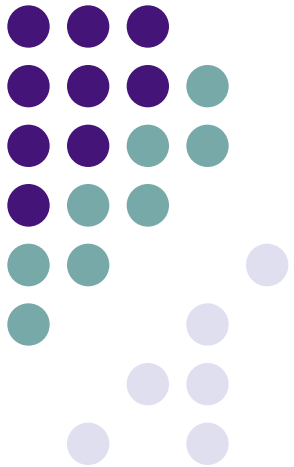
XSLT: XML Transformation



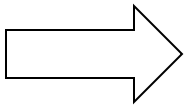
IST 516

Penn State

Dongwon Lee, Ph.D.♪

Cascading Style Sheets

- CSS separates low-level formatting information from HTML



- HTML+CSS

```
<html>
 <head>
    <style type="text/css">
      h1 {color:blue;}
      h1 u {color:red;}
      h1 i {color:green;}
    </style>
  </head>

<body>
    <h1><u>CSS</u>
Example for
<i>IST Class</i></h1>
  </body>
</html>
```

PENNSTATE

Then, What About XML File?

- XML files contain only “data”, not the “formatting” information
- XML itself does NOT do much useful things
  - Just bunch of tags and enclosed text
- Apps do interesting stuff using XML files
- Eg, If data in XML files need:
  - To be stored in RDBMS → transform XML files into relational records and SQL statements
  - To be shown on web browsers → transform XML files into HTML files
  - To be emailed to publisher → transform XML files into PDF or LaTeX files

PENNSTATE

Eg, Showing XML on Browsers?

- How can we control the look of XML files then?

- Step 1: transform XML into HTML
- Step 2: generate CSS code

- Step 1: XSLT
- Step 2: XML-FO
- Both together: XSL

Conceptually, one could think as follows:

