**http://www.sqlservercentral.com/articles/T-SQL/68927/**
Printed 2010/01/04 06:20AM

# Using T-SQL to Transform XML Data to a Relational Format

**By Nasir Mirza, 2009/12/01**

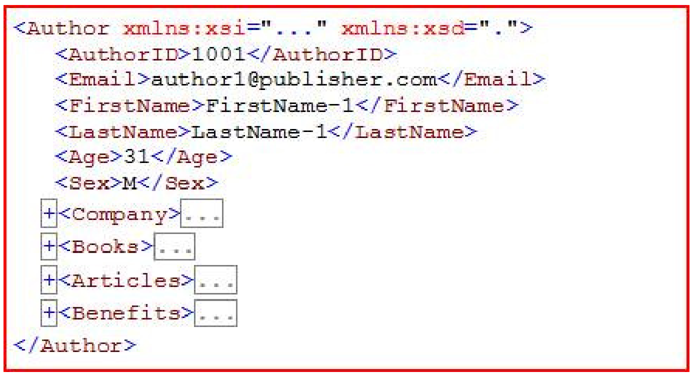
In certain situations there may be a need to parse back XML data stored in a database into a normalized table structure. There are multiple ways to achieve this, e.g. using the XML features of the Dataset or with the SSIS XML data source. This article describes use of XPath queries with CROSSAPPLY for parsing XML data into normalized relational data. In this sample project we will use the author list scenario described in the next section.

## AuthorList sample scenario

The XML to Relational sample scenario is based on the **AuthorList** table that has an **AuthorProfile** column of the XML data type. The AuthorProfile stores the profile of an author, which includes author's company details, books authored and any co-authors involved. It also contains information about articles written by an author, publishers of these articles and lastly the benefits an author is entitled to. The AuthorList table has this structure:

```
Create Table dbo.AuthorList
(
      AuthorID intPrimary Key,
      DOJ Datetime,
      UpdatedDate Datetime,
      AuthorProfile XML
);
GO
```

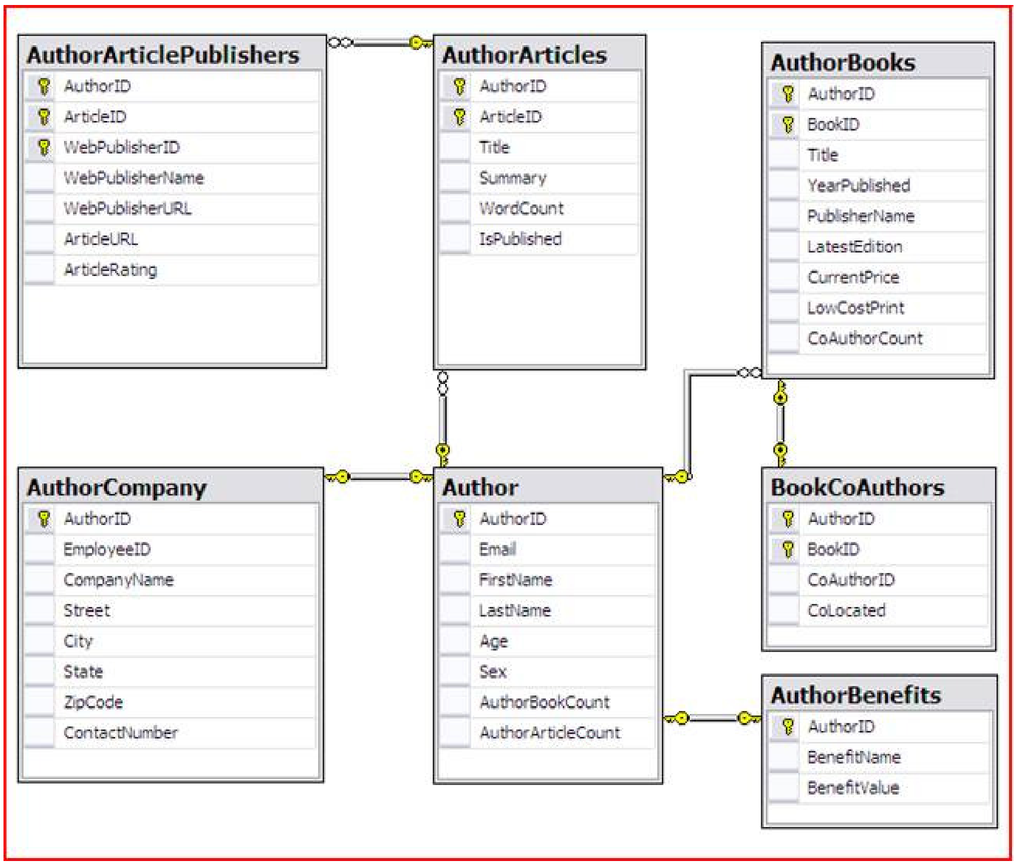
The AuthorProfile XML data type high level structure is shown below:



```
<Author xmlns:xsi="..." xmlns:xsd=".">
   <AuthorID>1001</AuthorID>
   <Email>author1@publisher.com</Email>
   <FirstName>FirstName-1</FirstName>
   <LastName>LastName-1</LastName>
   <Age>31</Age>
   <Sex>M</Sex>
  +<Company>...
  +<Books>...
  +<Articles>...
  +<Benefits>...
</Author>
```

