SQL Server™ 2005: XML Capabilities

Table of Contents

SQL Server 2005: XML Capabilities 3
Lab Setup 4
Exercise 1 Storing XML 5
Exercise 2 XML Schemas 7
Exercise 3 Server Side Querying 10
Exercise 4 XML DataType 17

SQL Server 2005: XML Capabilities

Objectives

NOTE: This lab focuses on the concepts in this module and as a result may not comply with Microsoft® security recommendations.

NOTE: The SQL Server 2005 labs are based on beta builds of the product. The intent of these labs is to provide you with a general feel of some of the planned features for the next release of SQL Server. As with all software development projects, the final version may differ from beta builds in both features and user interface. For the latest details on SQL Server 2005, please visit http://www.microsoft.com/sql/2005/.

After completing this lab, you will be able to:

- Store XML data in a table
- Use XML Schemas to validate the XML data that you store in SQL Server 2005
- Query XML data using a variety of mechanisms
- Modify XML data

Scenario

SQL Server 2005 will introduce significant enhances in native XML support including having an XML datatype. This lab will provide you with exercises on working with native XML data in a SQL Server database including storing XML data, validating data using XML schemas, querying mechanisms and modifying the XML data in-place.

Prerequisites

- Prior working experience with SQL Server 7.0 or SQL Server 2000
- Prior working experience with T-SQL
- Basic Microsoft Windows® networking concepts
- Basic understanding of XML, XPath, XQuery, and XML Schemas

Estimated Time to Complete This Lab

60 Minutes

Lab Setup

Tasks	Detailed Steps
1. Log in.	**1.** Log in using the **Administrator** user account. The password is **Pass@word1**.

Exercise 1
Storing XML

Scenario

By providing native support for XML, SQL Server 2005 not only allows you to store and retrieve XML data, but it also allows you to perform queries based on the contents of the XML, and to modify the XML in place. In this first exercise, you will see how you can create tables that store XML, and how you can retrieve XML data.

Tasks	Detailed Steps
1. Open SQL Server Management Studio and connect to your server. NOTE: You can find the completed lab solution in C:\SQL Labs\Lab Projects\XML Lab\Solution\XMLLabSolution.ssmssln.	**1.** Click **Start | All Programs | Microsoft SQL Server 2005 CTP | SQL Server Management Studio**. **2.** If you see a **Connect to Server** dialog box, confirm that the **Server type** is **Database Engine** and that **Windows Authentication** is selected for **Authentication**. For the **Server name** setting, type **localhost**, and click **Connect**. **3.** Click **File | Open | Project/Solution**. **4.** Navigate to **C:\SQL Labs\Lab Projects\XML Lab**. **5.** Select **XMLLab.ssmssln**, and then click **Open**.
2. Create a table with an XML column.	**1.** In the Solution Explorer, expand **XMLLab**, expand **Queries**, and then double-click **StoringXML.sql.** **2.** You'll see a **Connect to Server** dialog box. Confirm that the **Server type** is **Database Engine** and that **Windows Authentication** is selected for **Authentication**. For the **Server name** setting, type **localhost**, and click **Connect**. **3.** Select the following section of code, and press **F5**. `USE AdventureWorks` `GO` **4.** Select the following code after the comment "-- Exercise 1, Task2: Create a table with an XML column" and press **F5.** `CREATE TABLE ProductDocs (ID INT IDENTITY PRIMARY KEY,` `ProductDoc XML NOT NULL)` `GO` The ProductDoc column is designed to store an XML document or a fragment of an XML document.