CSS = Style Sheets for HTML
XSL = Style Sheets for XML

PENNSTATE

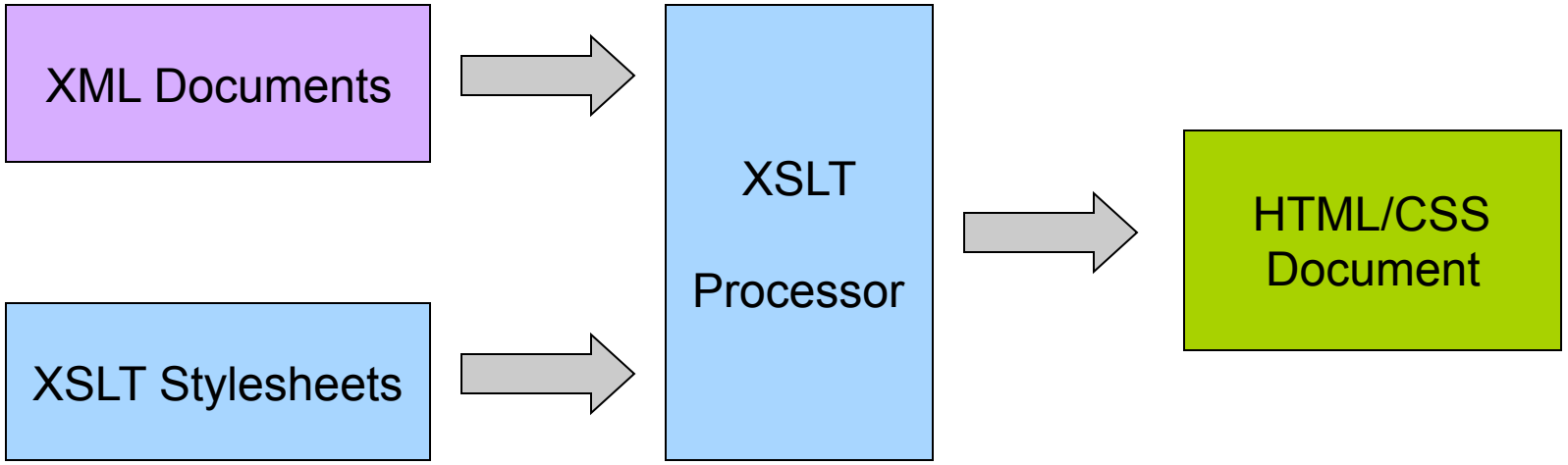
XSL

- eXtensible Stylesheet Language (XSL) consists of:
  - XSLT: "XSL Transformation"
  - XSL-FO: "XSL Formatting Object" (low-level formatting information with HTML+CSS)
  - XPath: to specify the portion of XML document to which the transformation and formatting are applied
- XSLT: the focus of this lecture

PENNSTATE

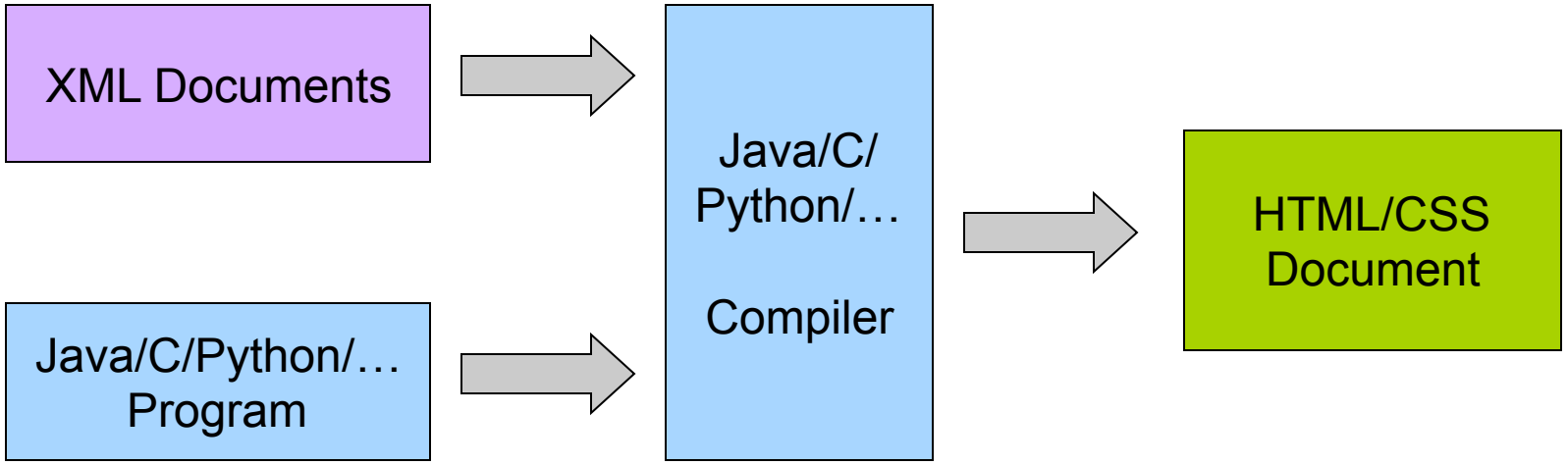
Alternatives of XML Transformation

- XSLT handles “transforming” XML data into something else (eg, HTML, text, SQL, …)



Alternatives of XML Transformation

- Java/C/Perl/Python/etc programs can "transform" XML data into something else too



Difference?

- Using programming languages is
  - More powerful s.t. developers have more fine-grained control
  - More customized transformation
- Using XSLT is
  - More generic and declarative
  - Arguably easier than writing programs
- Analogy
  - XML files ⇔ Relational tables
  - XSLT ⇔ SQL
  - Java code ⇔ JDBC code

PENNSTATE

How Does XSLT Work?

- Using XPath expressions, one defines parts of XML data that match one or more predefined templates
- When a match is found, XSLT will transform the matching part of the XML data into the result output
- XSLT code consists of:
  - XSLT constructs as in <xsl: …> format
  - Everything else (eg, HTML tags, SQL statements, Python codes, …) that one wants to generate

PENNSTATE

How Does XSLT Work?