The sample code contains one example XML file (**XMLToRelational_Author.xml**) of AuthorProfile. In the following sections of this article, we will go over each of these XML segments in the AuthorProfile XML column and demonstrate the T-SQL queries for populating corresponding relational tables. The entire working sample of the code can be downloaded from the Resources section.

In this sample we will use the .value, .query (data ('')), .query (count ('')), and .nodes functions of XPath query as well as CROSSAPPLY. Additionally FLWOR may be of interest to look at. Our target relational structure to populate from AuthorProfile XML column is represented next.



## XML to Relational Transformation

So far we saw the XML source that needs to be consumed and the relational structure we want to populate using T-SQL. In the rest of this article we will go step by step, pick up a section of AuthorProfile XML and show how to perform the XML to relational transformation.

### Author

The Author segment of the AuthorProfile XML column contains AuthorID, Email, FirstName, LastName, Age and Sex elements. Its structure is show below:

```
<Authorxmlns:xsi="..." xmlns:xsd="..." version="1">
   <AuthorID>1001</AuthorID>
   <Email>author1@publisher.com</Email>
   <FirstName>FirstName-1</FirstName>
   <LastName>LastName-1</LastName>
   <Age>31</Age>
   <Sex>M</Sex>
</Author>
```

We will populate the Author table with the above XML segment. The structure of the table to populate is shown below. There is one to one mapping between the XML elements and the table columns except for the AuthorBookCount and AuthorArticleCount. These two columns are filled with calculated values to show number of books and articles written by a particular author.

```
CreateTable dbo.Author
(
  AuthorID int,
  Email varchar(255),
  FirstName varchar(100),
  LastName varchar(100),
  Age int,
  Sex char(1),
  AuthorBookCount int,
  AuthorArticleCount int
);
```

The T-SQL query for performing this transformation would be:

```
--Author parsing query
SELECT
   AuthorProfile.value('/Author[1]/AuthorID[1]', 'int')As [AuthorID]
  ,AuthorProfile.value('/Author[1]/Email[1]', 'varchar(255)')As [Email]
  ,AuthorProfile.value('/Author[1]/FirstName[1]', 'varchar(100)')As [FirstName]
  ,AuthorProfile.value('/Author[1]/LastName[1]', 'varchar(100)')As [LastName]
  ,AuthorProfile.value('/Author[1]/Age[1]', 'int')As [Age]
  ,AuthorProfile.value('/Author[1]/Sex[1]', 'char(1)')As [Sex]
  ,Cast(Cast(AuthorProfile.query('count(/Author/Books/Book)')Asvarchar(5))As smallint)As [AuthorBookCount]
  ,Cast(Cast(AuthorProfile.query('count(/Author/Articles/Article)')Asvarchar(5))As smallint)As [AuthorArticleCount]
FROM[dbo].[AuthorList]
```

