Pro SQL Server 2008 XML

ISBN-13: 978-1-59059-983-9

ISBN-10: 1-59059-983-7

ISBN-13 (electronic): 978-1-4302-0630-9

ISBN-10 (electronic): 1-4302-0630-6

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jonathan Gennick
Technical Reviewer: Fabio Claudio Ferracchiati
Editorial Board: Clay Andres, Steve Anglin, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Matthew Moodie, Joseph Ottinger, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Kylie Johnston
Copy Editor: Kim Benbow
Associate Production Director: Kari Brooks-Copony
Production Editor: Liz Berry
Compositor/Artist: Kinetic Publishing Services, LLC
Proofreader: April Eddy
Indexer: Becky Hornyak
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski
Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at http://www.apress.com/info/bulksales.

The source code for this book is available to readers at http://www.apress.com.

CHAPTER 1

Enter XML

Welcome to *Pro SQL Server 2008 XML*. This book will cover the basics and advanced topics of SQL Server–based XML development, including a review of legacy XML support in prior versions of SQL Server, and a discussion of new features introduced in SQL Server 2005 and SQL Server 2008. Throughout this book I will discuss the implementation of several advanced XML standards via SQL Server, including XPath, XQuery, XSLT, XML Schema, and XML DML. I will also provide coverage of other advanced XML-related topics, like SQLCLR and client-side .NET XML capabilities and Microsoft's .NET LINQ to XML technology.

Throughout this book, I will provide step-by-step code samples to demonstrate the concepts presented. All samples are designed to run with the Microsoft AdventureWorks sample database, unless otherwise noted. The sample code from this book is available for download at the Apress web site (`www.apress.com/download`).

ADVENTUREWORKS SAMPLE DATABASE

As mentioned, the code samples in this book are designed to be run against the Microsoft AdventureWorks sample database, unless otherwise specified in the text. Microsoft has decided to offer the SQL Server 2008 version of the AdventureWorks sample database and applications on its CodePlex web site. The URL is `www.codeplex.com/MSFTDBProdSamples`. If you don't yet have the AdventureWorks database installed on a test server, I highly recommend that you visit CodePlex and download it. The AdventureWorks database and business scenarios are documented on the MSDN web site at `msdn2.microsoft.com/en-us/library/ms124501.aspx`.

In this chapter, I'll provide a brief background of XML in SQL Server, a quick primer on the World Wide Web Consortium (W3C) XML Recommendation, a comparison of XML to other data formats, and a brief overview of XML functionality in SQL Server 2008.

Looking Back at SQL Server XML

When SQL Server 2000 was released, Microsoft was just beginning a big push to thoroughly immerse its entire product line in XML. XML was integrated into SQL Server 2000 by adding the `FOR XML` clause to the `SELECT` statement and adding access to various Component Object Model (COM) components via stored procedures and functions. Some XML support was

provided through integration with Internet Information Services (IIS). XML data in SQL Server 2000 was truly a second-class citizen, driving many developers to avoid using SQL Server for all but the simplest of XML storage and retrieval tasks. The main problems with SQL Server 2000 XML support included the following:

- **Limited functionality.** SQL Server 2000 XML support was provided primarily by the `FOR XML` clause of the `SELECT` statement, the `OPENXML` function, and a couple of stored procedures to create XML documents in memory and remove them when finished. There was no built-in Transact-SQL (T-SQL) support for querying or modifying XML data.
- **Complicated to use.** SQL Server 2000 XML support relied on the old-style Large Object (LOB) data types, including `TEXT` and `NTEXT`. SQL Server 2000 LOB data types were kludgy at best.
- **Inefficient implementation.** SQL Server 2000 XML support also relied heavily on COM components and external libraries, making it much less efficient than a "native" solution. Additionally, if you failed to explicitly remove a document from memory, you were likely to cause more than a few server-side memory leaks.

SQL Server 2008 ups the ante by providing efficient native T-SQL support for XML, with XML-centric improvements to T-SQL statements, built-in XPath, XQuery, and XML DML support, the native `xml` data type, XML indexes, XML views, and more. SQL Server 2008's SQLCLR integration can also help make XML manipulation even more flexible as you'll see in Chapter 8. This book is an in-depth exploration of SQL Server 2008's powerful XML functionality.

What Is XML?

No discussion of SQL Server XML capabilities would be complete without a discussion of the underlying technology, XML. In this section, I'll discuss the W3C XML Recommendation.