- Typical Workflow
  - One prepares a source XML data: foo.xml
  - One writes XSLT codes: foo.xsl
  - Link XSLT codes to the XML data by adding one line into the XML data

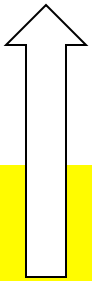
```
<!-- foo.xml -->
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="foo.xml"?>
<foo> … </foo>
```

  - View the XML data with XSLT-capable s/w such as Firefox or IE

XSLT Declaration

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
<xsl:template match="emp"> …
```

emp.xsl: transformation logic goes here



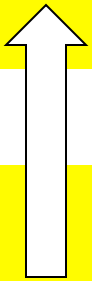
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<emp> … </emp>
```

emp.xml: input XML document about <emp> elements

XSLT Declaration w. Namespace

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl=http://www.w3.org/1999/XSL/Transform
   xmlns:pike="http://pike.psu.edu">
<xsl:template match="pike:emp"> …
```

emp.xsl: transformation logic goes here



```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp.xsl"?>
<emp xmlns="http://pike.psu.edu"> … </emp>
```

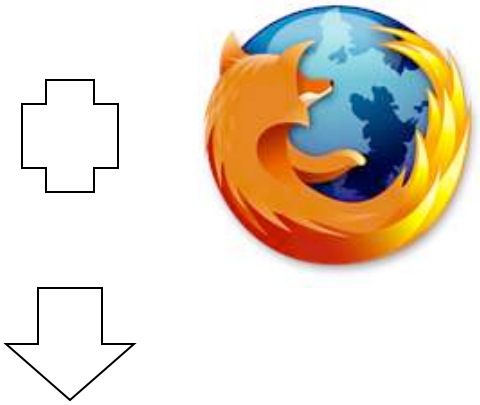
emp.xml: input XML document about <emp> elements

XSLT Processing

- Most modern web browsers have built-in XSLT engine/processor

emp.xml

```
<?xml version="1.0”?>
<?xml-stylesheet type="text/xsl" href=”emp.xsl"?>
<emp> … </emp>
```

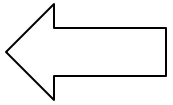


Firefox applies transformation logic in “emp.xsl” to XML data in “emp.xml” file and display the output on the fly

PENNSTATE

Versions

- Influenced by DSSSL--An SGML transformation and styling language
- V 1.0: 1999
  - http://www.w3.org/TR/xslt
  - Still widely used and supported
  - Most modern web browsers support this
  - Uses XPath 1.0
- V 2.0: 2007
  - Latest: http://www.w3.org/TR/xslt20/
  - Users XPath 2.0



Focus of this slide

PENNSTATE

<xsl:> elements

- +100 built-in XSLT elements perform different functions
- Format: <**xsl:** name attr="value">
- Often "value" is
  - XPath expression to return node-set
  - Boolean expression to return TRUE or FALSE
- By convention, tags starting with "<xsl: …> namespace are XSLT commands
- Other regular tags are HTML tags

PENNSTATE

<xsl:output>

<xsl:output method="value"/>

- Specifies the output format of XSLT transformation
- "Value" can be
  - xml
  - html
  - text
- Eg, <xsl:output method="html"/> indicates that the output of the transformation would be HTML file

PENNSTATE

<xsl:text>

<xsl:text> contents </xsl:text>

- “contents” between <xsl:text> and </xsl:text> are verbatim copied to the output
- Attribute: disable-output-escaping="yes|no”
  - Optional
  - “yes" indicates that special characters (like "<") should be output as is
  - "no" indicates that special characters (like "<") should be output as "<”
  - Default is "no".

PENNSTATE

HTML Special Characters

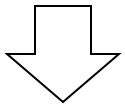
- Some special characters and symbols that cannot be easily generated by keyboards
  - Eg, Non-roman characters: **Düsseldorf**
- One can use HTML entities to enter them
- Eg,
  - & → &
  - < → <
  - > → >
  - á → á

Could be confused With tag notations of HTML and XML

PENNSTATE

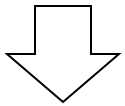
<xsl:text>