A few notes on the T-SQL for populating the Author table:

- **AuthorBookCount** and **AuthorArticleCount** are both type casted in two steps, first from XML to varchar and then to smallint. This is because the XML type cannot be converted directly to numeric types.
- Static typing rules require explicitly specifying the path expression that returns a singleton. Therefore, the additional **[1]** is specified with each XML element at the end of the path expression. For example, to refer to the Title of the second book under the first instance of the Books tag, we would use the path expression like: '/Author[1]/Books[1]/Book**[2]**/Title[1]'

**Author Company**

The Company node of the AuthorProfile XML contains the EmployeeID, CompanyName, Street, City, State, ZipCode and Contact Number. The exact XML segment is shown below:

```
 <Author xmlns:xsi="http://..." xmlns:xsd="http://..." version="1">
 <Company>
    <EmployeeID>2001</EmployeeID>
    <CompanyName>Company One</CompanyName>
    <Street>Street-1</Street>
    <City>City-1</City>
    <State>State-1</State>
    <ZipCode>1001</ZipCode>
    <ContactNumber>111-111-1111</ContactNumber>
  </Company>
</Author>
```

The elements of the Company node would be used to populate the AuthorCompany table. The structure of AuthorCompany table is shown here:

```
 Create Table dbo.AuthorCompany
 (
       AuthorID int,
       EmployeeID int,
       CompanyName varchar(100),
       Street varchar(100),
       City varchar(100),
       [State] varchar(25),
       ZipCode varchar(10),
       ContactNumber varchar(12)
 );
```

The T-SQL XPath query for populating the AuthorCompany table. This one is fairly straight forward.

```
--AuthorCompany parsing query
 SELECT
  AuthorProfile.value('/Author[1]/AuthorID[1]', 'int')As [AuthorID]
 ,AuthorProfile.value('/Author[1]/EmployeeID[1]','int') As [EmployeeID]
 ,AuthorProfile.value('/Author[1]/Company[1]/CompanyName[1]','varchar(100)') As [CompanyName]
 ,AuthorProfile.value('/Author[1]/Company[1]/Street[1]','varchar(100)') As [Street]
 ,AuthorProfile.value('/Author[1]/Company[1]/City[1]','varchar(100)') As [City]
 ,AuthorProfile.value('/Author[1]/Company[1]/State[1]','varchar(25)') As [State]
 ,AuthorProfile.value('/Author[1]/Company[1]/ZipCode[1]','varchar(10)') As [ZipCode]
 ,AuthorProfile.value('/Author[1]/Company[1]/ContactNumber[1]','varchar(12)') As [ContactNumber]
 FROM [dbo].[AuthorList]
```

**Author Books**

The Books XML node contains one or more books authored by an author. Each Book node has the BookID, Title, YearPublished, PublisherName, LatestEdition, CurrentPrice and LowCostPrint.

