### [Handling XML data in SQL Server](http://sathyadb.blogspot.in/2012/09/handling-xml-data-in-sql-server.html)

***XML :***

Extensible Markup Language (XML) has been widely adopted as a platform-independent format for data representation.

Before getting deep into XML,a little introduction to the structure of XML.

***Sample XML fragment:***

<?xml version = "1.0" encoding = "UTF-16"?>

<!—Student Information -->

<Studentinfo>

<Student>

<StudentID>1</StudentID>

<StudentName>Sathya</StudentName>

<CourseID>1</CourseID>

</Student>

<Student>

<StudentID>2</StudentID>

<StudentName>Deepak</StudentName>

<CourseID>2</CourseID>

</Student>

<Student>

<StudentID>3</StudentID>

<StudentName>sathish</StudentName>

<CourseID>3</CourseID>

</Student>

</Studentinfo>

In the above sample XML fragment ,

<Studentinfo> is the root node

<Student> is the element node,so in the above XML fragment,we havee 3 element nodes.

<StudentID> ,

<StudentName> ,

<CourseID> .. are the attribute nodes

1,sathya,1,.. are the attribute values.

<Student> is the start tag & </Student> is the end tag of the element node.

Similarly,

<Studentinfo> is the start tag & </Studentinfo> is the end tag of root node.

To put it simple in database terminologies,consider

root node as the database name,element node as the table name,attribute node as column name & attribute values as column values.

<!—Student Information --> are XML comments denoted by <!-- and --> delimiters

<?xml version = "1.0" encoding = "UTF-16"?> are XML processing instructions  marked by <? and ?> delimiters.

A processing instruction is a means to provide additional metadata to a processing application.

***FOR XML clause:***

For getting relational data in the form of XML, **FOR XML clause** was introduced.

Using FOR XML clause,we can represent relational data in the form of XML in two ways:  
 1.)    Attribute Centric   
2.)    Element Centric

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Sample data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Let us create sample datas for all below XML examples:

I have created sample data considering simple data model of students,courses offered,subjects in that course & marks details.

CREATE DATABASE TEST\_XML

GO

USE TEST\_XML

GO

CREATE TABLE Course

(

CourseID INT IDENTITY(1,1)NOT NULL,

CourseName VARCHAR(200) NOT NULL,

CONSTRAINT PK\_Course PRIMARY KEY(CourseID)

)

INSERT INTO Course (CourseName) SELECT 'Electronics Engineering'

INSERT INTO Course (CourseName) SELECT 'Computer Science'

INSERT INTO Course (CourseName) SELECT 'Civil Engineering'

CREATE TABLE Student

(

StudentID BIGINT IDENTITY(1,1)NOT NULL CONSTRAINT PK\_Student PRIMARY KEY(StudentID),

StudentName VARCHAR(200) NOT NULL,

CourseID INT NOT NULL CONSTRAINT FK\_Student\_CourseID FOREIGN KEY REFERENCES Course(CourseID)

)

INSERT INTO Student SELECT 'Sathya',1

INSERT INTO Student SELECT 'Deepak',2

INSERT INTO Student SELECT 'sathish', 3

CREATE TABLE Subject

(

SubjectID INT IDENTITY NOT NULL CONSTRAINT PK\_Subject PRIMARY KEY(SubjectID),

SubjectName VARCHAR(200)

)

INSERT INTO Subject (SubjectName) SELECT ('Electronics and Communication')

INSERT INTO Subject (SubjectName) SELECT ('Circuit Analysis')

INSERT INTO Subject (SubjectName) SELECT ('Mobile Communication')

INSERT INTO Subject (SubjectName) SELECT ('Data Structure')

INSERT INTO Subject (SubjectName) SELECT ('Java')

INSERT INTO Subject (SubjectName) SELECT ('Database Management System')

INSERT INTO Subject (SubjectName) SELECT ('Soil Mechanics')

INSERT INTO Subject (SubjectName) SELECT ('Steel Design')

INSERT INTO Subject (SubjectName) SELECT ('Concrete Design')

CREATE TABLE CourseSubject

(

CourseID INT CONSTRAINT FK\_CourseSubject\_CourseID FOREIGN KEY REFERENCES Course(CourseID),

SubjectID INT CONSTRAINT FK\_CourseSubject\_SubjectID FOREIGN KEY REFERENCES Subject(SubjectID)

)

INSERT INTO CourseSubject (CourseID,SubjectID) SELECT 1,1

INSERT INTO CourseSubject (CourseID,SubjectID) SELECT 1,2

INSERT INTO CourseSubject (CourseID,SubjectID) SELECT 1,3

INSERT INTO CourseSubject (CourseID,SubjectID) SELECT 2,4

INSERT INTO CourseSubject (CourseID,SubjectID) SELECT 2,5

INSERT INTO CourseSubject (CourseID,SubjectID) SELECT 2,6

INSERT INTO CourseSubject (CourseID,SubjectID) SELECT 3,7

INSERT INTO CourseSubject (CourseID,SubjectID) SELECT 3,8

INSERT INTO CourseSubject (CourseID,SubjectID) SELECT 3,9

CREATE TABLE Mark

(

StudentID BIGINT CONSTRAINT FK\_Mark\_StudentID FOREIGN KEY REFERENCES Student(StudentID),

SubjectID INT CONSTRAINT FK\_Mark\_SubjectID FOREIGN KEY REFERENCES Subject(SubjectID),

Mark INT

)

INSERT INTO Mark (StudentID,SubjectID,Mark) SELECT 1,1,75

INSERT INTO Mark (StudentID,SubjectID,Mark) SELECT 1,2,80

INSERT INTO Mark (StudentID,SubjectID,Mark) SELECT 1,3,70

INSERT INTO Mark (StudentID,SubjectID,Mark) SELECT 2,4,80

INSERT INTO Mark (StudentID,SubjectID,Mark) SELECT 2,5,80

INSERT INTO Mark (StudentID,SubjectID,Mark) SELECT 2,6,90

INSERT INTO Mark (StudentID,SubjectID,Mark) SELECT 3,7,80

INSERT INTO Mark (StudentID,SubjectID,Mark) SELECT 3,8,80

INSERT INTO Mark (StudentID,SubjectID,Mark) SELECT 3,9,90

SELECT \* FROM Course

SELECT \* FROM Student

SELECT \* FROM Subject

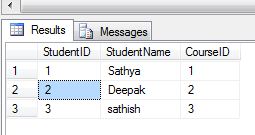
SELECT \* FROM CourseSubject

SELECT \* FROM Mark

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Sample data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

