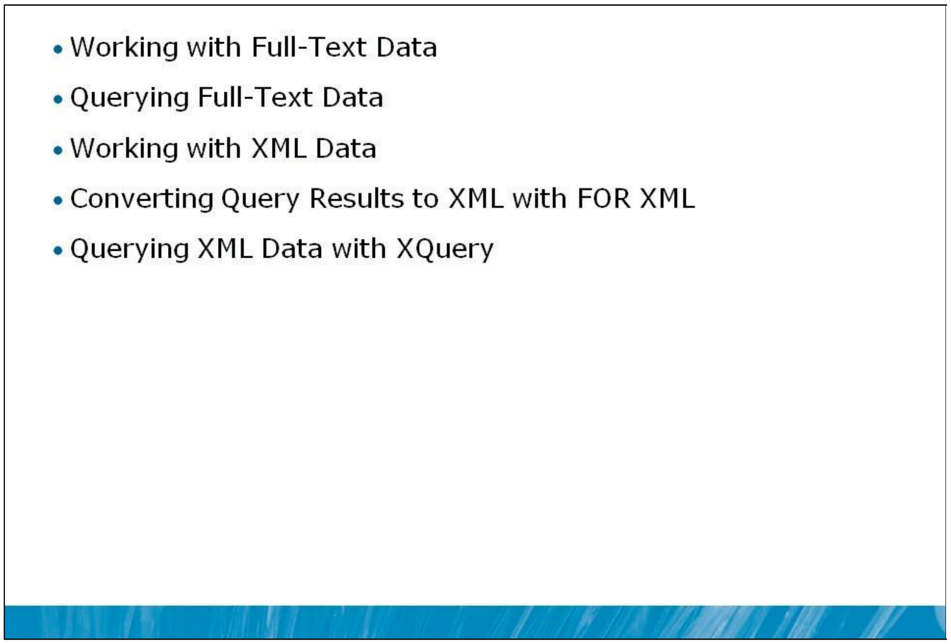
FIREBRAND

Appendix C

Writing Specialized Queries

Contents:

Module Overview



- Working with Full-Text Data
- Querying Full-Text Data
- Working with XML Data
- Converting Query Results to XML with FOR XML
- Querying XML Data with XQuery

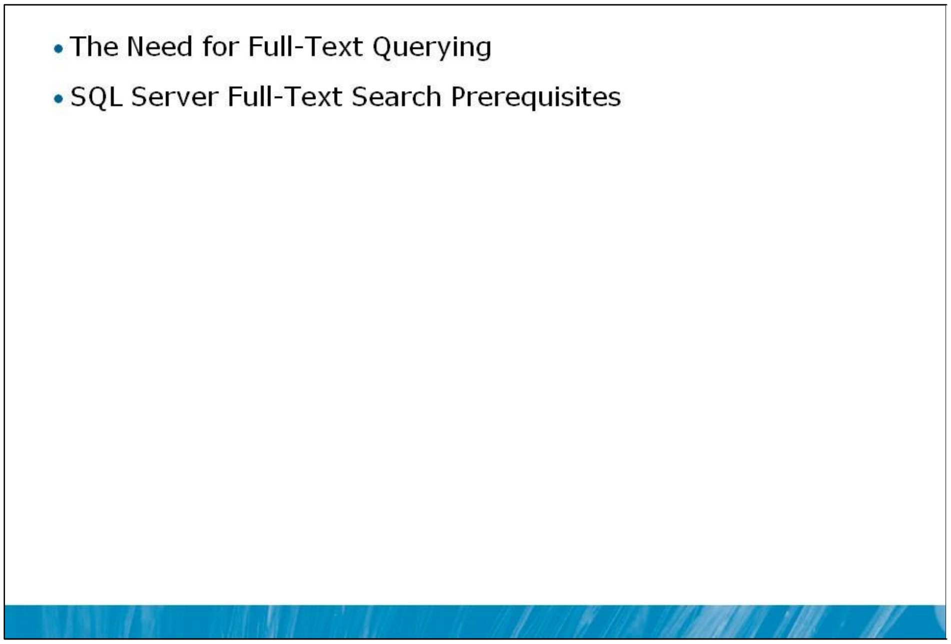
In this course, you have learned to write T-SQL queries against data stored in character, numeric, date, and time data types. Microsoft® SQL Server® also supports more specialized mechanisms for querying character data and includes additional data types, such as XML, that require other methods for querying them. In this module, you will learn how to use full-text querying when it is available in your databases and XML methods for working with XML data.

Objectives

After completing this module, you will be able to:

- Use full-text predicates in queries against databases enabled for full-text search.
- Write queries to retrieve tabular data and convert it to XML.
- Write queries that manipulate data stored in SQL Server's native XML data type.

Lesson 1
Working with Full-Text Data



- The Need for Full-Text Querying
- SQL Server Full-Text Search Prerequisites

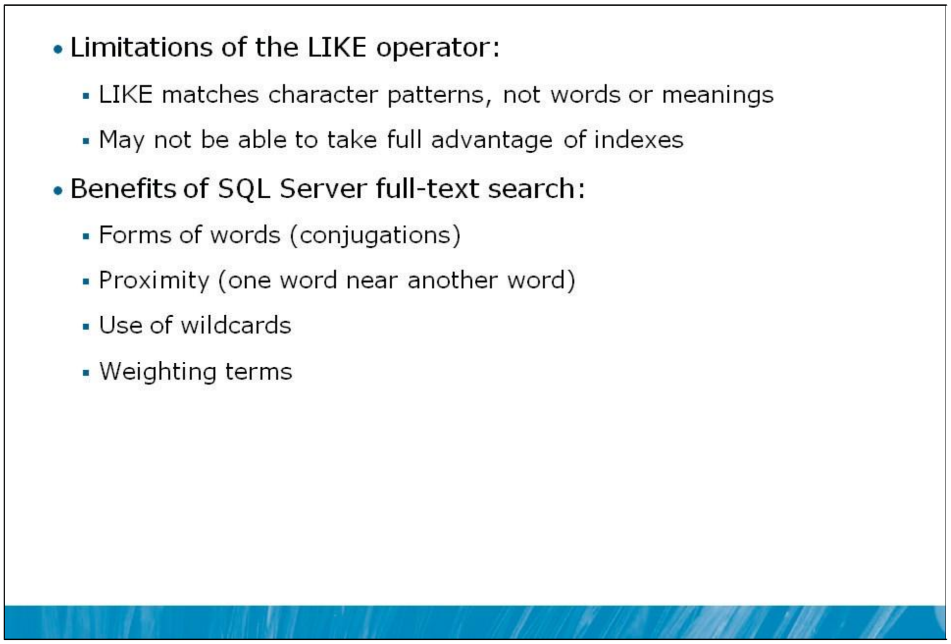
In this lesson, you will learn the background for querying with full-text search, including the benefits of full-text predicates compared to the LIKE predicate. You will also learn about what needs to be installed and configured on SQL Server to make full-text search available to your queries.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the benefits of full-text search compared to the T-SQL LIKE predicate.
- Describe the administrative tasks necessary to support full-text search.

The Need for Full-Text Querying



- Limitations of the LIKE operator:
  - LIKE matches character patterns, not words or meanings
  - May not be able to take full advantage of indexes
- Benefits of SQL Server full-text search:
  - Forms of words (conjugations)
  - Proximity (one word near another word)
  - Use of wildcards
  - Weighting terms

To better understand why full-text search is an attractive option for querying text stored in SQL Server, it might be useful to compare it to the alternative: the LIKE predicate you learned about earlier in this course. As you may recall, LIKE can be used to perform pattern matching in character columns. When used with wildcards (%) and pattern expressions, it can be very flexible in finding matching text.

However, LIKE has two major shortcomings:

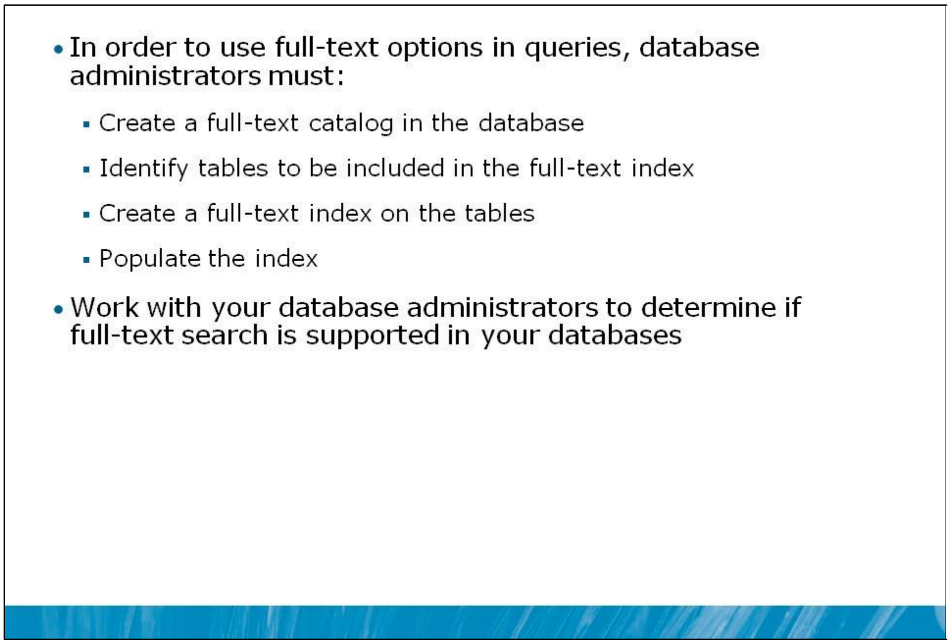
- LIKE matches strings at the character level, not the word level, and therefore cannot be used to find variations on a word. For example, a query ending in *WHERE productname LIKE '%race%';* can find products with "race" in their name, such as "racer" or "raced", but not "racing".
- If the LIKE predicate uses a wildcard as the leading character, SQL Server is unable to search an index on the column to improve the performance of the query. As a data volume grows, the penalty for scanning an entire table grows with it.

SQL Server full-text search (which is sometimes referred to as iFTS) is designed to address both issues. By extracting words from the source columns and creating special indexes, it can more quickly locate rows with words matching the search expression. And by treating words as words and not just strings of characters, SQL Server full-text search provides additional capabilities:

- The ability to find rows with forms of words, such as inflections or synonyms.
- The ability to find and rank rows based on the proximity of search terms to each other.
- The ability to use leading wildcards to identify prefixes, without the penalty LIKE suffers.
- The ability to assign relative weights to key words and rank rows based on the total weight.
- The ability to customize lists of "noise" words that should be ignored when indexing.

Note This module will cover how to search for word forms (inflections) and find words near each other (proximity). Searching with prefixes, assigning weights, and ranking results are beyond the scope of this course.

SQL Server Full-Text Search Prerequisites



- In order to use full-text options in queries, database administrators must:
  - Create a full-text catalog in the database
  - Identify tables to be included in the full-text index
  - Create a full-text index on the tables
  - Populate the index
- Work with your database administrators to determine if full-text search is supported in your databases

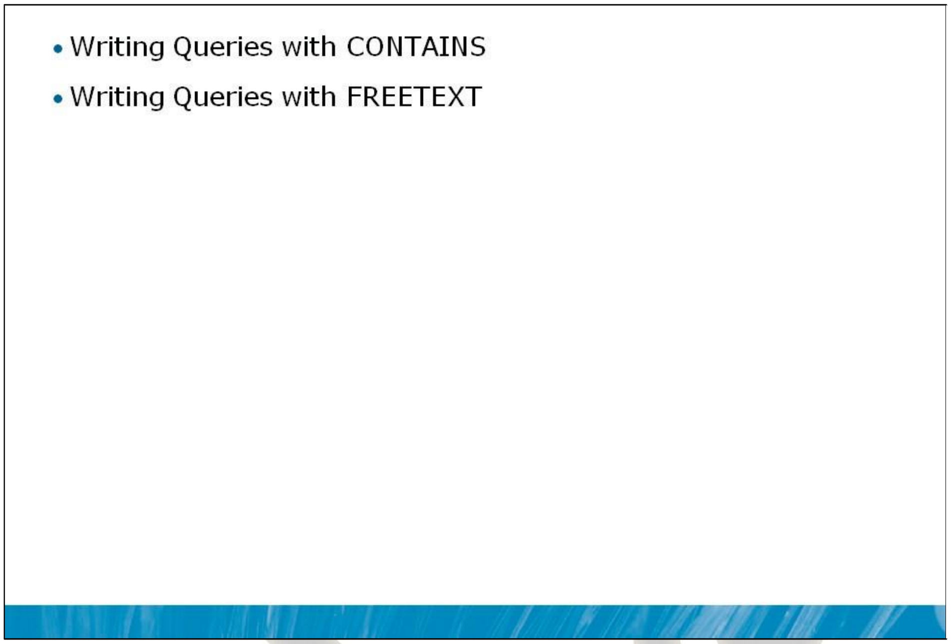
In order to take advantage of full-text search in SQL Server queries, several prerequisite steps must be completed by a database administrator. While the details of these steps are beyond the scope of this class, they are presented here at a high level for your understanding. To use full-text options in queries, database administrators must:

1. **Install full-text search at the instance level.** Full-text search is an optional component of SQL Server and must be installed on the instance to be used. This can be done when the SQL Server database engine is initially installed, or it can be added later through the SQL Server Installation Center.
2. **Create a full-text catalog in the user database.** One or more catalogs may be created in a user database to support full-text search. The catalog logically contains the full-text indexes and other system objects used by full-text search.
3. **Create one or more full-text indexes.** Full-text indexes contain information about the location of indexed words and related primary keys within the indexed data.
4. **Identify and include tables and columns in a full-text index.** Supported data types include character, Unicode, XML, and other large object (LOB) columns.
5. **Populate the index.** Administrators can choose among a number of automatic, scheduled, and manual methods for full-text search to crawl and index the words in your tables.

Only after these prerequisites are met will you have access to full-text data in your queries.

