

XML 배우기

1) <https://www.w3.org/TR/xml>

2) <https://www.iro.umontreal.ca/~lapalme/ForestInsteadOfTheTrees/HTML/ch01.html>

I. 들어가기

XML has been developed to facilitate the annotation of information to be shared between computer systems

XML은 데이터를 저장하고 전달할 목적으로 만들어졌으며, 저장되는 데이터의 구조를 기술하기 위한 언어입니다.

XML은 EXtensible Markup Language의 약자로, 수많은 응용 분야에서 데이터를 저장하고 전달하는 중요한 역할을 맡고 있습니다.

1. XML 선언

XML 문서는 맨 첫 줄에 <xml>태그를 사용하여 XML 문서임을 명시해야 합니다.

이것을 XML 프롤로그(prolog)라고 하며, 이때 사용되는 <xml>태그의 이름은 소문자 xml로만 사용해야 합니다.

XML 프롤로그의 문법은 다음과 같습니다.

```
<?xml version="XML문서버전" encoding="문자셋" standalone="yes|no"?>
```

=version 속성에는 XML 문서에 사용된 XML의 버전을 명시합니다.

=encoding 속성에는 XML 문서의 문자셋(character set)을 명시하며, 기본값은 UTF-8로 설정됩니다.

=standalone 속성은 XML 문서가 외부 DTD(Document Type Definition)와 같은 외부 소스의 데이터에 의존하고 있는 문서인지 아닌지를 XML 파서(parser)에 알려주는 역할을 합니다.

=이 속성의 기본값은 no이며, yes로 설정하면 이 문서를 파싱(parsing)할 때 참조해야 할 외부 소스가 없다는 것을 의미합니다.

예제: book.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!-- xml doc 임을 선언 -->
```