The Author Books segment of the AuthorProfile XML column

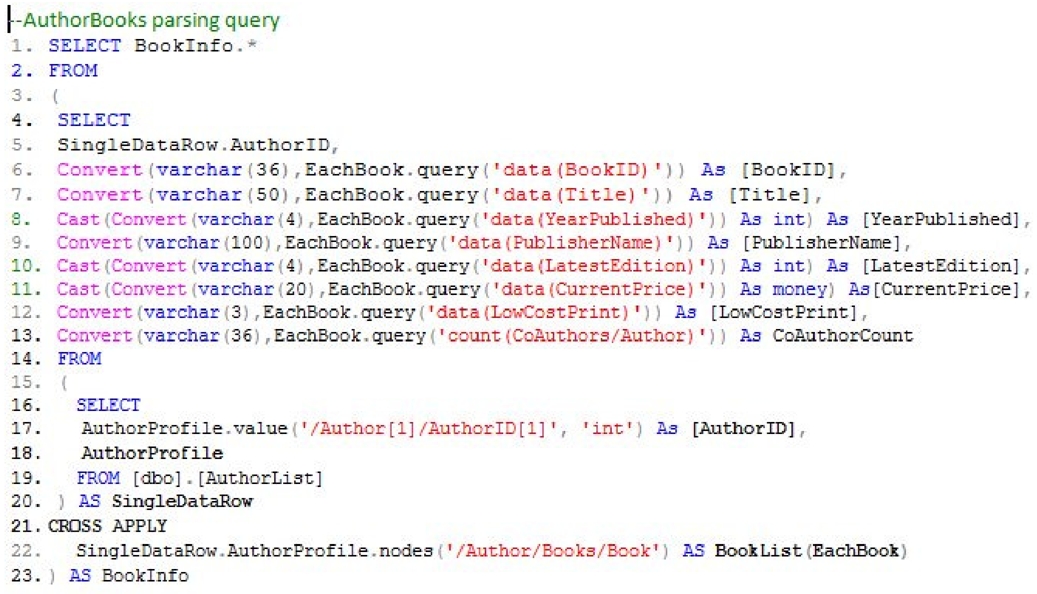
```
 <Author xmlns:xsi="http://..." xmlns:xsd="http://..." version="1">
  <Books>
    <Book>
      <BookID>249DA9BD-2DCD-4FF2-BB87-170B17A58070</BookID>
      <Title>Title-1</Title>
      <YearPublished>2001</YearPublished>
      <PublisherName>Publisher One</PublisherName>
      <LatestEdition>2008</LatestEdition>
      <CurrentPrice>59.60</CurrentPrice>
      <LowCostPrint>Yes</LowCostPrint>
    </Book>
```

```
      ...
   </Books>
  </Author>
```

The AuthorBooks table has the structure to accommodate each of the element values of the Book node in XML. In addition to that, there is a calculated column for holding the number of co-authors, if any, for a particular book.

```
Create Table dbo.AuthorBooks
 (
       AuthorID int,
       BookID varchar(36),
       Title varchar(50),
       YearPublished int,
       PublisherName varchar(100),
       LatestEdition int,
       CurrentPrice money,
       LowCostPrint varchar(3),
       CoAuthorCount int
 );
```

The AuthorBooks table is populated with T-SQL query that uses CROSSAPPLY.



```
--AuthorBooks parsing query
1.  SELECT BookInfo.*
2.  FROM
3.  (
4.   SELECT
5.   SingleDataRow.AuthorID,
6.   Convert(varchar(36),EachBook.query('data(BookID)')) As [BookID],
7.   Convert(varchar(50),EachBook.query('data(Title)')) As [Title],
8.   Cast(Convert(varchar(4),EachBook.query('data(YearPublished)')) As int) As [YearPublished],
9.   Convert(varchar(100),EachBook.query('data(PublisherName)')) As [PublisherName],
10.  Cast(Convert(varchar(4),EachBook.query('data(LatestEdition)')) As int) As [LatestEdition],
11.  Cast(Convert(varchar(20),EachBook.query('data(CurrentPrice)')) As money) As[CurrentPrice],
12.  Convert(varchar(3),EachBook.query('data(LowCostPrint)')) As [LowCostPrint],
13.  Convert(varchar(36),EachBook.query('count(CoAuthors/Author)')) As CoAuthorCount
14.  FROM
15.  (
16.    SELECT
17.    AuthorProfile.value('/Author[1]/AuthorID[1]', 'int') As [AuthorID],
18.    AuthorProfile
19.    FROM [dbo].[AuthorList]
20.  ) AS SingleDataRow
21. CROSS APPLY
22.    SingleDataRow.AuthorProfile.nodes('/Author/Books/Book') AS BookList(EachBook)
23. ) AS BookInfo
```

A few notes on the AuthorBooks T-SQL query above:

- The line numbers 16-19 show the innermost query of the overall statement. This sub-query has an alias name **SingleDataRow** and it returns the **AuthorID** and **AuthorProfile** XML columns. This result set is further cross applied (line numbers 21-22) with each Book node (/Author/Books/Book) for every data row in SingleDataRow.
- The result set after CROSS APPLY is given the alias name BookList (EachBook). In the outer SELECT statement at line number 4, the EachBook alias is used to access the various columns from the CROSS APPLY output.
- The SELECT statement at line number 4 parses each Book node, returned by CROSS APPLY, to extract the various book details. It also counts number of CoAuthors for each of the book if there are any.
- The result set prepared between line numbers 4-22 is wrapped as a sub-query with the alias name **BookInfo**. This may be required only in case of the need to join the whole result set with other tables before consuming it in scenarios like INSERT…SELECT, DELETE FROM or UPDATE FROM. If that is not intended then the lines 4-22 are sufficient.

**Book CoAuthors**

The AuthorProfile XML contains CoAuthors node for each book by an author. A book can have one or more co-authors. For each co-author, the AuthorID and CoLocated information is preserved. In order to populate BookCoAuthors table, we will need to extract the details from the XML root note for the AuthorID, from Book node for the BookID and from CoAuthors sub-nodes for the AuthorID and CoLocated fields. Doing this involves reading from two levels of nested nodes. In terms of the T-SQL we will cross apply twice to get out of the AuthorProfile XML what is needed for populating the BookCoAuthors table.

```
 <Author xmlns:xsi="http://..." xmlns:xsd="http://..." version="1">
  <Books>
```

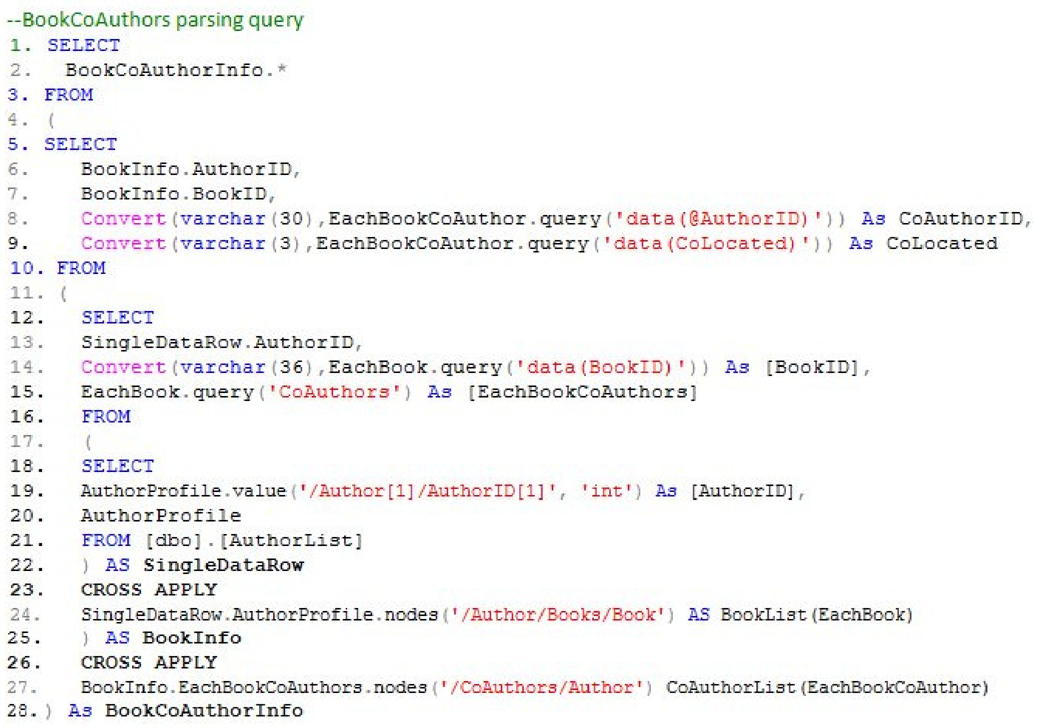
```
    <Book>
      <CoAuthors>
        <AuthorAuthorID="1002">
          <CoLocated>Yes</CoLocated>
        </Author>
         ...
      </CoAuthors>
    </Book>
  </Books>
 </Author>
```

The BookCoAuthors table has the AuthorID, BookID, CoAuthorID and CoLocated fields.