I will show you the resultset of this below query in XML format in attribute centric & element centric

SELECT \* FROM Student

[](http://1.bp.blogspot.com/-bVa9e1FjAzU/UEMTeDnN2gI/AAAAAAAAAGg/1d4mUAuYZEk/s1600/centricxml.JPG)

***Example for Attribute centric:***

Execute the below query,

SELECT \* FROM Student

FOR XML RAW

You will notice each row of relational data is represented by each row of XML as shown below:

***OUTPUT:***

<row StudentID="1" StudentName="Sathya" CourseID="1" />

<row StudentID="2" StudentName="Deepak" CourseID="2" />

<row StudentID="3" StudentName="sathish" CourseID="3" />

***Example for Element centric:***

Execute the below query,

SELECT \* FROM Student

FOR XML RAW ,ELEMENTS

You will notice each row of relational data is represented by each XML element fragments as shown below:

***OUTPUT:***

<row>

<StudentID>1</StudentID>

<StudentName>Sathya</StudentName>

<CourseID>1</CourseID>

</row>

<row>

<StudentID>2</StudentID>

<StudentName>Deepak</StudentName>

<CourseID>2</CourseID>

</row>

<row>

<StudentID>3</StudentID>

<StudentName>sathish</StudentName>

<CourseID>3</CourseID>

</row>

*Types of FOR XML clause:*

FOR XML RAW

FOR XML AUTO

FOR XML EXPLICIT

FOR XML PATH

Let us see,how the resultset of below query ,which is relational data is converted to XML by using different types of FOR XML clause at the end of the query .

SELECT

STD.StudentID ,

STD.StudentName ,

SUB.SubjectID ,

SUB.SubjectName ,

M.Mark

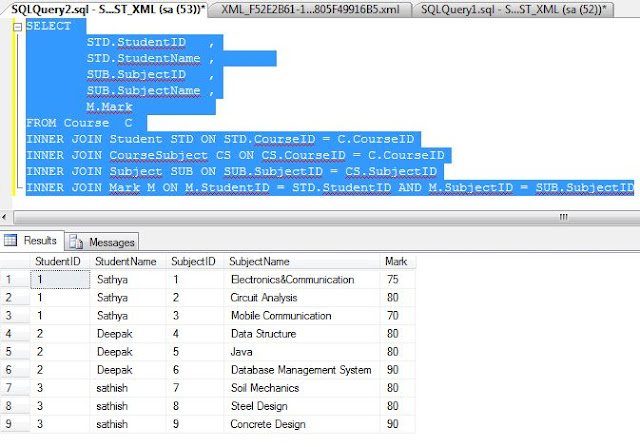
FROM Course C

INNER JOIN Student STD ON STD.CourseID = C.CourseID

INNER JOIN CourseSubject CS ON CS.CourseID = C.CourseID

INNER JOIN Subject SUB ON SUB.SubjectID = CS.SubjectID

INNER JOIN Mark M ON M.StudentID = STD.StudentID AND M.SubjectID = SUB.SubjectID

[](http://1.bp.blogspot.com/-uxQ-c4Z6BgI/UEMUCSQJZNI/AAAAAAAAAGo/Hj5A-OV7phc/s1600/path1.JPG)

***FOR XML RAW:***

FOR XML  clause returns attribute centric XML data by default.

Example 1 - **FOR XML RAW :**

SELECT

STD.StudentID ,

STD.StudentName ,

SUB.SubjectID ,

SUB.SubjectName ,

M.Mark

FROM Course C

INNER JOIN Student STD ON STD.CourseID = C.CourseID

INNER JOIN CourseSubject CS ON CS.CourseID = C.CourseID

INNER JOIN Subject SUB ON SUB.SubjectID = CS.SubjectID

INNER JOIN Mark M ON M.StudentID = STD.StudentID AND M.SubjectID = SUB.SubjectID

FOR XML RAW

***OUTPUT:***

<row StudentID="1" StudentName="Sathya" SubjectID="1" SubjectName="Electronics and Communication" Mark="75" />

<row StudentID="1" StudentName="Sathya" SubjectID="2" SubjectName="Circuit Analysis" Mark="80" />

<row StudentID="1" StudentName="Sathya" SubjectID="3" SubjectName="Mobile Communication" Mark="70" />

<row StudentID="2" StudentName="Deepak" SubjectID="4" SubjectName="Data Structure" Mark="80" />

<row StudentID="2" StudentName="Deepak" SubjectID="5" SubjectName="Java" Mark="80" />

<row StudentID="2" StudentName="Deepak" SubjectID="6" SubjectName="Database Management System" Mark="90" />

<row StudentID="3" StudentName="sathish" SubjectID="7" SubjectName="Soil Mechanics" Mark="80" />

<row StudentID="3" StudentName="sathish" SubjectID="8" SubjectName="Steel Design" Mark="80" />

<row StudentID="3" StudentName="sathish" SubjectID="9" SubjectName="Concrete Design" Mark="90" />

***Example 2 - FOR XML RAW:***

***Step 1:***

UPDATE Mark SET Mark = NULL WHERE StudentID = 3 AND SubjectID = 9

***Step 2:***

let us see ,how FOR XML RAW returns element centric XML data usingELEMENTS directive and NULL is handled using XSINIL option.