Tasks	Detailed Steps
3. Store XML in the table. NOTE: To view the full text of an XML data type column in the results grid, you can click the XML value, which appears as a hyperlink.	**1.** Select the SQL statement in section "Exercise 1, Task 3, Step 1…," and press **F5** to execute it: `INSERT INTO ProductDocs VALUES('` `  <Product>` `  <ProductID>1</ProductID>` `  <ProductName>Chai</ProductName>` `  <SupplierID>1</SupplierID>` `  <CategoryID>1</CategoryID>` `  <QuantityPerUnit>10 boxes x 20 bags</QuantityPerUnit>` `  <UnitPrice>18.0000</UnitPrice>` `  <UnitsInStock>39</UnitsInStock>` `  <UnitsOnOrder>0</UnitsOnOrder>` `  <ReorderLevel>10</ReorderLevel>` `  <Discontinued>0</Discontinued>` `  </Product>` `')` **2.** Next, retrieve the XML that has just been inserted using a standard SQL SELECT statement. Select the following SQL statement after the "Exercise 1, Task 3, Step 2…" comment and press **F5** to execute: `SELECT ID,ProductDoc FROM ProductDocs WHERE ID = 1` `GO` As you can see, XML can be retrieved from the database just like any other value.

Exercise 2
XML Schemas

Scenario

In this exercise, you will store an XML Schema in SQL Server 2005 to validate the XML in a column.

Schemas allow XML to be validated against a known structure. Since schemas are stored in the database, the database can ensure that an XML document conforms to a given structure at the time that it is inserted.

Tasks	Detailed Steps
1. Store a schema in SQL Server.	**1.** In the **Solution Explorer**, double-click **Schemas.sql.** **2.** If you see a **Connect to Server** dialog box, confirm that the **Server type** is **Database Engine** and that **Windows Authentication** is selected for **Authentication**. For the **Server name** setting, type **localhost**, and click **Connect.** **3.** Highlight the following section of code, and press **F5**. `USE AdventureWorks` `GO` **4.** Select the SQL statement in section “Exercise 2, Task 1, Step 4…”, and press **F5** to execute it. `CREATE XML SCHEMA COLLECTION ProductSchema AS '` `<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"` `  targetNamespace="http://www.microsoft.com/schemas/adventure-works/products"` `  xmlns:prod="http://www.microsoft.com/schemas/adventure-works/products">` `  <xs:element name="Product">` `  <xs:complexType>` `  <xs:sequence>` `  <xs:element ref="prod:ProductID" />` `  <xs:element ref="prod:ProductName" />` `  <xs:element ref="prod:SupplierID" />` `  <xs:element ref="prod:CategoryID" />` `  <xs:element ref="prod:QuantityPerUnit" />` `  <xs:element ref="prod:UnitPrice" />` `  <xs:element ref="prod:UnitsInStock" />` `  <xs:element ref="prod:UnitsOnOrder" />` `  <xs:element ref="prod:ReorderLevel" />` `  <xs:element ref="prod:Discontinued" />` `  </xs:sequence>` `  </xs:complexType>` `  </xs:element>` `  <xs:element name="ProductID" type="xs:integer" />` `  <xs:element name="ProductName" type="xs:string" />` `  <xs:element name="SupplierID" type="xs:integer" />` `  <xs:element name="CategoryID" type="xs:integer" />`

Tasks	Detailed Steps
	<xs:element name="QuantityPerUnit" type="xs:string" /> <xs:element name="UnitPrice" type="xs:double" /> <xs:element name="UnitsInStock" type="xs:integer" /> <xs:element name="UnitsOnOrder" type="xs:integer" /> <xs:element name="ReorderLevel" type="xs:integer" /> <xs:element name="Discontinued" type="xs:boolean" /> </xs:schema> ' SELECT * FROM sys.xml_schema_collection This code stores a Schema definition in SQL Server 2005. The **SELECT** statement will show you all of the schemas that are currently stored in the database. When you create XML columns, you can indicate that the data they contain must conform to a particular schema. **5.** Select the SQL statement in section "Exercise 2, Task 1, Step 5…", and press **F5** to execute it: DROP TABLE dbo.ProductDocs GO CREATE TABLE ProductDocs (ID INT IDENTITY PRIMARY KEY, ProductDoc XML(ProductSchema) NOT NULL) GO When XML data is inserted into the table, the schema collection you created is used to validate the data. If the data does not conform to the schema, it will be rejected.
2. Use the schema to validate XML.	**1.** Select the SQL statement in section "Exercise 2, Task 2, Step 1 …", and press **F5** to execute it. INSERT INTO ProductDocs VALUES(' <Product xmlns= "http://www.microsoft.com/schemas/adventure-works/products"> <ProductID>1</ProductID> <SupplierID>1</SupplierID> <CategoryID>1</CategoryID> <QuantityPerUnit>10 boxes x 20 bags</QuantityPerUnit> <UnitPrice>18.0000</UnitPrice> <UnitsInStock>39</UnitsInStock> <UnitsOnOrder>0</UnitsOnOrder> <ReorderLevel>10</ReorderLevel> <Discontinued>0</Discontinued> </Product> ')