```
<?xml-stylesheet type="text/xsl" href="book.xsl"?>
```

```
<!-- xml doc의 stylesheet doc.를 참조한다 -->
```

```
<!-- 이 statement가 있어야 xml doc를 view할 수 있다 -->
```

```
<bookstore> <!-- root element 이다 -->
```

```
<book category="cooking">
```

```
<!-- 1st parent element, 그것의 속성 category는 문자값 cooking을 -->
```

```
<title lang="en">Everyday Italian</title>
```

```
<!-- title은 children element, 그것의 속성 lang은 문자값 en을 -->
```

```
<author>Giada De Laurentiis</author>
```

```
<year>2005</year>
```

```
<price>30.00</price>
```

```
</book>
```

```
<book category="children">
```

```
<!-- 2nd parent element, 그것의 속성 category는 문자값 children을 -->
```

```
<title lang="en">Harry Potter</title>
```

```
<author>J K. Rowling</author>
```

```
<year>2005</year>
```

```
<price>29.99</price>
```

```
</book>
```

```
</bookstore>
```

2. XML 문법: HTML과의 차이

1. 모든 XML 요소는 종료 태그를 가져야 합니다.
2. XML 태그는 대소문자를 구분합니다.
3. XML에서 속성값은 반드시 따옴표로 감싸야 합니다
4. XML에서는 띄어쓰기를 인식합니다.

3. XML 네임스페이스(namespace)

XML 네임스페이스는 XML 요소 간의 이름에 대한 충돌을 방지해 주는 방법을 제공합니다.

XML 네임스페이스는 요소의 이름과 속성의 이름을 하나의 그룹으로 묶어주어 이름에 대한 충돌을 해결합니다.

4. element name: xmlns:prefix="URI">

XML 네임스페이스의 선언은 xmlns나 xmlns:로 시작합니다

XML 네임스페이스는 URI(Uniform Resource Identifiers)로 식별됩니다.

예제:

<root

xmlns:a="https://www.w3.org/TR/html5/"

xmlns:b="http://codingsam.com/xml/physical/">

<a:body>

<a:h1>html에서의 제목</a:h1>

<a:p>html에서의 단락</a:p>

</a:body>

<b:body>

<b:arm>70</b:arm>

<b:leg>110</b:leg>

</b:body>

</root>

II. XML schema의 두 가지

1. DTD : 일반적인 문서 타입 정의(document type definition)
2. XML 스키마(XSD)

1. DTD

문서 타입 정의(DTD)는 XML 문서의 구조 및 해당 문서에서 사용할 수 있는 적절한 요소와 속성을 정의합니다.

DTD를 사용하여 새로운 XML 문서의 구조를 정의함으로써 새로운 문서 타입을 만들 수 있습니다.

<!DOCTYPE 루트요소 DTD식별자 [선언1 선언2 ...]>

=DTD는 <!DOCTYPE 으로 시작합니다.

=루트(root) 요소는 XML 파서(parser)에 명시된 루트 요소부터 파싱(parsing)을 시작하라고 알려주는 역할을 합니다.

=DTD 식별자는 프로그램 외부에 존재하는 DTD 파일을 위한 식별자입니다.

만약에 DTD 식별자가 외부 주소를 가리키고 있으면, 그것을 외부 서브셋(subset)이라고 합니다.

괄호([]) 안에는 내부 서브셋(subset)이라 불리는 추가로 선언한 엔티티(entity)의 리스트가 존재합니다.

DTD 내부 서브셋(subset):

DTD가 XML 파일 내부에서 선언되면, 그 선언은 반드시 <!DOCTYPE>안에 위치해야 합니다.

예제

<food.dtd>

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<!DOCTYPE food [

<!ELEMENT food (name,type,cost)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT type (#PCDATA)>

<!ELEMENT cost (#PCDATA)>

<food>

 <name>상추</name>

 <type>야채</type>

 <cost>2000</cost>

</food>

DTD 외부 서브셋(subset)

DTD가 XML 파일 외부에서 선언되면, <!DOCTYPE>은 반드시 외부 DTD 파일의 주소 정보를 포함해야 합니다.

이러한 외부 DTD 파일은 .dtd 확장자를 사용하여 저장합니다.

예제: data.xml

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE food SYSTEM "food.dtd">

```
<food>
  <name>상추</name>
  <type>야채</type>
  <cost>2000</cost>
</food>
```

예제: food.dtd

```
<!ELEMENT food (name,type,cost)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT cost (#PCDATA)>
```

DTD에서 요소의 콘텐츠에는 PCDATA, 그리고 속성의 속성값으로는 CDATA만 사용.

DTD에서 엔티티는 XML 문서나 DTD 내부에 선언할 수도 있으며, 파일 외부에 선언할 수도 있습니다.

내부 엔티티 선언

```
<!ENTITY 엔티티이름 "엔티티값">
```

이렇게 선언된 내부 엔티티는 XML 문서에서 AND기호(&) + 엔티티 이름 + 세미콜론(:)의 형식으로 사용합니다.

DTD 예제

```
<!ENTITY css "Cascading Style Sheets">
```

XML 예제

```
<lecture>&css;</lecture> // <lecture>Cascading Style Sheets</lecture>
```

외부 엔티티 선언

```
<!ENTITY 엔티티이름 SYSTEM "URI또는URL">
```

이렇게 선언된 외부 엔티티는 XML 문서에서 AND기호(&) + 엔티티 이름 + 세미콜론(:)의 형식으로 사용합니다.

DTD 예제

```
<!ENTITY html SYSTEM "http://codingsam.com/xml/html.dtd">
```

XML 예제

```
<lecture>&html;</lecture> // <lecture>HyperText Markup Language</lecture>
```

파라미터 엔티티의 선언

DTD 문서에서만 사용하기 위해 선언한 엔티티를 파라미터 엔티티라고 합니다.

DTD 내부에 선언되는 파라미터 엔티티는 다음과 같은 문법으로 선언할 수 있습니다.