XML is designed to be a simple, fast, and flexible text format derived from Standard Generalized Markup Language (SGML), as defined by the ISO (International Organization for Standardization) 8879 standard. The XML Recommendation, and its related recommendations, are maintained by the W3C, a standards body with the mission of developing interoperable technologies for the World Wide Web. The W3C XML 1.0 specification defines a set of rules for adding structure and context to data through the use of markup. In addition to the XML 1.0 specification, the W3C has proposed dozens of additional XML-based specifications to standardize data transfer and sharing between applications, XML processing, querying, sharing, and manipulation. The latest versions of the XML 1.0 and XML 1.1 Recommendations are available at `www.w3.org/TR/xml` and `www.w3.org/TR/xml11`, respectively.

Work on the XML recommendation initially began in 1996, when it was chartered by the W3C. Design work on XML continued through 1997, and XML 1.0 became a formal W3C Recommendation in early 1998. Though largely defined as a subset of SGML, XML 1.0 also adapted technology from various other sources, including the Text Encoding Initiative (TEI), Hypertext Markup Language (HTML), and Extended Reference Concrete Syntax (ERCS), among others. During the creation of the XML recommendation, the ISO SGML standard was updated to maintain consistency with XML. The XML 1.1 Recommendation adds support for additional character sets, additional encodings, and extended support for control characters in currently supported encodings. Generally speaking, unless you have a specific need for the capabilities of XML 1.1 (such as Unicode 2.0 or Extended Binary Coded Decimal Interchange Code [EBCDIC]

control character support), it is recommended that you use XML 1.0. SQL Server 2008 supports the XML 1.0 Recommendation.

RECOMMENDATIONS VS. STANDARDS

In W3C terms, a "recommendation" is equivalent to a "standard" put forth by a sanctioned standards organization like ISO or ANSI (American National Standards Institute). Presumably the W3C chose to describe their work in terms of recommendations instead of standards because the W3C is a voluntary organization with no power to enforce acceptance of their standards. Individuals, organizations, and standards bodies are free to accept or ignore W3C recommendations at will. XML and its related recommendations have, however, been adopted as industry standards.

The XML 1.0 Recommendation was created with a very specific set of design goals in mind. The following is a summary of these goals:

- **XML should be easy to process.** XML can be processed with simple custom-made string parsers, although there are a wide variety of prebuilt parsers freely available to facilitate XML processing.
- **XML should be straightforward to use over the Internet.** XML is created with plain text, generally using one of several predefined standardized character sets (UTF-8, UTF-16, and so on). It is designed for easy transmission through firewalls and over the Internet using standardized Internet protocols like HTTP (Hypertext Transfer Protocol). Binary formatted data often requires heavy manipulation for transmission using HTTP (or other protocols) over the Internet.
- **XML should be human legible.** XML is, by its plain-text nature and self-documenting structure, easy for humans to read. This makes debugging problematic XML easier than debugging binary data files.
- **XML should be easy to create.** Unlike proprietary binary data formats, XML can be easily created by anyone with a simple text editor, a more complex XML-specific editor, or an automated process.
- **XML should support a wide variety of applications.** Because it is designed to be flexible and extensible, and because it inherently supports international character sets, a very wide variety of applications can use XML.
- **XML standards should be formal and concise.** This design goal was created to ensure that the XML standard was "programmer-friendly." The idea behind this goal required eliminating "consultant-speak" and "pretty-talk" from the standard and instead providing formal notations and syntax that XML implementers could use to create standardized products quickly and efficiently.
- **XML should be compatible with SGML.** XML was designed with cooperation of the SGML standard committee, so XML is compatible with the SGML standard. In fact, compatibility was so important that some portions of the SGML standard were changed to ensure compatibility during development of the XML standard.

- **XML should have a minimal amount of optional features, ideally zero.** This design goal was a response to SGML's history of adding optional features in an attempt to make SGML as general-purpose as possible. The problem is that these optional features often made document interchange impossible. XML eliminates these types of optional features; any XML parser should be able to read any XML document as long as it follows the standard.
- **XML should be designed and standardized quickly.** This goal was adopted in order to put a standard in place before the big software developers adopted a wide range of conflicting proprietary standards to accomplish similar goals. Work began on XML in mid-1996, with the first working draft presented later that year. XML 1.0 was adopted as a W3C Recommendation in early 1998.

■ **Note** I've presented these design goals in order of importance, as I see it anyway, for SQL Server–based XML programmers. This order differs from the order in which they are presented in the XML Recommendation and may not reflect others' view of the importance of each.

To meet these goals, the designers of XML made several design decisions, including the decision that terseness of marked up XML was not important. This directly contradicts many other formats that strive to store their data in terse and compact formats. As a result, XML data often has a proportionately large quantity of markup information included in it. The nature of XML data, however, does tend to make it highly compressible by modern data compression algorithms, if storage space is a high priority.