Tasks	Detailed Steps
	This XML should return the following error because it does not conform to the schema: `Msg 6965, Level 16, State 1, Line 1` `XML Validation: Invalid content. Expected element(s):http://www.microsoft.com/schemas/adventure-works/products:ProductName where element 'http://www.microsoft.com/schemas/adventure-works/products:SupplierID' was specified. Location: /*:Product[1]/*:SupplierID[1]` The error is returned because the schema specifies that the XML must contain a "ProductName" element immediately following ProductID, and this XML document is missing that element. **2.** Select the code in section "Exercise 2, Task 2, Step 2 …", and press **F5** to execute it: `INSERT INTO ProductDocs VALUES('` `  <Product xmlns="http://www.microsoft.com/schemas/adventure-works/products">` `  <ProductID>1</ProductID>` `  <ProductName>Chai</ProductName>` `  <SupplierID>1</SupplierID>` `  <CategoryID>1</CategoryID>` `  <QuantityPerUnit>10 boxes x 20 bags</QuantityPerUnit>` `  <UnitPrice>18.0000</UnitPrice>` `  <UnitsInStock>39</UnitsInStock>` `  <UnitsOnOrder>0</UnitsOnOrder>` `  <ReorderLevel>10</ReorderLevel>` `  <Discontinued>0</Discontinued>` `  </Product>` `')` This XML conforms to the schema, so it is inserted without error.

Exercise 3
Server Side Querying

Scenario

In this exercise, you will learn the various mechanisms that can be used to query the XML stored in SQL Server 2005. Storing XML is effective if you can perform queries on the contents of the XML itself. For example, if you are storing documents as XML, you may want to return only the headings, in order to build an outline.

Tasks	Detailed Steps
1. Selecting XML.	**1.** In the **Solution Explorer**, double-click **Querying.sql.** **2.** If you see a **Connect to Server** dialog box, confirm that the **Server type** is **Database Engine** and that **Windows Authentication** is selected for **Authentication**. For the **Server name** setting, type **localhost**, and click **Connect.** **3.** Highlight the following section of code, and press **F5**. `USE AdventureWorks` `GO` **4.** Type the following SQL statement in section "Exercise 3, Task 1, Step 4…", then select the statement and press **F5** to execute it. `SELECT * FROM Sales.Individual WHERE CustomerID = 11000` The Demographics column should contain XML demographics data, like the following: `<IndividualSurvey xmlns=` `"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey">` `<TotalPurchaseYTD>8248.99</TotalPurchaseYTD>` `<DateFirstPurchase>2001-07-22Z</DateFirstPurchase>` `<BirthDate>1966-04-08Z</BirthDate>` `<MaritalStatus>M</MaritalStatus>` `<YearlyIncome>75001-100000</YearlyIncome>` `<Gender>M</Gender>` `<TotalChildren>2</TotalChildren>` `<NumberChildrenAtHome>0.0000000000</NumberChildrenAtHome>` `<Education>Bachelors </Education>` `<Occupation>Professional</Occupation>` `<HomeOwnerFlag>1</HomeOwnerFlag>` `<NumberCarsOwned>0.0000000000</NumberCarsOwned>` `<CommuteDistance>1-2 Miles</CommuteDistance>` `</IndividualSurvey>` As you saw in Exercise 1, you can query a table, and retrieve the XML contained in a column. In addition, you can perform queries based on the actual contents of the XML.