SELECT

STD.StudentID ,

STD.StudentName ,

SUB.SubjectID ,

SUB.SubjectName ,

M.Mark

FROM Course C

INNER JOIN Student STD ON STD.CourseID = C.CourseID

INNER JOIN CourseSubject CS ON CS.CourseID = C.CourseID

INNER JOIN Subject SUB ON SUB.SubjectID = CS.SubjectID

INNER JOIN Mark M ON M.StudentID = STD.StudentID AND M.SubjectID = SUB.SubjectID

FOR XML RAW ,ELEMENTS XSINIL

***OUTPUT:***

<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>1</StudentID>

<StudentName>Sathya</StudentName>

<SubjectID>1</SubjectID>

<SubjectName>Electronics and Communication</SubjectName>

<Mark>75</Mark>

</row>

<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>1</StudentID>

<StudentName>Sathya</StudentName>

<SubjectID>2</SubjectID>

<SubjectName>Circuit Analysis</SubjectName>

<Mark>80</Mark>

</row>

<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>1</StudentID>

<StudentName>Sathya</StudentName>

<SubjectID>3</SubjectID>

<SubjectName>Mobile Communication</SubjectName>

<Mark>70</Mark>

</row>

<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>2</StudentID>

<StudentName>Deepak</StudentName>

<SubjectID>4</SubjectID>

<SubjectName>Data Structure</SubjectName>

<Mark>80</Mark>

</row>

<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>2</StudentID>

<StudentName>Deepak</StudentName>

<SubjectID>5</SubjectID>

<SubjectName>Java</SubjectName>

<Mark>80</Mark>

</row>

<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>2</StudentID>

<StudentName>Deepak</StudentName>

<SubjectID>6</SubjectID>

<SubjectName>Database Management System</SubjectName>

<Mark>90</Mark>

</row>

<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>3</StudentID>

<StudentName>sathish</StudentName>

<SubjectID>7</SubjectID>

<SubjectName>Soil Mechanics</SubjectName>

<Mark>80</Mark>

</row>

<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>3</StudentID>

<StudentName>sathish</StudentName>

<SubjectID>8</SubjectID>

<SubjectName>Steel Design</SubjectName>

<Mark>80</Mark>

</row>

<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>3</StudentID>

<StudentName>sathish</StudentName>

<SubjectID>9</SubjectID>

<SubjectName>Concrete Design</SubjectName>

<Mark xsi:nil="true" />

</row>

***FOR XML AUTO:***

Used to return query resultset as nested XML elements.

Similar to FOR XML RAW,but FOR XML AUTO automatically takes element node name as table alias name & attribute node name as column name in SELECT statement.

FOR XML AUTO also return nested attribute centric XML data by default.

***EXAMPLE – 1 :FOR XML AUTO***

SELECT

STUDENT.StudentID ,

STUDENT.StudentName ,

SUBJECT.SubjectID ,

SUBJECT.SubjectName ,

MARK.Mark

FROM Course COURSE

INNER JOIN Student STUDENT ON STUDENT.CourseID = COURSE.CourseID

INNER JOIN CourseSubject CS ON CS.CourseID = COURSE.CourseID

INNER JOIN Subject SUBJECT ON SUBJECT.SubjectID = CS.SubjectID

INNER JOIN Mark MARK ON MARK.StudentID = STUDENT.StudentID AND MARK.SubjectID = SUBJECT.SubjectID

FOR XML AUTO

***OUTPUT:***

<STUDENT StudentID="1" StudentName="Sathya">

<SUBJECT SubjectID="1" SubjectName="Electronics and Communication">

<MARK Mark="75" />

</SUBJECT>

<SUBJECT SubjectID="2" SubjectName="Circuit Analysis">

<MARK Mark="80" />

</SUBJECT>

<SUBJECT SubjectID="3" SubjectName="Mobile Communication">

<MARK Mark="70" />

</SUBJECT>

</STUDENT>

<STUDENT StudentID="2" StudentName="Deepak">

<SUBJECT SubjectID="4" SubjectName="Data Structure">

<MARK Mark="80" />

</SUBJECT>

<SUBJECT SubjectID="5" SubjectName="Java">

<MARK Mark="80" />

</SUBJECT>

<SUBJECT SubjectID="6" SubjectName="Database Management System">

<MARK Mark="90" />

</SUBJECT>

</STUDENT>

<STUDENT StudentID="3" StudentName="sathish">

<SUBJECT SubjectID="7" SubjectName="Soil Mechanics">

<MARK Mark="80" />

</SUBJECT>

<SUBJECT SubjectID="8" SubjectName="Steel Design">

<MARK Mark="80" />

</SUBJECT>

<SUBJECT SubjectID="9" SubjectName="Concrete Design">

<MARK />

</SUBJECT>

</STUDENT>

***EXAMPLE – 2 :FOR XML AUTO***

SELECT

STUDENT.StudentID ,

STUDENT.StudentName ,

SUBJECT.SubjectID ,

SUBJECT.SubjectName ,

MARK.Mark

FROM Course COURSE

INNER JOIN Student STUDENT ON STUDENT.CourseID = COURSE.CourseID

INNER JOIN CourseSubject CS ON CS.CourseID = COURSE.CourseID

INNER JOIN Subject SUBJECT ON SUBJECT.SubjectID = CS.SubjectID

INNER JOIN Mark MARK ON MARK.StudentID = STUDENT.StudentID AND MARK.SubjectID = SUBJECT.SubjectID