Defining XML Data

XML data is composed of several types of items:

- The Document Type Definition (DTD) is a special structure used to define entity declarations and structure validation information (note that SQL Server XML does not support the validation aspect).
- XML elements are containers for character data (CDATA) content in XML documents. Each XML element is defined by matching start and end tags used to encapsulate other XML elements and data. XML elements provide structure to XML data.
- XML attributes are closely tied to elements. Attributes provide additional context, content, and metadata to your XML markup data.
- XML comments are denoted by <!-- and --> delimiters. XML provides support for comments to allow developers to add human-readable documentation to their XML data.
- XML processing instructions are marked by <? and ?> delimiters. A processing instruction is a means to provide additional metadata to a processing application.
- XML character references and entity character references are constructs that allow you to insert special characters in your XML data.

Consider the simple XML document in Listing 1-1, which represents a simple selection of high-grossing movies in XML format.

Listing 1-1. *Sample Movies XML Document*

```
<?xml version="1.0" encoding="UTF-16"?>
<!-- High-grossing movie listing -->
<movies>
    <film>
        <?style superhero?>
        <name>Spider-Man</name>
        <releaseDate>2002-05-03-05:00</releaseDate>
        <gross area="world-wide">821706375.00</gross>
        <gross area="domestic">403706375.00</gross>
        <director>Sam Raimi</director>
        <cast>
            <actor>Maguire, Tobey</actor>
            <actor>Dafoe, Willem</actor>
            <actor>Dunst, Kirsten</actor>
        </cast>
    </film>
    <film>
        <?style superhero-sequel?>
        <name>Spider-Man 2</name>
        <releaseDate>2004-06-30-05:00</releaseDate>
        <gross area="world-wide">783924485.00</gross>
        <gross area="domestic">373585825.00</gross>
        <director>Sam Raimi</director>
        <cast>
            <actor>Maguire, Tobey</actor>
            <actor>Franco, James</actor>
            <actor>Dunst, Kirsten</actor>
            <actor>Molina, Alfred</actor>
        </cast>
    </film>
    <film>
        <?style superhero-sequel?>
        <name>Spider-Man 3</name>
        <releaseDate>2007-05-04-05:00</releaseDate>
        <gross area=”world-wide”>888977494.00</gross>
        <gross area=”domestic”>336027292.00</gross>
        <director>Sam Raimi</director>
        <cast>
            <actor>Maguire, Tobey</actor>
            <actor>Dunst, Kirsten</actor>
            <actor>Franco, James</actor>
            <actor>Church, Thomas Haden</actor>
        </cast>
```

```
    </film>
</movies>
```

This simple example of a well-formed XML document includes several elements, including the root element `<movies>`, the `<film>` elements nested within it, and the subelements nested within them. This hierarchical structure is standard fare for XML, although XML data does not have to have a strongly regular structure as in the example. For instance, you could have `<cast>` (or other) elements outside of the `<film>` elements if it made sense for your application.

The example also includes XML comments, which are included within the `<!--` and `-->` comment indicators. XML comments are nodes that are processed by XML parsers like other XML nodes. Comment nodes are normally not used by applications during processing because they contain human-readable comments. Finally, the example also contains processing instructions that can be used by the application during processing.

XML Requirements

The sample XML in Listing 1-1 does not include any declarations. Declarations are created via DTDs. SQL Server supports a very small subset of DTDs, allowing you to expand entity references in your XML data and assign default values to attributes. I will discuss DTDs further in Chapter 3.

■ **Note** SQL Server does not use DTDs to constrain the format of XML data or the content of elements and attributes. This is a common usage of DTDs as defined by the W3C XML 1.0 Recommendation. To constrain the content and structure of your XML, use XML schema collections instead, which are a much more powerful solution.

I also did not include character references in the example. Character references come in two forms: *character entity references* and *numeric character references*. XML defines a small set of predeclared character entity references so that you can include otherwise reserved characters in your XML data. The process of converting special characters in XML to character references is known as *entitizing*. XML data that contains nonentitized special characters will cause XML parsers to reject the XML during processing. Table 1-1 lists the predeclared character entity references supported by XML.

Table 1-1. *XML Predeclared Character Entity References*

Entity	Description
`&amp;`	The `&amp;` entity is used when you want to include the ampersand (`&`) in your XML data.
`&lt;`	The `&lt;` entity is used when you want to include the less-than sign (`<`) in your XML data.
`&gt;`	The `&gt;` entity is used when you want to include the greater-than sign (`>`) in your XML data.
`&apos;`	The `&apos;` entity is used when you want to include the apostrophe (`'`) in your XML data.
`&quot;`	The `&quot;` entity is used when you want to include the quotation mark (`"`) in your XML data.