The BookCoAuthors Table

```
Create Table BookCoAuthors
(
      AuthorID int,
      BookID varchar(36),--comes from /Author/Books/Book node
      CoAuthorID varchar(36),--comes from /Author/Books/Book/CoAuthors/Author node
      CoLocated varchar(3)
);
```

The T-SQL query to cross apply twice to build result set for populating BookCoAuthors is below:



```
--BookCoAuthors parsing query
1. SELECT
2.   BookCoAuthorInfo.*
3. FROM
4. (
5. SELECT
6.    BookInfo.AuthorID,
7.    BookInfo.BookID,
8.    Convert(varchar(30),EachBookCoAuthor.query('data(@AuthorID)')) As CoAuthorID,
9.    Convert(varchar(3),EachBookCoAuthor.query('data(CoLocated)')) As CoLocated
10. FROM
11. (
12.   SELECT
13.   SingleDataRow.AuthorID,
14.   Convert(varchar(36),EachBook.query('data(BookID)')) As [BookID],
15.   EachBook.query('CoAuthors') As [EachBookCoAuthors]
16.   FROM
17.   (
18.   SELECT
19.   AuthorProfile.value('/Author[1]/AuthorID[1]', 'int') As [AuthorID],
20.   AuthorProfile
21.   FROM [dbo].[AuthorList]
22.   ) AS SingleDataRow
23.   CROSS APPLY
24.   SingleDataRow.AuthorProfile.nodes('/Author/Books/Book') AS BookList(EachBook)
25.   ) AS BookInfo
26.   CROSS APPLY
27.   BookInfo.EachBookCoAuthors.nodes('/CoAuthors/Author') CoAuthorList(EachBookCoAuthor)
28.) As BookCoAuthorInfo
```

A few notes on the BookCoAuthors T-SQL query:

- The lines 18-21 in the above statement are executed first. These return the AuthorID and the AuthorProfile XML columns. This result set is then cross applied with each Book node (/Author/Books/Book) and given the alias name SingleDataRow.
- The result of line numbers 18-24 contain the AuthorID, BookID and the CoAuthors XML segment (if present) under each book.
- The result set of lines 12-25 is further cross applied with each co-author node (/CoAuthors/Author), and the final result set contains the AuthorID, BookID, CoAuthorID and CoLocated columns.
- The result set produced between lines 5-27 is wrapped into sub-query with alias name of BookCoAuthorInfo. As described in the AuthorBooks before, this sub-query wrapping may be required only when there is a need to join the result set further with other tables.

**Author Articles**

The Articles XML node contains details about one or more articles written by an author. The AuthorArticles segment of AuthorProfile XML column

```
<Author xmlns:xsi="http://..." xmlns:xsd="http://..." version="1">
 <Articles>
   <Article>
     <ArticleID>4CE268B4-B74C-49F3-A241-3DCF6B093DCC</ArticleID>
     <Title>Title-One</Title>
     <Summary>Title-One Summary</Summary>
```

```
      <WordCount>2000</WordCount>
      <IsPublished>Yes</IsPublished>
    </Article>
     ...
  </Articles>
 </Author>
```

The AuthorArticles table has fields that match the Article node elements in the XML. For populating this table we will need to read the AuthorID from Author node and cross apply the results with each Article node in /Author/AuthorArticles.

```
Create Table dbo.AuthorArticles
 (
       AuthorID int,
       ArticleID varchar(36),--comes from /Author/AuthorArticles/Article
       Title varchar(150),
       Summary varchar(500),
       WordCount int,
       IsPublished varchar(3)
 );
```