FOR XML AUTO,ELEMENTS XSINIL

***OUTPUT:***

<STUDENT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>1</StudentID>

<StudentName>Sathya</StudentName>

<SUBJECT>

<SubjectID>1</SubjectID>

<SubjectName>Electronics and Communication</SubjectName>

<MARK>

<Mark>75</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>2</SubjectID>

<SubjectName>Circuit Analysis</SubjectName>

<MARK>

<Mark>80</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>3</SubjectID>

<SubjectName>Mobile Communication</SubjectName>

<MARK>

<Mark>70</Mark>

</MARK>

</SUBJECT>

</STUDENT>

<STUDENT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>2</StudentID>

<StudentName>Deepak</StudentName>

<SUBJECT>

<SubjectID>4</SubjectID>

<SubjectName>Data Structure</SubjectName>

<MARK>

<Mark>80</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>5</SubjectID>

<SubjectName>Java</SubjectName>

<MARK>

<Mark>80</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>6</SubjectID>

<SubjectName>Database Management System</SubjectName>

<MARK>

<Mark>90</Mark>

</MARK>

</SUBJECT>

</STUDENT>

<STUDENT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>3</StudentID>

<StudentName>sathish</StudentName>

<SUBJECT>

<SubjectID>7</SubjectID>

<SubjectName>Soil Mechanics</SubjectName>

<MARK>

<Mark>80</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>8</SubjectID>

<SubjectName>Steel Design</SubjectName>

<MARK>

<Mark>80</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>9</SubjectID>

<SubjectName>Concrete Design</SubjectName>

<MARK>

<Mark xsi:nil="true" />

</MARK>

</SUBJECT>

</STUDENT>

***EXAMPLE – 3: FOR XML AUTO :***

SELECT

STUDENT.StudentID ,

STUDENT.StudentName ,

SUBJECT.SubjectID ,

SUBJECT.SubjectName ,

MARK.Mark

FROM Course COURSE

INNER JOIN Student STUDENT ON STUDENT.CourseID = COURSE.CourseID

INNER JOIN CourseSubject CS ON CS.CourseID = COURSE.CourseID

INNER JOIN Subject SUBJECT ON SUBJECT.SubjectID = CS.SubjectID

INNER JOIN Mark MARK ON MARK.StudentID = STUDENT.StudentID AND MARK.SubjectID = SUBJECT.SubjectID

FOR XML AUTO,ELEMENTS XSINIL,TYPE,XMLSCHEMA

o   The TYPE option, when specified, returns your FOR XML result as an xml data type instance.

o   XMLSCHEMA to get schema along with XML result.

***OUTPUT:***

<xsd:schema xmlns:schema="urn:schemas-microsoft-com:sql:SqlRowSet1" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:sqltypes="http://schemas.microsoft.com/sqlserver/2004/sqltypes" targetNamespace="urn:schemas-microsoft-com:sql:SqlRowSet1" elementFormDefault="qualified">

<xsd:import namespace="http://schemas.microsoft.com/sqlserver/2004/sqltypes" schemaLocation="http://schemas.microsoft.com/sqlserver/2004/sqltypes/sqltypes.xsd" />

<xsd:element name="STUDENT">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="StudentID" type="sqltypes:bigint" nillable="1" />

<xsd:element name="StudentName" nillable="1">

<xsd:simpleType>

<xsd:restriction base="sqltypes:varchar" sqltypes:localeId="1033" sqltypes:sqlCompareOptions="IgnoreCase IgnoreKanaType IgnoreWidth" sqltypes:sqlSortId="52">

<xsd:maxLength value="200" />

</xsd:restriction>

</xsd:simpleType>

</xsd:element>

<xsd:element ref="schema:SUBJECT" minOccurs="0" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

<xsd:element name="SUBJECT">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="SubjectID" type="sqltypes:int" nillable="1" />

<xsd:element name="SubjectName" nillable="1">

<xsd:simpleType>

<xsd:restriction base="sqltypes:varchar" sqltypes:localeId="1033" sqltypes:sqlCompareOptions="IgnoreCase IgnoreKanaType IgnoreWidth" sqltypes:sqlSortId="52">

<xsd:maxLength value="200" />

</xsd:restriction>

</xsd:simpleType>

</xsd:element>

<xsd:element ref="schema:MARK" minOccurs="0" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

<xsd:element name="MARK">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Mark" type="sqltypes:int" nillable="1" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:schema>

<STUDENT xmlns="urn:schemas-microsoft-com:sql:SqlRowSet1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>1</StudentID>

<StudentName>Sathya</StudentName>

<SUBJECT>

<SubjectID>1</SubjectID>

<SubjectName>Electronics&amp;Communication</SubjectName>

<MARK>

<Mark>75</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>2</SubjectID>

<SubjectName>Circuit Analysis</SubjectName>

<MARK>

<Mark>80</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>3</SubjectID>

<SubjectName>Mobile Communication</SubjectName>

<MARK>

<Mark>70</Mark>

</MARK>

</SUBJECT>

</STUDENT>

<STUDENT xmlns="urn:schemas-microsoft-com:sql:SqlRowSet1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>2</StudentID>

<StudentName>Deepak</StudentName>

<SUBJECT>

<SubjectID>4</SubjectID>

<SubjectName>Data Structure</SubjectName>

<MARK>

<Mark>80</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>5</SubjectID>

<SubjectName>Java</SubjectName>

<MARK>

<Mark>80</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>6</SubjectID>

<SubjectName>Database Management System</SubjectName>

<MARK>

<Mark>90</Mark>

</MARK>

</SUBJECT>

</STUDENT>

<STUDENT xmlns="urn:schemas-microsoft-com:sql:SqlRowSet1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<StudentID>3</StudentID>

<StudentName>sathish</StudentName>

<SUBJECT>

<SubjectID>7</SubjectID>

<SubjectName>Soil Mechanics</SubjectName>

<MARK>

<Mark>80</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>8</SubjectID>

<SubjectName>Steel Design</SubjectName>

<MARK>

<Mark>80</Mark>

</MARK>

</SUBJECT>

<SUBJECT>

<SubjectID>9</SubjectID>

<SubjectName>Concrete Design</SubjectName>

<MARK>

<Mark xsi:nil="true" />

</MARK>

</SUBJECT>

  </STUDENT>

***FOR XML EXPICIT:***

Using FOR XML EXPLICIT is little different,we can define the path (node structure)of output XML .

But its complicated because we have to use nested SELECT statements using UNION ALL to define path of output XML.

***EXAMPLE FOR XML EXPLICIT :***

IF you have a look at the below example,

 Tag & Parent are used to nest subsequent SELECT statements.

In the first SELECT statement itself we can define the path (node structure) of output XML in the below format

**ElementName!TagNumber!AttributeName!Directive**

Each SELECT statement represents an Element node,so we are assigning values to each SELECT statement accordingly.

