http://www.sqlservercentral.com/articles/XML/61333/
Printed 2007/11/07 02:21PM

XML Workshop X - Working with namespaces

By Jacob Sebastian, 2007/10/18

Introduction

XML uses the concept of *Namespaces* to avoid ambiguity of elements. It is possible that an *XML* document is created with two elements having the same name, but referring to or originated from two different contexts. A common example that most tutorials use is of an *XML* document which has two elements named "*table*", one referring to an *HTML* table (having rows and columns) and the other referring to a physical wooden table (say a study table). In this case, each element needs to be distinctly marked to avoid ambiguity.

The primary consumer of an *XML* document is a piece of software or program. Most of the times, it will be a software or application that will read the data from an *XML* document. Often, software is not intelligent enough to resolve ambiguity by using contextual references, as humans do. A human can easily differentiate between table 1 (two rows, 3 columns) and table 2 (4 legs, made of steel). But often, software cant do that. An *XML* parser cannot resolve ambiguity the way we humans often do. Hence *XML* specification introduced the concept of namespaces, using which, elements belonging to a specific context are identified distinctly.

Resolving Ambiguity

The first time you used this must be in your first program or your first *SQL Query*. However, we may not have paid attention to that. When we write a *TSQL* Query, we use aliases to make sure that the columns are taken from the correct table. If you have a query which involves multiple tables, and one of the columns that you are using exists in more than one table, you *MUST* qualify the column with table alias.

Let us look at an example:

```
 1 WITH
 2 -- "Employee" Table
 3 Employees AS (
 4     SELECT 'J001' AS Code, 'Jacob' AS EmployeeName, 1 AS Dept
 5     UNION SELECT 'S001', 'Sebastian', 2
 6 ),
 7 -- "Departments" table
 8 Departments AS (
 9     SELECT 1 AS Code, 'IT' AS DepartmentName
10     UNION SELECT 2, 'Finance'
11 )
12
13 /*
14 Select values from the two CTEs that we just created.
15 */
16
17 SELECT      *
18 FROM Employees e
19 INNER JOIN Departments d ON d.Code = e.dept
20 WHERE code IS NOT NULL
21
```

```
/*
OUTPUT:

Msg 209, Level 16, State 1, Line 20
Ambiguous column name 'code'.
*/
```

This is a very common error that all of us must have seen a million times. And everyone of you know how to fix it. Here is the correct code.

```
WITH
-- "Employee" Table
Employees AS (
    SELECT 'J001' AS Code, 'Jacob' AS EmployeeName, 1 AS Dept
    UNION SELECT 'S001', 'Sebastian', 2
),
-- "Departments" table
Departments AS (
    SELECT 1 AS Code, 'IT' AS DepartmentName
    UNION SELECT 2, 'Finance'
)

/*
Select values from the two CTEs that we just created.
*/

SELECT     *
FROM Employees e
INNER JOIN Departments d ON d.Code = e.dept
WHERE e.code IS NOT NULL

/*
OUTPUT:

Code EmployeeName Dept        Code        DepartmentName
---- ------------ ----------- ----------- --------------
J001 Jacob        1           1           IT
S001 Sebastian    2           2           Finance

(2 row(s) affected)
*/
```

The same happens with other programming languages too. It is quite natural that you will end up writing an application that uses more than one class having the same name. In such cases, we use namespaces to differentiate between the two classes. Here is a *VB.NET* example.

```
Namespace internet
    Public Class Connection
        Public Provider As String
        Public Speed As String
    End Class
End Namespace
```

```
Namespace Database
    Public Class Connection
        Public Provider As String
        Public Protocol As String
        Public Authentication As String
    End Class
End Namespace
```

```
Module Module1

    Sub Main()
        Dim c1 As New Database.Connection
        Dim c2 As New internet.Connection

        c1.Authentication = "Windows"
        c1.Protocol = "TCP/IP"
        c1.Provider = "SQL Server Client Provider"

        c2.Provider = "World Wide Internet Providers"
        c2.Speed = "512 KBPS"
    End Sub

End Module
```

The above sample shows two classes named *Connection*. The ambiguity is avoided by using a namespace name. I am sure, none of you need a detailed explanation about it. This is something that all of us do every day and is part of our day-to-day programming life.

