierID** column, and return the inserted value by using the **returnid** attribute.
3. Save the file and close Notepad.

► **To use the supplier updategram**

1. Open the file AddSupplier.htm with Notepad from the folder *<install folder>\Labs\Lab07\Starter*. Note that this page sends two parameters to the InsertSupplier.xml Updategram.
2. Close Notepad and double-click **AddSupplier.htm** to open it with Internet Explorer.
3. Confirm that the **Company Name** field is filled in for you, and enter your own name in the **Contact Name** field.
4. Click **Add Supplier**. A newsupplier XML document containing a **returnid** attribute with the IDENTITY value used should be returned.
5. Close Internet Explorer and use SQL Server Query Analyzer to confirm that the data has been inserted into the Suppliers table in the Northwind Traders database.

Exercise 2

Deleting Data with an Updategram

In this exercise, you will create an Updategram to delete data from the Northwind Traders database.

Scenario

Customers with no outstanding orders can remove themselves from the Northwind Traders database over the Internet. You must develop an Updategram that enables the deletion of a customer record.

► To add a customer to the database

1. Run the script InsertCustomer.vbs from the folder Lab07\Starter.
2. Use SQL Server Query Analyzer to confirm that a customer with the CustomerID **PLACEHOLDER** has been added to the Customers table in the Northwind Traders database.

► To create an updategram to delete a customer record

You can examine the solution for this procedure by viewing the file DeleteCustomer.xml in the folder *<install folder>\Labs\Lab07\Solution*.

1. Open the file DeleteCustomer.xml with Notepad from the folder *<install folder>\Labs\Lab07\Starter*.
2. Between the `<deletecustomer>` elements, add XML code to create an Updategram that will delete the record from the **Customers** table with the **CustomerID** value that is passed to the updategram as a parameter.
3. Save the file and close Notepad.

► To use the deletecustomer updategram

1. Open the file RemoveCustomer.htm with Notepad from the folder *<install folder>\Labs\Lab07\Starter*. Note that this page sends a **CustomerID** parameter to the **DeleteCustomer.xml** Updategram.
2. Close Notepad and double-click **RemoveCustomer.htm** to open it with Internet Explorer.
3. Confirm that the **CustomerID** field contains the value **PLACEHOLDER**, and click **Unregister Me**. An empty **deletecustomer** XML document should be returned.
4. Close Internet Explorer and use SQL Server Query Analyzer to confirm that the data has been deleted from the customer table in the Northwind Traders database.

Exercise 3

Updating Data with an Updategram

In this exercise, you will create an Updategram to modify data in the Northwind Traders database.

Scenario

Northwind Traders users need to be able to modify their customer details through the Northwind Traders Web site. You must create an Updategram that will allow customers to change their **ContactName** information.

► To create an updategram to update the Customers table

You can examine the solution for this procedure by viewing the file `UpdateCustomer.xml` in the folder `<install folder>\Labs\Lab07\Solution`.

1. Open the file `UpdateCustomer.xml` with Notepad from the folder `<install folder>\Labs\Lab07\Starter`.
2. Between the `<updatecustomer>` elements, add XML code to create an Updategram that accepts two parameters. The Updategram should update the record from the **Customers** table with the **CustomerID** value that is passed to the Updategram as the first parameter, by changing the **ContactName** field to the value passed as the second parameter.
3. Save the file and close Notepad.

► To use the updatecustomer updategram

1. Open the file `UpdateCustomer.htm` with Notepad from the folder `<install folder>\Labs\Lab07\Starter`. Note that this page sends a **CustomerID** parameter and a **ContactName** parameter to the `UpdateCustomer.xml` Updategram.
2. Close Notepad and double-click **UpdateCustomer.htm** to open it with Internet Explorer.
3. Confirm that the **CustomerID** field contains the value "ALFKI", and type your own name in the **ContactName** field.
4. Click **Submit**. An empty **updatecustomer** XML document should be returned.
5. Close Internet Explorer and use SQL Server Query Analyzer to confirm that the data has been updated in the Northwind Traders database.

◆ Inserting XML Data with the Bulk Load Facility

Topic Objective

To introduce the topics in this section.

Lead-in

You can use the Bulk Load facility to insert large volumes of XML data into SQL Server.