```
<xsl:text disable-output-escaping="yes">
        <MyTag>would appear as it is</MyTag>
</xsl:text>
```



<MyTag>would appear as it is</MyTag>

```
<xsl:text disable-output-escaping="no">
      <MyTag>HTML Entities would appear</MyTag>
</xsl:text>
```



<MyTag> HTML Entities would appear</MyTag>

PENNSTATE

<xsl:template>

- XSLT consists of rules, called "templates"
- Each template contains an individual transformation logic to apply when nodes are matched by the given XPath expression
- <xsl:template> builds such templates

```
<xsl:template match="XPath">
        transformation logic here
</xsl:template>
```

  - Eg, <xsl: template **match="/"**> matches the whole document

PENNSTATE

<xsl:value-of>

<xsl:value-of select="XPath"/>

- Extract single value of XML elements that match the given Xpath
  - XPath should return the single value
- Eg, <xsl:value-of select="//cd[1]/title"/> extracts the single value of the "title" element of the first appearing "cd" element

Example: <xsl:template> (zvon.org)

XML Source

```
<source>

<AAA id="a1" pos="start">
   <BBB id="b1"/>
   <BBB id="b2"/>
</AAA>
<AAA id="a2">
   <BBB id="b3"/>
   <BBB id="b4"/>
   <CCC id="c1">
      <DDD id="d1"/>
   </CCC>
   <BBB id="b5">
      <CCC id="c2"/>
   </BBB>
</AAA>

</source>
```

XSLT stylesheet

```
<xsl:stylesheet version = '1.0'
   xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:template match="BBB">
   <div style="color:purple">
      <xsl:value-of select="name()"/>
      <xsl:text> id=</xsl:text>
      <xsl:value-of select="@id"/>
   </div>
</xsl:template>

<xsl:template match="/source/AAA/CCC/DDD">
   <p style="color:red">
      <xsl:value-of select="name()"/>
      <xsl:text> id=</xsl:text>
      <xsl:value-of select="@id"/>
   </p>
</xsl:template>


</xsl:stylesheet>
```

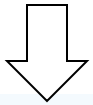
PENNSTATE

Example: <xsl:template> (zvon.org)

By default, the processing
Is by document order

Output

```
<div style="color:purple">BBB id=b1</div>

<div style="color:purple">BBB id=b2</div>



<div style="color:purple">BBB id=b3</div>

<div style="color:purple">BBB id=b4</div>


<p style="color:red">DDD id=d1</p>


<div style="color:purple">BBB id=b5</div>
```



HTML view

BBB id=b1
BBB id=b2
BBB id=b3
BBB id=b4

DDD id=d1

BBB id=b5

PENNSTATE

<xsl:for-each>

```
<xsl:for-each select="XPath"/>
```

- Select every XML element that matches the given XPath
- Allows "looping" in XSLT

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

Node-set (ie, multiple <cd> elements) are returned here

PENNSTATE

Example: <xsl:for-each> (zvon.org)

XML Source

```
<source>

<AAA id="a1" pos="start">
   <BBB id="b1"/>
   <BBB id="b2"/>
</AAA>
<AAA id="a2">
   <BBB id="b3"/>
   <BBB id="b4"/>
   <CCC id="c1">
      <DDD id="d1"/>
   </CCC>
   <BBB id="b5">
      <CCC id="c2"/>
   </BBB>
</AAA>

</source>
```

XSLT stylesheet

```
<xsl:stylesheet version = '1.0'
   xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:template match="/">
   <xsl:for-each select="//BBB">
      <DIV style="color:red">
         <xsl:value-of select="name()"/>
         <xsl:text> id=</xsl:text>
         <xsl:value-of select="@id"/>
      </DIV>
   </xsl:for-each>
   <xsl:for-each select="source/AAA/CCC">
      <DIV style="color:navy">
         <xsl:value-of select="name()"/>
         <xsl:text> id=</xsl:text>
         <xsl:value-of select="@id"/>
      </DIV>
   </xsl:for-each>
</xsl:template>