Note For more information on the setup and administration of full-text search in SQL Server, see Microsoft Course 10776: *Developing Microsoft® SQL Server® 2012 Databases*. See also Books Online at http://msdn.microsoft.com/en-us/library/ms142497(v=SQL.110).aspx.

Lesson 2
Querying Full-Text Data



- Writing Queries with CONTAINS
- Writing Queries with FREETEXT

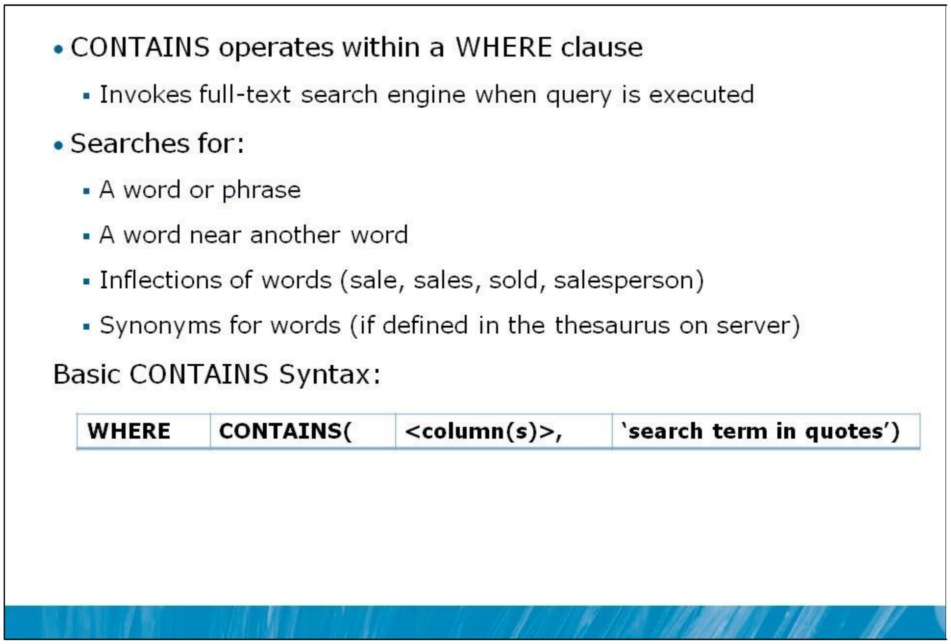
In this lesson, you will learn the fundamentals of writing full-text queries in SQL Server 2012. SQL Server full-text search provides several predicates and functions for use in queries: CONTAINS, CONTAINSTABLE, FREETEXT, and FREETEXTTABLE. In this lesson, you will learn how to use the CONTAINS and FREETEXT predicates in the WHERE clause of T-SQL SELECT statements.

Lesson Objectives

After completing this lesson, you will be able to:

- Use the CONTAINS predicate to match words in full-text queries.
- Use the FREETEXT predicate to match meanings in full-text queries.

Writing Queries with CONTAINS



- CONTAINS operates within a WHERE clause
  - Invokes full-text search engine when query is executed
- Searches for:
  - A word or phrase
  - A word near another word
  - Inflections of words (sale, sales, sold, salesperson)
  - Synonyms for words (if defined in the thesaurus on server)

Basic CONTAINS Syntax:

WHERE	**CONTAINS(**	**<column(s)>,**	**'search term in quotes')**

The SQL Server full-text search predicate CONTAINS extends the WHERE clause of your T-SQL queries, invoking the full-text search engine and returning rows that match your search phrase. CONTAINS returns exact or literal matches at the word level, not the character level, in contrast to the LIKE predicate.

CONTAINS provides powerful capabilities for matching:

- A word or phrase in an indexed column, including Boolean operators
- A word based on proximity to another word
- Inflections, or word-stemming forms, of a word
- Synonyms and replacements for a word
- Prefix searches similar to a leading wildcard using LIKE

Basic CONTAINS syntax:

```
WHERE CONTAINS(<column(s)>, '<search_term_in_single_quotes>')
```

CONTAINS syntax with search phrase:

```
WHERE CONTAINS(<column(s)>, '<word1> <word2>')
```

CONTAINS syntax with proximity:

```
WHERE CONTAINS(<column(s)>, '<word1> NEAR <word2>')
```

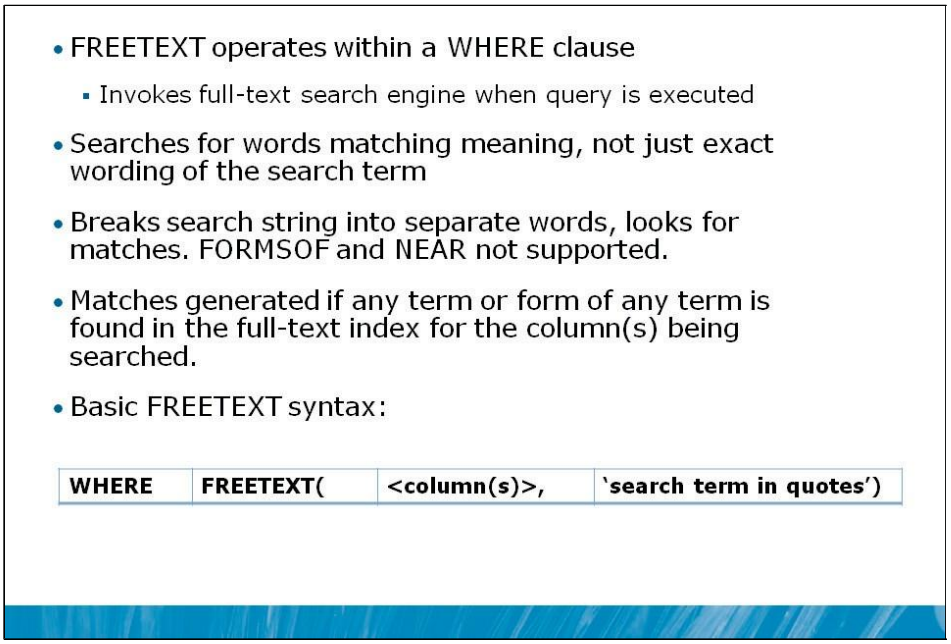
Note Proximity searches can also include ranking values to indicate relative proximity. The CONTAINSTABLE predicate is used for this. For more information on CONTAINSTABLE, see Books Online at http://msdn.microsoft.com/en-us/library/ms189760(v=SQL.110).aspx.

CONTAINS syntax with word-stemming:

```
WHERE CONTAINS(<column(s)>, 'FORMSOF(INFLECTIONAL,<word>')
```

Note CONTAINS also provides advanced features such as term weighting, synonym, and expansion, which are beyond the scope of this course. For more information, see Microsoft Course 10776A: *Developing Microsoft® SQL Server® 2012 Databases*. See also "Supported Forms of Query Terms (Full-Text Search)" in Books Online at http://msdn.microsoft.com/en-us/library/cc879300(v=SQL.110).aspx.

Writing Queries with FREETEXT



- FREETEXT operates within a WHERE clause
  - Invokes full-text search engine when query is executed
- Searches for words matching meaning, not just exact wording of the search term
- Breaks search string into separate words, looks for matches. FORMSOF and NEAR not supported.
- Matches generated if any term or form of any term is found in the full-text index for the column(s) being searched.
- Basic FREETEXT syntax:

WHERE	**FREETEXT(**	**<column(s)>,**	**'search term in quotes')**

The SQL Server full-text search predicate FREETEXT extends the WHERE clause of your T-SQL queries, invoking the full-text search engine and returning rows that match your search phrase. Unlike CONTAINS, FREETEXT returns expanded matches at the word level, not only exact matches. FREETEXT will break apart a phrase and search for each term separately, unlike CONTAINS.

FREETEXT provides many capabilities for matching, including:

- A word or phrase in an indexed column, including Boolean operators
- Conjugations of nouns, including plurals, gender, and case
- Conjugations of verbs, including tenses
- Synonyms and replacements for a word
- Fuzzy matching of phrases, compared to exact matches in CONTAINS

Note FREETEXT does not provide proximity searching using NEAR or inflections with FORMSOF. CONTAINS provides this capability.

Basic FREETEXT syntax:

```
WHERE FREETEXT(<column(s)>, '<search_term_in_single_quotes>')
```

FREETEXT will return all forms of the search term, per the rules of the language settings in the database. For example, a search for "race" will return "race", "races", "racing", "raced", and other forms.

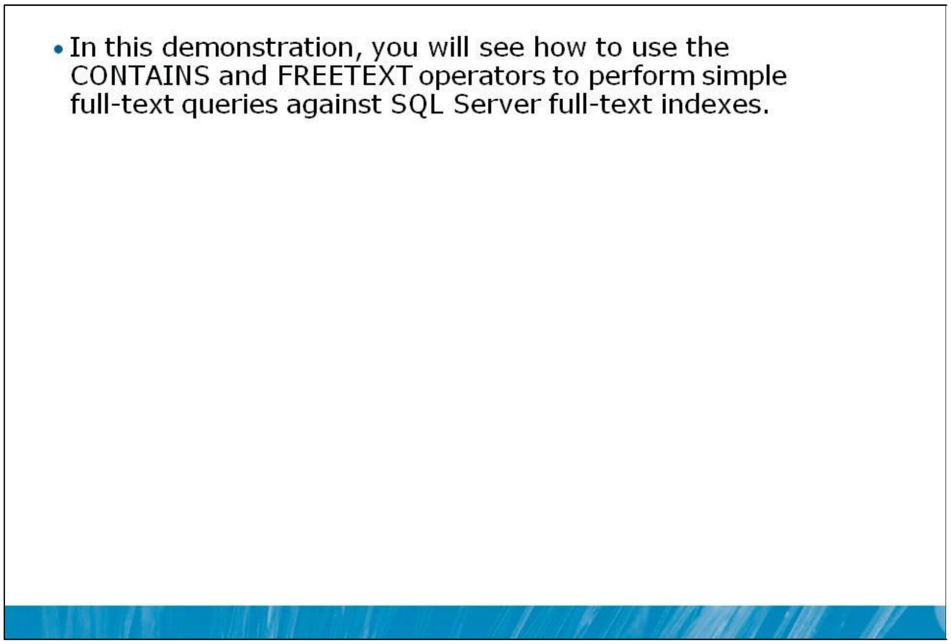
FREETEXT syntax with search phrase:

```
WHERE FREETEXT (<column(s)>, '<word1> <word2>')
```

With a search phrase, FREETEXT will return instances of each word as well as inexact matches for the phrase itself. A search for "road racing" will return "road", "roads", "race", "racing", "road racing", "racing on roads", and other variations.

Note FREETEXT does not rank results. You may wish to explore the FREETEXTTABLE function, which provides ranking. FREETEXTTABLE is beyond the scope of this class. For more information, see Books Online at http://msdn.microsoft.com/en-us/library/ms177652(v=SQL.110).aspx.

Demonstration: Querying Full-Text Data



- In this demonstration, you will see how to use the CONTAINS and FREETEXT operators to perform simple full-text queries against SQL Server full-text indexes.

Demonstration Steps

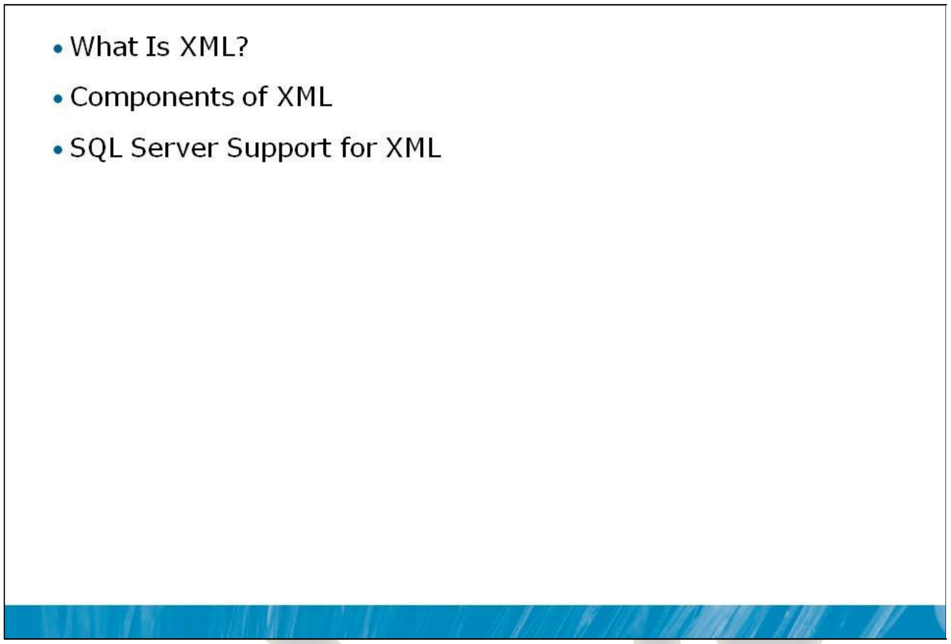
1. On the virtual machine, click **Start**, click **All Programs**, click **Microsoft SQL Server 2012**, and click **SQL Server Management Studio**. In the **Connect to Server** window, type **Proseware** in the **Server name** text box and click **Connect**. On the **File** menu, click **Open** and click **Project/Solution**. Navigate to F:\10774A_Labs\10774A_AC_PRJ\10774A_AC_PRJ.ssmssln and click **Open**.