The T-SQL query for populating AuthorArticles table has the sub-query that gets the AuthorID and AuthorProfile XML columns. This result set is given the alias name SingleDataRow and cross applied with each Article in /Author/Articles.

```
 --AuthorArticles
 SELECT
       ArticleInfo.*
 FROM
 (
   SELECT
    SingleDataRow.AuthorID,
    Convert(varchar(36),EachArticle.query('data(ArticleID)')) As [ArticleID],
    Convert(varchar(150),EachArticle.query('data(Title)'))As [Title],
    Convert(varchar(500),EachArticle.query('data(Summary)')) As [Summary],
    Cast(Convert(varchar(6),EachArticle.query('data(WordCount)'))As int)As [WordCount],
    Convert(varchar(3),EachArticle.query('data(IsPublished)')) As [IsPublished]
  FROM
     (
     SELECT
       AuthorProfile.value('/Author[1]/AuthorID[1]', 'int')As [AuthorID],
       AuthorProfile
     FROM [dbo].[AuthorList]
    ) AS SingleDataRow
    CROSS APPLY
    SingleDataRow.AuthorProfile.nodes('/Author/Articles/Article')AS ArticleList(EachArticle)
)AS ArticleInfo
```

**Article Publishers**

Each Article node in the AuthorProfile XML has the WebPublishers segment. There can be one or more WebPublisher nodes for each Article node. The elements inside each WebPublisher node would be read to populate the AuthorArticlePublishers table. The AuthorArticlePublishers segment of the AuthorProfile XML column

```
 <Author xmlns:xsi="http://..." xmlns:xsd="http://..." version="1">
  <Articles>
    <Article>
      <WebPublishers>
        <WebPublisher>
          <WebPublisherID>26145DCA-2A94-4A45-BB3E-4F6ED2C26152</WebPublisherID>
          <WebPublisherName>Web Publisher One</WebPublisherName>
          <WebPublisherURL>http://www.webpublisherone.com</WebPublisherURL>
          <ArticleURL>http://www.webpublisherone.com/4CE268B4-B74C-49F3-A241-3DCF6B093DCC</ArticleURL>
          <ArticleRating>5</ArticleRating>
        </WebPublisher>
         ...
      </WebPublishers>
    </Article>
  </Articles>
 </Author>
```

The AuthorArticlePublishers table will be populated with each WebPublisher node in /Author/AuthorArticles/Article/WebPublishers segment.

```
Create Table AuthorArticlePublishers
 (
  AuthorID int,
  ArticleID varchar(36),--comes from /Author/AuthorArticles/Article
 WebPublisherID varchar(36), --comes from Author/AuthorArticles/Article/WebPublishers/WebPublisher
  WebPublisherName varchar(255),
  WebPublisherURL varchar(255),
  ArticleURL varchar(255),
  ArticleRating int
 );
```

The T-SQL query for populating the AuthorArticlesPublishers table has a sub-query that gets the AuthorID and AuthorProfile XML columns. This result set is given the alias name SingleDataRow and cross applied with each Article in /Author/Articles. Subsequently, whole result set is further cross applied with each WebPublisher node in /WebPublishers segment.

```
 --AuthorArticlePublishers
 SELECT
       ArticlePublisherInfo.*
 FROM
 (
 SELECT
ArticleInfo.AuthorID,
ArticleInfo.ArticleID,
 Convert(varchar(36),EachArticlePublisher.query('data(WebPublisherID)'))As WebPublisherID,
 Convert(varchar(255),EachArticlePublisher.query('data(WebPublisherName)'))As WebPublisherName,
 Convert(varchar(255),EachArticlePublisher.query('data(WebPublisherURL)'))As WebPublisherURL,
 Convert(varchar(255),EachArticlePublisher.query('data(ArticleURL)'))As ArticleURL,
 Cast(Convert(varchar(6),EachArticlePublisher.query('data(ArticleRating)'))As int)As ArticleRating
 FROM
 (
  SELECT
       SingleDataRow.AuthorID,
      Convert(varchar(36),EachArticle.query('data(ArticleID)'))As [ArticleID],
       EachArticle.query('WebPublishers')As [EachArticlePublishers]
   FROM
   (
   SELECT
       AuthorProfile.value('/Author[1]/AuthorID[1]', 'int')As [AuthorID],
       AuthorProfile
  FROM [dbo].[AuthorList]
  ) AS SingleDataRow
  CROSS APPLY
  SingleDataRow.AuthorProfile.nodes('/Author/Articles/Article') AS ArticleList(EachArticle)
 ) AS ArticleInfo
 CROSS APPLY
  ArticleInfo.EachArticlePublishers.nodes('/WebPublishers/WebPublisher') WebPublisherList(EachArticlePublisher)
 ) As ArticlePublisherInfo
```