- Bulk Loading XML Data Using the Bulk Load Facility
- Using the **SQLXMLBulkLoad** Object
- **SQLXMLBulkLoad** and Mapping Schemas
- Bulk Load Options
- Handling Bulk Load Errors

*******ILLEGAL FOR NON-TRAINER USE*******

You use the Bulk Load facility to insert large volumes of XML data into SQL Server tables. The Bulk Load facility provides a scriptable object to programmatically read an XML file or stream. You then use a mapping schema to insert the data into the appropriate tables in the database.

This section describes the use of the **SQLXMLBulkLoad** object, and discusses issues involved in using this object to bulk load data.

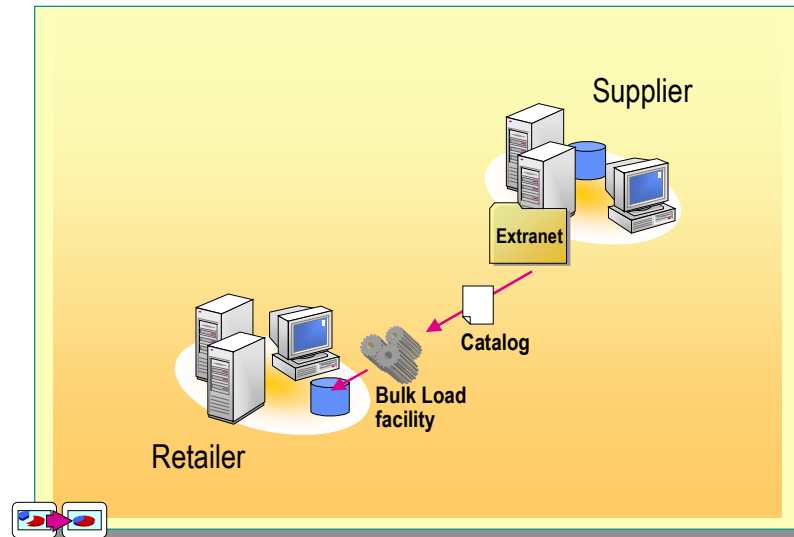
Bulk Loading XML Data Using the Bulk Load Facility

Topic Objective

To describe a scenario where the Bulk Load facility can be used.

Lead-in

You can use the XML Bulk Load facility to import large volumes of XML data into SQL Server.



*****ILLEGAL FOR NON-TRAINER USE*****

Delivery Tip

This slide builds automatically to show a catalog being downloaded from a supplier and imported into a retailer database.

You can restart the build by pressing the LEFT ARROW key on your keyboard.

You can use the XML Bulk Load facility to import large volumes of XML data into SQL Server.

For example, a retailer could download catalog data from a supplier in XML format, and use a script to import the data into a local database.

Consider the following facts about the Bulk Load facility:

- The Bulk Load facility can be used from any COM-aware programming language, including Microsoft Visual Basic® Scripting Edition (VBScript), Visual Basic, Microsoft Visual C++®, and Microsoft Visual J++®.
- Before importing XML data by using the Bulk Load facility, you must define a mapping schema that maps the XML data to columns in the SQL Server database.
- The Bulk Load facility requires an OLE DB connection to SQL Server.

Using the SQLXMLBulkLoad Object

Topic Objective

To describe the SQLXMLBulkLoad object.

Lead-in

You can script the SQLXMLBulkLoad object to import XML data into SQL Server.

■ Creating a SQLXMLBulkLoad Object

```
Set objBL=CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
```

■ Loading Data Using the Execute Method

- Mapping schema and XML source parameters

```
objBL.Execute "c:\productschema.xml", "c:\products.xml"
```

*****ILLEGAL FOR NON-TRAINER USE*****

You can script the **SQLXMLBulkLoad** object to import XML data into SQL Server. This provides a high-performance approach to loading XML data.

Creating a SQLXMLBulkLoad Object

You can instantiate the **SQLXMLBulkLoad** object in any COM-aware language by using the ProgID "SQLXMLBulkLoad.SQLXMLBulkLoad". After you have created the **SQLXMLBulkLoad** object, it must be connected to a database by using one of the following two approaches:

- Assign an OLE DB connection string to the **ConnectionString** property
- Assign an existing ADO **Command** object to the **ConnectionCommand** property.

Loading Data Using the Execute Method

XML data is loaded into the database by calling the **Execute** method of the **SQLXMLBulkLoad** object. This method accepts the following two parameters:

- **bstrSchemaFile** A mapping schema used to map the XML data to tables in the database
- **vDataFile** A file or stream containing the XML data to be imported

The following example shows how you can use VBScript to import XML data into a SQL Server database by using the **SQLXMLBulkLoad** object.