2. On the **View** menu, click **Solution Explorer**.

3. Open the 11 – Demonstration A.sql script file.

4. Follow the instructions contained within the comments of the script file.

Lesson 3
Working with XML Data



- What Is XML?
- Components of XML
- SQL Server Support for XML

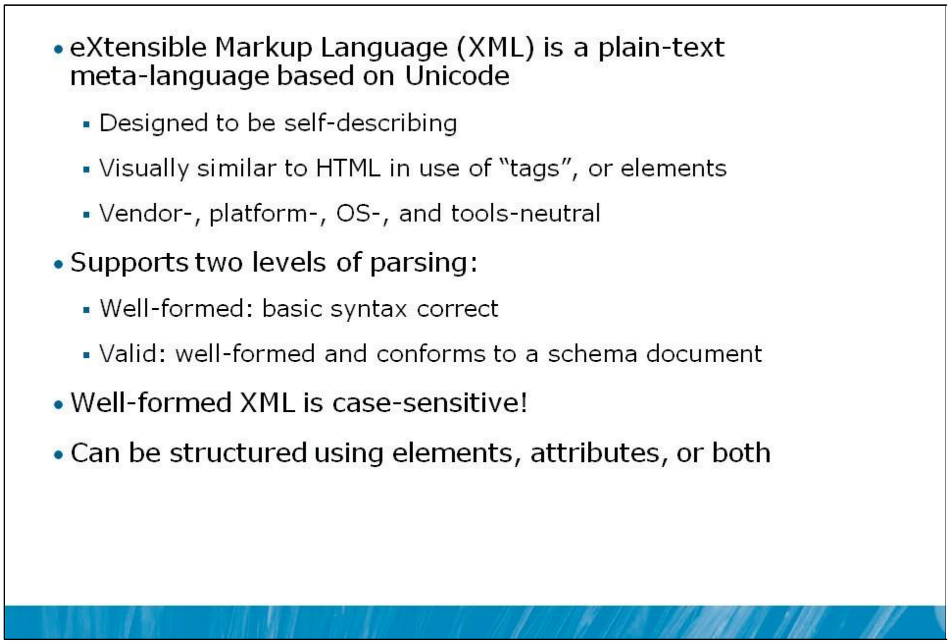
In this lesson, you will learn the fundamentals of XML support in SQL Server, including the basic components and terminology of XML. You will also see some of the uses for XML in SQL Server 2012.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the eXtensible Markup Language (XML).
- Describe the basic components of XML.
- Describe the uses of XML in SQL Server.

What Is XML?



- eXtensible Markup Language (XML) is a plain-text meta-language based on Unicode
  - Designed to be self-describing
  - Visually similar to HTML in use of "tags", or elements
  - Vendor-, platform-, OS-, and tools-neutral
- Supports two levels of parsing:
  - Well-formed: basic syntax correct
  - Valid: well-formed and conforms to a schema document
- Well-formed XML is case-sensitive!
- Can be structured using elements, attributes, or both

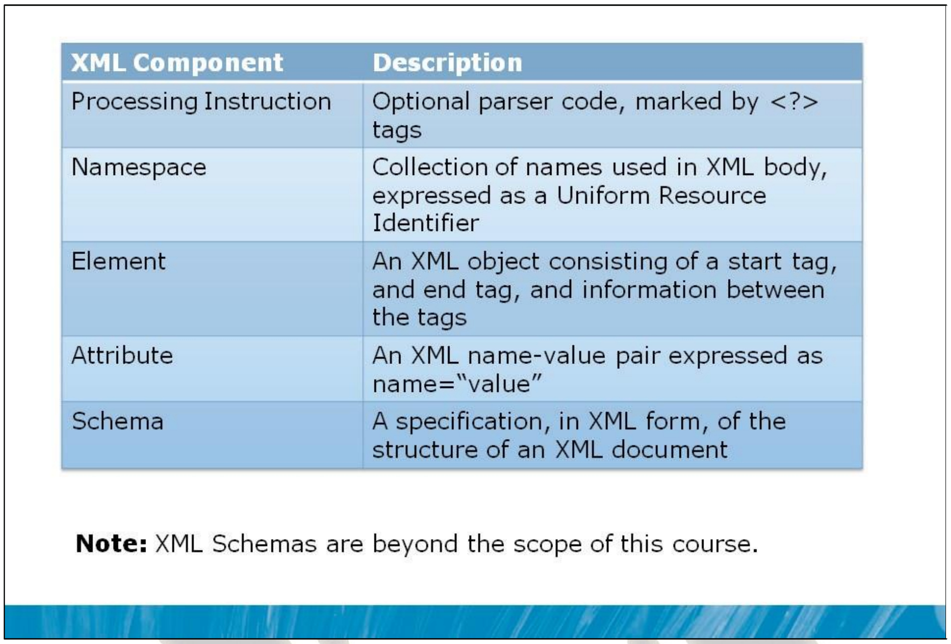
XML is a text format for describing and storing data. An industry standard adopted by Microsoft for many products including SQL Server, XML is designed to be vendor-neutral, operating-system-neutral, language-neutral, and accessible to any tool that can work with Unicode text. You may have previously worked with other markup languages, such as Hypertext Markup Language (HTML), which describes the formatting and layout of data on a web page. HTML includes a set of predefined tags for this purpose. However, XML is designed to provide meaning about the data itself through a syntax similar to HTML's "tags", or elements. Unlike HTML, XML includes very few predefined elements.

The specification for XML includes several basic syntax requirements you will need to observe in your SQL Server XML queries. These can be expressed as levels of parsing:

- **Well-formed XML,** which requires case-sensitivity, properly nested elements, and a single root element that contains all the other elements.
- **Valid XML,** which requires that the XML be well-formed and that it conform to an XML Schema document. (In SQL Server, XML Schemas may be stored in a database as XML Schema Collections or included inline in an XML document.)

Note For the purposes of this course, you will only work with well-formed XML. You can work with XML Schemas in Microsoft Course 10776A: *Developing a Microsoft® SQL Server® 2012 Database.*

Components of XML



XML Component	Description
Processing Instruction	Optional parser code, marked by <?> tags
Namespace	Collection of names used in XML body, expressed as a Uniform Resource Identifier
Element	An XML object consisting of a start tag, and end tag, and information between the tags
Attribute	An XML name-value pair expressed as name="value"
Schema	A specification, in XML form, of the structure of an XML document

Note: XML Schemas are beyond the scope of this course.

There are a few basic components of XML that you will work with in SQL Server:

XML Component	Description
Processing Instruction	Also called a prolog, this section is marked by <?> tags and contains information for the XML parser.
Namespace	Collection of names used in the body of the XML document, expressed as a Uniform Resource Identifier (URI). Used to distinguish between duplicate and potentially ambiguous element and attribute names. Use of namespaces required in XQuery expressions.
Element	The basic building block of XML, an object delimited with <> brackets forming start and end tags with information between.
Attribute	An optional method for expressing an element as a name-value pair, delimited with quotation marks.
Schema	A specification, in XML form, of the structure of an XML document. XML Schemas are beyond the scope of this course.

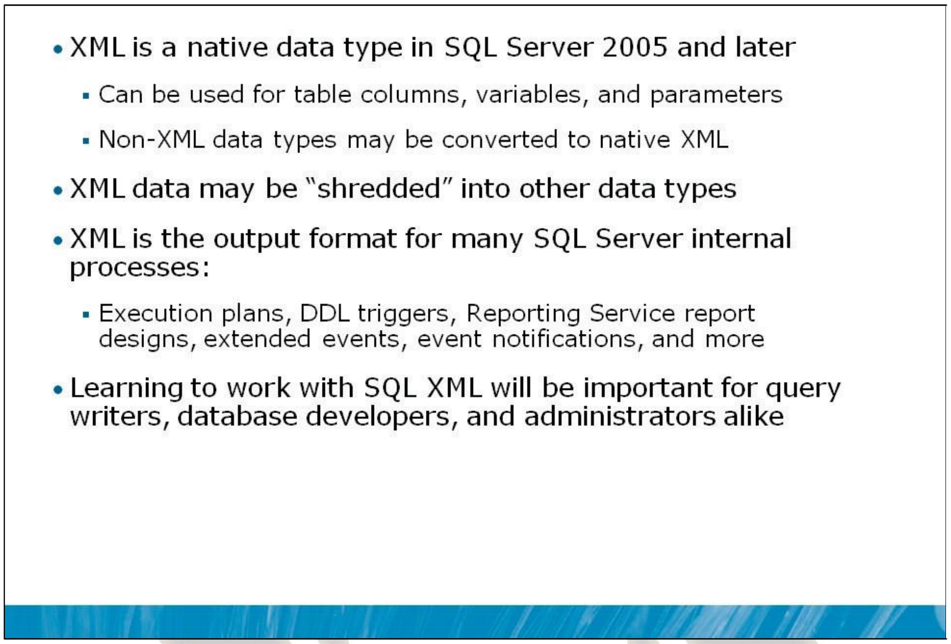
Note Not all of these components will be present in all of your work with XML in SQL Server. For example, the FOR XML clause does not generate a prolog, which may cause errors with some XML parsers.

The following sample XML includes a section for a processing instruction, nested elements, and the use of attributes to represent an order ID number. Note that all the line breaks, indentations, and white space have been added for readability, but are not required by an XML parser:

```
<?xml version="1.0" encoding="utf-8"?>
<Orders>
      <Order OrderID="10248">
      <CustID>85</CustID>
      <OrdDate>2006-07-04</OrdDate>
      </Order>
      <Order OrderID="10249">
      <CustID>79</CustID>
      <OrdDate>2006-07-05</OrdDate>
      </Order>
</Orders>
```

For More Information The complete XML specification can be found at http://www.w3c.org.

SQL Server Support for XML



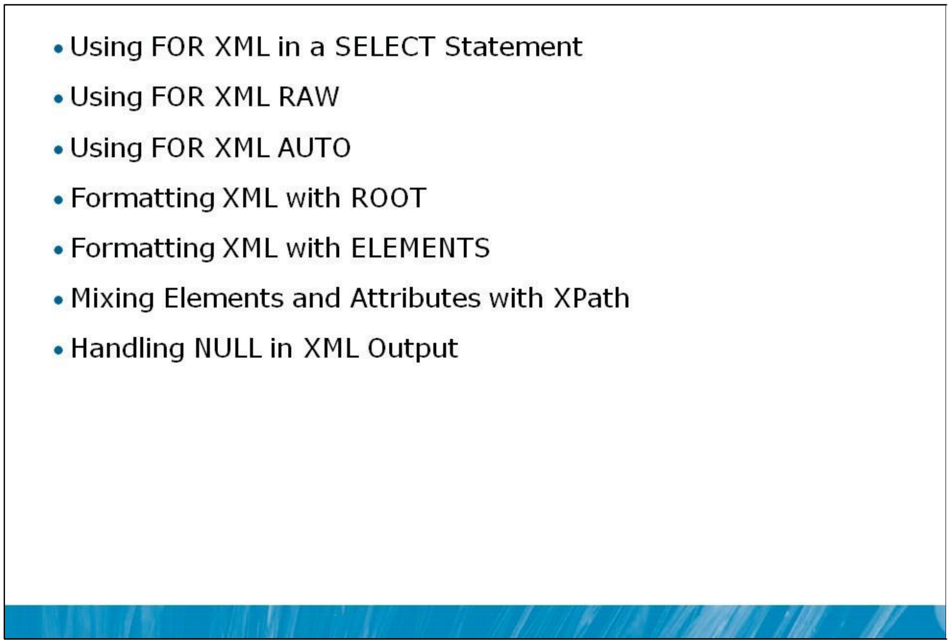
- XML is a native data type in SQL Server 2005 and later
  - Can be used for table columns, variables, and parameters
  - Non-XML data types may be converted to native XML
- XML data may be “shredded” into other data types
- XML is the output format for many SQL Server internal processes:
  - Execution plans, DDL triggers, Reporting Service report designs, extended events, event notifications, and more
- Learning to work with SQL XML will be important for query writers, database developers, and administrators alike

In this course, you will encounter XML in two main contexts: as a data format that your queries return and as a data type in columns that you will directly query. Additionally, XML has a number of uses in aspects of SQL Server:

- As a native data type, XML can be used for variables and stored procedure parameters as well as for storage in columns. The XML data type also provides support for indexes and type validation through schemas.
- XML may be decomposed or "shredded" into individual components, and then manipulated as expressions, stored in tables, etc.
- As an output format for execution plans.
- As the output format of SQL Server system functions, such as the EVENT_DATA returned by some triggers, event notifications, and extended events.
- As the storage format of a number of SQL Server technologies, such as Reporting Services report definitions and XMLA scripts in Analysis Services.
- As an input format for data converted to other data types via the OPENXML function.

As a result of the spread of XML throughout the SQL Server ecosystem, it will be useful for all database professionals to learn to work with XML.

Lesson 4
Converting Query Results to XML with FOR XML



- Using FOR XML in a SELECT Statement
- Using FOR XML RAW
- Using FOR XML AUTO
- Formatting XML with ROOT
- Formatting XML with ELEMENTS
- Mixing Elements and Attributes with XPath
- Handling NULL in XML Output

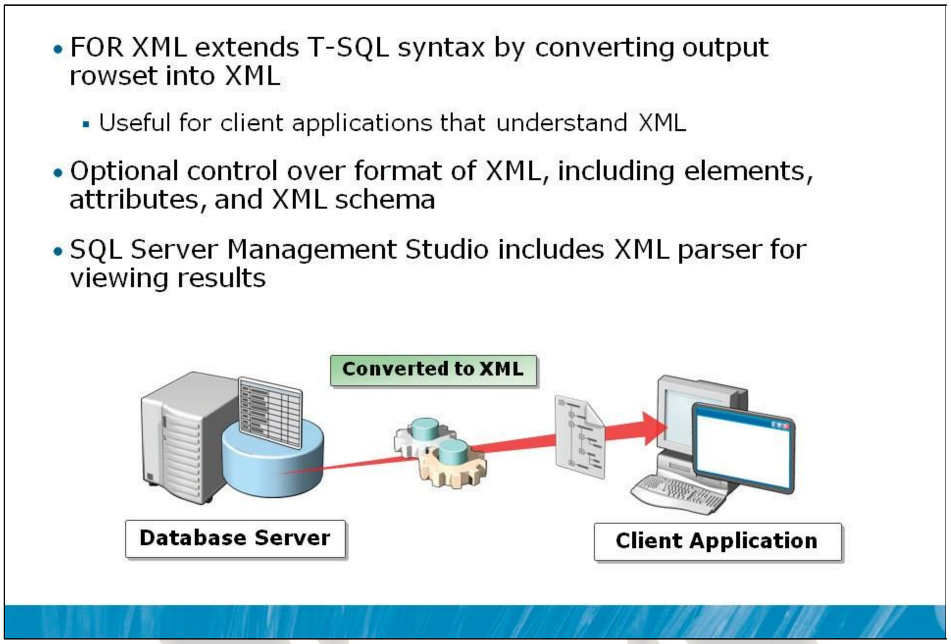
In this lesson, you will learn how to convert the output of relational queries into an XML result.