Tasks	Detailed Steps
2. Querying with XPath.	**1.** Select the SQL statement in section "Exercise 3, Task 2, Step 1…", and press **F5** to execute it: `SELECT TOP 10 Demographics.query('` `  declare default element namespace` `"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey";` `  /IndividualSurvey/YearlyIncome')` `FROM Sales.Individual` This query returns the **YearlyIncome** elements from the **IndividualSurvey** documents stored in the **Individual** table in the **Sales** relational schema. As the Demographics column has a schema collection associated to it you need to declare the namespace to use. In the code above you declare the namespace as the **default element namespace**. This will bind all the elements in the subsequent query to that particular namespace. **2.** Select the SQL statement in section "Exercise 3, Task 2, Step 2 …", and press **F5** to execute it: `SELECT TOP 10 Demographics.value('` `  declare default element namespace` `"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey";` `  (/IndividualSurvey/YearlyIncome)[1]',` `  'varchar(250)')` `FROM Sales.Individual` The "[1]" is added at the end of the path expression in the value() method to explicitly indicate that the path expression returns a singleton. The previous query returned the matching XML elements. This query returns only the text of those elements: `75001-100000` `50001-75000` `50001-75000` `50001-75000` `75001-100000` `50001-75000` `50001-75000` `50001-75000` `50001-75000` `50001-75000` **3.** Copy the SQL statement from section "Exercise 3, Task 2, Step 1…" into section "Exercise 3, Task 2, Step 3…", and modify it to return each TotalChildren element. **4.** Copy the SQL statement from section "Exercise 3, Task 2, Step 2 …" into section "Exercise 3, Task 2, Step 4…", and modify it to return the value of each TotalChildren element. You can also use XPath expressions to limit the results based on the contents of the XML. **5.** Select the SQL statement in section "Exercise 3, Task 2, Step 5…", and press **F5** to execute it: `SELECT Demographics.value('` `  declare default element namespace` `"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey";`

Tasks	Detailed Steps
3. Querying with XQuery.	The Resume column of the JobCandidate table contains a resume stored as an XML document. The following is an example of such a resume:

```
<ns:Resume
xmlns:ns="http://schemas.microsoft.com/sqlserver/2004/07/adventu
re-works/Resume">
  <ns:Name>
    <ns:Name.Prefix />
    <ns:Name.First>Shai</ns:Name.First>
    <ns:Name.Middle />
    <ns:Name.Last>Bassli</ns:Name.Last>
    <ns:Name.Suffix />
  </ns:Name>
  <ns:Skills>
I am an experienced and ...
    </ns:Skills>
  <ns:Employment>
    <ns:Emp.StartDate>2000-06-01Z</ns:Emp.StartDate>
    <ns:Emp.EndDate>2002-09-30Z</ns:Emp.EndDate>
    <ns:Emp.OrgName>Wingtip Toys</ns:Emp.OrgName>
    <ns:Emp.JobTitle>Lead Machinist</ns:Emp.JobTitle>
    <ns:Emp.Responsibility> Supervised work of ...
    </ns:Emp.Responsibility>

<ns:Emp.FunctionCategory>Production</ns:Emp.FunctionCategory>

<ns:Emp.IndustryCategory>Manufacturing</ns:Emp.IndustryCategory>
    <ns:Emp.Location>
      <ns:Location>
        <ns:Loc.Country>US </ns:Loc.Country>
        <ns:Loc.State>MI </ns:Loc.State>
        <ns:Loc.City>Saginaw</ns:Loc.City>
      </ns:Location>
    </ns:Emp.Location>
  </ns:Employment>
    .
    .
    .
  <ns:Education>
    <ns:Edu.Level>Bachelor</ns:Edu.Level>
    <ns:Edu.StartDate>1990-09-15Z</ns:Edu.StartDate>
    <ns:Edu.EndDate>1994-05-10Z</ns:Edu.EndDate>
    <ns:Edu.Degree>Bachelor of Science</ns:Edu.Degree>
    <ns:Edu.Major>Mechanical Engineering</ns:Edu.Major>
    <ns:Edu.Minor />
    <ns:Edu.GPA>3.2</ns:Edu.GPA>
    <ns:Edu.GPAScale>4</ns:Edu.GPAScale>
    <ns:Edu.School>Midwest State University</ns:Edu.School>
    <ns:Edu.Location>
      <ns:Location>
        <ns:Loc.Country>US </ns:Loc.Country>
        <ns:Loc.State>IA </ns:Loc.State>
        <ns:Loc.City>Ames</ns:Loc.City>
```