Example

```
Dim objBL
Set objBL=CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBL.ConnectionString = "PROVIDER=sqloledb;" & _
    "DATA SOURCE=server1;" & _
    "INITIAL CATALOG=retaildb" & _
    "INTEGRATED SECURITY=sspi;"
objBL.Execute "c:\productschema.xml", "c:\products.xml"
```

SQLXMLBulkLoad and Mapping Schemas

Topic Objective

To explain how the SQLXMLBulkLoad object handles annotations in mapping schemas.

Lead-in

The SQLXMLBulkLoad object interprets the annotations in the mapping schema.

- **is-constant nodes are ignored**
- **Elements with map-field = "0" are ignored**
 - Optionally loaded into overflow-field column
- **Use the key-ordering rule with relationships**
 - The primary key attribute in the parent should be specified before the child element
 - If an explicit foreign key value is not specified in the child element, the primary key from the parent will be inserted

*****ILLEGAL FOR NON-TRAINER USE*****

The **SQLXMLBulkLoad** object interprets the annotations in the mapping schema and loads the data accordingly. Each time an opening tag for an element that corresponds to a table is read, a record is generated for that table. When the closing tag for the element is read, the row is inserted into the database.

Using is-constant Annotations

If a node in the schema is assigned the attribute **is-constant="1"**, the **SQLXMLBulkLoad** object ignores the node and does not attempt to generate a row for it. This annotation is commonly used for root elements or container elements that do not map to data in the database.

Using map-fields Annotations

Data in the schema annotated with the attribute **map-fields="0"** are not mapped to data in the database. The **SQLXMLBulkLoad** object ignores these fields unless the parent element in the schema has an overflow-field annotation specifying a column for unconsumed XML data. In this case, the data in the unmapped field is inserted into the overflow column.

Using the Key-Ordering Rule

When a schema contains a hierarchical XML representation of a relationship between SQL Server tables, the child element may not include the foreign key value required in the corresponding table.

In any primary key table element containing a foreign key table element, the primary key field should be declared before the child element. This allows the **SQLXMLBulkLoad** object to insert the primary key from the parent element into the foreign key column in the table represented by the child element.

If the child element contains a value for the foreign key field, the value in the child element is inserted into the table. Otherwise, the primary key from the parent table is used.

For example, the following XML document and schema could be used to bulk load data into the Categories and Products tables. Because the <Product> element does not include the CategoryID value, the primary key from the <Category> element is used.

Example Document

```
<?xml version="1.0"?>
<Catalog>
  <Category CategoryID="1" CategoryName="Computers">
    <Product ProductID="1" ProductName="Desktop Computer"/>
    <Product ProductID="2" ProductName="Laptop Computer"/>
  </Category>
  <Category CategoryID="2" CategoryName="Monitors">
    <Product ProductID="5" ProductName="15 inch Monitor"/>
    <Product ProductID="6" ProductName="17 inch Monitor"/>
  </Category>
</Catalog>
```

Example Schema

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">

  <ElementType name="Catalog" sql:is-constant="1">
    <element type="Category"/>
  </ElementType>

  <ElementType name="Category" sql:relation="Categories">
    <AttributeType name="CategoryID"/>
    <AttributeType name="CategoryName"/>

    <attribute type="CategoryID" sql:field="CategoryID"/>
    <attribute type="CategoryName" sql:field="CategoryName"/>
    <element type="Product">
      <sql:relationship
        key-relation="Categories"
        key="CategoryID"
        foreign-key="CategoryID"
        foreign-relation="Products"/>
    </element>
  </ElementType>

  <ElementType name="Product" sql:relation="Products">
    <AttributeType name="ProductID"/>
    <AttributeType name="ProductName"/>

    <attribute type="ProductID" sql:field="ProductID"/>
    <attribute type="ProductName" sql:field="ProductName"/>
  </ElementType>