</xsl:stylesheet>
```

PENNSTATE

Example: <xsl:for-each> (zvon.org)

Output

```
<DIV style="color:red">BBB id=b1</DIV>
<DIV style="color:red">BBB id=b2</DIV>
<DIV style="color:red">BBB id=b3</DIV>
<DIV style="color:red">BBB id=b4</DIV>
<DIV style="color:red">BBB id=b5</DIV>
<DIV style="color:navy">CCC id=c1</DIV>
```

HTML view

BBB id=b1
BBB id=b2
BBB id=b3
BBB id=b4
BBB id=b5
CCC id=c1

PENNSTATE

<xsl:sort>

```
<xsl:sort select="XPath"/>
```

- Sort elements based on element defined by XPath
- Often used together with <xsl:for-each>

```
<xsl:for-each select="//cd">
        <xsl:sort select="artist"/>
</xsl:for-each>
```

Sort based on this element

  - ➔ Among many <cd> nodes matching "//cd", sort them according to the values of <artist> sub-element of <cd>

PENNSTATE

<xsl:if>

```
<xsl:if test="boolean expression">
        contents if TURE
</xsl:if>
```

- Conditional action using XPath expression

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

PENNSTATE

<xsl:choose>

```
<xsl:choose>
        <xsl:when test="expression"> … </xsl:when>
        <xsl:otherwise> … </xsl:otherwise>
</xsl:choose>
```

- Similar to CASE statements in C, Java, etc

```
<xsl:for-each select="//cd">
        <xsl:choose>
                <xsl:when test="price = 10"> ONE </xsl:when>
                <xsl:otherwise> TWO </xsl:otherwise>
        </xsl:choose>
</xsl:for-each>
```

<xsl:apply-templates>

- Applies the specified template to current element or its children
- The "select" attribute makes <xsl:apply-templates> to process only children elements

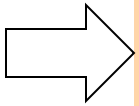
```
<xsl:template match="//cd">
        <xsl:apply-templates select="title"/>
        <xsl:apply-templates select="artist"/>
</xsl:template>
...
<xsl:template match="title"> … </xsl:template>
...
<xsl:template match="artist"> … </xsl:template>
```

Processing order is enforced: <title>, followed by <artist>

Example #1: XML → XML

Input (person.xml)

```
<?xml version="1.0"?>

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

<persons>
  <person username="JS1">
    <fname>John</fname>
    <family-name>Smith</family-
    name>
  </person>
  <person username="MI1">
    <fname>Morka</fname>
    <family-name>Ismincius</
    family-name>
  </person>
</persons>
```



Output (person2.xml)

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

<root>
  <firstname
    nickname="JS1">John</
    firstname>
  <firstname
    nickname="MI1">Morka</
    firstname>
</root>
```