Numeric character references look similar to character entity references, but instead of a name, they are represented by &# followed by a decimal or hexadecimal number and a semicolon. Numeric character references can be used to represent any valid Unicode character, even those that already have a predeclared character entity reference. The less-than character (<) can be represented using any of the following character references in your XML data:

```
&lt;
&#60;
&#x3c;
```

The first character reference is the predeclared character entity reference for the less-than sign. The second is the decimal numeric character reference for the same character. The final example shows the hexadecimal version of the numeric character reference for that character. Notice that the hexadecimal version has a lowercase x character between the number sign (#) and the first hexadecimal digit of the character reference. The lowercase x indicates that the numeric character reference is hexadecimal instead of decimal. Table 1-2 shows a small selection of common numeric character references.

Table 1-2. *Sample of Common Numeric Character References*

Description	Symbol	Code
quote	"	"
ampersand	&	&
less-than	<	<
greater-than	>	>
cent sign	¢	¢
pound symbol	£	£
yen symbol	¥	¥
copyright	©	©
registered	®	®
degrees	°	°
plus-minus	±	±
superscript-2	²	²
superscript-3	³	³
fraction-1/4	¼	¼
fraction-1/2	½	½
fraction-3/4	¾	¾
times	×	×
divide	÷	÷
pi	π	π
ndash	–	–
mdash	—	—
euro symbol	€	€
trademark	™	™

XML VS. HTML

HTML coders will recognize similarities between XML and HTML immediately. Both are markup languages based on subsets of the grand-daddy of markup languages, SGML. Their common ancestry means they share similar element, attribute, and comment delimiters. In both markup languages elements define the overall structure of the document. That's where the similarity ends, however.

These two markup languages are designed for completely different purposes. The purpose of HTML is to format data for display. HTML is a standard with predefined tags and attributes, is not case sensitive, and does not preserve white space. XML, on the other hand, is designed to format and structure data. XML carries no requirement to carry additional formatting information, and it has no predefined tags—you create your own tags based on your XML application. XML is also case sensitive and preserves white space.

The Extensible HTML (XHTML) Recommendation is an XML application that redefines HTML in terms of XML. Because it is based on XML, XHTML is stricter than plain HTML in terms of structure, format, and case sensitivity; although XHTML does support less strict validation modes for backward-compatibility purposes.

Well-Formed and Valid XML

XML data comes in one of two basic forms. XML data can be represented as a fragment or it can be a well-formed document. The SQL Server `xml` type can handle both forms of XML data automatically. XML data must meet the following criteria to be considered well-formed:

1. The XML data must contain one or more elements.
2. The XML data must contain one, and only one, root element. This is the element that contains all other elements within the XML document.
3. All elements within the XML document must be properly nested within one another.

XML structure and content can further be constrained by assigning an `xml` instance to an XML schema collection. The XML schema collection contains XML schemas that are defined per the W3C XML Schema Recommendation. XML schemas provide a flexible and powerful tool for constraining XML data. I will detail SQL Server support for the W3C XML Schema Recommendation in Chapter 4.

■ **Note** Although the XML recommendation specifies using DTDs for simple XML document validation, SQL Server supports only a limited subset of DTDs. SQL Server does not support DTD XML structure and content validation.

Considering Other Formats

Although this book is about SQL Server XML functionality, I don't want you to walk away with the idea that XML is the only game in town. The sample data I provided in Listing 1-1 can always be represented in a more terse and compact format (remember in XML, terseness is not considered important). For instance, the same data represented in the common Comma Separated Values (CSV) format used by Microsoft Excel might look like Listing 1-2.

Listing 1-2. *Sample Movie Data in CSV Format*

```
Spider-Man,2002-05-03-05:00,821706375.00,403706375.00,Sam Raimi,"Maguire,
Tobey","Dafoe, Willem","Dunst, Kirsten",""
Spider-Man 2,2004-06-30-05:00,783924485.00,373585825.00,Sam Raimi,"Maguire,
Tobey","Franco, James","Dunst, Kirsten","Molina, Alfred"
Spider-Man 3,2007-05-04-05:00, 888977494.00, 336027292.00,Sam Raimi,"Maguire,
Tobey","Dunst, Kirsten","Franco, James","Church, Thomas Haden"
```

The CSV format, though much more compact than the XML version, removes the context, structure, and self-documenting tags provided by the XML. It can also be more difficult to expand the CSV version to include more fields, add optional data to records, or modify and troubleshoot the existing format. For instance, consider Listing 1-3, which adds an additional `<actor>` element, expands the `<actor>` elements of the XML version to include an `actor_id` attribute and a list of additional movies some actors have acted in.