SELECT 1 AS Tag,

NULL AS Parent,

0 AS [STUDENTINFO!1!SORT!HIDE],

NULL AS [STUDENTINFO!1!],

NULL AS [student!2!ID],

NULL AS [student!2!name],

NULL AS [subject!3!ID],

NULL AS [subject!3!Name]

UNION ALL

SELECT 2 AS Tag,

1 AS Parent,

STD.StudentID + 1,

NULL,

STD.StudentID AS [student!2!ID],

STD.StudentName AS [student!2!name],

NULL ,

NULL

FROM Student STD

UNION ALL

SELECT 3 AS Tag,

2 AS Parent,

STD.StudentID + 1,

NULL,

NULL,

NULL,

SUB.SubjectID AS [subject!3!ID],

SUB.SubjectName As [subject!3!Name]

FROM Course C

INNER JOIN Student STD ON STD.CourseID = C.CourseID

INNER JOIN CourseSubject CS ON CS.CourseID = C.CourseID

INNER JOIN Subject SUB ON SUB.SubjectID = CS.SubjectID

INNER JOIN Mark M ON M.StudentID = STD.StudentID AND M.SubjectID = SUB.SubjectID

ORDER BY [STUDENTINFO!1!SORT!HIDE]

FOR XML EXPLICIT

In the above example [STUDENTINFO!1!SORT!HIDE] is nothing but used to sort the elements and HIDE directive is used to hide the sorting column.

***OUTPUT:***

<STUDENTINFO>

<student ID="1" name="Sathya">

<subject ID="1" Name="Electronics and Communication" />

<subject ID="2" Name="Circuit Analysis" />

<subject ID="3" Name="Mobile Communication" />

</student>

<student ID="2" name="Deepak">

<subject ID="4" Name="Data Structure" />

<subject ID="5" Name="Java" />

<subject ID="6" Name="Database Management System" />

</student>

<student ID="3" name="sathish">

<subject ID="7" Name="Soil Mechanics" />

<subject ID="8" Name="Steel Design" />

<subject ID="9" Name="Concrete Design" />

</student>

</STUDENTINFO>

***FOR XML PATH:***

Though both FOR XML EXPLICIT & XML PATH perform same operation.

FOR XML PATH is very simple in defining path(node structure)of XML output.

Using FOR XML PATH,we can define path(node structure)of XML output in single SELECT as shown below:

SELECT

STD.StudentID "@StudentID" ,

STD.StudentName "@StudentName",

SUB.SubjectID "Student/ID",

SUB.SubjectName "Student/Name",

M.Mark "Student/Mark"

FROM Course C

INNER JOIN Student STD ON STD.CourseID = C.CourseID

INNER JOIN CourseSubject CS ON CS.CourseID = C.CourseID

INNER JOIN Subject SUB ON SUB.SubjectID = CS.SubjectID

INNER JOIN Mark M ON M.StudentID = STD.StudentID AND M.SubjectID = SUB.SubjectID

FOR XML PATH ('Student')

@StudentID - @ indicates that it should be attribute.

***OUTPUT:***

<Student StudentID="1" StudentName="Sathya">

<Student>

<ID>1</ID>

<Name>Electronics and Communication</Name>

<Mark>75</Mark>

</Student>

</Student>

<Student StudentID="1" StudentName="Sathya">

<Student>

<ID>2</ID>

<Name>Circuit Analysis</Name>

<Mark>80</Mark>

</Student>

</Student>

<Student StudentID="1" StudentName="Sathya">

<Student>

<ID>3</ID>

<Name>Mobile Communication</Name>

<Mark>70</Mark>

</Student>

</Student>

<Student StudentID="2" StudentName="Deepak">

<Student>

<ID>4</ID>

<Name>Data Structure</Name>

<Mark>80</Mark>

</Student>

</Student>

<Student StudentID="2" StudentName="Deepak">

<Student>

<ID>5</ID>

<Name>Java</Name>

<Mark>80</Mark>

</Student>

</Student>

<Student StudentID="2" StudentName="Deepak">

<Student>

<ID>6</ID>

<Name>Database Management System</Name>

<Mark>90</Mark>

</Student>

</Student>

<Student StudentID="3" StudentName="sathish">

<Student>

<ID>7</ID>

<Name>Soil Mechanics</Name>

<Mark>80</Mark>

</Student>

</Student>

<Student StudentID="3" StudentName="sathish">

<Student>

<ID>8</ID>

<Name>Steel Design</Name>

<Mark>80</Mark>

</Student>

</Student>

<Student StudentID="3" StudentName="sathish">

<Student>

<ID>9</ID>

<Name>Concrete Design</Name>

</Student>

</Student>

### [XQUERY,XPATH,XMLSCHEMA,XML INDEX](http://sathyadb.blogspot.in/2012/09/xqueryxpathxmlschemaxml-index_6.html)

***(XQUERY,XPATH,XMLSCHEMA,XML INDEX)***

Ø  XQuery is ***the*** language for querying XML data.

   Ø  XPath was designed to navigate an XML document to retrieve the documents elements and attributes.

     Ø  XQuery is built on XPath expressions.

Ø To put it simple XQuery,XPath are used to traverse through XML document/fragment to fetch/modify attribute values or element nodes.

***XQuery FLWOR Expression:***

* for - (optional) binds a variable to each item returned by the in expression
* let - (optional) to assign value to variable
* where - (optional) specifies a criteria
* order by - (optional) specifies the sort-order of the result
* return - specifies what to return in the result.

**[Using XML Type Methods:](http://www.blogger.com/blogger.g?blogID=954987329016201413)**

Ø  The xml data type features several built-in methods that allow you to manipulate XML instance data.

These methods allow you to query, modify, or shred your XML data into relational form.

**query()**   -The query() method allows you to perform an XQuery on your xml instance.

The result returned is untyped XML.

**Syntax**: DbObject.query('XQuery')

**value()**  - The value() method allows you to perform an XQuery on your xml instance and returns a scalar value cast to a SQL Server data type.

**Syntax:** DbObject.value('XQuery', 'SqlType')

**exist()** - The exist() method allows you to specify an XQuery on your xml instance and returns a SQL bit value of 1 if the XQuery returns a result, 0 if the XQuery returns no result, or NULL if the xml instance is NULL.

**Syntax** :DbObject.exist('XQuery')

**modify()**                 - The modify() method allows you to execute XML Data Manipulation Language (XML DML) statements against an xml instance. The modify() method can only  be used with a SET clause or statement.