Lesson Objectives

After completing this lesson, you will be able to:

- Write queries that retrieve tabular data and return it as XML.
- Describe the differences between FOR XML clauses using the RAW, AUTO, and PATH options.
- Choose the proper FOR XML options to represent data as elements or attributes.
- Handle NULLs in XML output.

Using FOR XML in a SELECT Statement



- FOR XML extends T-SQL syntax by converting output rowset into XML
  - Useful for client applications that understand XML
- Optional control over format of XML, including elements, attributes, and XML schema
- SQL Server Management Studio includes XML parser for viewing results

Converted to XML

Database Server

Client Application

In the SELECT statements you have written so far in this course, the results have been returned in a tabular format. FOR XML is an extension to the T-SQL SELECT syntax that converts the tabular output of a SELECT statement into a stream of XML. This will allow client applications that need data in an XML format to receive it without further conversion needed.

Note As previously mentioned, the FOR XML clause does not generate a prolog (processor instructions), that many XML parsers, such as the one used by Internet Explorer, require. Some further modification of the XML will be necessary to create well-formed XML. The XML parser used by SQL Server Management Studio (SSMS) will accept XML without a prolog, but will mark the XML with syntax errors.

To use a FOR XML clause, add it to the end of a SELECT statement, along with one of its options: RAW, AUTO, PATH, or EXPLICIT. You can add further options to control the format and representation of the data in XML.

Basic syntax of the FOR XML clause:

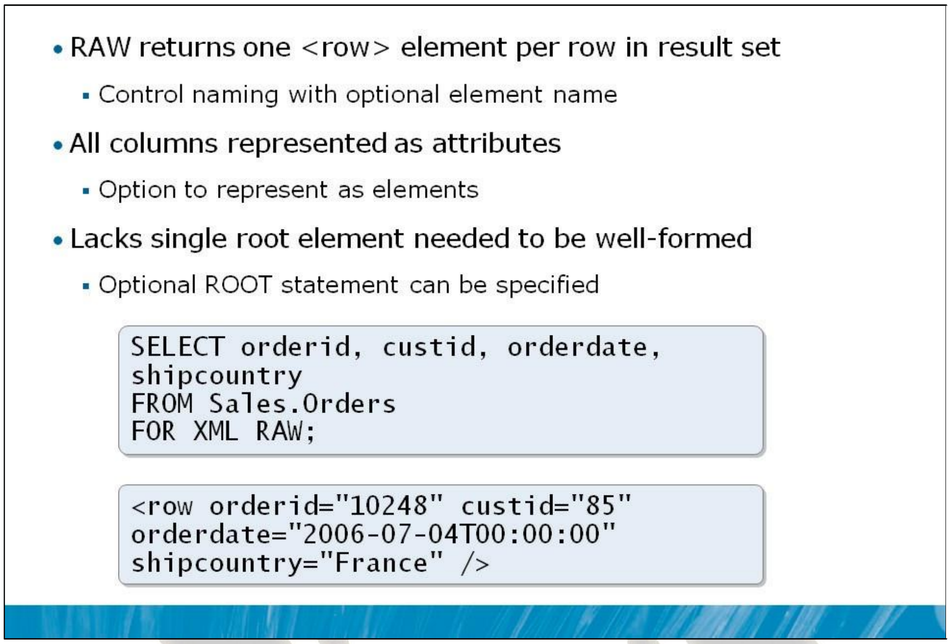
```
SELECT <select_list>
FROM   <table_source>
FOR XML RAW [('ElementName')] |
        AUTO , ELEMENTS [ XSINIL] |
        PATH [ ('ElementName') ] [ , ELEMENTS [ XSINIL][ , ROOT [ ('RootName') ] ];
```

In this lesson, you will see the use of the RAW, AUTO, and PATH modes, as well as the use of XSINIL, ROOT, and ELEMENTS. (EXPLICIT mode is beyond the scope of this course.)

For More Information See "Use FOR XML and OPENXML to Publish and Process XML Data" in Books Online at http://msdn.microsoft.com/en-us/library/ms191268(v=SQL.110).aspx and "FOR XML (SQL Server)" at http://msdn.microsoft.com/en-us/library/ms178107(v=SQL.110).aspx.

Question: What applications in your environment cannot query SQL Server but can read XML directly?

Using FOR XML RAW



- RAW returns one <row> element per row in result set
  - Control naming with optional element name
- All columns represented as attributes
  - Option to represent as elements
- Lacks single root element needed to be well-formed
  - Optional ROOT statement can be specified

```
SELECT orderid, custid, orderdate,
shipcountry
FROM Sales.Orders
FOR XML RAW;
```

```
<row orderid="10248" custid="85"
orderdate="2006-07-04T00:00:00"
shipcountry="France" />
```

The RAW mode of FOR XML provides conversion from tabular data to XML. By default, each row in the source is returned as an element named <row>. No root element is created, and each column value is represented as an attribute. The results are not well-formed.

The following example shows the minimal conversion that the RAW mode returns, using default options:

```
SELECT TOP(5) orderid, custid, shipcountry
FROM Sales.Orders
FOR XML RAW;
```

Returns:

```
<row orderid="10248" custid="85" shipcountry="France" />
<row orderid="10249" custid="79" shipcountry="Germany" />
<row orderid="10250" custid="34" shipcountry="Brazil" />
<row orderid="10251" custid="84" shipcountry="France" />
<row orderid="10252" custid="76" shipcountry="Belgium" />
```

Note The results above are limited to five rows for brevity.

In the following example, the RAW mode is being applied to the results of a JOIN operation:

```
SELECT TOP(5) o.orderid, c.custid, c.companyname
FROM Sales.Orders AS o JOIN Sales.Customers AS c
ON o.custid=c.custid
FOR XML RAW;
```

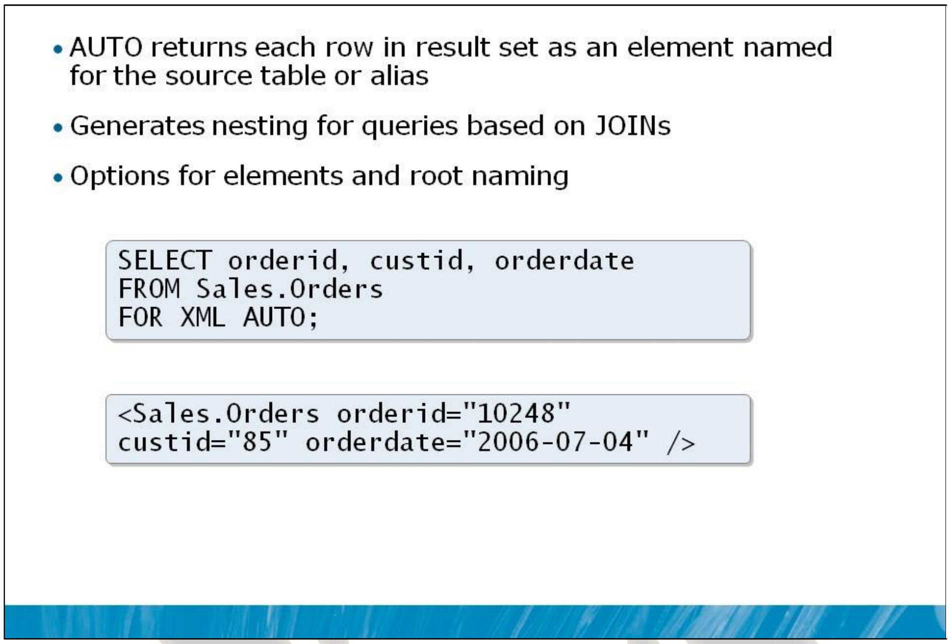
This returns the following results. Note that the results are not hierarchical, but rather are flattened per row:

```
<row orderid="10643" custid="1" companyname="Customer NRZBB" />
<row orderid="10692" custid="1" companyname="Customer NRZBB" />
<row orderid="10702" custid="1" companyname="Customer NRZBB" />
<row orderid="10835" custid="1" companyname="Customer NRZBB" />
<row orderid="10952" custid="1" companyname="Customer NRZBB" />
```

You will learn how to control the representation of the rows, add a root element, and use elements instead of attributes later in this lesson.

For More Information See Use "RAW Mode with FOR XML" in Books Online at http://msdn.microsoft.com/en-us/library/ms175140(v=SQL.110).aspx.

Using FOR XML AUTO



- AUTO returns each row in result set as an element named for the source table or alias
- Generates nesting for queries based on JOINs
- Options for elements and root naming

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
FOR XML AUTO;
```

```
<Sales.Orders orderid="10248"
custid="85" orderdate="2006-07-04" />
```

Like the RAW mode, the AUTO mode converts tabular data to XML. By default, each row in the source is returned as an element named for the row's source table or alias. No root element is created, and each column value is represented as an attribute. The results are not well-formed.

The following query uses the AUTO mode’s default options for the tabular-to-XML conversion:

```
SELECT TOP(5) orderid, custid, shipcountry
FROM Sales.Orders
FOR XML AUTO;
```

The query returns:

```
<Sales.Orders orderid="10248" custid="85" shipcountry="France" />
<Sales.Orders orderid="10249" custid="79" shipcountry="Germany" />
<Sales.Orders orderid="10250" custid="34" shipcountry="Brazil" />
<Sales.Orders orderid="10251" custid="84" shipcountry="France" />
<Sales.Orders orderid="10252" custid="76" shipcountry="Belgium" />
```

Note The results above are limited to five rows to fit the page.

In this query, the AUTO mode is being applied to the results of a JOIN operation:

```
SELECT TOP(5) o.orderid, c.custid, c.companyname
FROM Sales.Orders AS o JOIN Sales.Customers AS c
ON o.custid=c.custid
FOR XML AUTO;
```

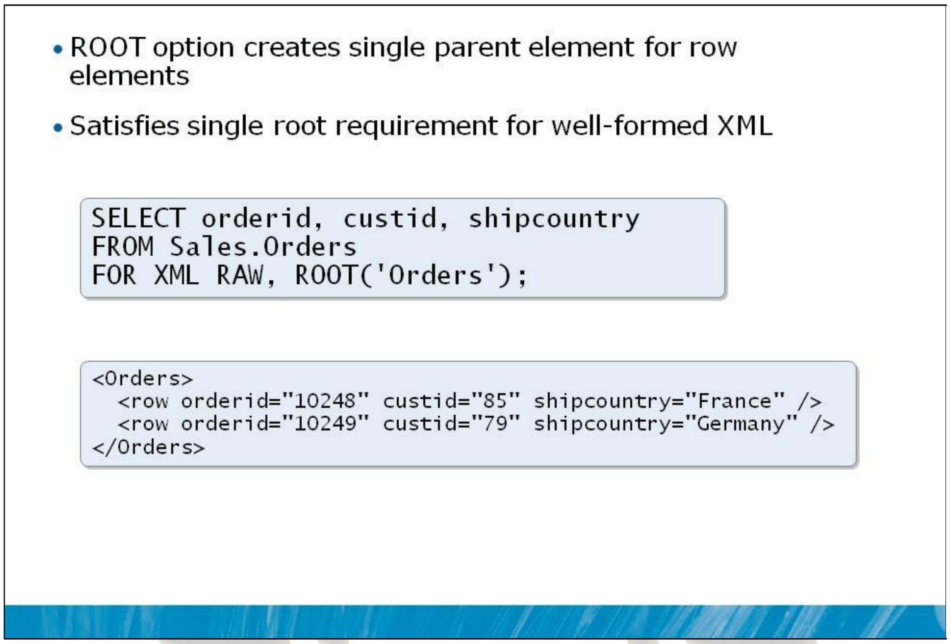
The query returns the following results. Note that the results represent a hierarchy of parent table to child table. The order is determined by the order of the columns in the SELECT list.