```
<!ENTITY %엔티티이름 "엔티티값">
```

파라미터 엔티티는 엔티티 이름 앞에 퍼센트(%) 기호가 들어가는 것을 제외하면, 일반 엔티티와 같은 방법으로 사용됩니다.

2. XSD 개요

XML Schema describes the structure of an XML document.

XML Schema language is also referred to as XML Schema Definition (XSD).

XSD는 XML 스키마 정의(XML Schema Definition)를 의미합니다.

XSD는 XML 문서의 구조 및 해당 문서가 포함할 수 있는 적절한 요소와 속성을 명시합니다.

XML에서 스키마를 정의할 때는 XSD뿐만 아니라 앞서 배운 DTD를 사용할 수도 있습니다.

XSD 요소

XSD 문법

모든 XSD 문서의 루트 요소는 <schema>요소입니다.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://codingsam.com"
  xmlns="http://codingsam.com"
  elementFormDefault="qualified">

  ...

</xs:schema>
```

=xmlns:xs 속성은 XSD의 요소와 타입에 사용할 W3C의 XML 스키마 네임스페이스.
=targetNamespace 속성은 요소를 정의할 XML 스키마 네임스페이스.
=xmlns 속성은 기본 XML 스키마 네임스페이스.
=elementFormDefault 속성은 해당 스키마를 이용해 선언한 XML 문서의 모든 요소가 네임스페이스를 만족한다는 것을 명시합니다.

다음 예제는 간단한 1) .xml 2) .dtd 3) .xsd 파일 예제입니다.

1) 예제: food.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<food>
  <name>상추</name>
  <type>야채</type>
  <cost>2000</cost>
</food>
```

2) food.xml 문서의 요소들을 정의한 DTD 파일 예제입니다.

예제: food.dtd

```
<!ELEMENT food (name,type,cost)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT cost (#PCDATA)>
```

3) food.xml 문서의 요소들을 정의한 XSD 파일 예제입니다.

예제: food.xsd

```
<!--xml 선언부 -->
<?xml version="1.0" encoding="UTF-8" ?>

<!--schema 선언부 시작 -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  <!-- schema 요소는 모든 xml 요소의 root 요소이다. 이 스키마에서 사용된 모든 요소와
  data types는 http://www.w3.org/2001/XMLSchema“ namespace에서 온 것이다.-->
  <!-- 이 namespace에서 온 요소와 data types는 xs:라는 접두사를 가져야 한다 -->

  targetNamespace="http://codingsam.com"
  <!-- (note, to, from, heading, body)로 정의된 요소들은 “https://www.w3schools.com”
  namespace에서 온 것이다 -->
```

```

    xmlns="http://codingsam.com"
<!-- default namespace 이다 -->

    elementFormDefault="qualified">
<!-- 이 스키마에서 선언된 XML instance document에서 사용된 어떠한 요소도 자격이
있는 namespace이어야 한다 -->

<!-- schema 선언부 끝 -->

<xs:element name="food">
<!-- 이름이 note인 부모 요소이다 -->

    <xs:complexType>
<!-- note element는 to, from, heading, body elements를 가지고 있으므로 complex
type이다 -->
<!-- simple element: 단지 text만 포함할 수 있고, 어떠한 다른 요소나 속성을 갖지 못한다
-->

        <xs:sequence>
<!-- sequence indicator: child elements는 선언된 순서에 따라 나타난다 -->

            <xs:element name="name" type="xs:string"/ default="red">
<!-- data types: xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time -->
<!-- 속성에도 적용된다: xs:attribute name+"lang" type="xs:string" fixed="EN" -->
<!-- default value는 다른 어떤 값이 정해지지 않았을 때 자동적으로 요소에 할당된다. -->

                <xs:element name="type" type="xs:string" fixed="red"/>
<!-- fixed value 역시 요소에 자동적으로 요소에 할당되며 다른 값을 지정할 수 없다 -->

                    <xs:element name="cost" type="xs:string"/>

            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>

```

1+2)외부 DTD 파일을 포함하는 XML 파일 예제입니다.

예제: food.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
```



```
<!DOCTYPE food SYSTEM "food.dtd">
<food>
  <name>상추</name>
  <type>야채</type>
  <cost>2000</cost>
</food>
```

1+3) 외부 XSD 파일을 포함하는 XML 파일 예제입니다.