</Schema>
```

Note For information about how other annotations are handled by the **SQLXMLBulkLoad** object, see the XML for SQL documentation.

Bulk Load Options

Topic Objective

To explain some of the options available when using the **SQLXMLBulkLoad** object.

Lead-in

You can use a number of properties to set options when using the **SQLXMLBulkLoad** object.

- **Generate the database schema**
 - **SGDropTables** property
 - **SchemaGen** property
 - **SGUseID** property
- **Assign NULL with the KeepNulls property**
- **Assign IDENTITY values with the KeepIdentity property**

*****ILLEGAL FOR NON-TRAINER USE*****

The **SQLXMLBulkLoad** object has a number of properties that you can use to control its behavior. This provides a number of useful options when bulk loading data.

Generating the Database Schema

By default, the **SQLXMLBulkLoad** object is used to import data into existing database tables. You can change this behavior by setting the following properties:

- **SchemaGen**

Setting this property to **True** causes the tables mapped by the schema to be created, if they do not already exist.

- **SGDropTables**

Setting this property to **True** (together with the **SchemaGen** property) causes all tables mapped by the schema to be dropped and recreated.

- **SGUseID**

Setting this property to **True** causes primary key constraints to be created based on any **id** attributes defined in the schema.

In addition, setting the **BulkLoad** property to **False** causes the creation of the tables mapped in the schema, without importing any data.

Assigning NULL Values

By default, if a column in a table has no matching element or attribute in the XML document that is being loaded, the default value for the column is inserted. If no default value is defined, NULL is inserted. Setting the **KeepNulls** property to **True** causes a NULL value to be inserted for unmapped columns, even if they have default values defined.

Assigning IDENTITY Values

By default, any values in an XML document that map to IDENTITY columns are inserted instead of the automatically generated IDENTITY value. If you want to ignore the value in the XML document and use the system-generated value, you must set the **KeepIdentity** property to **False**.

Note The **KeepIdentity** property is an “all-or-nothing” option. If you set **KeepIdentity** to **False**, automatic values are generated for all tables with IDENTITY columns.

Example

The following bulk load process drops and recreates the table, inserts NULL for any unspecified columns, and uses the system-generated IDENTITY values:

```
Dim objBL
Set objBL=CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
With objBL
    .ConnectionString ="PROVIDER=sqlolddb;" & _
        "DATA SOURCE=server1;" & _
        "INITIAL CATALOG=retaildb" & _
        "INTEGRATED SECURITY=sspi;"

    .SchemaGen=True
    .SGDropTables=True
    .KeepNulls=True
    .KeepIdentity=False
End With
objBL.Execute "c:\productschema.xml", "c:\products.xml"
```

Handling Bulk Load Errors

Topic Objective

To discuss how to handle bulk load errors.

Lead-in

Errors can occur during a bulk load process.

■ Logging Errors

- Always specify the `ErrorLogFile` property

■ Maintaining Relational Integrity

- `IgnoreDuplicateKeys` property
- `CheckConstraints` property

■ Using Transactions

- `Transaction` property

*****ILLEGAL FOR NON-TRAINER USE*****

Be aware that errors can occur during a bulk load process. Errors can happen as a result of incorrectly defined schemas, invalid XML data, or violations of the database's referential integrity. You can control the way the **SQLXMLBulkLoad** object handles errors in a number of ways.

Logging Errors

You should always specify the **ErrorLogFile** property when using the **SQLXMLBulkLoad** object. This causes any errors to be logged to a text file.

Maintaining Referential Integrity

To enforce referential integrity, you can control error-handling behavior by using the following properties:

■ IgnoreDuplicateKeys

By default, an attempt to insert a duplicate primary key results in an error, and the entire bulk load operation is stopped. Setting the **IgnoreDuplicateKeys** property to **True** results in the failure of the individual insertion without stopping the entire import.

■ CheckConstraints

Setting the **CheckConstraints** property to **True** causes primary/foreign key constraints to be enforced during insertions. Constraint violation results in an error.

Using Transactions

You can use transactions to ensure the robustness of the import process. When a transaction is used, the bulk load operation either completely succeeds or completely fails.

The following properties are used to control transactional behavior of the **SQLXMLBulkLoad** object:

■ Transaction

Setting this property to **True** causes a transaction to be used. Each row is stored in a temporary file until the bulk load process is complete. The entire import process is rolled back if the operation fails.

■ TempFilePath

This property is used to control the location of the temporary file used when the **Transaction** property is set to **True**. When bulk loading on the local computer, this should be a local file path. When importing data to a remote server a UNC path (`\\servername\sharename`) must be used.

Note Using transactions causes a performance overhead. As a result, a transactional importing of a large volume of data will take longer than a non-transactional importing of the same data.

The following example shows how you can handle errors when using the **SQLXMLBulkLoad** object.

Example