Tasks	Detailed Steps

```
        </ns:Location>
      </ns:Edu.Location>
    </ns:Education>
    <ns:Address>
      <ns:Addr.Type>Home</ns:Addr.Type>
      <ns:Addr.Street>567 3rd Ave</ns:Addr.Street>
      <ns:Addr.Location>
        <ns:Location>
          <ns:Loc.Country>US </ns:Loc.Country>
          <ns:Loc.State>MI </ns:Loc.State>
          <ns:Loc.City>Saginaw</ns:Loc.City>
        </ns:Location>
      </ns:Addr.Location>
      <ns:Addr.PostalCode>53900</ns:Addr.PostalCode>
      <ns:Addr.Telephone>
        <ns:Telephone>
          <ns:Tel.Type>Voice</ns:Tel.Type>
          <ns:Tel.IntlCode>1</ns:Tel.IntlCode>
          <ns:Tel.AreaCode>276</ns:Tel.AreaCode>
          <ns:Tel.Number>555-0114</ns:Tel.Number>
        </ns:Telephone>
      </ns:Addr.Telephone>
    </ns:Address>
    <ns:EMail>Shai@Example.com</ns:EMail>
    <ns:WebSite />
</ns:Resume>
```

The resume contains seven sections: Name, Skills, Employment, Education, Address, EMail, and WebSite.

XQuery allow you to create complex SQL-like queries for an XML document. Using XQuery, you can retrieve information and perform transformations. Also, using the DML extensions to XQuery in SQL Server 2005, you can modify XML documents in-place.

1. Using this query, retrieve just the employer names from the resume documents. Select the SQL statement in section “Exercise 3, Task 3, Step 1…”, and press **F5** to execute:

```
SELECT JobCandidateID, Resume.query('
   declare default element namespace
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/Resume";
   for $employer in /Resume/Employment/Emp.OrgName
   return $employer
   ')
FROM HumanResources.JobCandidate
```

2. XQuery allows you to perform queries that are much more complex than XPath queries, using a syntax that is often more readable. For example, in this step, you will return job candidates’ employers in the manufacturing industry .Select the SQL statement in section “Exercise 3, Task 3, Step 2”, and press **F5** to execute it:

```
SELECT JobCandidateID, Resume.query('
```

Tasks	Detailed Steps
	`` declare default element namespace "http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"; for $employer in /Resume/Employment where data($employer/Emp.IndustryCategory) = "Manufacturing" return $employer/Emp.OrgName ') FROM HumanResources.JobCandidate ``` **3.** Likewise, in this step, you will return job candidates' employers in the city of Renton. .Select the SQL statement in section "Exercise 3, Task 3, Step 3", and press **F5** to execute it: ``` SELECT JobCandidateID, Resume.query(' declare default element namespace "http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume"; for $employer in /Resume/Employment where data($employer/Emp.Location/Location/Loc.City) = "Renton" return $employer/Emp.OrgName ') FROM HumanResources.JobCandidate ``` **4.** You can also use XQuery to perform transformations on an XML document. In this step, output the employer name in a custom element that you create as part of your XQuery. Select the SQL statement in section "Exercise 3, Task 3, Step 4…", and press **F5** to execute it. This should produce results similar to the following output in the second column of the result set: ``` <EmployerName xmlns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume">Wingtip Toys</EmployerName> <EmployerName xmlns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume">Blue Yonder Airlines</EmployerName> <EmployerName xmlns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/Resume">City Power and Light</EmployerName> ``` As you can see, the output contains the EmployerName element, which was not part of the input.
4. Creating XML Indexes.	You can increase the performance of XML queries by creating an index on the XML column. In this exercise, you will work with a subset of the XML. This will save time, and give you a set of data that you can modify. **1.** Select the SQL statement in section "Exercise 3, Task 4…", and press **F5** to execute it. ``` SELECT * INTO Individual1 FROM Sales.Individual ```

