Transformation Languages

http://goo.gl/8e4OJS

CS5200 DBMS
Bruce Chhay

Transformation Languages

L1. XPath and XSLT

XML

- XML: Extensible Markup Language.
- Document object model:
  - Interpreted as a tree of node elements.
  - Tags, attributes, values.
  - Format for human readable and machine readable arbitrary data structures.

```
<?xml version="1.0" encoding="UTF-8"?>
<BlogApplication>
  <BlogUsers>
    <BlogUser date="2016-04-02">
      <UserName>username1</UserName>
      <FirstName>First1</FirstName>
      <LastName>Last1</LastName>
    </BlogUser>
    ...
  </BlogUsers>
  <BlogPosts>
    <BlogPost>
      <PostId>1</PostId>
      <UserName>username1</UserName>
      <Title>title1</Title>
      <Content>content1</Content>
      <Comments>
        <Comment>
          <CommentId>1</CommentId>
          <UserName>username4</UserName>
          <Content>comment1</Content>
        </Comment>
        <Comment>
          <CommentId>2</CommentId>
          <UserName>username4</UserName>
          <Content>comment2</Content>
        </Comment>
      </Comments>
    </BlogPost>
    ...
  </BlogPosts>
</BlogApplication>
```

XML

- XML: Extensible Markup Language.
- Document object model:
  - Interpreted as a tree of node elements.
  - Tags, attributes, values.
  - Format for human readable and machine readable arbitrary data structures.

```
<?xml version="1.0" encoding="UTF-8"?>
<BlogApplication>
  <BlogUsers>
    <BlogUser date="2016-04-02">
      <UserName>username1</UserName>
      <FirstName>First1</FirstName>
      <LastName>Last1</LastName>
    </BlogUser>
    ...
  </BlogUsers>
  <BlogPosts>
    <BlogPost>
      <PostId>1</PostId>
      <UserName>username1</UserName>
      <Title>title1</Title>
      <Content>content1</Content>
      <Comments>
        <Comment>
          <CommentId>1</CommentId>
          <UserName>username4</UserName>
          <Content>comment1</Content>
        </Comment>
        <Comment>
          <CommentId>2</CommentId>
          <UserName>username4</UserName>
          <Content>comment2</Content>
        </Comment>
      </Comments>
    </BlogPost>
    ...
  </BlogPosts>
</BlogApplication>
```

XML

- XML: Extensible Markup Language.
- Document object model:
  - Interpreted as a tree of node elements.
  - Tags, attributes, values.
  - Format for human readable and machine readable arbitrary data structures.

```
<?xml version="1.0" encoding="UTF-8"?>
<BlogApplication>
  <BlogUsers>
    <BlogUser date="2015-04-02">
      <UserName>username1</UserName>
      <FirstName>First1</FirstName>
      <LastName>Last1</LastName>
    </BlogUser>
    ...
  </BlogUsers>
  <BlogPosts>
    <BlogPost>
      <PostId>1</PostId>
      <UserName>username1</UserName>
      <Title>title1</Title>
      <Content>content1</Content>
      <Comments>
        <Comment>
          <CommentId>1</CommentId>
          <UserName>username4</UserName>
          <Content>comment1</Content>
        </Comment>
        <Comment>
          <CommentId>2</CommentId>
          <UserName>username4</UserName>
          <Content>comment2</Content>
        </Comment>
      </Comments>
    </BlogPost>
    ...
  </BlogPosts>
</BlogApplication>
```

XML

- XML: Extensible Markup Language.
- Document object model:
  - Interpreted as a tree of node elements.
  - Tags, attributes, values.
  - Format for human readable and machine readable arbitrary data structures.

```
<?xml version="1.0" encoding="UTF-8"?>
<BlogApplication>
  <BlogUsers>
    <BlogUser date="2015-04-02">
      <UserName>username1</UserName>
      <FirstName>First1</FirstName>
      <LastName>Last1</LastName>
    </BlogUser>
    ...
  </BlogUsers>
  <BlogPosts>
    <BlogPost>
      <PostId>1</PostId>
      <UserName>username1</UserName>
      <Title>title1</Title>
      <Content>content1</Content>
      <Comments>
        <Comment>
          <CommentId>1</CommentId>
          <UserName>username4</UserName>
          <Content>comment1</Content>
        </Comment>
        <Comment>
          <CommentId>2</CommentId>
          <UserName>username4</UserName>
          <Content>comment2</Content>
        </Comment>
      </Comments>
    </BlogPost>
    ...
  </BlogPosts>
</BlogApplication>
```