**Syntax** :DbObject.modify('XQuery')

**nodes() -** The nodes() method allows you to shred xml instances. *Shredding* is the process of converting your XML data to relational form.

**Syntax** :DbObject.nodes('XQuery') AS TableAlias(ColumnAlias)

**XML DML Keywords:**

**Delete** - Deletes the node specified by an XQuery path expression.

**Insert** -  Inserts one or more nodes as the children or siblings of a node specified

by an XQuery path expression.

**replace value of**    -  Updates the value of a node specified by an XQuery path.

**--Examples for XQUERY & XPATH**

--lets us create sample data

USE TEST\_XML

GO

CREATE TABLE College\_Master

(College\_ID INT NOT NULL,

College\_Name VARCHAR(20),

College\_Details XML)

INSERT INTO College\_Master VALUES (1,'BCC UNIVERSITY','<STUDENTINFO>

<student ID="1" name="Sathya">

<subject ID="1" Name="Electronics and Communication" />

<subject ID="2" Name="Circuit Analysis" />

<subject ID="3" Name="Mobile Communication" />

</student>

<student ID="2" name="Deepak">

<subject ID="4" Name="Data Structure" />

<subject ID="5" Name="Java" />

<subject ID="6" Name="Database Management System" />

</student>

<student ID="3" name="sathish">

<subject ID="7" Name="Soil Mechanics" />

<subject ID="8" Name="Steel Design" />

<subject ID="9" Name="Concrete Design" />

</student>

</STUDENTINFO>')

INSERT INTO College\_Master VALUES (2,'ICC UNIVERSITY','<STUDENTINFO>

<student ID="1" name="Karthi">

<subject ID="1" Name="Electronics and Communication" />

<subject ID="2" Name="Circuit Analysis" />

<subject ID="3" Name="Mobile Communication" />

</student>

<student ID="2" name="Vikram">

<subject ID="4" Name="Data Structure" />

<subject ID="5" Name="Java" />

<subject ID="6" Name="Database Management System" />

</student>

<student ID="3" name="Anandh">

<subject ID="7" Name="Soil Mechanics" />

<subject ID="8" Name="Steel Design" />

<subject ID="9" Name="Concrete Design" />

</student>

</STUDENTINFO>')

SELECT \* FROM College\_Master

--query() & value() examples:

--query() returns typed XML

SELECT College\_Details.query('/STUDENTINFO/student') FROM College\_Master

WHERE College\_ID = 1;

--value() returns single value of mentioned data type

SELECT College\_Details.value('(/STUDENTINFO/student/@name)[1]','varchar(20)') FROM College\_Master

WHERE College\_ID = 1;

--XPath examples :

--to get all below nodes from root node - STUDENTINFO

 SELECT College\_Details.query('/STUDENTINFO') FROM College\_Master

WHERE College\_ID = 1;

--to get all element nodes named 'student'

 SELECT College\_Details.query('//student') FROM College\_Master

WHERE College\_ID = 1;

-- to get student node with ID = 1

 SELECT College\_Details.query('//student[@ID = 1]') FROM College\_Master

WHERE College\_ID = 1;

--to get student node at position = 1 from root node

 SELECT College\_Details.query('/STUDENTINFO/student[position()=1]') FROM College\_Master

WHERE College\_ID = 1;

-- to get student node at last position from root node

 SELECT College\_Details.query('/STUDENTINFO/student[last()]') FROM College\_Master

WHERE College\_ID = 1;

--to return unknown nodes from root node - STUDENTINFO

 SELECT College\_Details.query('/STUDENTINFO/\*') FROM College\_Master

WHERE College\_ID = 1;

--to return unknown nodes from XML column - College\_Details

 SELECT College\_Details.query('//\* ') FROM College\_Master

WHERE College\_ID = 1;

--XQuery FLWOR example:

--from resultset of $i ,assigning it to variable $j using let clause,

Getting top two students by using where & order by clause.

SELECT College\_Details.query('for $i in /STUDENTINFO/student

let $j:= $i

where ($j/@ID) <= 2

order by ($j/@ID)[1] ascending

return $j')

FROM College\_Master

WHERE College\_ID = 1;

--Example:

                for XQuery built-in functions & operators:

-- Sql:column can be used to access non-xml columns in a relational table.

--Example:

SELECT College\_Details.query ('

<College\_Master>

<College\_Details> {

sql:column("College\_ID"),

sql:column("College\_Name")

} </College\_Details>

</College\_Master>')

FROM College\_Master

WHERE College\_ID = 1;

-- Sql:variable can be used to use sql variables inside query()

--Example:

DECLARE @ID INT = 1;

SELECT College\_Details.query('/STUDENTINFO/student[@ID = sql:variable("@ID")]') FROM College\_Master

WHERE College\_ID = 1;

--string()

 SELECT College\_Details.query('string((/STUDENTINFO/student[1]/@name)[1])') FROM College\_Master

WHERE College\_ID = 1;

--data()

 SELECT College\_Details.query('data(//student/@name)') FROM College\_Master WHERE College\_ID = 1;

--Operators examples:

**Value comparison** - Value comparison operators help compare atomic values.

Example: checking first student id = 1

SELECT College\_Details.query('(/STUDENTINFO/student/@ID)[1] eq 1')

FROM College\_Master

WHERE College\_ID = 1

SELECT College\_Details.query('(/STUDENTINFO/student/@ID)[1] eq 1')

FROM College\_Master

WHERE College\_ID = 1

**Node comparison**- node comparison operator is to compare two nodes to determine if they represent the same node or not.

Example:comparing nodes

SELECT College\_Details.query('

if ( (/STUDENTINFO/student)[1] is (//student)[1] )

then

<Result>Nodes are equal</Result>

else

<Result>Nodes are not equal</Result>

') as Result

FROM College\_Master

WHERE College\_ID = 1

SELECT College\_Details.query('

if ( (/STUDENTINFO/student)[1] is (//subject)[1] )

then

<Result>Nodes are equal</Result>

else

<Result>Nodes are not equal</Result>

') as Result

FROM College\_Master

WHERE College\_ID = 1

--XML DML examples

--modify() is used for insert/update/delete operations

--to insert 'Remarks' node after first 'student' node

UPDATE College\_Master

SET College\_Details.modify('

insert(<Remarks> Passed </Remarks>)

after(/STUDENTINFO/student)[1]')

WHERE College\_ID = 1;

SELECT \* FROM College\_Master WHERE College\_ID = 1;

--to update 'Remarks' node value & also notice that text() is used to fetch text value of node.