예제: food.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<food
xmlns="http://codingsam.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<!-- XMLSchema-instance namespace를 이용할 경우: xsi 접두사 사용 -->
  <!-- instance document란 데이터를 포함하고 있는 xml 파일이다 -->

  xsi:schemaLocation="http://codingsam.com food.xsd">
  <!-- xsi:schemaLocation attribute은 space로 분리된 두 개의 값을 갖는다: 1) 사용할
namespace, 2) 그 namespace용으로 사용하기 위한 XML schema의 위치 -->
  <!-- xsi:noNamespaceSchemaLocation="example.xsd"도 xsi의 또 다른 속성임 -->
    <name>상추</name>
    <type>야채</type>
    <cost>2000</cost>
  </food>
```

XSD 기본 타입(primitive datatype)

네임스페이스 선언

사용한 접두사를 붙여야 합니다.

```
<xsi:element name="player" type="xs:string"/>
```

문자열(string)

문자열 타입은 문자, 줄 바꿈 문자(line feed), 행 복귀 문자(carriage return), 탭 문자(tab) 등을 포함할 수 있습니다.

```
<xsi:element name="student" type="xs:string"/>
```

위의 XSD 예제에서 정의한 요소는 XML 문서에서 다음과 같이 표현될 수 있습니다.

날짜와 시간(dateTime)

yyyy-mm-ddThh:mm:ss

- yyyy-mm-dd : yyyy년 mm월 dd일을 나타냅니다.
- T : 시간 부분이 시작됨을 알려주며, 반드시 표기해야 하는 문자입니다.
- hh:mm:ss : hh시 mm분 ss초를 나타냅니다.

<xs:element name="deadline" type="xs:dateTime"/>

XML 예제

<deadline>2017-07-01T09:00:00</deadline>

실수(decimal)

<xs:element name="answer" type="xs:decimal"/>

XML 예제

<answer>12.5</answer>

불리언(boolean)

<xs:attribute name="disabled" type="xs:boolean"/>

XML 예제

<rank disabled="true">10</rank>

anyURI: anyURI 타입은 통합 자원 식별자(URI)를 표현할 수 있습니다.

XSD 예제

<xs:attribute name="src" type="xs:anyURI"/>

XML 예제

<image src="http://codingsam.com/xml/images/uri.png" />

단순 타입 요소의 선언

문법:

```
<xs:element name="요소이름" type="요소타입"/>
```

단순 타입 요소를 XSD에서 어떻게 선언하는지를 보여주는 예제

XML 예제

```
<player>홍길동</player>
```

```
<rank>24</rank>
```

```
<goal>13:15:00</goal>
```

XSD 예제

```
<xs:element name="player" type="xs:string"/> // 문자열 타입
```

```
<xs:element name="rank" type="xs:integer"/> // 숫자 타입
```

```
<xs:element name="goal" type="xs:time"/> // 시간 타입
```

XSD 속성

단순 타입(simple type) 요소는 속성을 가질 수 없습니다.

만약에 단순 타입의 요소가 속성을 가지게 되면, 그 요소는 복합 타입(complex type)이 됩니다.

복합 타입(complex type) 요소란 자식 요소나 속성을 포함하는 요소를 의미합니다.

XSD에서 복합 타입 요소는 다음과 같이 구분할 수 있습니다.

1. 빈 요소
2. 자식 요소만을 포함하는 요소
3. 텍스트만을 포함하는 요소
4. 자식 요소와 텍스트를 모두 포함하는 요소

하지만 속성(attribute) 그 자체는 언제나 단순 타입(simple type)으로 선언됩니다.

속성의 선언

문법

```
<xs:attribute name="속성이름" type="속성타입"/>
```

XML 예제

```
<priority rating="3">middle</priority>
```

XSD 예제

```
<xs:attribute name="rating" type="xs:integer"/>
```

빈 요소의 선언

XSD에서 빈 요소(empty element)란 자식 요소는 가지지 않고, 오직 속성만을 가지는 요소를 의미합니다. 빈 요소는 복합 타입 요소이므로, 반드시 하나 이상의 속성을 가지고 있어야 합니다.

XML 예제

```
<red redValue="139" />
```

위의 XML 예제는 XSD에서 아래처럼 선언됩니다.

XSD 예제

```
<xs:element name="red">
  <xs:complexType>
    <xs:attribute name="redValue" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

자식 요소만을 포함하는 요소

XSD에서 가장 일반적으로 정의되는 요소 중의 하나가 바로 자식 요소만을 포함하는 요소입니다.