Tasks	Detailed Steps
	`` WHERE CustomerID < 11100 ALTER TABLE Individual1 ADD PRIMARY KEY(CustomerID) CREATE PRIMARY XML INDEX idx_Demographics ON Individual1(Demographics) ``` When you create an index, all the XML in the column is parsed, and a node table is created. The node table provides a way to improve the performance of XML searches.
5. Promoting nodes to columns.	While you can query directly against the contents of an XML document, there is a significant performance penalty for doing so. By promoting the TotalChildren element to its own column, you can dramatically reduce the number of reads. **1.** Select the SQL Statement in section "Exercise 3, Task 5, Step 1", and press **F5** to execute it: ``` SELECT CustomerID, Demographics.value(' declare default element namespace "http://schemas.microsoft.com/sqlserver/2004/07/adventure- works/IndividualSurvey"; (/IndividualSurvey/TotalChildren)[1]', 'int') As TotalChildren, Demographics INTO Individual2 FROM Individual1 GO CREATE INDEX idx_TotalChildren ON Individual2 (TotalChildren) ``` You can now query based on the new promoted TotalChildren column. **2.** Select the SQL statements in section "Exercise 3, Task 5, Step 2", and press **F5** to execute: ``` SELECT Demographics.value(' declare default element namespace "http://schemas.microsoft.com/sqlserver/2004/07/adventure- works/IndividualSurvey"; (/IndividualSurvey/YearlyIncome)[1]', 'varchar(20)') as YearlyIncome FROM Individual2 WHERE TotalChildren > 1 ``` There are times when you do not know enough about the structure of an XML document to promote any nodes to columns. However, if you do know the structure of the document, and there are certain nodes that will be frequently queried, you should strongly consider using this optimization.
6. Modifying XML documents in place.	You can use the Data Modification Language (DML) extensions to XQuery to modify an XML document in place. **1.** You can modify the contents of an element in an XML column. Select the SQL

Tasks	Detailed Steps
	statement in section "Exercise 3, Task 6, Step 1…", and press **F5** to execute it: `UPDATE Individual1` `SET Demographics.modify(` `' declare default element namespace` `"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey";` `replace value of (/IndividualSurvey/TotalPurchaseYTD)[1]` `with 10')` `WHERE CustomerID = 11000` This statement changes the value of an element in place. This is much more efficient than retrieving the entire XML document, making a small change, and then updating the entire XML field. **2.** You can also delete nodes from an existing XML document. Select the SQL Statement in section "Exercise 3, Task 6, Step 2…", and press **F5** to execute it. `UPDATE Individual1` `SET Demographics.modify(` `' declare default element namespace` `"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey";` `delete /IndividualSurvey/Hobby[2]')` `WHERE CustomerID = 11000` This query deletes the second Hobby element for the specified individual. Again, it is significantly more efficient to modify an XML document in place, rather than extracting the entire document to make modifications.

Exercise 4
XML DataType

Scenario

In this exercise, you will learn to use the XML datatype.

Tasks	Detailed Steps
1. Selecting XML.	**1.** In the **Solution Explorer**, double-click **XML Datatype.sql.** **2.** If you see a **Connect to Server** dialog box, confirm that the **Server type** is **Database Engine** and that **Windows Authentication** is selected for **Authentication**. For the **Server name** setting, type **localhost**, and click **Connect.** **3.** Highlight the following section of code, and press **F5**. `USE AdventureWorks` `GO` In this exercise, you will be working with data from the Person.Address and HumanResources.Employee tables, so take a minute to become familiar with the columns. **4.** Type the following SQL statement in section "Exercise 4, Task 1, Step 4…", and then select the statement and press **F5** to execute it: `SELECT * FROM Person.Address` **5.** Select the SQL statement in section "Exercise 4, Task 1, Step 5…", and press **F5** to execute it. `SELECT * FROM HumanResources.Employee` **6.** You can create variables of type XML, and store data in them. The data can come either from an XML column, or from the results of a FOR XML statement.Select the SQL statement in section "Exercise 4, Task 1, Step 6…", and press **F5** to execute it: `DECLARE @x XML` `SET @x = (SELECT * FROM Address WHERE AddressID = 1 FOR XML AUTO,` `ELEMENTS, TYPE)` `SELECT @x` This should return the following results: `<Person.Address>` `  <AddressID>1</AddressID>`