UPDATE College\_Master

SET College\_Details.modify('

replace value of (/STUDENTINFO/Remarks/text())[1]

with "Passed with Distinction" ')

WHERE College\_ID = 1;

SELECT \* FROM College\_Master WHERE College\_ID = 1;

--to delete 'Remarks' node

UPDATE College\_Master

SET College\_Details.modify('delete(/STUDENTINFO/Remarks)[1]')

WHERE College\_ID = 1;

SELECT \* FROM College\_Master WHERE College\_ID = 1;

--exist() example:

--returns 1 if exists

SELECT College\_Details.exist('/STUDENTINFO/student[@name = "Sathya"]') FROM College\_Master

WHERE College\_ID = 1;

--returns 0 if not exists

SELECT College\_Details.exist('/STUDENTINFO/student[@name = "sathya"]') FROM College\_Master

WHERE College\_ID = 1;

--nodes() used for shredding XML data into columns of relational table

Examples:

--to shred StudentID, StudentName into Student table from XML column.

CREATE TABLE Student\_Master (ID INT,NAME NVARCHAR(30))  
  
INSERT INTO Student\_Master  
SELECT  
 tab.col.value('@ID[1]', 'INT') StudentID  
,tab.col.value('@name[1]', 'NVARCHAR(30)') StudentName  
FROM College\_Master CROSS APPLY College\_Details.nodes('/STUDENTINFO/student') tab(col)  
WHERE College\_ID = 1   
  
SELECT \* FROM Student\_Master

--to shred SubjectID, SubjectName into Subject table from XML column.

CREATE TABLE Subject\_Master (ID INT,NAME NVARCHAR(30))  
  
INSERT INTO Subject\_Master  
SELECT  
 tab.col.value('@ID[1]', 'INT') SubjectID  
,tab.col.value('@Name[1]', 'NVARCHAR(30)') SubjectName  
FROM College\_Master CROSS APPLY College\_Details.nodes('/STUDENTINFO/student/subject') tab(col)   
WHERE College\_ID = 1  
  
SELECT \* FROM Subject\_Master

Using OPENXML is another approach for shredding XML data into columns of relational table

Example:

DECLARE @DocHandle int

DECLARE @XmlDocument nvarchar(1000)

SET @XmlDocument = N'<STUDENTINFO>

<student ID="1" name="Sathya">

<subject ID="1" Name="Electronics and Communication" />

<subject ID="2" Name="Circuit Analysis" />

<subject ID="3" Name="Mobile Communication" />

</student>

<student ID="2" name="Deepak">

<subject ID="4" Name="Data Structure" />

<subject ID="5" Name="Java" />

<subject ID="6" Name="Database Management System" />

</student>

<student ID="3" name="sathish">

<subject ID="7" Name="Soil Mechanics" />

<subject ID="8" Name="Steel Design" />

<subject ID="9" Name="Concrete Design" />

</student>

</STUDENTINFO>'

EXEC sp\_xml\_preparedocument @DocHandle OUTPUT, @XmlDocument

SELECT \* FROM OPENXML (@DocHandle, '/STUDENTINFO/student',1)

WITH (ID INT,

name VARCHAR(30))

EXEC sp\_xml\_removedocument @DocHandle

Ø  The sp\_xml\_preparedocument stored procedure stores the prepared XML document in SQL Server's internal cache.

Ø  Calling the sp\_xml\_removedocument stored procedure is necessary in order to remove the prepared XML document from the cache.

Ø  When OPENXML approach is used,according to SQL Server 2005 Books on Line, one-eighth of the total memory available to SQL Server may be used by the MSXML parser.

***XML SCHEMA COLLECTION:***

XML Schema is used to

                       validate structure of XML,

                       validate  data types and

                       to perform restriction on values.

To  associate XSD schemas with a variable or a column of **xml** type use  XML schema collection

If  a variable or a column is associated with XML schema then it called typed xml,if not it is called untyped XML.

o    **CREATE XML SCHEMA COLLECTION** - Imports schema components into a database.

o    **ALTER XML SCHEMA COLLECTION** - Modifies the schema components in an existing XML schema collection.

o    **DROP XML SCHEMA COLLECTION** - Deletes a complete XML schema collection and all its components.

--Examples:

--you can create xml schema,typing it manually or you generate it automatically with help of various tools.

--step 1: create xml schema

CREATE XML SCHEMA COLLECTION Studentinfo\_studentID1\_Schema

AS

'<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="STUDENTINFO">

<xs:complexType>

<xs:sequence>

<xs:element name="student" maxOccurs="3" minOccurs="0">

<xs:complexType>

<xs:sequence>

<xs:element name="subject" maxOccurs="unbounded" minOccurs="0">

<xs:complexType>

<xs:simpleContent>

<xs:extension base="xs:string">

<xs:attribute type="xs:byte" name="ID" use="optional"/>

<xs:attribute type="xs:string" name="Name" use="optional"/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

</xs:element>

</xs:sequence>

<xs:attribute type="xs:byte" name="ID" use="optional"/>

<xs:attribute type="xs:string" name="name" use="optional"/>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>'

--step 2: create table with xml column registered with xml schema

CREATE TABLE XML\_Schema\_test

(

ID INT PRIMARY KEY,

College\_Details XML(Studentinfo\_studentID1\_Schema)

);

--step 3:insert with 4 <student> element nodes

--it will throw error because in schema,we have mentioned that max occurance of <student> = 3

-- " <xs:element name="student" maxOccurs="3" minOccurs="0"> "