```
<o orderid="10643">
  <c custid="1" companyname="Customer NRZBB" />
</o>
<o orderid="10692">
  <c custid="1" companyname="Customer NRZBB" />
</o>
<o orderid="10702">
  <c custid="1" companyname="Customer NRZBB" />
</o>
<o orderid="10835">
  <c custid="1" companyname="Customer NRZBB" />
</o>
<o orderid="10952">
  <c custid="1" companyname="Customer NRZBB" />
</o>
```

You will learn how to control the representation of the rows, add a root element, and use elements instead of attributes later in this lesson.

For More Information See "Use AUTO Mode with FOR XML" in Books Online at http://msdn.microsoft.com/en-us/library/ms188273(v=SQL.110).aspx.

Formatting XML with ROOT



- ROOT option creates single parent element for row elements
- Satisfies single root requirement for well-formed XML

```
SELECT orderid, custid, shipcountry
FROM Sales.Orders
FOR XML RAW, ROOT('Orders');
```

```
<Orders>
  <row orderid="10248" custid="85" shipcountry="France" />
  <row orderid="10249" custid="79" shipcountry="Germany" />
</Orders>
```

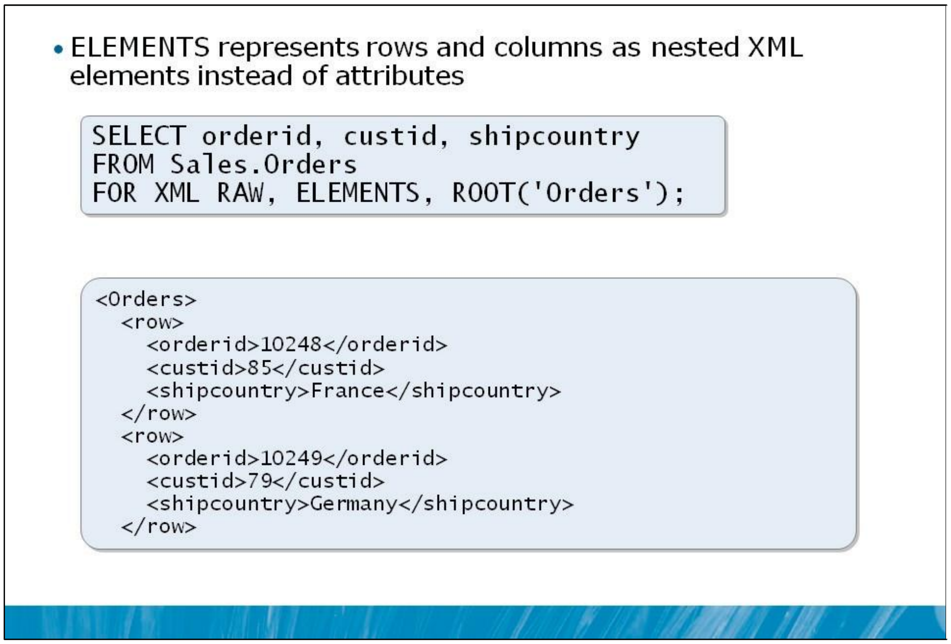
To make the results of your FOR XML clause comply better with the requirements of well-formed XML, you will want to add a single root element, under which all the other elements will be nested. To accomplish this, use the ROOT('<root_name>') option, which is supported by FOR XML RAW, FOR XML AUTO, and FOR XML PATH:

```
SELECT orderid, custid, shipcountry
FROM Sales.Orders
FOR XML RAW, ROOT('Orders');
```

This returns the following results, which show a newly created root element named Orders:

```
<Orders>
  <row orderid="10248" custid="85" shipcountry="France" />
  <row orderid="10249" custid="79" shipcountry="Germany" />
  <row orderid="10250" custid="34" shipcountry="Brazil" />
  <row orderid="10251" custid="84" shipcountry="France" />
  <row orderid="10252" custid="76" shipcountry="Belgium" />
</Orders>
```

Formatting XML with ELEMENTS



- ELEMENTS represents rows and columns as nested XML elements instead of attributes

```
SELECT orderid, custid, shipcountry
FROM Sales.Orders
FOR XML RAW, ELEMENTS, ROOT('Orders');
```

```
<Orders>
  <row>
    <orderid>10248</orderid>
    <custid>85</custid>
    <shipcountry>France</shipcountry>
  </row>
  <row>
    <orderid>10249</orderid>
    <custid>79</custid>
    <shipcountry>Germany</shipcountry>
  </row>
```

You may have noticed that, by default, FOR XML RAW and FOR XML AUTO represent the values of each column as an attribute (i.e., as a name-value pair) delimited with double quotation marks. Some applications may require that elements be represented in element-centric form. To accomplish this, use the ELEMENTS option, which is supported by FOR XML RAW, FOR XML AUTO, and FOR XML PATH:

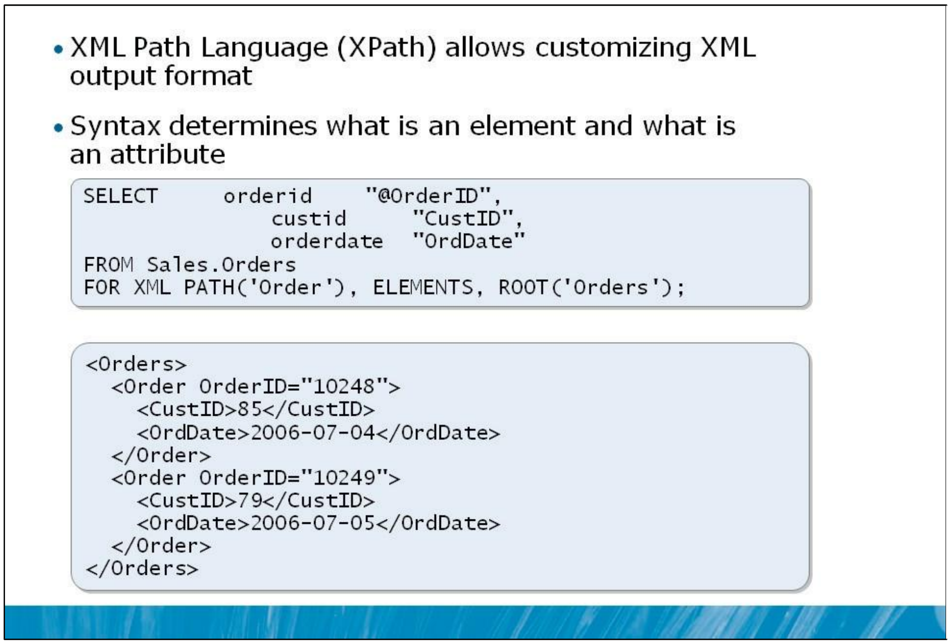
```
SELECT orderid, custid, shipcountry
FROM Sales.Orders
FOR XML RAW, ELEMENTS, ROOT('Orders');
```

This returns the following results, which show each column's value represented as an element nested within the element representing the row:

```
<Orders>
  <row>
    <orderid>10248</orderid>
    <custid>85</custid>
    <shipcountry>France</shipcountry>
  </row>
  <row>
    <orderid>10249</orderid>
    <custid>79</custid>
    <shipcountry>Germany</shipcountry>
  </row>
  <row>
    <orderid>10250</orderid>
    <custid>34</custid>
    <shipcountry>Brazil</shipcountry>
  </row>
  <row>
    <orderid>10251</orderid>
    <custid>84</custid>
    <shipcountry>France</shipcountry>
  </row>
  <row>
    <orderid>10252</orderid>
    <custid>76</custid>
    <shipcountry>Belgium</shipcountry>
  </row>
</Orders>
```

Note To mix elements and attributes, you can use XPath, which is covered in the next topic.

Mixing Elements and Attributes with XPath



- XML Path Language (XPath) allows customizing XML output format
- Syntax determines what is an element and what is an attribute

```
SELECT     orderid    "@OrderID",
                custid      "CustID",
                orderdate   "OrdDate"
FROM Sales.Orders
FOR XML PATH('Order'), ELEMENTS, ROOT('Orders');
```

```
<Orders>
  <Order OrderID="10248">
    <CustID>85</CustID>
    <OrdDate>2006-07-04</OrdDate>
  </Order>
  <Order OrderID="10249">
    <CustID>79</CustID>
    <OrdDate>2006-07-05</OrdDate>
  </Order>
</Orders>
```

SQL Server 2005 introduced support for XML Path Language (XPath), which is a query language for retrieving nodes from an XML document. When used in a FOR XML clause, the PATH mode allows you to mark up columns in the SELECT clause using special notation. When the query is processed, the PATH mode interprets the markup and uses it to determine which columns should be represented as elements, which should be represented as attributes, and even allows for arbitrary hierarchies to be created within the XML stream.

Some of the possible notation options for the PATH mode include:

Column notation	*Description*
@<alias>	Returns the value of the column as an attribute of the row element named <alias>.
<alias>	Returns the value of the column as a child element of the row element named <alias>.
<parent_alias>/<alias>	Creates a child element of the row named <parent_alias> and nests a child element with the value of the column named <alias>.
text()	Returns the value of the column as a text node to the XML stream.

In the following example, the orderid column will be represented as an attribute of the row named OrderId, and the custid and shipcountry columns will be represented as child elements in the resulting XML stream:

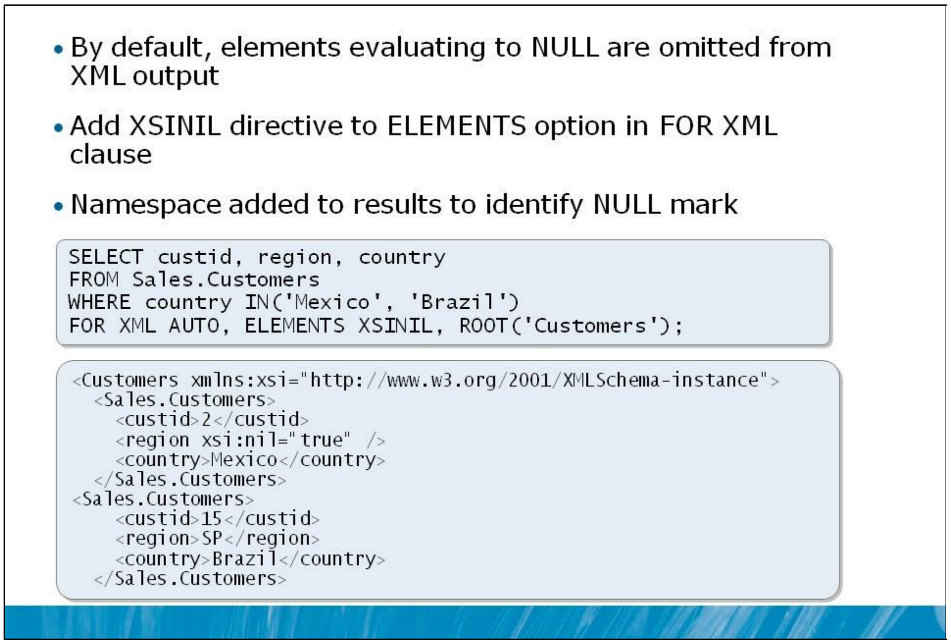
```
SELECT orderid       "@OrderID",
       custid        "CustID",
       shipcountry   "ShipCountry"
FROM Sales.Orders
FOR XML PATH('Order'), ELEMENTS, ROOT('Orders');
```

The results:

```
<Orders>
  <Order OrderID="10248">
    <CustID>85</CustID>
    <ShipCountry>France</ShipCountry>
  </Order>
  <Order OrderID="10249">
    <CustID>79</CustID>
    <ShipCountry>Germany</ShipCountry>
  </Order>
  <Order OrderID="10252">
    <CustID>76</CustID>
    <ShipCountry>Belgium</ShipCountry>
  </Order>
</Orders>
```

For More Information See "Use PATH Mode with FOR XML" in Books Online at http://msdn.microsoft.com/en-us/library/ms189885(v=SQL.110).aspx.

Handling NULL in XML Output



- By default, elements evaluating to NULL are omitted from XML output
- Add XSINIL directive to ELEMENTS option in FOR XML clause
- Namespace added to results to identify NULL mark

```
SELECT custid, region, country
FROM Sales.Customers
WHERE country IN('Mexico', 'Brazil')
FOR XML AUTO, ELEMENTS XSINIL, ROOT('Customers');
```

```
<Customers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Sales.Customers>
    <custid>2</custid>
    <region xsi:nil="true" />
    <country>Mexico</country>
  </Sales.Customers>
<Sales.Customers>
    <custid>15</custid>
    <region>SP</region>
    <country>Brazil</country>
  </Sales.Customers>
```

In a FOR XML operation, any results within a query that evaluate to NULL are removed from the resulting XML by default. This is due to a silent default modifier, ABSENT

The following query includes customers that have placed no orders:

```
SELECT c.custid, c.companyname, o.orderid, o.orderdate
FROM Sales.Customers AS c LEFT OUTER JOIN Sales.Orders AS o ON c.custid =o.custid
FOR XML RAW, ROOT('CustOrders');
```

In the results, note the missing orderid and orderdate attributes for the last two customers:

```
<CustOrders>
  <row custid="9" companyname="Customer RTXGC" orderid="11076" orderdate="2008-05-06" />
  <row custid="7" companyname="Customer QXVLA" orderid="10265" orderdate="2006-07-25" />
  <row custid="22" companyname="Customer DTDMN" />
  <row custid="57" companyname="Customer WVAXS" />
</CustOrders>
```

This will also occur when using the ELEMENTS option:

```
SELECT c.custid, c.companyname, o.orderid, o.orderdate
FROM Sales.Customers AS c LEFT OUTER JOIN Sales.Orders AS o ON c.custid =o.custid
FOR XML RAW, ELEMENTS, ROOT('CustOrders');
```

The results:

```
<CustOrders>
<row>
    <custid>9</custid>
    <companyname>Customer RTXGC</companyname>
    <orderid>11076</orderid>
    <orderdate>2008-05-06</orderdate>
  </row>
  <row>
    <custid>65</custid>
    <companyname>Customer NYUHS</companyname>
    <orderid>11077</orderid>
    <orderdate>2008-05-06</orderdate>
  </row>
  <row>
    <custid>22</custid>
    <companyname>Customer DTDMN</companyname>
  </row>
  <row>
    <custid>57</custid>
    <companyname>Customer WVAXS</companyname>
  </row>
</CustOrders>
```

To force FOR XML to include elements for values evaluating to NULL, add the XSINIL directive to the ELEMENTS option:

```
SELECT c.custid, c.companyname, o.orderid, o.orderdate
FROM Sales.Customers AS c LEFT OUTER JOIN Sales.Orders AS o
ON c.custid =o.custid
FOR XML RAW, ELEMENTS XSINIL, ROOT('CustOrders');
```

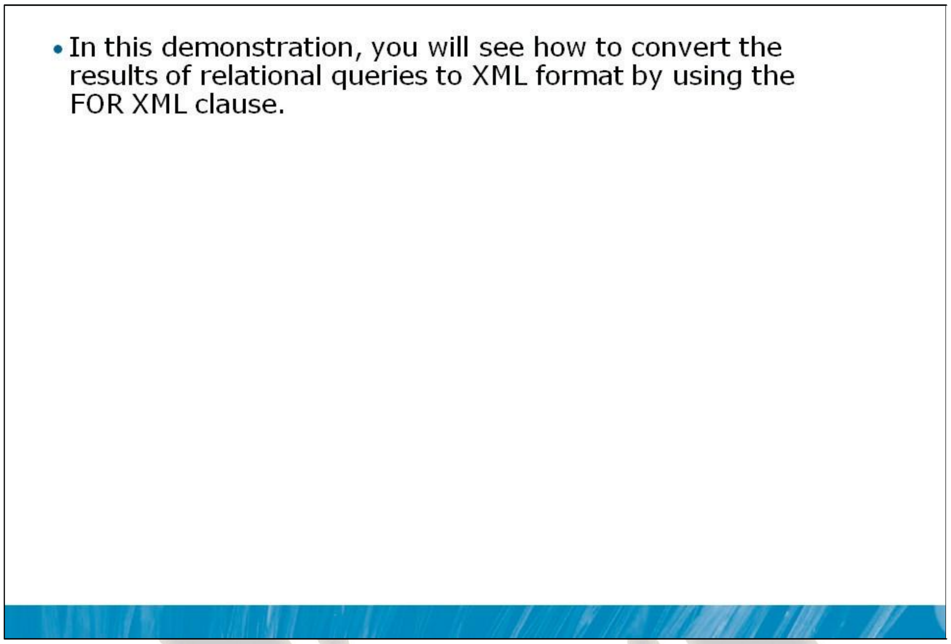
The results will include not only elements marked with nil="true" but also a namespace declaration in the root element that defines the source of the nil element:

```
<CustOrders xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <custid>85</custid>
    <companyname>Customer ENQZT</companyname>
    <orderid>10248</orderid>
    <orderdate>2006-07-04</orderdate>
  </row>
  <row>
    <custid>79</custid>
    <companyname>Customer FAPSM</companyname>
    <orderid>10249</orderid>
    <orderdate>2006-07-05</orderdate>
  </row>
  <row>
    <custid>22</custid>
    <companyname>Customer DTDMN</companyname>
    <orderid xsi:nil="true" />
    <orderdate xsi:nil="true" />
  </row>
  <row>
    <custid>57</custid>
    <companyname>Customer WVAXS</companyname>
    <orderid xsi:nil="true" />
    <orderdate xsi:nil="true" />
  </row>
</CustOrders>
```

Note A discussion of namespaces is beyond the scope of this topic.

For More Information See "Generate Elements for NULL Values with the XSINIL Parameter" in Books Online at http://msdn.microsoft.com/en-us/library/ms178079(v=SQL.110).aspx.

Demonstration: Converting Query Results to XML with FOR XML



- In this demonstration, you will see how to convert the results of relational queries to XML format by using the FOR XML clause.

Demonstration Steps

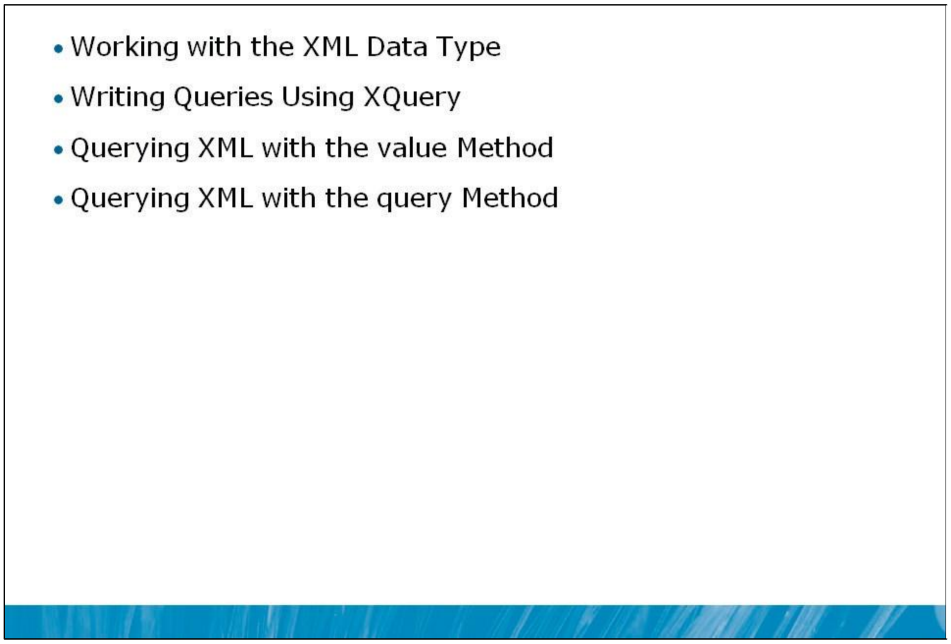
1. On the virtual machine, click **Start**, click **All Programs**, click **Microsoft SQL Server 2012**, and click **SQL Server Management Studio**. In the **Connect to Server** window, type **Proseware** in the **Server name** text box and click **Connect**. On the **File** menu, click **Open** and click **Project/Solution**. Navigate to F:\10774A_Labs\10774A_AC_PRJ\10774A_AC_PRJ.ssmssln and click **Open**.

2. On the **View** menu, click **Solution Explorer**.

3. Open the 21 – Demonstration B.sql script file.

4. Follow the instructions contained within the comments of the script file.

Lesson 5
Querying XML Data



- Working with the XML Data Type
- Writing Queries Using XQuery
- Querying XML with the value Method
- Querying XML with the query Method

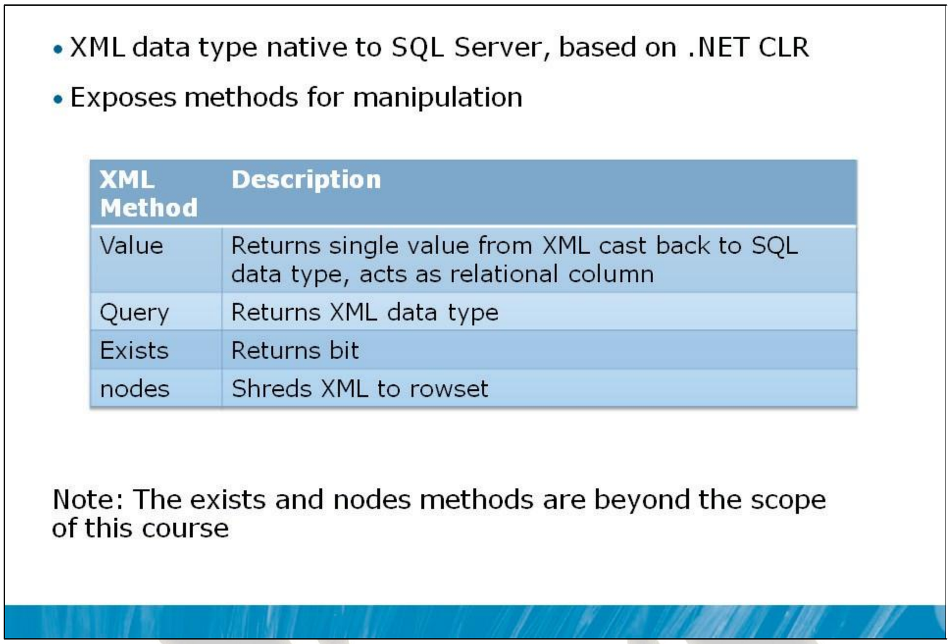
In this module, you have learned how to convert tabular query results into XML. Now you will see how to query XML data in its native form, without conversion. To do so, you will learn the basics of XQuery, the standards-based query language of XML.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the capabilities and usage of the XML data type.
- Describe the use of XQuery components in relational T-SQL queries.
- Write queries against XML data using the value method.
- Write queries against XML data using the query method.

Working with the XML Data Type



- XML data type native to SQL Server, based on .NET CLR
- Exposes methods for manipulation

XML Method	Description
Value	Returns single value from XML cast back to SQL data type, acts as relational column
Query	Returns XML data type
Exists	Returns bit
nodes	Shreds XML to rowset

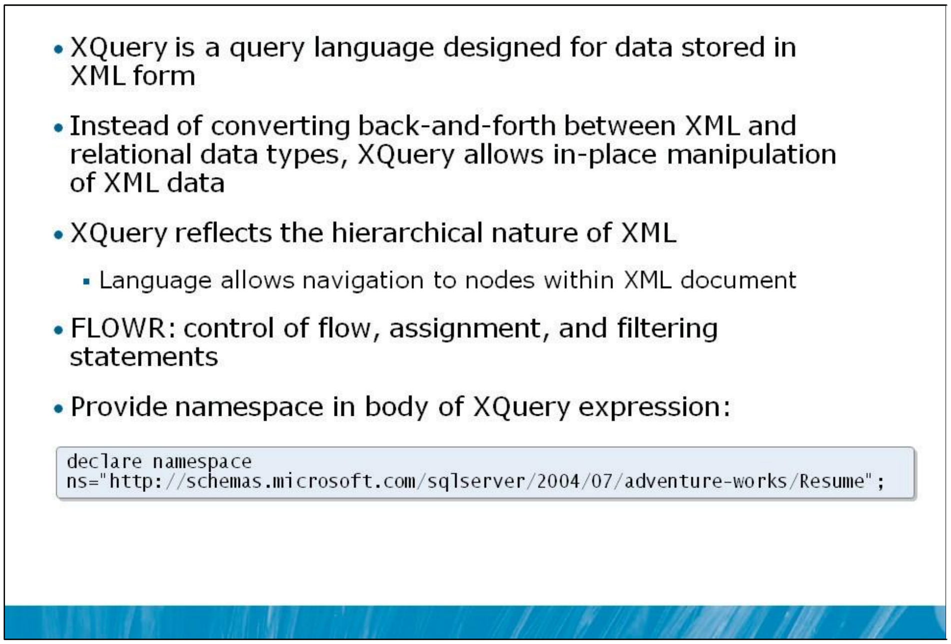
Note: The exists and nodes methods are beyond the scope of this course

SQL Server 2005 introduced support for a native XML data type. The xml type provides a number of capabilities, including xml-specific indexes and support for validation through XML Schemas. By storing XML data natively in an xml data type and directly manipulating it, less effort can be spent converting it back and forth between text and other relational data types. In order to accomplish this, you will learn about some of the methods exposed by the xml data type:

xml Data Type Method	Description
value	Returns a single value from an XQuery expression, cast back to a SQL data type. Displayed as relational column in the query results.
query	Returns untyped (not validated) XML from an XQuery expression as the xml data type.
exist	Checks for existence of a node within an XML document and returns a bit.
nodes	Shreds XML to rowsets, converting XML data to tabular data.
modify	Provides data manipulation support for inserting, updating, and deleting XML elements.

For More Information Additional reading on xml data type methods can be found in Books Online at http://msdn.microsoft.com/en-us/library/ms190798(v=SQL.110).aspx.

Writing Queries Using XQuery



- XQuery is a query language designed for data stored in XML form
- Instead of converting back-and-forth between XML and relational data types, XQuery allows in-place manipulation of XML data
- XQuery reflects the hierarchical nature of XML
    - Language allows navigation to nodes within XML document
- FLOWR: control of flow, assignment, and filtering statements
- Provide namespace in body of XQuery expression:

```
declare namespace
ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
```

To retrieve data from within XML stored in a column or variable of the xml data type, you can use the value and query methods provided. While the syntax for XQuery is very different from T-SQL, many of the same core elements exist in both languages: You need to provide a description of what you want retrieved (as in the T-SQL SELECT clause), the source of the XML data (similar to the T-SQL FROM clause), and typically a predicate to filter the results (similar to the WHERE clause in T-SQL). However, there are also some significant differences:

- The body of the XQuery expression is a string, delimited by single quotation marks and enclosed within parentheses, as expressed in this pseudo-code:

```
SELECT <relational_column_name>,<xml_column_name>.value('<XQuery_expression>') AS
Result
FROM <table_source>;
```

- XQuery provides its own operator characters and its own navigational notation for traversing the hierarchy contained within an XML document. For example, you use the string "eq" instead of an equals sign "=" and forward slashes "/" to separate the names of elements, forming a navigation path. The following snippet of XML illustrates this, showing the navigation from a root element Orders down to a child element Order down to another child OrderDate:

```
/Orders/Order/OrderDate
```

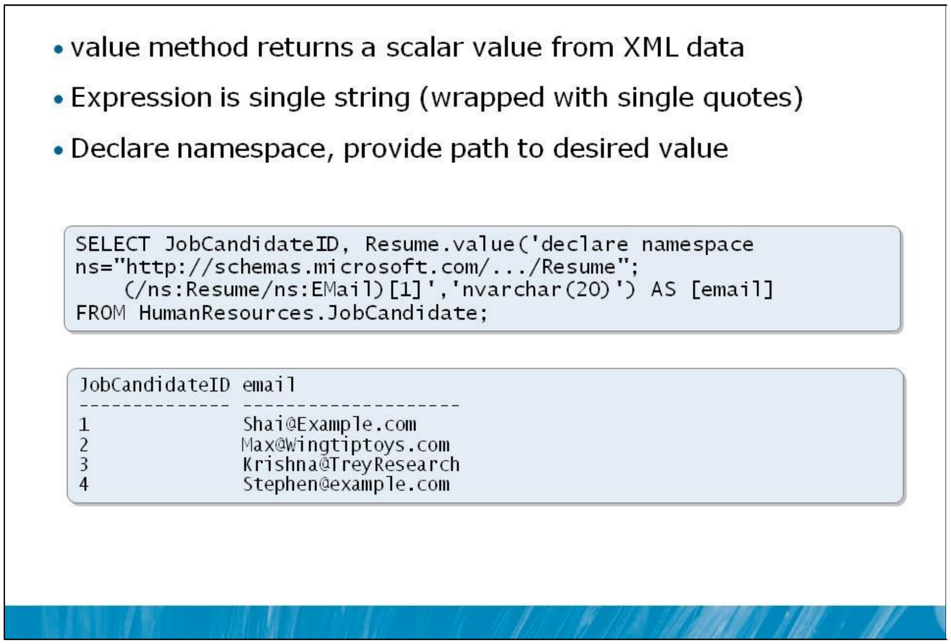
- XQuery expressions require a prolog in the form of an XML namespace declaration. This might appear as follows:

```
declare namespace ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/Resume";
```

- XQuery expressions can include control-of-flow components, string literal assignment, filtering, and sorting through FLWOR (FOR, LET, ORDER BY, WHERE, RETURN) statements.

Note The use of FLWOR is beyond the scope of this course. For more information, see course 10776A: *Implementing a SQL Server 2012 Database*. For more information on the XQuery language and its elements, see Books Online at http://msdn.microsoft.com/en-us /library/ms189075(v=SQL.110).aspx.

Querying XML Data with the value Method



- value method returns a scalar value from XML data
- Expression is single string (wrapped with single quotes)
- Declare namespace, provide path to desired value

```
SELECT JobCandidateID, Resume.value('declare namespace
ns="http://schemas.microsoft.com/.../Resume";
    (/ns:Resume/ns:EMail)[1]','nvarchar(20)') AS [email]
FROM HumanResources.JobCandidate;
```

```
JobCandidateID email
-------------- --------------------
1              Shai@Example.com
2              Max@Wingtiptoys.com
3              Krishna@TreyResearch
4              Stephen@example.com
```

The value method of the xml data type allows you to return a scalar (single) value from XML data. The value method takes two parameters:

1. An XQuery expression, wrapped in single quotes, including a namespace declaration in the prolog.
2. A SQL Server data type to which the results will be cast.

A simplified view of the syntax appears as follows:

```
SELECT <xmlcolumn>.value('<namespace declaration>;(<XQuery expression with single
result>','<data_type_to_cast_to') AS <alias>
```

The following query uses XQuery and the value method to return email addresses from the Resume column of the AdventureWorks HumanResources.JobCandidate table. (The Resume column uses the xml data type.)