**Author Benefits**

The last segment in the AuthorProfile XML is the Benefits node. An author could be entitled to one or more benefits. Each Benefit element has the Name and the Value property.

```
<Authorxmlns:xsi="..." xmlns:xsd="..." version="1">
  <Benefits>
    <BenefitName="HospitalizationLimit" Value="500000"/>
    <BenefitName="MedicalConsultationLimit" Value="12000"/>
    <BenefitName="LifeInsuranceCoverage" Value="1000000"/>
    <BenefitName="MonthlyTelephone" Value ="5000"/>
    <BenefitName="DependentMemberCount" Value="5"/>
    ...
  </Benefits>
</Author>
```

The AuthorBenefits table will hold the AuthorID and benefit details for each author.

```
Create Table AuthorBenefits
 (
       AuthorID int,
       BenefitName varchar(255),
       BenefitValue int
 );
```

The T-SQL query to read each /Benefit element from /Author/Benefits is shown below:

```
 --AuthorBenefits parsing query
 SELECT
   SingleRow.AuthorID,
  Convert(varchar(36),EachBenefit.query('data(@Name)'))As BenefitName,
  Cast(Convert(varchar(36),EachBenefit.query('data(@Value)'))As int)As BenefitValue
 FROM
 (
       SELECT
             AuthorProfile.value('/Author[1]/AuthorID[1]', 'int')As [AuthorID],
             AuthorProfile
       FROM dbo.AuthorList
 ) As SingleRow
```

```
CROSS APPLY
SingleRow.AuthorProfile.nodes('/Author/Benefits/Benefit') BenefitList(EachBenefit)
```

In a scenario where we want to filter out XML nodes, the FLWOR style statement can be used. The below T-SQL statement has a FLWOR query that filters out the Benefit nodes with a Value > 2200. We can also have filter based on string data. In the below T-SQL statement there is FLWOR clause that can be replaced with **where data($AuthorBenefitList/@Name) = "HospitalizationLimit"** to return the Benefit nodes having the Name as HospitalizationLimit.

```
--AuthorBenefits parsing query using FLWOR style query
SELECT
  SingleRow.AuthorID,
 Convert(varchar(36),EachBenefit.query('data(@Name)'))As BenefitName,
 Cast(Convert(varchar(36),EachBenefit.query('data(@Value)'))As int)As BenefitValue
FROM
(
      SELECT
            AuthorProfile.value('/Author[1]/AuthorID[1]', 'int')As [AuthorID]
            ,AuthorProfile.query('
                 for $AuthorBenefitList in /Author[1]/Benefits[1]/Benefit
                       where data($AuthorBenefitList/@Value) > 2200
                       return $AuthorBenefitList
                 ') As AuthorBenefitsColumn
      FROM dbo.AuthorList
) As SingleRow
CROSS APPLY
     SingleRow.AuthorBenefitsColumn.nodes('/Benefit') BenefitList(EachBenefit)
```

## Conclusion

In this article it was demonstrated how we can use inbuilt features of SQL Server query engine to manipulate the XML data. One of the important differences between XML and relational storage is the way data relationships are maintained. When data is stored as XML, entity relationships are implicitly preserved by the very nature of XML structure, which is hierarchical format. On the contrary, when data is stored in relational format, entity relationships are explicitly maintained by foreign key relationships. Since SQL 2005 lot of XML features have been added to the core engine that allows us to manipulate XML data without the need of adding application layer for such transformations. Appropriate XML indexes can be created to boost the performance of such transformations from the XML to the relational format.