```
Dim objBL
Set objBL=CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
With objBL
    .ConnectionString ="PROVIDER=sqloledb;" & _
        "DATA SOURCE=server1;" & _
        "INITIAL CATALOG=retaildb" & _
        "INTEGRATED SECURITY=sspi;"
    .ErrorLogFile="c:\importerrors.log"
    .IgnoreDuplicateKeys=True
    .CheckConstraints=True
    .Transaction=True
    .TempFilePath="\\Server1\TempShare"
End With
objBL.Execute "c:\productschema.xml", "c:\products.xml"
```

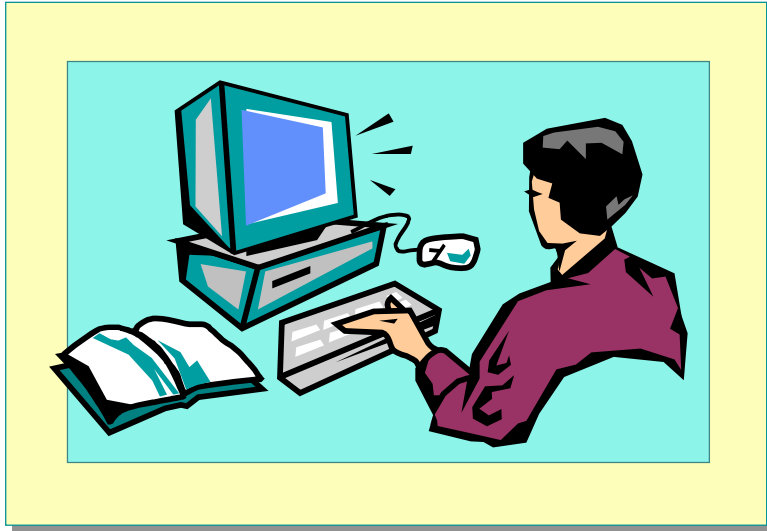
Lab 7.2: Bulk Loading XML Data

Topic Objective

To introduce the lab.

Lead-in

In this lab, you will load XML data into SQL Server by using the XML Bulk Load facility.



*****ILLEGAL FOR NON-TRAINER USE*****

Explain the lab objectives.

Objectives

After completing this lab, you will be able to load XML data into SQL Server by using the XML Bulk Load facility.

Prerequisites

Before working on this lab, you must have an understanding of annotated XML-Data Reduced (XDR) Schemas.

For More Information Read the section “Performing Bulk Load of XML” in the XML for SQL Web Update documentation.

Scenario

Some data in the Northwind Traders database must be regularly loaded from XML files. For example, new employee records are loaded from XML personnel records, and the catalog is updated by importing category and product data from an XML catalog file.

Estimated time to complete this lab: 15 minutes

Exercise 1

Importing Data from an XML File

In this exercise, you will use the XML Bulk Load facility to import employee data from an XML file.

Scenario

When new employees join Northwind Traders, their employee details are stored in a personnel database. After the employees' initial training, these details are exported to an XML file to be loaded into the Northwind Traders database.

► To examine the schema and data files

1. Open the file `PersonnelSchema.xml` with Internet Explorer from the folder `<install folder>\Labs\Lab07\Starter`. This file defines the schema for a list of employees.
2. Open the file `NewEmployees.xml` with Internet Explorer from the folder `<install folder>\Labs\Lab07\Starter`. This file contains a list of employees based on the `personnelschema.xml` schema.
3. Close Internet Explorer.

► To import the employee data by using the XML Bulk Load facility

You can examine a solution to this procedure by viewing the file `ImportEmployees.vbs` in the folder `<install folder>\Labs\Lab07\Solution`.

1. Open the file `ImportEmployees.vbs` file with Notepad from the folder `Lab07\Starter`. Note that a utility routine to identify the current directory is provided.
2. Find the comment **Add Your Code Here**, and add code to do the following tasks:
 - Instantiate the `objBulkLoad` variable as a **SQLXMLBulkLoad** object.
 - Set the **ConnectionString** property to connect the component to the Northwind Traders database on the local server by using Integrated Security.
 - Use the `strSchemaFile` and `strXMLFile` variables to execute the bulk load process.
3. Save the file and close Notepad.
4. Double-click the script **ImportEmployees.vbs** to execute it.
5. Use SQL Server Query Analyzer to confirm that the records have been imported into the **Employees** table in the Northwind Traders database.

Exercise 2

Importing Data and Generating a Database Schema

In this exercise, you will import data by using the Bulk Load facility while dropping the necessary tables and generating the database schema.

Scenario

The Northwind Traders catalog needs to be periodically updated. The new catalog data is represented in an XML document, which must be imported into the Categories and Products tables. The existing tables must be dropped and recreated as part of the import process.

► To examine the schema and data files

1. Open the file CatalogSchema.xml with Internet Explorer from the folder *<install folder>\Labs\Lab07\Starter*. This file defines the schema for a list of categories and products. (The data maps to tables named NewCategories and NewProducts, respectively.)
2. Open the file CatalogData.xml with Internet Explorer from the folder *<install folder>\Labs\Lab07\Starter*. This file contains a list of categories and products based on the CatalogSchema.xml schema. (The large binary fields are category pictures.)
3. Close Internet Explorer.

► To import the catalog data and generate the database schema

You can examine a solution to this procedure by viewing the file ImportCatalog.vbs in the folder *<install folder>\Labs\Lab07\Solution*.