XML

- XML: Extensible Markup Language.
- Document object model:
  - Interpreted as a tree of node elements.
  - Tags, attributes, values.
  - Format for human readable and machine readable arbitrary data structures.

```
<?xml version="1.0" encoding="UTF-8"?>
<BlogApplication>
  <BlogUsers>
    <BlogUser date="2015-04-02">
      <UserName>username1</UserName>
      <FirstName>First1</FirstName>
      <LastName>Last1</LastName>
    </BlogUser>
    ...
  </BlogUsers>
  <BlogPosts>
    <BlogPost>
      <PostId>1</PostId>
      <UserName>username1</UserName>
      <Title>title1</Title>
      <Content>content1</Content>
      <Comments>
        <Comment>
          <CommentId>1</CommentId>
          <UserName>username4</UserName>
          <Content>comment1</Content>
        </Comment>
        <Comment>
          <CommentId>2</CommentId>
          <UserName>username4</UserName>
          <Content>comment2</Content>
        </Comment>
      </Comments>
    </BlogPost>
    ...
  </BlogPosts>
</BlogApplication>
```

XPath

- XPath: expression language for selecting nodes and values in an XML doc.
- XPath 2.0: http://www.w3.org/TR/xpath20/

XPath Syntax (Selecting)

Expression	Example	Description
/	/	Root node
path/to/node	BlogUsers/BlogUser/UserName	Descend into the specified node path to select UserName nodes.
path//node	BlogUsers//UserName	Recursively descend into path to select all UserName nodes under BlogUsers.
*	BlogUsers/*	Wildcard for nodes and attributes. Selects all child elements under BlogUsers.
|	path/a | path/b	“OR” condition for using multiple expressions.
.		Select current node.
..		Select parent node.

XPath Syntax (Predicate/Filtering)

Expression	Examples	Description
+ - * div mod		Basic math operators.
< > <= >= = != and or ()		Boolean operators.
[]		Filtering or array. (See below.)
node/text()	BlogPosts/BlogPost[UserName/ text()='username1']/PostId	Return PostId nodes where the text value of the BlogPosts/BlogPost/UserName nodes are “username1”.
arrays	BlogUsers/BlogUser[2]/UserName BlogUser[last()] BlogUser[position() < 6] BlogPosts/BlogPost[Comments/ count(Comment) > 1]	Position index. Return UserName node within second BlogUser node. Return last BlogUser node. Return first five BlogUser nodes. Return BlogPost nodes with more than one comment.
@attribute	BlogUser[@date] BlogUser[@date="2015-04-02"]	Matches the specified attribute. Return all BlogUser nodes containing attr. Return all BlogUser nodes where the ‘date’ attribute value is "2015-04-02".

XSLT

- XSLT: Extensible Stylesheet Language Transformations.
- XSLT 2.0: http://www.w3.org/TR/xslt20/
- Language for transforming XML docs to other XML docs.
- Declarative. Based on pattern matching.
- Processing flow:
  Given an XSLT template, transform input XML docs into output XML docs.
  Input XML docs parsed as a tree, and then XSLT declaration, including XPath patterns, is used to process the tree and output the result.
- Uses:
  XML -> XML
  (DB,API,... -> XML) XML -> HTML
  Java library http://docs.oracle.com/javase/7/docs/api/javax/xml/transform/package-summary.html

XSL Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Root element of a stylesheet: Define namespaces (e.g. xls), define version (2.0 is current), optionally exclude namespaces. -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" exclude-result-prefixes="xs" version="2.0">
    <!-- Output format, for example xml, html, or text. -->
    <xsl:output method="html" indent="yes"></xsl:output>
    <!-- Template to apply when "match" pattern is encountered. The "match" pattern is an XPath expression. A single stylesheet can contain multiple templates. Within a template, you can apply the other templates to the current element (and its child nodes) with: <xsl:apply-templates select="sub-template-pattern"/> -->
    <xsl:template match="/BlogApplication">
        <!-- Within a template, you can use a mix of HTML and XSL. The "select" patterns are XPath expressions. -->
        <xsl:for-each select="BlogUsers/BlogUser">
            ...
        </xsl:for-each>
        ...
    </xsl:template>
</xsl:stylesheet>
```

XSL Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Root element of a stylesheet: Define namespaces (e.g. xls), define version (2.0 is current), optionally exclude namespaces. -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" exclude-result-prefixes="xs" version="2.0">
    <!-- Output format, for example xml, html, or text. -->
    <xsl:output method="html" indent="yes"></xsl:output>
    <!-- Template to apply when "match" pattern is encountered. The "match" pattern is an XPath expression. A single stylesheet can contain multiple templates. Within a template, you can apply the other templates to the current element (and its child nodes) with: <xsl:apply-templates select="sub-template-pattern"/> -->
    <xsl:template match="/BlogApplication">
        <!-- Within a template, you can use a mix of HTML and XSL. The "select" patterns are XPath expressions. -->
        <xsl:for-each select="BlogUsers/BlogUser">
            ...
        </xsl:for-each>
        ...
    </xsl:template>
</xsl:stylesheet>
```