Example adopted from
http://en.wikipedia.org/wiki/XSLT

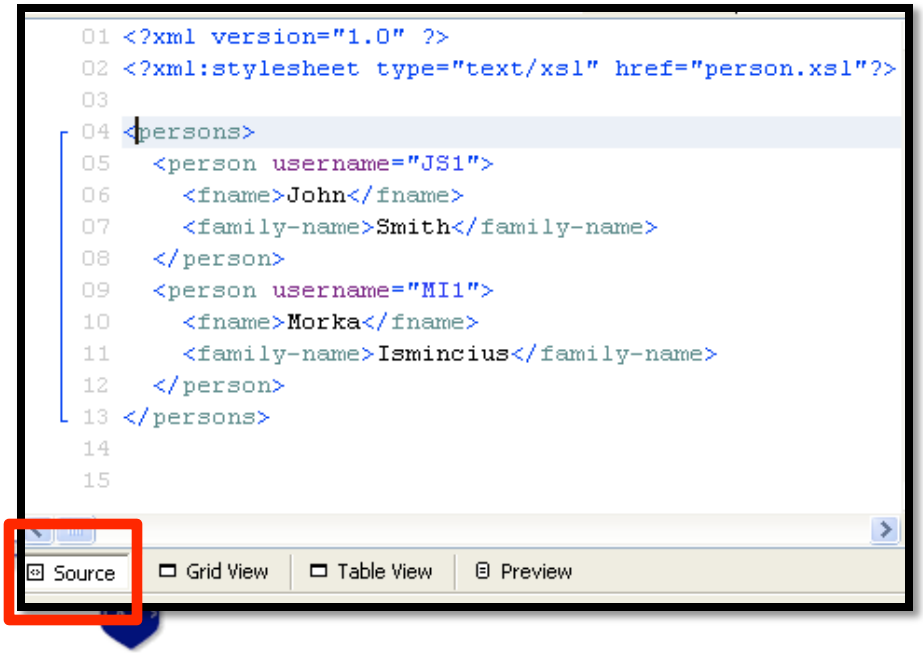
Example #1: XML → XML

Transformation Logic (person.xsl)

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

    <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/persons">
        <root>
            <xsl:apply-templates select="person"/>
        </root>
    </xsl:template>
    <xsl:template match="person">
        <firstname nickname="{@username}">
            <xsl:value-of select="fname" />
        </firstname>
    </xsl:template>

</xsl:stylesheet>
```

Example #1: XML → XML

- To begin the transformation
  - Put both person.xml and person.xsl into the same folder
  - View person.xml using a web browser OR
  - Load person.xml into XMLPad→ "Preview" option

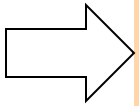



```
<?xml version="1.0"
    encoding="UTF-8"?>

<root>
  <firstname
    nickname="JS1">John</
    firstname>
  <firstname
    nickname="MI1">Morka<
    /firstname>
</root>
```

Example #2: XML → HTML

Input (card.xml)

```
<?xml version="1.0"?>

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

<card type="simple">
   <name>Dongwon Lee</
    name>
   <title>Associate Prof.</title>
   <email>dlee@ist.psu.edu</
    email>
   <phone>(814) 111-2222</
    phone>
</card>
```



Output (card.html)

```
<html xmlns="http://
   www.w3.org/1999/xhtml">

<title>business card</title>
<body bgcolor="gray">
   <h1>Dongwon Lee</h1>
   <h3><i>Associate Prof.</
   i></h3>
   <p>email: <a
   href="mailto:dlee@ist.psu.ed
   u"><tt>dlee@ist.psu.edu</
   tt></a></p>
   <p>phone: (814) 111-2222</
   p>
</body>
</html>
```

Example #2: XML → HTML

Transformation Logic (card.xsl)

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
   Transform"    version="1.0">

  <xsl:output method="html"/>

  <xsl:template match="card[@type='simple']">
    <html xmlns="http://www.w3.org/1999/xhtml">
      <title>business card</title><body bgcolor="gray">
        <xsl:apply-templates select="name"/>
        <xsl:apply-templates select="title"/>
        <xsl:apply-templates select="email"/>
        <xsl:apply-templates select="phone"/>
    </body></html>
  </xsl:template>
```

Example #2: XML → HTML

Transformation Logic (card.xsl)

```
<xsl:template match="name">
   <h1><xsl:value-of select="text()"/></h1>  </xsl:template>

<xsl:template match="title">
   <h3><i><xsl:value-of select="text()"/></i></h3>
   </xsl:template>

<xsl:template match="email">
    <p>email: <a href="mailto:{text()}"><tt>
   <xsl:value-of select="text()"/>     </tt></a></p>
  </xsl:template>

<xsl:template match="phone"> <p>phone:
   <xsl:value-of select="text()"/></p></xsl:template>

</xsl:stylesheet>
```