Listing 1-3. *Modified Sample XML*

```
<?xml version="1.0" encoding="UTF-16"?>
<!-- High-grossing movie listing -->
<movies>
    <film>
        <?style superhero?>
        <name>Spider-Man</name>
        <releaseDate>2002-05-03-05:00</releaseDate>
        <gross area="world-wide">821706375.00</gross>
        <gross area="domestic">403706375.00</gross>
        <director>Sam Raimi</director>
        <cast>
            <actor actor_id="MA011">Maguire, Tobey
                <movie>Spider-Man 3</movie>
                <movie>The Good German</movie>
                <movie>Spider-Man 2</movie>
                <movie>Seabiscuit</movie>
            </actor>
            <actor actor_id="DA982">Dafoe, Willem
                 <movie>Clear and Present Danger</movie>
                 <movie>Mississippi Burning</movie>
                 <movie>Platoon</movie>
            </actor>
            <actor actor_id="DU208">Dunst, Kirsten
                 <movie>Spider-Man 3</movie>
                 <movie>Interview with the Vampire</movie>
            </actor>
        </cast>
    </film>
</movies>
```

These modifications to the XML data make it harder (though not impossible) to represent the data as a single CSV file. CSV format also eliminates additional information about the contents, including the hierarchical structure of the data, comments, and the character set encoding information. Every time the structure of the data changes (e.g., more fields are added or fields are rearranged), the application parsing and querying the data must be modified, often significantly. Finally, the simple CSV format in the example requires you to reference fields by their ordinal position, which can lead to subtle errors and bugs that are difficult to troubleshoot.

The XML version of the data, on the other hand, allows you to query the data elements by name and relative position. For instance, to retrieve the director name of the movie *Spider-Man* in the XML file, you would essentially say "retrieve the `<director>` element of the `<film>` where the `<name>` is 'Spider-Man.'"

Bear in mind also that CSV is not a standardized format. It is a de facto format conjured up in the early days of computing. A CSV file created on a Windows PC might not be readable on a UNIX computer, and vice versa. Even CSV files created by different Microsoft programs can prove incompatible with one another. This makes it nearly impossible to create a general-purpose CSV parser. Text-based formats like Windows initialization (`.INI`) files suffer from similar limitations.

Binary formats, like the proprietary formats used by COM-related technologies and even office productivity applications like Microsoft Excel are much more compact than XML, CSV, and other text-based formats; but they suffer some drawbacks as well. These formats are not as easy to transmit over the Internet using standard protocols like HTTP, each format requires its own proprietary parsing tools to "look inside" the data, and they tend to be much harder to modify and debug. Figure 1-1 shows a portion of the binary data that composes the sample in Excel worksheet format.

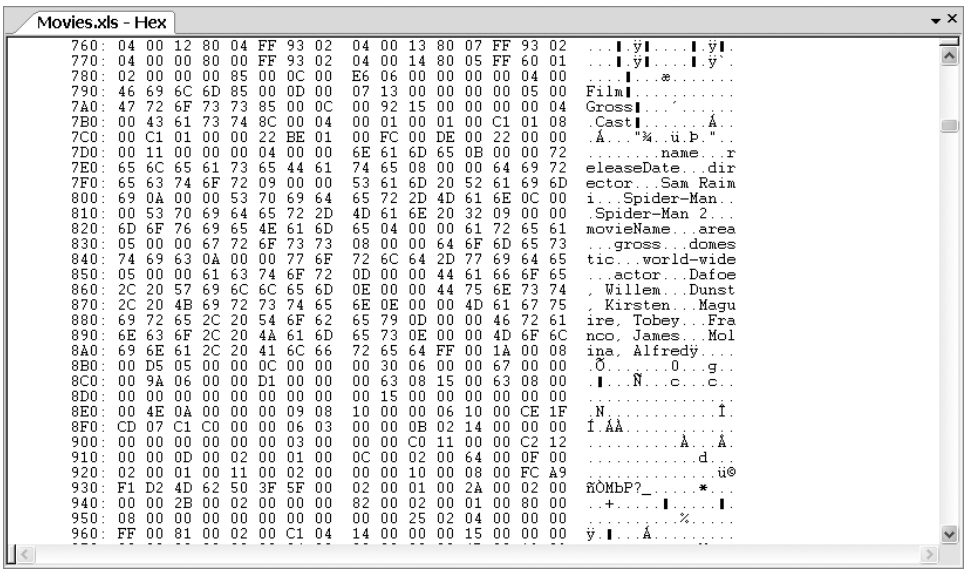
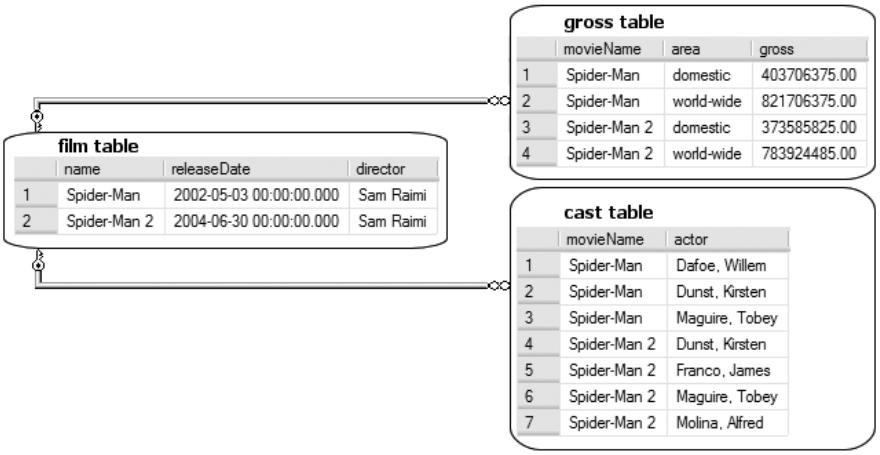