```
1)  SELECT JobCandidateID, Resume.value('
2)  declare namespace ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
    works/Resume";
3)  (/ns:Resume/ns:EMail)[1]','nvarchar(20)') AS [email]
4)  FROM HumanResources.JobCandidate;
```

Let's examine the components of this query. Note that line numbers have been added for clarity. Also note that the namespace declaration, which wraps from line 2 to line 3, is required to be on one line in order to execute. Let's look at the query line by line:

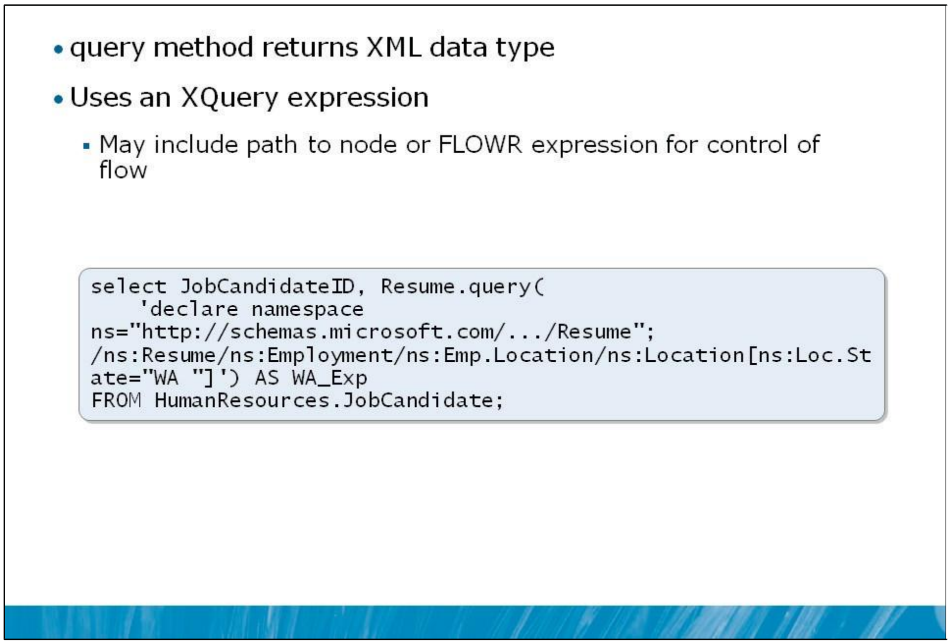
1. The SELECT clause invokes the value method of the Resume column and opens the query.
2. The namespace for the element names is declared. For the purposes of this example, the namespace definition has been retrieved from the AdventureWorks database through querying system views. You may need to ask your administrator for the proper Uniform Resource Identifier (URI) for your data. The declaration defines the prefix "ns" for elements in the namespace, used later in the query.

3. The XQuery expression, which navigates to the EMail element, appears in parentheses. It is followed by an index number in square brackets [1]. This tells the query to retrieve the first email address it finds at that path. The rest of the line sets nvarchar(2) as the data type to cast the results to, and creates an alias for the column.

A snippet of the results, executed against the AdventureWorks2008R2 database, appear below:

```
JobCandidateID email
-------------- --------------------
1              Shai@Example.com
2              Max@Wingtiptoys.com
3              Krishna@TreyResearch
4              Stephen@example.com
```

Querying XML Data with the query Method



- query method returns XML data type
- Uses an XQuery expression
  - May include path to node or FLOWR expression for control of flow

```
select JobCandidateID, Resume.query(
    'declare namespace
ns="http://schemas.microsoft.com/.../Resume";
/ns:Resume/ns:Employment/ns:Emp.Location/ns:Location[ns:Loc.St
ate="WA "]') AS WA_Exp
FROM HumanResources.JobCandidate;
```

The query method of the xml data type returns results in the xml data type (unlike the value method) from XML data. The query method takes a single parameter, an XQuery expression, wrapped in single quotes, including a namespace declaration in the prolog.

A simplified view of the syntax appears as follows:

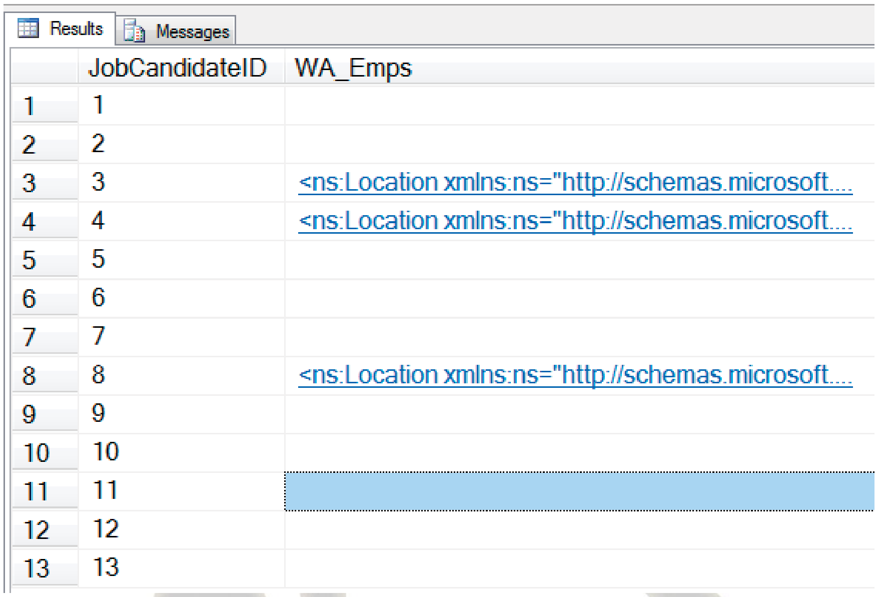
```
SELECT <xmlcolumn>.query('<namespace declaration>;<XQuery expression>');
```

The query method allows you to build string output in your XML results, as well as retrieve values from within the XML data through a navigation path or FLWOR language.

The following example retrieves job candidates from the HumanResources.JobCandidate table. When a candidate's resume includes work experience in the state of Washington, related location information is returned. Note the use of the equality predicate [ns:Loc.State="WA "] to filter results:

```
select JobCandidateID, Resume.query('declare namespace
ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume";
/ns:Resume/ns:Employment/ns:Emp.Location/ns:Location[ns:Loc.State="WA "]') AS WA_Emps
FROM HumanResources.JobCandidate;
```

The results, as viewed in SSMS, look like this:

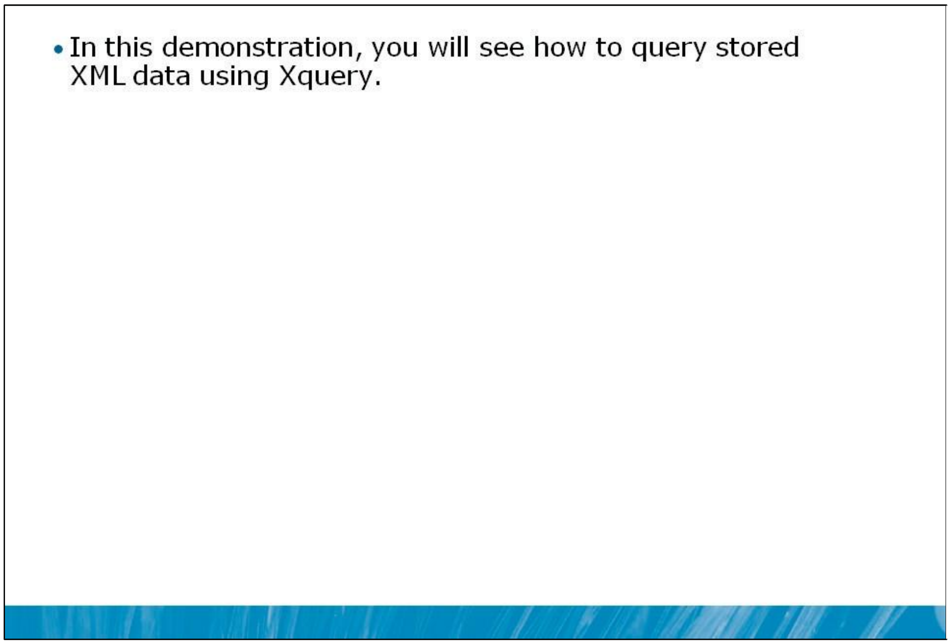


	JobCandidateID	WA_Emps
1	1	
2	2	
3	3	<ns:Location xmlns:ns="http://schemas.microsoft....
4	4	<ns:Location xmlns:ns="http://schemas.microsoft....
5	5	
6	6	
7	7	
8	8	<ns:Location xmlns:ns="http://schemas.microsoft....
9	9	
10	10	
11	11	
12	12	
13	13	

Note that the results from the Resume column are represented in the xml data type. Clicking one of the result links yields the following:

```
<ns:Location xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/Resume">
  <ns:Loc.CountryRegion>US </ns:Loc.CountryRegion>
  <ns:Loc.State>WA </ns:Loc.State>
  <ns:Loc.City>Renton</ns:Loc.City>
</ns:Location>
<ns:Location xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/Resume">
  <ns:Loc.CountryRegion>US </ns:Loc.CountryRegion>
  <ns:Loc.State>WA </ns:Loc.State>
  <ns:Loc.City>Bellevue</ns:Loc.City>
</ns:Location>
<ns:Location xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/Resume">
  <ns:Loc.CountryRegion>US </ns:Loc.CountryRegion>
  <ns:Loc.State>WA </ns:Loc.State>
  <ns:Loc.City>Kent</ns:Loc.City>
</ns:Location>
```

Demonstration: Querying XML Data



- In this demonstration, you will see how to query stored XML data using Xquery.

Demonstration Steps

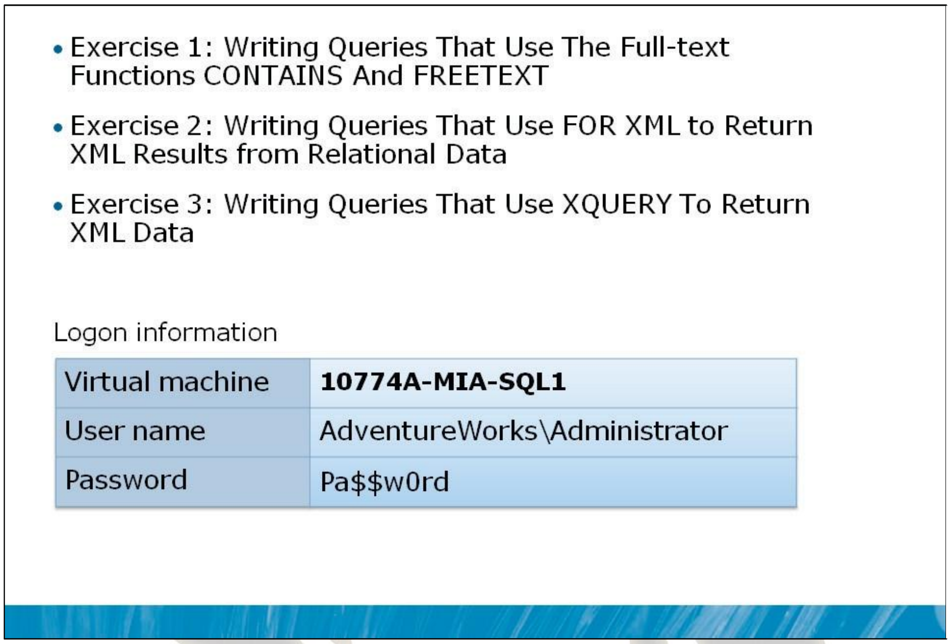
1. On the virtual machine, click **Start**, click **All Programs**, click **Microsoft SQL Server 2012**, and click **SQL Server Management Studio**. In the **Connect to Server** window, type **Proseware** in the **Server name** text box and click **Connect**. On the **File** menu, click **Open** and click **Project/Solution**. Navigate to F:\10774A_Labs\10774A_AC_PRJ\10774A_AC_PRJ.ssmssln and click **Open**.

2. On the **View** menu, click **Solution Explorer**.

3. Open the 31 – Demonstration C.sql script file.

4. Follow the instructions contained within the comments of the script file.

Lab: Writing Specialized Queries



- Exercise 1: Writing Queries That Use The Full-text Functions CONTAINS And FREETEXT
- Exercise 2: Writing Queries That Use FOR XML to Return XML Results from Relational Data
- Exercise 3: Writing Queries That Use XQUERY To Return XML Data

Logon information

Virtual machine	**10774A-MIA-SQL1**
User name	AdventureWorks\Administrator
Password	Pa$$w0rd

Lab Setup

For this lab, you will use the available virtual machine environment. Before you begin the lab, you must complete the following steps:

1. On the host computer, click **Start**, point to **Administrative Tools**, and click **Hyper-V Manager**.
2. Maximize the **Hyper-V Manager** window.
3. If the virtual machine **10774A-MIA-DC1** is not started:
   - In the **Virtual Machines** list, right-click **10774A-MIA-DC1** and click **Start**.
   - Right-click **10774A-MIA-DC1** and click **Connect**.
   - In the **Virtual Machine Connection** window, wait until the “Press CTRL+ALT+DELETE to log” on message appears, and then close the **Virtual Machine Connection** window.
4. If the virtual machine **10774A-MIA-SQL1** is not started:
   - In the **Virtual Machines** list, right-click **10774A-MIA-SQL1** and click **Start**.
   - Right-click **10774A-MIA-SQL1** and click **Connect**.
   - In the **Virtual Machine Connection** window, wait until the “Press CTRL+ALT+DELETE to log on” message appears, and then close the **Virtual Machine Connection** window.
5. In **Virtual Machine Connection** window, click the **Revert** toolbar icon.
6. If you are prompted to confirm that you want to revert, click **Revert**. Wait for the revert action to complete.

7. If you are not already logged on:
   - In the **Virtual Machine Connection** window, click the **Ctrl-Alt-Delete** menu item on the **Action** menu.
   - Click **Switch User**, and then click **Other User**.
   - Log on using the following credentials:
     - User name: **AdventureWorks\Administrator**
     - Password: **Pa$$w0rd**
8. In the **Virtual Machine Connection** window, click **Full Screen Mode** on the **View** menu.
9. If the **Server Manager** window appears, check the **Do not show me this console at logon** check box and close the **Server Manager** window.
10. On the virtual machine, click **Start**, click **All Programs**, click **Microsoft SQL Server 2012**, and click **SQL Server Management Studio**.
11. In the **Connect to Server** window, type **Proseware** in the **Server name** text box.
12. Click the **Options** button. Under **Connection Properties**, select **<Browse server>** in the **Connect to database** list. Choose **Yes** when prompted for the connection to the database. Under **User Databases**, select the **TSQL2012** database.
13. Choose the authentication type depending on the type of Microsoft SQL version:
    - For an on-premises Microsoft SQL Server instance, click the **Login** tab, select **Windows Authentication** in the **Authentication** list, and click **Connect**.
    - For Microsoft SQL Azure™, click the **Login** tab, select **SQL Server Authentication** in the **Authentication** list, type your login name in the **Login** text box and the password in the **Password** text box, and click **Connect**.

Lab Scenario

You are a business analyst for Adventure Works who will be writing reports using corporate databases stored in SQL Server 2012. You have been provided with a set of business requirements for data and you will write T-SQL queries to retrieve the specified data from the databases. Some of the data has been specially indexed to support more sophisticated linguistic queries. Other portions of the data have been stored in a native XML format. You will need to use the appropriate techniques for each type of data.

> **Important:** When comparing your results with the provided sample outputs, the column ordering and total number of affected rows should always match. However, remember that the order of the rows in the output of a query without an ORDER BY clause is not guaranteed. Therefore, the order of the rows in the sample outputs may be different than yours. Also, the answer outputs include abbreviated results.

Exercise 1: Writing Queries That Use the Full-Text Functions CONTAINS and FREETEXT

Scenario

The marketing department needs a report from the AdventureWorks2008R2 database regarding the product descriptions. Because you need to apply efficient filtering for large character strings, you will use the full-text functions based on the prepared full-text catalog and indexes.

The main tasks for this exercise are as follows:

1. Write a CONTAINS query to retrieve rows based on one or two words.
2. Compare the CONTAINS and the FREETEXT function.

X Task 1: Write a simple CONTAINS query

- Open the project file F:\10774A_Labs\10774A_AC_PRJ\10774A_AC_PRJ.ssmssln and the T-SQL script 51 - Lab Exercise 1.sql. To set your database context to that of the AdventureWorks2008R2 database, highlight the statement **USE AdventureWorks2008R2;** and execute the highlighted code. After executing this statement, the AdventureWorks2008R2 database should be selected in the **Available Databases** box. In subsequent exercises, you will simply be instructed to ensure that you are connected to the AdventureWorks2008R2 database.

- Check to see if the population of the full-text index is complete. Execute the provided T-SQL statement and observe the column has_crawl_completed. The value should be 1 to indicate that the population is complete.

```
SELECT
   is_enabled, has_crawl_completed, crawl_type, crawl_start_date, crawl_end_date
FROM sys.fulltext_indexes
WHERE OBJECT_NAME(object_id) = N'Document';
```

- Write a SELECT statement against the Production.Document table and retrieve the Title and DocumentSummary columns. Filter the result on the DocumentSummary column to contain the word "Adventure".

- Execute the written statement and compare the results that you got with the desired results shown in the file 52 - Lab Exercise 1 - Task 1 Result.txt.

X Task 2: Write a CONTAINS query to search for two words

- Modify the query in task 1 to filter the DocumentSummary column to contain the words "Adventure Works" and "pedals".

- Execute the written statement and compare the results that you got with the desired results shown in the file 53 - Lab Exercise 1 - Task 2 Result.txt.

X Task 3: Write queries to highlight the difference between the CONTAINS and FREETEXT functions

- Write a SELECT statement that uses the CONTAINS function against the DocumentSummary table to retrieve the Title and DocumentSummary columns. Filter the result on the DocumentSummary column to contain the word "component".
- Execute the written statement and compare the results that you got with the desired results shown in the file 54 - Lab Exercise 1 - Task 3_1 Result.txt.
- Modify the query to use the FREETEXT function instead of the CONTAINS function.
- Execute the written statement and compare the results that you got with the desired results shown in the file 55 - Lab Exercise 1 - Task 3_2 Result.txt.
- What is the difference between the functions CONTAINS and FREETEXT?

Results: After this exercise, you should be able to create basic queries using the full-text functions.

Exercise 2: Writing Queries That Use FOR XML to Return XML Results from Relational Data

Scenario

The IT department would like to exchange the invoiced orders information with the external payment provider using the XML structure. You will query the relational data and return it as XML.

The main tasks for this exercise are as follows:

1. Write a SELECT statement to retrieve information about the sales staff and return it as XML.
2. Write a SELECT statement to retrieve orders with order details for a specific year and return it as an XML.

X Task 1: Write a basic SELECT statement and return the result as XML

- Open the project file F:\10774A_Labs\10774A_AC_PRJ\10774A_AC_PRJ.ssmssln and the T-SQL script 61 - Lab Exercise 2.sql. Ensure that you are connected to the AdventureWorks2008R2 database.
- Write a SELECT statement to retrieve the BusinessEntityID, SalesQuota, Bonus, CommissionPct, SalesYTD, and SalesLastYearcustid columns from the Sales.SalesPerson table. Return the result as XML using the AUTO option.
- Execute the written statement and compare the results that you got with the desired results shown in the file 62 - Lab Exercise 2 - Task 1 Result.txt. You can display the XML structure if you click the column in the query result pane.

X Task 2: Return the XML result based on the ELEMENTS option

- Modify the SELECT statement in task 1 to include each column as a separate element inside the XML.
- Execute the written statement and compare the results that you got with the recommended results shown in the file 63 - Lab Exercise 2 - Task 2 Result.txt.

X Task 3: Add a root member and modify the element's name

- Modify the SELECT statement in task 2 to include a root element named SalesPersons. Also modify it so that each row has the XML element “Employee”.
- Execute the written statement and compare the results that you got with the recommended results shown in the file 64 - Lab Exercise 2 - Task 3 Result.txt.
- Execute the T-SQL statement without the FOR XML output. Observe the missing values in the specific columns. Are these columns present in the XML output?

X Task 4: Return the elements with the NULL inside the XML output

- Modify the SELECT statement in task 3 to include the information about the NULLs in the XML output.
- Execute the written statement and compare the results that you got with the recommended results shown in the file 65 - Lab Exercise 2 - Task 4 Result.txt.

X *Task 5: Prepare a SELECT statement to exchange orders information*

- Write a SELECT statement against the Sales.SalesOrderHeader and Sales.SalesOrderDetail tables to retrieve the SalesOrderNumber, OrderDate, CustomerID, SalesOrderDetailID, ProductID, OrderQty, and UnitPrice columns. Use the aliases OrderHeader and OrderDetails for needed tables. Filter the query to include only the orders with the document date of 31 July 2008. Order the result by the SalesOrderNumber column and return it as XML using the AUTO option. Add a root element called Orders.

- Execute the written statement and compare the results that you got with the recommended results shown in the file 66 - Lab Exercise 2 - Task 5_1 Result.txt.

- Change the order of the columns in the SELECT clause by specifying first the columns from the Sales.SalesOrderDetails table and then the columns from the Sales.SalesOrderHeader table.

- Execute the written statement and compare the results that you got with the recommended results shown in the file 67 - Lab Exercise 2 - Task 5_2 Result.txt.

- Is the order of the columns in the SELECT clause important when returning the result using FOR XML AUTO?

Results: After this exercise, you should know how to return the relational data as an XML.

Exercise 3: Writing Queries That Use XQuery to Return XML Data

Scenario

You have learned how to return the relational data as XML data. Now you will have to learn how to query the XML data to efficiently read the elements inside the xml data type. The HR department has stored the resume information of each job candidate as an xml data type. You will have to retrieve different elements and values from the Resume column inside the HumanResources.JobCandidate table.

The main tasks for this exercise are as follows:

1. Learn how to navigate the XML structure on a sample provided by the IT department.
2. Write a SELECT statement against the HumanResources.JobCandidate table to retrieve elements and values inside the XML data.

X Task 1: Write simple SELECT statements using XQuery

- Open the project file F:\10774A_Labs\10774A_AC_PRJ\10774A_AC_PRJ.ssmssln and the T-SQL script 71 - Lab Exercise 3.sql. Ensure that you are connected to the AdventureWorks2008R2 database.
- The IT department has provided you with sample code that stores a simple XML structure inside the variable @x. Write a SELECT statement against the XML stored in the variable @x to retrieve columns that contain:
  - The complete sequence. Use the alias completesequence.
  - The customer name element (/root/customer/name). Use the alias elementcustname.
  - The value of the customer name element (/root/customer/name). Use the alias elementcustnamevalue.
  - A new element with an XML based on name and age element. Use the alias newelement.

```
<result>Customer name is Albert and he is 32 years old.</result>
```

- Highlight the provided T-SQL code together with the written query and execute them.
- Observe and compare the results that you got with the recommended results shown in the file 72 - Lab Exercise 3 - Task 1 Result.txt.

X Task 2: Write a SELECT statement to retrieve the first name XML element

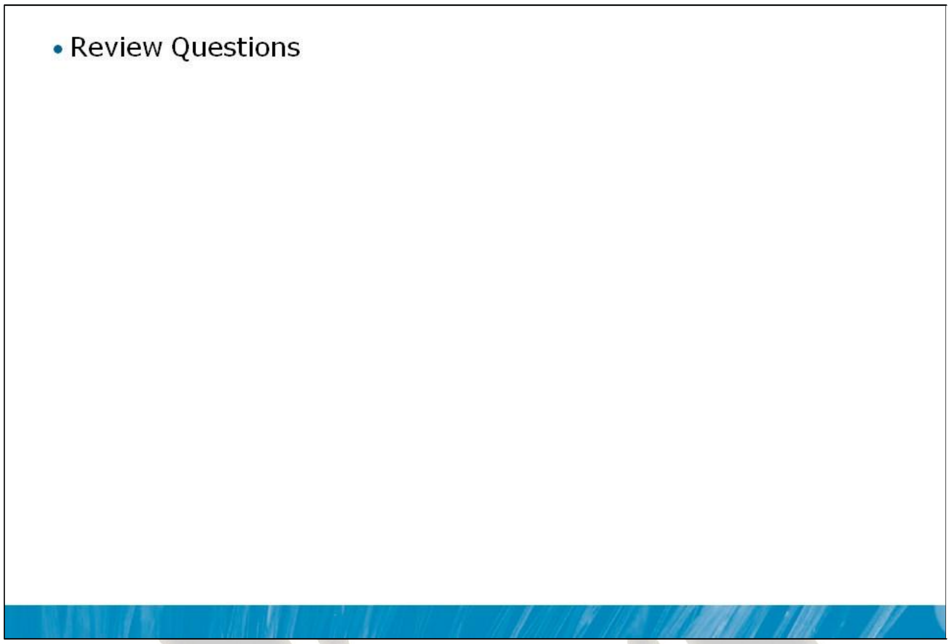
- Write a SELECT statement against the HumanResources.JobCandidate table and return the columns:
  - JobCandidateID
  - Resume
  - A calculated column in which XQuery is used to retrieve only the element ns:Resume/ns:Name/ns:Name.First from the Resume column. Use the alias XMLFirstName. Declare the namespace ns as http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume.
- Execute the written statement and compare the results that you got with the recommended results shown in the file 73 - Lab Exercise 3 - Task 2 Result.txt. Notice that the data type of the calculated column is xml.

X Task 3: Modify a SELECT statement to retrieve the value of the first name XML element

- Modify the T-SQL statement in task 2 so that the calculated column returns the value of the element ns:Resume/ns:Name/ns:Name.First as a data type NVARCHAR(50).
- Execute the written statement and compare the results that you got with the recommended results shown in the file 74 - Lab Exercise 3 - Task 3 Result.txt.

Results: After this exercise, you should have a basic understanding of how to use XQuery.

Module Review



- Review Questions

Review Questions

1. When would you use CONTAINS as opposed to FREETEXT when using full-text search?

2. Which FOR XML option allows you to specify which columns are represented as elements and which columns are represented as attributes?