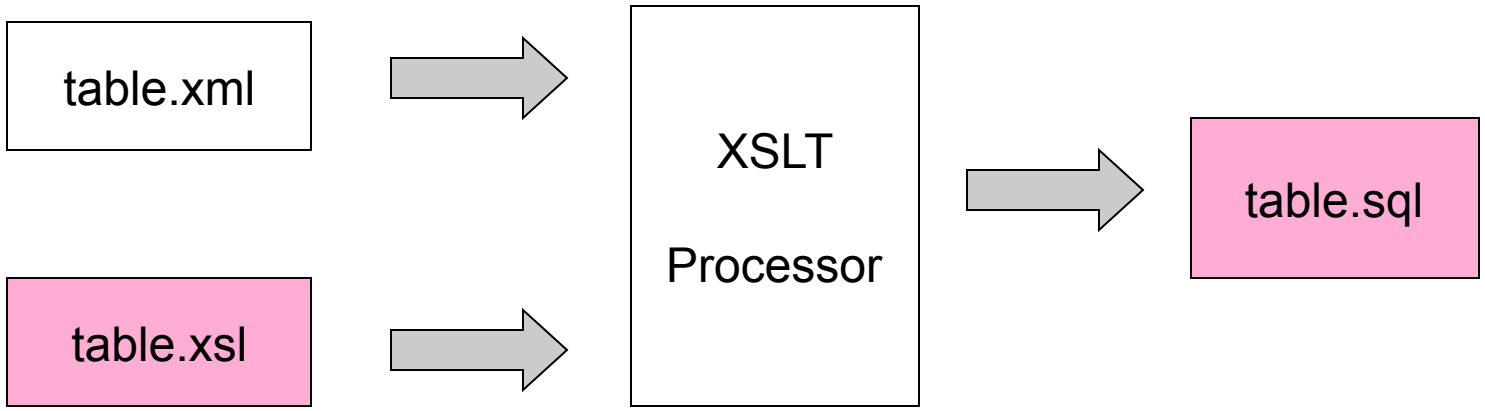
Example #2: XML → HTML

- To begin the transformation
  - Put both card.xml and card.xsl into the same folder
  - View card.xml using a web browser OR
  - Load card.xml into XMLPad→ "Preview" option

PENNSTATE

Example #3: XML → SQL

- Input: table.xml
- Goal: to load this data into MySQL
- Issue: MySQL can't take an XML file
- Task: Transform data in table.xml into SQL INSERT statements
- Eg,



PENNSTATE

Example #3: XML → SQL

Input Data (table.xml)

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
    href="table.xsl"?>
<employee>
 <tuple>
  <id>1</id>
  <name>
   <ln>Lee</ln> <fn>Dongwon</fn>
  </name>
  <salary>500000</salary>
  <manager>199</manager>
  <birthyear>1980</birthyear>
  <startyear>2002</startyear>
 </tuple>
 <tuple>
  <id>2</id>
  <name>
   <ln>Luo</ln><fn>Robert</fn>
  </name>
…
```

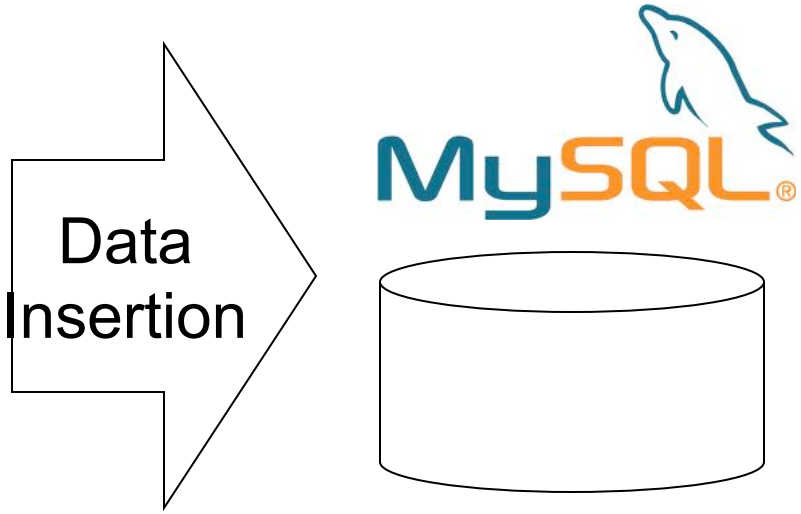
Transformation Logic (table.xsl)

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

 <xsl:template match="tuple">
  INSERT INTO employee VALUES (
  <xsl:value-of select="id"/>,
  "<xsl:value-of select="name/ln"/>,
  <xsl:value-of select="name/fn"/>",
  <xsl:value-of select="salary"/>,
  <xsl:value-of select="birthyear"/>,
  <xsl:value-of select="startyear"/>);
 </xsl:template>

</xsl:stylesheet>
```