XSL Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Root element of a stylesheet: Define namespaces (e.g. xls), define version (2.0 is current), optionally exclude namespaces. -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" exclude-result-prefixes="xs" version="2.0">
    <!-- Output format, for example xml, html, or text. -->
    <xsl:output method="html" indent="yes"></xsl:output>
    <!-- Template to apply when "match" pattern is encountered. The "match" pattern is an XPath expression. A single stylesheet can contain multiple templates. Within a template, you can apply the other templates to the current element (and its child nodes) with: <xsl:apply-templates select="sub-template-pattern"/> -->
    <xsl:template match="/BlogApplication">
        <!-- Within a template, you can use a mix of HTML and XSL. The "select" patterns are XPath expressions. -->
        <xsl:for-each select="BlogUsers/BlogUser">
            ...
        </xsl:for-each>
        ...
    </xsl:template>
</xsl:stylesheet>
```

XSL Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Root element of a stylesheet: Define namespaces (e.g. xls), define version (2.0 is current), optionally exclude namespaces. -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" exclude-result-prefixes="xs" version="2.0">
    <!-- Output format, for example xml, html, or text. -->
    <xsl:output method="html" indent="yes"></xsl:output>
    <!-- Template to apply when "match" pattern is encountered. The "match" pattern is an XPath expression. A single stylesheet can
contain multiple templates. Within a template, you can apply the other templates to the current element (and its child nodes) with:
<xsl:apply-templates select="sub-template-pattern"/> -->
    <xsl:template match="/BlogApplication">
        <!-- Within a template, you can use a mix of HTML and XSL. The "select" patterns are XPath expressions. -->
        <xsl:for-each select="BlogUsers/BlogUser">
            ...
        </xsl:for-each>
        ...
    </xsl:template>
</xsl:stylesheet>
```

Setup

- Make sure the Java JDK is installed (part of Java SE, needed for Eclipse and Application Development module).
- Install Saxon-HE: www.saxonica.com/download/java.xml, eventually.
- Documentation: http://www.saxonica.com/documentation/#!using-xsl/commandline
- Example of running:

  java -cp "saxon-he-12.4.jar" net.sf.saxon.Transform -s:"BlogApplication.xml" -xsl:"BlogApplication.xsl" -o:"results_xsl.html"

- Or run in Eclipse.

  Different transformation engine…
  Java API for XML Parsing (JAXP)

Example

- XML input (BlogApplication.xml): http://goo.gl/rtWpuy
- For each and every BlogUser, list all their BlogPosts.
  XSLT (BlogApplication.xsl): http://goo.gl/UJnU3R
- XML output (results_xsl.html): http://goo.gl/RXmH3r

Exercises

In one table:

1. For each and every BlogUser, map the UserName and FirstName to an array of post titles they commented on (empty if no post comments), and an array of the comment id and comment content.
   Expected output: http://goo.gl/SXcSup
   Solution: http://goo.gl/o8UcWD
2. Now “unroll” the array and flatten the table so that each comment id/content appears in a separate row. Exclude users that have no comments.
   Expected output: http://goo.gl/LuV6cJ
   Solution: http://goo.gl/M0qxhT
3. Same as #2, except include all users, even those with no comments.
   Expected output: https://goo.gl/GkzMS7
   Solution: http://goo.gl/8fgrnb

Transformation Languages

L2. XQuery

XQuery

- Extract data from XML WWW doc by querying and transforming XML data into other formats.
- Can use declarative SQL-like FLWOR expression.
- Uses XPath for pattern matching in XML docs.
- XQuery 1.0: http://www.w3.org/TR/xquery/
- 1.0 is most common, but 3.0 is latest: https://www.w3.org/TR/xquery-3/

FLWOR

FOR: iterate over a sequence.