Figure 1-1. *Sample Excel spreadsheet binary data*

Since we're using SQL Server, it's important to note that data is often better represented using the relational model. The XML sample presented in this chapter was chosen specifically because it is relatively easy to convert to relational format. Figure 1-2 shows the sample data when converted to tables in a relational database.



gross table

	movieName	area	gross
1	Spider-Man	domestic	403706375.00
2	Spider-Man	world-wide	821706375.00
3	Spider-Man 2	domestic	373585825.00
4	Spider-Man 2	world-wide	783924485.00

film table

	name	releaseDate	director
1	Spider-Man	2002-05-03 00:00:00.000	Sam Raimi
2	Spider-Man 2	2004-06-30 00:00:00.000	Sam Raimi

cast table

	movieName	actor
1	Spider-Man	Dafoe, Willem
2	Spider-Man	Dunst, Kirsten
3	Spider-Man	Maguire, Tobey
4	Spider-Man 2	Dunst, Kirsten
5	Spider-Man 2	Franco, James
6	Spider-Man 2	Maguire, Tobey
7	Spider-Man 2	Molina, Alfred

Figure 1-2. *Sample data represented in relational database tables*

Whether your data is better represented using XML or the relational model is a question that must be answered on a per-project basis, guided by your business rules and requirements. Keep in mind that these two choices do not have to exist exclusive of one another, however. SQL Server offers tools to convert relational data to XML and vice versa, and to manipulate and query both relational data and XML. In the next section, you'll look at some things to take into consideration when deciding whether or not XML is the best choice for representing your data.

When to Use XML

When should you choose XML format over the other possibilities? Some of the factors that can help guide your decision toward using XML to represent data include the following:

- You need a platform-independent model that requires a portable data format.
- Your data is inherently ordered, and the position of data elements provides additional context. Hierarchical data is a prime example where the position of data elements provides context.
- You anticipate querying or updating your data based on its structure.
- Your data is sparse or you anticipate significant structure changes.
- Your data is semi-structured; for example, it has no fixed schema, an implicit or irregular structure, or is nested and heterogeneous.
- Your data represents a containment hierarchy.

SQL Server 2008's native XML manipulation and querying facilities are particularly useful in the following situations:

- You want to manipulate or share XML data while taking advantage of SQL Server's transactional capabilities.
- You want to take advantage of SQL Server's administrative functionality to back up, recover, or replicate your XML data.
- You need to ensure that your server-side XML data is valid and well-formed.
- You want your XML and relational data to interoperate.
- You want to take advantage of SQL Server's XQuery and XPath capabilities.
- You want to optimize your queries of XML data using indexes and SQL Server's query optimizer.

XML is a particularly good choice as a communication format for loosely coupled systems or when trying to make legacy systems communicate with modern servers. It's also an excellent choice when storing data for applications that lend themselves to using data in a marked-up format, such as document management and presentation applications. Modeling semistructured data in an ad hoc fashion also lends itself well to using XML. There are many standard applications for XML already defined in almost every industry, so XML is often an excellent choice when standard communication formats and data sharing are high priorities.

Whether, and when, to use XML is the subject of much debate among SQL Server professionals. There are situations where XML is not the best tool for the job, including the following:

- Your data is highly structured and dense (non-sparse).
- Order is unimportant to your data.
- You do not want to query data based on its structure.
- Your data is best represented using entity references.