Tasks	Detailed Steps

```
  <AddressLine1>1970 Napa Ct.</AddressLine1>
  <City>Bothell</City>
  <StateProvinceID>79</StateProvinceID>
  <PostalCode>98011</PostalCode>
  <rowguid>9AADCB0D-36CF-483F-84D8-585C2D4EC6E9</rowguid>
  <ModifiedDate>1998-01-04T00:00:00</ModifiedDate>
</Person.Address>
```

The results of the FOR XML query were stored in an XML variable, and the contents of the variable were then output in XML format. The XML data type provides all of the functionality available for XML columns in tables. As a result, you can use any methods on this data type that you can on XML columns.

7. In this query, data is loaded into an XML variable, and the query method is used to output a specific XML document. Select the SQL statement in section "Exercise 4, Task 1, Step 7…", and press **F5** to execute it:

```
DECLARE @x XML
SET @x = (SELECT TOP 50 City FROM Person.Address FOR XML
AUTO, ELEMENTS, TYPE)
SELECT @x.query('
   <Cities>
   {
       for $city in /Person.Address/City
       return $city
   }
   </Cities>
')

This should return the following results:
<Cities>
  <City>Ottawa</City>
  <City>Burnaby</City>
  <City>Dunkerque</City>
 ...
 (lines deleted for brevity...)
 ...
  <City>Paris</City>
  <City>Versailles</City>
  <City>Croix</City>
</Cities>
```

Using a combination of XQuery and the XML datatype opens the door to powerful creation and manipulation of XML data in T-SQL code.

8. In addition, the FOR XML clause supports nested SQL statements, allowing you to join data from tables to create hierarchies. Select the SQL statement in section "Exercise 4, Task 1, Step 8…", and press **F5** to execute it:

```
DECLARE @x XML
SET @x = (
   SELECT Employee.*, EmployeeAddress.AddressID, (
       SELECT *
       FROM Person.Address Address
```

Tasks	Detailed Steps

```
      WHERE EmployeeAddress.AddressID = Address.AddressID
      FOR XML AUTO, ELEMENTS, TYPE
   ) AS Addresses
    FROM HumanResources.Employee AS Employee
    INNER JOIN HumanResources.EmployeeAddress AS
EmployeeAddress
    ON Employee.EmployeeID = EmployeeAddress.EmployeeID
    WHERE Employee.EmployeeID = 1
   FOR XML AUTO, ELEMENTS, TYPE
)
SELECT @x
```

This should return the following results:

```
<Employee>
  <EmployeeID>1</EmployeeID>
  <NationalIDNumber>14417807</NationalIDNumber>
  <ContactID>1209</ContactID>
  <LoginID>adventure-works\guy1</LoginID>
  <ManagerID>16</ManagerID>
  <Title>Production Technician - WC60</Title>
  <BirthDate>1972-05-15T00:00:00</BirthDate>
  <MaritalStatus>M</MaritalStatus>
  <Gender>M</Gender>
  <HireDate>1996-07-31T00:00:00</HireDate>
  <SalariedFlag>0</SalariedFlag>
  <VacationHours>21</VacationHours>
  <SickLeaveHours>30</SickLeaveHours>
  <CurrentFlag>1</CurrentFlag>
  <rowguid>AAE1D04A-C237-4974-B4D5-935247737718</rowguid>
  <ModifiedDate>2004-07-31T00:00:00</ModifiedDate>
  <EmployeeAddress>
    <AddressID>61</AddressID>
    <Addresses>
      <Address>
        <AddressID>61</AddressID>
        <AddressLine1>7726 Driftwood Drive</AddressLine1>
        <City>Monroe</City>
        <StateProvinceID>79</StateProvinceID>
        <PostalCode>98272</PostalCode>
        <rowguid>07373E01-BD99-405A-996A-
AB68984423C3</rowguid>
        <ModifiedDate>1996-07-24T00:00:00</ModifiedDate>
      </Address>
    </Addresses>
  </EmployeeAddress>
</Employee>
```