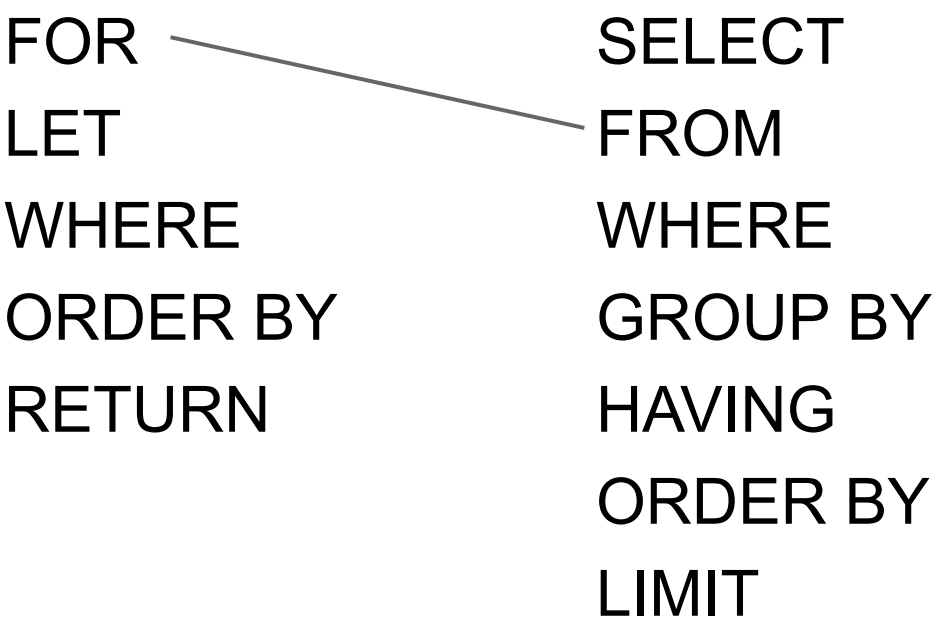
LET: variable declarations.

WHERE: filtering conditions.

ORDER BY: sort order.

RETURN: result output.

FLWOR vs. SELECT



FLWOR vs. SELECT

FOR	SELECT
LET	FROM
WHERE	WHERE
ORDER BY	GROUP BY
RETURN	HAVING
	ORDER BY
	LIMIT

FLWOR vs. SELECT

FOR distinct-values()
LET
WHERE
ORDER BY
RETURN

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT

FLWOR vs. SELECT

FOR
LET
WHERE
ORDER BY
RETURN

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT

FLWOR vs. SELECT

FOR	SELECT
LET	FROM
WHERE	WHERE
ORDER BY	GROUP BY
RETURN	HAVING
	ORDER BY
	LIMIT

FLWOR vs. SELECT

(FOR
LET
WHERE
ORDER BY
RETURN)
[position() = n to m]

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT

Setup

- Make sure the Java JDK is installed (part of Java SE, needed for Eclipse and Application Development module).
- Install Saxon HE: http://saxon.sourceforge.net/.
- Documentation: http://www.saxonica.com/documentation/#!using-xquery/commandline
- Examples of running:
  - Query an input file with query string and output to stdout:
    java -cp "saxon-he-12.4.jar" net.sf.saxon.Query -s:"BlogApplication.xml"
    -qs:"<results>{/BlogApplication/BlogUsers/BlogUser[UserName='username3']/FirstName
    /text()}</results>"
    Good way to test your XPath expressions!
  - Query without input file (input document reference can be defined in .xquery file):
    java -cp "saxon-he-12.4.jar" net.sf.saxon.Query -q:"BlogApplication.xquery"
    -o:"results_xquery.html"

Example

XQuery (BlogApplication.xquery): http://goo.gl/TZVxue

Exercise

- Same output as XSLT Exercise 2, except with XQuery:
  For only BlogUsers with comments (INNER JOIN), map the UserName and FirstName to their comment id/content.
- Solution: http://goo.gl/D8ZLK3
- Try XSLT Exercise 3 in XQuery:
  For all BlogUsers (LEFT OUTER JOIN), map the UserName and FirstName to the comment id/content.
  Hint: Would an additional “for each BlogUser” clause be helpful?

Comparison: XSLT and XQuery

- Overlapping functionality. Intent: XSLT was made to transform docs for rendering and XQuery was for querying docs (like SQL, but for web docs).
- FLWOR is similar to SELECT statement, so easier for us to learn.
- Same data model, both use XPath, similar libraries.
- XSLT has templating and re-use.
- XSLT has a larger community and is further developed. Better support in different programming languages. Also expressed as XML, so can be easily integrated with application stacks. So you should probably use XSLT.