Additionally, if you do not intend to query or manipulate your XML data on SQL Server, but just want to store and retrieve it, you should use the `varchar` or `nvarchar` data types instead of SQL Server's `xml` data type. This is most often the case when you simply use SQL Server as a storage repository for your XML data and perform all your XML querying and manipulation in the middle tier or in a client-side application. Also store your XML data as `varchar` or `nvarchar` if you need to store it exactly, as a character-for-character copy, for auditing purposes or for regulatory compliance. This is a common scenario when you are storing data about digital financial transactions, and you need to maintain an audit trail of the transactions but do not need to manipulate or query the data in the XML.

EXACTLY XML

The `xml` data type does not necessarily store an exact character-for-character copy of your XML. The XML Schema Recommendation specifies that XML is converted to an abstract tree-like representation, known as an XML Information Set (Infoset) for persistence or querying purposes. The XQuery Data Model (XDM) Recommendation builds on the XML Infoset and extends the XML Schema model even further. The Infoset allows for loss of characters or other internal manipulations on the XML elements, so long as the meaning of the content is not lost or distorted. In some situations, like regulatory compliance (e.g., compliance with the Sarbanes-Oxley Act of 2002) and auditing, maintaining just the "meaning" is not good enough. You must maintain literal copies of your data. In instances where this is a requirement, it makes sense to store your XML data using the `varchar` or `nvarchar` data type, not the `xml` data type. I will discuss the XQuery 1.0 and XQuery 2.0 Data Model in more detail later in Chapter 4, during the discussion of XML schema collections.

Some database developers and administrators are vehemently opposed to storing XML data in a relational database. Fortunately for them, they do not necessarily have to store XML in the database in order to take advantage of SQL Server's XML querying and manipulation capabilities. The `xml` data type can be used to declare variables that can be used for querying, validating, and manipulating XML data server side. It can also be used to declare parameters for functions and stored procedures.

What's New in SQL Server 2008 XML

SQL Server 2008 provides several enhancements over SQL Server 2000 in terms of XML support and some enhancements over SQL Server 2005. While much of the backward-compatible XML-specific functionality from SQL Server 2000 is available in SQL Server 2008, most of it has been deprecated in favor of the new features and functionality. This section gives a broad overview of the major enhancements to XML support, which include the following items:

- New `xml` data type
- XML schema collections
- XML indexes
- `FOR XML` enhancements, including XPath support in the `FOR XML PATH` clause
- XQuery and XML DML support
- SQLCLR `xml` data type support
- Improvements to legacy XML functionality, including improvements to the `sp_xml_preparedocument` procedure
- HTTP Simple Object Access Protocol (SOAP) endpoints

In addition, there have been enhancements made to the `xml` data type implementation over the SQL Server 2005 version, including the following items:

- Full support for the XML Schema `xs:date`, `xs:time`, and `xs:dateTime` data types has been added in SQL Server 2008.
- Support has been added for XML Schema–defined lists of unions and unions of lists.
- Improvements to XML Schema union types have been implemented.
- Support has been added for XQuery FLWOR expression `let` clauses.

SQL Server 2008's SQLCLR functionality also provides extensive support for XML via .NET Framework 2.0 classes. With .NET 2.0, enhancements were made to existing .NET 1.1 classes, and new classes and features have been added. These new features are discussed in greater detail in Chapter 8.

The xml Data Type

Prior to SQL Server 2005, SQL Server provided extremely limited support for storing, managing, and manipulating XML data. SQL Server 2000 implemented its XML capabilities through implementation of the `FOR XML` clause and kludgy LOB data type operations combined with specialized system-stored procedures. SQL Server 2005 introduced the `xml` data type, promoting XML data storage and manipulations to first-class status in SQL Server.

The `xml` data type remains one of the most important XML-specific features in SQL Server 2008. The `xml` data type supports the storage of typed XML documents and fragments that have been validated against an XML schema collection and untyped XML data which has not. The `xml` data type can be used to declare columns in a table, T-SQL variables, parameters, and as the return type of a function. Data can also be cast to and from the `xml` data type. In addition, the `xml` data type brings with it a set of methods useful for querying, shredding, and manipulating XML data. The `xml` data type is described in detail in Chapter 3.

XML Schema Collections

XML schema collections provide the ability to validate your documents for well-formedness and validity according to XML schemas that follow the W3C XML Schema standard. SQL Server 2008 provides support for creating server-side XML schema collections that can be used to validate XML documents for structure and data typing. A very powerful feature, XML schema collections and the W3C XML Schema Recommendation are discussed in detail in Chapter 4.