XML 예제

```
<physical>
  <height>180</height>
  <waist>32</waist>
</physical>
```

XSD 예제

```
<xs:element name="physical">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="height" type="xs:integer"/>
      <xs:element name="waist" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

위의 예제에서 sequence 지시자는 body 타입의 요소는 <height>요소와 <waist>요소 순으로 자식 요소를 가져야 합니다. sequence 지시자는 자식 요소가 명시된 순서대로만 나타낼 수 있다는 것을 명시하는 지시자입니다.

텍스트만을 포함하는 요소

XSD에서 텍스트만을 포함하는 요소는 <simpleContent>요소를 사용하여 선언합니다.

이때 <simpleContent>요소 안에는 <extension>요소나 <restriction>요소 중 하나가 반드시 선언되어야 합니다.

그래야만 <simpleContent>요소를 사용하여 기초가 되는 단순 타입 요소를 extension하거나 restriction할 수 있기 때문입니다.

XML 예제

```
<currency country="kor">10000</currency>
```

위의 XML 예제는 XSD에서 아래처럼 선언됩니다.

XSD 예제

```
<xs:element name="currency">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

자식 요소와 텍스트를 모두 포함하는 요소

XSD에서는 자식 요소, 속성 그리고 텍스트까지 모두 포함한 복합 타입 요소를 선언할 수 있습니다.

XML 예제

```
<student>
  학생의 이름은 <name>홍길동</name>이고,
  성별은 <gender>남자</gender>이며,
  나이는 <age>15</age>살 입니다.
</student>
```

위의 XML 예제는 XSD에서 아래처럼 선언됩니다.

XSD 예제

```
<xs:element name="student">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="gender" type="xs:string"/>
      <xs:element name="age" type="xs:positiveInteger"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

위의 XML 예제에서 자식 요소들 사이에 나타나는 텍스트를 표현하기 위해서는 반드시 XSD에서 mixed 속성값을 true로 설정해야 합니다.

임의의 요소 및 속성 사용

<any>요소는 해당 XSD 파일에서 선언되지 않은 요소를 사용하여 XML 문서를 확장할 수 있게 해줍니다. 또한, **<anyAttribute>요소**는 해당 XSD 파일에서 선언되지 않은 속성을 사용하여 XML 문서를 확장할 수 있도록 도와줍니다. 이러한 <any>요소와 <anyAttribute>요소는 XML 문서의 확장성을 더욱 좋게 만들어 줍니다.

다음 예제에서 student.xsd를 만족하는 student 요소는 세 번째 요소로 어떤 요소가 나와도 되며, 또한 세 번째 요소가 나오지 않아도 됩니다.

예제: student.xsd

```
<xs:element name="student">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="gender" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

예제: address.xsd

```
<xs:element name="address">
  <xs:complexType>
    <xs:sequence>
```

```

        <xs:element name="zipcode" type="xs:integer"/>
        <xs:element name="si" type="xs:string"/>
        <xs:element name="gu" type="xs:string"/>
        <xs:element name="street" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
</xs:element>

```

다음 XML 예제들은 모두 위에서 살펴본 student.xsd를 만족하는 XML 예제입니다.

예제: student.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<student
xmlns="http://codingsam.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://codingsam.com student.xsd"
xsi:schemaLocation="http://codingsam.com address.xsd"
>
    <name>길동</name>
    <gender>남자</gender>
    <address>
        <zipcode>12345</zipcode>
        <si>서울시</si>
        <gu>강남구</gu>
        <street>테헤란로</street>
    </address>
</student>

```

3. XSL(eXtensible Stylesheet Language)

XSLT is a language for transforming XML documents.

XPath is a language for navigating in XML documents.

XQuery is a language for querying XML documents.

CSS가 HTML 문서를 위한 스타일 시트 언어라면, XSL은 XML 문서를 위한 스타일 시트 언어입니다.

XSL의 구성

1. XSLT : XSL Transformations를 의미하며, XML 문서를 다른 구조의 문서로 변환시키기 위한 언어입니다.

2. XPath : XML 문서의 특정 요소나 속성에 접근하기 위한 경로를 지정하는 언어입니다.

선언: 루트 요소

XSLT 문서의 루트(root) 요소는 <xsl:stylesheet>요소나 <xsl:transform>요소로 표현할 수 있습니다. 또한, 루트 요소의 시작 태그에는 W3C XSLT 네임스페이스를 반드시 같이 명시해 줘야 합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
...
</xsl:stylesheet>
```