INSERT INTO XML\_Schema\_test SELECT 1,

'<STUDENTINFO>

<student ID="1" name="Sathya">

<subject ID="1" Name="Electronics and Communication" />

<subject ID="2" Name="Circuit Analysis" />

<subject ID="3" Name="Mobile Communication" />

</student>

<student ID="2" name="Deepak">

<subject ID="4" Name="Data Structure" />

<subject ID="5" Name="Java" />

<subject ID="6" Name="Database Management System" />

</student>

<student ID="3" name="sathish">

<subject ID="7" Name="Soil Mechanics" />

<subject ID="8" Name="Steel Design" />

<subject ID="9" Name="Concrete Design" />

</student>

<student ID="4" name="Sunny">

<subject ID="7" Name="Soil Mechanics" />

<subject ID="8" Name="Steel Design" />

<subject ID="9" Name="Concrete Design" />

</student>

</STUDENTINFO>

'

-- try again with 3 <student> element nodes

INSERT INTO XML\_Schema\_test SELECT 1,

'<STUDENTINFO>

<student ID="1" name="Sathya">

<subject ID="1" Name="Electronics and Communication" />

<subject ID="2" Name="Circuit Analysis" />

<subject ID="3" Name="Mobile Communication" />

</student>

<student ID="2" name="Deepak">

<subject ID="4" Name="Data Structure" />

<subject ID="5" Name="Java" />

<subject ID="6" Name="Database Management System" />

</student>

<student ID="3" name="sathish">

<subject ID="7" Name="Soil Mechanics" />

<subject ID="8" Name="Steel Design" />

<subject ID="9" Name="Concrete Design" />

</student>

</STUDENTINFO>

'

---try to do the below insert,you will get error,

--because of data type mismatch(datatype mentioned in schema & what we are inserting are different)

-- " <xs:attribute type="xs:string" name="Name" use="optional"/> "

INSERT INTO XML\_Schema\_test SELECT 1,

'<STUDENTINFO>

<student ID="1" name="Sathya">

<subject ID="1" Name="Electronics and Communication" />

<subject ID="2" Name="Circuit Analysis" />

<subject ID="3" Name="Mobile Communication" />

</student>

<student ID="2" name="Deepak">

<subject ID="4" Name="Data Structure" />

<subject ID="5" Name="Java" />

<subject ID="6" Name="Database Management System" />

</student>

<student ID="3" name="sathish">

<subject ID="7" Name="Soil Mechanics" />

<subject ID="8" Name="Steel Design" />

<subject ID="Nine" Name="Concrete Design" />

</student>

</STUDENTINFO>

'

-- try again with matching data type value

INSERT INTO XML\_Schema\_test SELECT 1,

'<STUDENTINFO>

<student ID="1" name="Sathya">

<subject ID="1" Name="Electronics and Communication" />

<subject ID="2" Name="Circuit Analysis" />

<subject ID="3" Name="Mobile Communication" />

</student>

<student ID="2" name="Deepak">

<subject ID="4" Name="Data Structure" />

<subject ID="5" Name="Java" />

<subject ID="6" Name="Database Management System" />

</student>

<student ID="3" name="sathish">

<subject ID="7" Name="Soil Mechanics" />

<subject ID="8" Name="Steel Design" />

<subject ID="9" Name="Concrete Design" />

</student>

</STUDENTINFO>

'

***XML index:***

  Types: Primary (base) and Secondary (PATH, VALUE, PROPERTY – Non-clustered on the primary)

  Storage: Increases XML storage cost. Estimate as 3 times the storage requirement of XML instance in base table.

  To create the primary XML index, table must have a clustered primary key

  Primary index has 11 columns, including id, nid (node), hid (hierarchy), value, lvalue, PK[n]

  The primary XML index on an XML column is a clustered index on an internal table known as the node table that users cannot use directly from their T-SQL statements.

  The primary XML index is a B+tree and its usefulness is due to the way that the optimizer creates a plan for the entire query.

  The primary XML index essentially contains one row for each node in the XML instance

  Once the primary XML index has been created, an additional three kinds of secondary XML index can be created.

  The secondary XML indexes assist in certain types of XQuery processing. These are called the PATH, PROPERTY, and VALUE indexes.

**XML Indexing – Secondary Indexes:**

* Path index - Good for path queries
* Value index - Optimized to find the value and we don’t know the path
* Property index - Optimized for name-value pair storage

**Examples:**

--create primary key to create primary xml index

ALTER TABLE College\_Master ADD CONSTRAINT PK\_College\_ID

PRIMARY KEY CLUSTERED (College\_ID);

--primary xml index

CREATE PRIMARY XML INDEX IDX\_College\_Details ON College\_Master(College\_Details)

GO

-- below query will display the columns in the primary XML index (node table)

SELECT \* FROM sys.columns c

JOIN sys.indexes i ON i.object\_id = c.object\_id

WHERE i.name = 'IDX\_College\_Details'

AND i.type = 1

--create secondary xml index

CREATE XML INDEX IDX\_SEC\_College\_Details ON College\_Master(College\_Details)

USING XML INDEX IDX\_College\_Details FOR PATH

--check execution plan of FLWOR expression example query

SELECT College\_Details.query('for $i in /STUDENTINFO/student

let $j:= $i

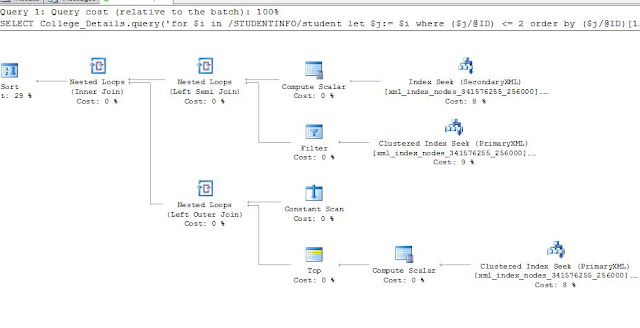
where ($j/@ID) <= 2

order by ($j/@ID)[1] ascending

return $j')

FROM College\_Master

WHERE College\_ID = 1;

[](http://2.bp.blogspot.com/-ZhL0txaFwL4/UEi9guUhcrI/AAAAAAAAAHQ/aRXjwWZMHao/s1600/xmlseek.JPG)