XML Indexes

In the SQL Server XML model, whenever you query or manipulate XML data, the data is first converted to a relational format in a process known as *shredding*. This process can be time consuming when manipulating large XML documents or when querying large numbers of `xml` data type instances. SQL Server 2008 supports indexing of `xml` data type columns. Indexing `xml` columns helps the SQL optimizer significantly improve query performance on XML data stored in the database. The performance is improved by building an index of your XML data by converting it to a relational format, a process known as *preshredding*. The XML index preshredding process

eliminates the shredding step during a query or XML data manipulation, resulting in much faster and less resource-intensive XML query operations. New DML statements have been added to T-SQL to make XML index management relatively easy. XML indexes will be discussed in detail in Chapter 7.

FOR XML

SQL Server includes improvements to the legacy `FOR XML` clause. One improvement is tighter integration with the new `xml` data type, including options to generate native `xml`-typed results. `FOR XML` results can be assigned to variables of the `xml` data type, with additional support for nesting `FOR XML` queries, an improvement on the SQL Server 2000 `FOR XML` clauses, which were limited only to the top level of a `SELECT` statement. The `FOR XML PATH` mode, also carried over from SQL Server 2005, is an improvement over the legacy `FOR XML EXPLICIT` mode. With built-in support for XPath-style expressions, `FOR XML PATH` makes generating XML in explicit structures much easier than was possible in SQL Server 2000.

The `FOR XML RAW` mode has also been improved with additional features, including the ability to rename the default `row` element name, the ability to explicitly specify the root node, and the ability to retrieve your data in an element-centric format. The `FOR XML AUTO` and `FOR XML EXPLICIT` modes have also been improved with additional options and settings.

While some options have been deprecated, several additional options have been added to the `FOR XML` clauses since the SQL Server 2000 version, including the `ELEMENTS XSINIL` option, which generates elements for `NULL`s in the result set, and `XMLSCHEMA`, which generates an inline XML Schema Definition (XSD) in your XML result. The power of the `FOR XML` clause will be explored in detail in Chapter 2.

XQuery and XML DML Support

The new `xml` data type provides several methods to allow querying and modification of XML data. These new methods, including the `query()`, `value()`, `exist()`, `nodes()`, and `modify()` methods, support XQuery querying, XML shredding, and XML DML manipulation of your XML data. The SQL Server 2008 XQuery implementation is a powerful subset of the W3C XML Query Language specification, featuring support for path expressions, FLWOR (for-let-where-order-by-return) expressions, standard functions and operators, and XML DML statements. XQuery will be discussed in depth in Chapter 5, and supported XQuery functions and operators and XML DML will be covered in Chapter 6.

HTTP SOAP Endpoints

A powerful feature introduced in SQL Server 2005, SQL Server 2008 continues providing support for native HTTP SOAP endpoints, which use the XML-based SOAP protocol. HTTP SOAP endpoints provide an efficient, secure, easy-to-configure option for providing SQL Server–based web service support. The built-in HTTP SOAP endpoint support makes it much easier to expose SQL Server functionality as web services than was previously possible via the Internet Information Server (IIS)–based web services available in SQL Server 2000. We will discuss HTTP SOAP endpoints in Chapter 9.

■ **Note** HTTP SOAP endpoints are not available in SQL Server 2008 Express Edition.

Summary

SQL Server 2008 XML functionality includes all of the functionality that was introduced with SQL Server 2005, improved legacy SQL 2000–compatible functionality, and dozens of new features and enhancements to existing functionality. This chapter provided an overview of XML, its history, and improvements to XML support in SQL Server 2008.

I talked about how XML stacks up to other data formats and looked at some of the items that you should consider when deciding whether or not XML fits in with your design goals. Along the way I contrasted XML to HTML, and considered instances when it makes sense to store your data in strict character format instead of using the `xml` data type. I also briefly considered alternatives to XML for data storage, manipulation, and transmission, and I also provided some general guidelines to help determine when and where SQL Server's XML capabilities can be useful.

In order to understand where you're going, it helps to know where you came from. With that in mind, I provided a brief overview of the history of SQL Server XML support going back to SQL Server 2000 functionality. I then looked forward with a summary of SQL Server 2008 XML enhancements, including the `xml` first-class data type, the enhanced `FOR XML` clause, XML schema collections, XML indexes, XQuery and XML DML support, and HTTP SOAP endpoints. Each of these topics, and much more, will be discussed in great detail in later chapters.

In the next chapter, I'll discuss the `FOR XML` clause and support for other legacy XML functionality, which was introduced in SQL Server 2000.