xml 다큐의 변환 단계

1) raw.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <cd>
    ....
  </cd>
</catalog>
```

2) raw.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    ....
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

3) .xsl 문서의 URI가 포함된 1)의 .xml

```
<?xml version="1.0" encoding="UTF-8"?>
```



```
<?xml-stylesheet type="text/xsl" href="raw.xsl"?>
<catalog>
  <cd>
    .....
  </cd>
</catalog>
```

템플릿

1) <xsl:template> element

XSLT 프로세서가 XSLT 문서에서 가장 먼저 찾는 요소는 바로 템플릿 요소입니다. 템플릿(template)은 특정 노드가 일치할 때 해당 노드에 적용할 규칙들을 포함하고 있습니다. 이러한 템플릿은 루트 노드에 포함된 콘텐츠를 어떻게 처리하고 변환할 것인지를 나타냅니다. XSLT 문서에서 템플릿(template)은 <xsl:template>요소로 표현할 수 있습니다. match 속성에는 속성값으로 템플릿과 연결할 XML 요소의 범위를 나타내는 XPath 표현식을 명시합니다.

```
<xsl:template match="/">  <!-- xsl의 규칙을 다큐 전체에 적용하라는 의미 -->
```

2) <xsl:value-of> element

선택된 노드의 값을 추출하기 위해 사용

```
.
.
.
<xsl:value-of select="catalog/cd/title"/>
<p/>
<xsl:value-of select="catalog/cd/artist"/>
.
.
.
```

3) <xsl:for-each> element

지정한 노드 세트의 모든 xml 요소들을 선택하는데 사용.

```
.
.
<xsl:for-each select="catalog/cd">
  <xsl:value-of select="catalog/cd/title"/>
<p/>
```

```

        <xsl:value-of select="catalog/cd/artist"/>
    </xsl:for-each>
    .
    .
    .

```

Filtering the Output

We can also filter the output from the XML file by adding a criterion to the select attribute in the <xsl:for-each> element.

```
<xsl:for-each select="catalog/cd[artist='Bob Dylan']">
```

Legal filter operators are:

```

= (equal)
!= (not equal)
< less than
> greater than

```

4) <xsl:sort> element

Be used to sort the output.

```

    .
    .
    <xsl:for-each select="catalog/cd">
        <xsl:sort select="artist"/>
        <xsl:value-of select="catalog/cd/title"/>
        <p/>
        <xsl:value-of select="catalog/cd/artist"/>
    </xsl:for-each>
    .
    .
    .

```

5) <xsl:if> element

Be used to put a conditional test against the content of the XML file.

```

<xsl:for-each select="catalog/cd">
    <xsl:if test="price > 10">
        <xsl:value-of select="price"/>
    </xsl:if>
</xsl:for-each>

```

6) <xsl:choose> element

Be used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests.

```
<xsl:for-each select="catalog/cd">
  <xsl:value-of select="title"/>
  <xsl:choose>
    <xsl:when test="price > 10">
      <xsl:value-of select="artist"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="artist"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:for-each>
```

5. XQuery

XQuery is to XML what SQL is to databases.

XQuery is designed to query XML data.

```
<예>
  for $x in doc("book.xml")/bookstore/book
  where $x/price>20
  order by $x/title
  return $x/title
```

FLWOR (pronounced "flower"): "For, Let, Where, Order by, Return".

For - selects a sequence of nodes

Let - binds a sequence to a variable

Where - filters the nodes

Order by - sorts the nodes

Return - what to return (gets evaluated once for every node)

End!!!