XML Namespaces

Let us create an *XML* representation of the above example. The *XML* may look like this.

```
<configuration>
  <connection>
    <provider>World Wide Internet Providers</provider>
    <speed>512 KBPS</speed>
  </connection>
  <connection>
    <provider>SQL Client Provider</provider>
    <protocol>TCP/IP</protocol>
    <Authentication>Windows</Authentication>
  </connection>
</configuration>
```

An application which reads information from the above *XML* will not be able to identify the correct element that it is supposed to read. It will either throw a validation error or will try to read information from the wrong element. By using namespaces, we can resolve the ambiguity in this example. Here is the new version of the *XML*.

```
<configuration
    xmlns:net="urn:www.dotnetquest.com/internetconnection"
    xmlns:db="urn:www.dotnetquest.com/databaseconnection">
  <net:connection>
    <net:provider>World Wide Internet Providers</net:provider>
    <net:speed>512 KBPS</net:speed>
  </net:connection>
  <db:connection>
    <db:provider>SQL Client Provider</db:provider>
    <db:protocol>TCP/IP</db:protocol>
    <db:Authentication>Windows</db:Authentication>
  </db:connection>
</configuration>
```

Generating XML

Let us see how to generate an *XML* structure like the one given above. First of all, let us create the tables that we need and populate them. [Code]

```
-- Create the table for Net Connection
CREATE TABLE NetConnection (
    Provider VARCHAR(50),
    Speed VARCHAR(20)
)

-- Populate the Table
INSERT INTO NetConnection(
    Provider,
    Speed )
SELECT
    'World Wide Internet Providers',
    '512 KBPS'

-- Create table for DB Connection
CREATE TABLE DbConnection (
    Provider VARCHAR(50),
    Protocol VARCHAR(20),
    [Authentication] VARCHAR(20)
)

-- Populate the Table
INSERT INTO DbConnection (
    Provider,
    Protocol,
    [Authentication] )
SELECT
    'SQL Client Provider',
    'TCP/IP',
    'Windows'
```

In the initial few sessions of the *XML Workshop*, we had seen several examples which explained how to generate XML data from the result of a query. If you do not know how to generate XML data from a TSQL query, you should refer to those examples now. You can find the examples here:

- XML Workshop I - Generating XML with FOR XML
- XML Workshop III - FOR XML PATH
- XML Workshop IV - FOR XML EXPLICIT

The examples we discussed in the above 3 sessions did not touch namespaces. The following example shows how to generate namespace information along with the *XML* data. [code]

```
WITH XMLNAMESPACES
(
    'urn:www.dotnetquest.com/internetconnection' AS net,
    'urn:www.dotnetquest.com/databaseconnection' AS db
)
SELECT
    net.Provider AS 'net:Connection/net:Provider',
    net.Speed AS 'net:Connection/net:Speed',
    db.Provider AS 'db:Connection/db:Provider',
    db.Protocol AS 'db:Connection/db:Protocol',
    db.[Authentication] AS 'db:Connection/db:Authentication'
    FROM NetConnection net
CROSS JOIN DbConnection db
FOR XML PATH('Configuration')
```

The above code generates the following *XML*. [xml]

```
<Configuration
  xmlns:db="urn:www.dotnetquest.com/databaseconnection"
```

```
3    xmlns:net="urn:www.dotnetquest.com/internetconnection">
4    <net:Connection>
5      <net:Provider>World Wide Internet Providers</net:Provider>
6      <net:Speed>512 KBPS</net:Speed>
7    </net:Connection>
8    <db:Connection>
9      <db:Provider>SQL Client Provider</db:Provider>
10     <db:Protocol>TCP/IP</db:Protocol>
11     <db:Authentication>Windows</db:Authentication>
12   </db:Connection>
13 </Configuration>
```

Conclusions

In this session we have seen how to generate *XML* data which contains namespace information. Namespace information is added to the generated *XML* by using the *WITH NAMESPACES* TSQL keyword.