1. Open the file ImportCatalog.vbs with Notepad from the folder Lab07\Starter.
2. Find the comment **Add Your Code Here**, and add code to do the following tasks:
 - Instantiate the objBulkLoad variable as a **SQLXMLBulkLoad** object.
 - Set the **ConnectionString** property to connect the component to the Northwind Traders database on the local server by using Integrated Security.
 - Set the appropriate properties to generate the database schema, dropping existing tables if necessary.
 - Use the strSchemaFile and strXMLFile variables to execute the bulk load process.
3. Save the file and close Notepad.
4. Double-click the script **ImportCatalog.vbs** to execute it.
5. Use SQL Server Query Analyzer to confirm that the records have been imported into newly created **NewCategories** and **NewProducts** tables in the Northwind Traders database.

Best Practices

Topic Objective

To describe best practices for using Updategrams and the Bulk Load facility.

Lead-in

Consider these best practices when using Updategrams or the Bulk Load facility.

- **Consider Updategram concurrency issues**
- **Allow POSTed Updategrams but not queries**
- **Define data types for date/time or UUID fields**

*****ILLEGAL FOR NON-TRAINER USE*****

Consider the following best practices when using Updategrams or the Bulk Load facility.

Consider Updategram Concurrency Issues

When updating records with an Updategram, you might want to address concurrency issues. SQL Server uses the information in the <updg:before> element to prevent overwriting of changes made to records by other users since the Updategram was submitted. You can control concurrency behavior by using the following techniques:

- Specify only the primary key in the <updg:before> element. This causes all changes to the record by other users (other than deletion) to be overwritten.
- Specify the primary key and any fields that are to be updated in the <updg:before> element. Any changes made by other users to the fields affected by the Updategram will cause the update to fail.
- Specify all fields in the <updg:before> element. Changes made by other users to any fields in the record will cause the update to fail.

Allow POSTed Updategrams

When you use an IIS virtual directory to publish a database, you can restrict the application of POST to only allow Updategrams to be POSTed and not URL queries. Restricting the use of POST to apply only to Updategrams provides a higher level of security than allowing any query to be executed over HTTP (although it does allow any update to be performed by using an Updategram). To allow POSTed Updategrams, set the **allow posted Updategrams** option for the virtual directory.

Define Data Types for Date/Time and UUID Fields

When defining schemas that will be used with the Bulk Load facility, any fields that contain date/time or UUID values should be defined by using the XML **dt:type** and **sql:datatype** attributes. This ensures that the correct data type is entered into the database.

Review

Topic Objective

To reinforce module objectives by reviewing key points.

Lead-in

The review questions cover some of the key concepts taught in the module.

- **Modifying Data with Updategrams**
- **Inserting XML Data with the Bulk Load Facility**

*****ILLEGAL FOR NON-TRAINER USE*****

1. You want to use an Updategram to modify data in a SQL Server database. How can you map elements and attributes in the Updategram to tables and columns in the database?

Use the default mapping, where the names of elements and attributes must match the names of the corresponding tables and columns, or specify a mapping schema.

2. How can you use an Updategram to insert a row into a table with an IDENTITY column, and retrieve the identity value used in the HTTP response?

Create an Updategram containing an <updg:after> element with no corresponding <updg:before> element. Use the at-identity attribute to insert the identity column, and the returnid attribute to receive the inserted identity value.

3. How can you match multiple <updg:before> elements with their corresponding <updg:after> elements?

Use the id attribute to assign a unique value to each element pair.

4. You want to use the Bulk Load facility to import data into a SQL Server database. What properties should you set to ensure that the tables are dropped and recreated?

SGDropTables and SchemaGen should both be set to TRUE.