Example #3: XML → SQL

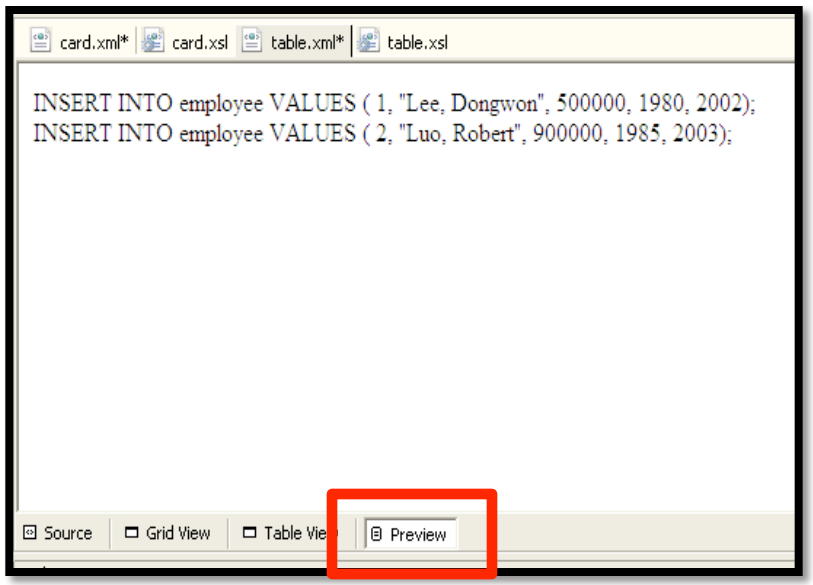
Output (table.sql)

```
INSERT INTO employee
    VALUES (1, 'Lee,
    Dongwon', 500000,
    199, 1980, 2002);

INSERT INTO employee
    VALUES (2, 'Luo,
    Robert', 900000, 1,
    1985, 2003);
```



Example #3: XML → SQL

- To begin the transformation
  - Put both table.xml and table.xsl into the same folder
  - View table.xml using a web browser OR
  - Load table.xml into XMLPad→ "Preview" option



Examples Download

- Download two examples from here:
  - http://pike.psu.edu/classes/ist516/latest/labs/xslt/
- Load them into XMLPad
- Play with different features of XSLT by modifying "*.xsl" files

Much More XSLT Features

- XSLT has more than 100 built-in functions
- Useful online XSLT tool at Zvon
  - http://www.zvon.org/xxl/XSLTutorial/Books/Output/contents.html
- Try: 1, 3, 6, 8, 9, 11, 14, 15, 18, 19, 26

XSLT Processors

- XSLT Processor (or Engine): program that can take XML and XSLT as input and apply the specified transformations
  - XT: James Clark's XSLT processor for Java
  - XALAN: Apache's XSLT processor for Java/C++
  - MSXML: Microsoft's XSLT processor

    Embedded in Microsoft IE web browser

- More here: http://www.w3c.org/Style/XSL/

Reference

- XSLT, 1st Edition, Doug Tidwell, O'Reilly, 2001
- W3Schools XSLT Tutorial
  - http://www.w3schools.com/xsl/
- Zvon XSLT Tutorial
  - http://www.zvon.org/xxl/XSLTutorial/Output/index.html

PENNSTATE