Data and Document Processing: XSLT and SQL

- ♡ *Introducing XML*
- ♡ *XPath*
- ♡ *Namespaces*
- ♡ *DTDs*
- ♡ *XML Schema*
- ♡ *XLink*
- ♡ *XBRL*
- ♡ *XML and Java*
- ♡ *XQuery*
- ♡ *XSLT*

XSLT and SQL

In these notes, we use our knowledge of SQL as the basis for exploring some more of the features of XSLT. In particular, we look at:

- Some sample SQL queries posed against the simple relational database shown in the handout.
  - We then compare these with some more examples of XSLT.
  - The XSLT examples are applied to an XML document that has been derived from the database.
  - This document also appears in the accompanying handout.
- The use of variables in XSLT.
- How you can direct XSLT to *sort* the document it generates.
- How *parameters* may be passed to an XSLT program.

Example 1: Describe all items of assessment and show the weighting attached to each.

```
Select   Title, Weight
From     Assessment

---------------
Title    Weight
---------------
JSP       10
XML       30
Exam      60
---------------
```

The query is answered quite simply by naming the columns required, and by naming the table in which the data will be found.

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.1">
<xsl:output indent="yes"/>

<xsl:template match="/">
<query>
<xsl:apply-templates select="sdb/Assessment/AssItem"/>
</query>
</xsl:template>

<xsl:template match="AssItem">
<row>
<title><xsl:value-of select="Title"/></title>
<weight><xsl:value-of select="Weight"/></weight>
</row>
</xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="utf-8"?>
<query>
   <row>
      <title>JSP</title>
      <weight>10</weight>
   </row>
   <row>
      <title>XML</title>
      <weight>30</weight>
   </row>
   <row>
      <title>Exam</title>
      <weight>60</weight>
   </row>
</query>
```

Another solution

```
1. <xsl:stylesheet
2.      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3.      version="1.1">
4. <xsl:output indent="yes"/>
5.
6. <xsl:template match="/">
7. <query>
8. <xsl:for-each select="sdb/Assessment/AssItem">
9. <row>
10. <title><xsl:value-of select="Title"/></title>
11. <weight><xsl:value-of select="Weight"/></weight>
12. </row>
13. </xsl:for-each>
14. </query>
15. </xsl:template>
16.
17. </xsl:stylesheet>
```

xsl:for-each

- How does this solution differ from the first one?
-
-
-

Example 2: Describe items of assessment weighted more than 25%.

```
Select   Title, Weight
From     Assessment
Where    Weight > 25

---------------
Title    Weight
---------------
XML       30
Exam      60
---------------
```

```
<?xml version="1.0" encoding="utf-8"?>
<query>
   <row>
      <title>XML</title>
      <weight>30</weight>
   </row>
   <row>
      <title>Exam</title>
      <weight>60</weight>
   </row>
</query>
```

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.1">
<xsl:output indent="yes"/>

<xsl:template match="/">
<query>
<xsl:for-each select="sdb/Assessment/AssItem[Weight>=25]">
<row>
<title><xsl:value-of select="Title"/></title>
<weight><xsl:value-of select="Weight"/></weight>
</row>
</xsl:for-each>
</query>
</xsl:template>

</xsl:stylesheet>
```

Example 3: We can evaluate statistical functions in SQL:

```
Select count(*), sum(Weight)
From   Assessment

-----------
  3     100
-----------
```

We can do something similar in XPath+XSLT:

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.1">
<xsl:output indent="yes"/>

<xsl:template match="/">
<query>
 <count>
  <xsl:value-of select="count(sdb/Assessment/AssItem)"/>
 </count>
 <sum>
  <xsl:value-of select="sum(sdb/Assessment/AssItem/Weight)"/>
 </sum>
</query>
</xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="utf-8"?>
<query>
   <count>3</count>
   <sum>100</sum>
</query>
```

≪•≫ *We can use functions (from XPath and from XSLT)*

- Data conversion: `number()`, `string()`
- String manipulation: `concat()`, `substring()`, `starts-with()`
- Aggregation: `count()`, `sum()`
- Information about context: `current()`, `last()`, `position()`
- Access to other documents: `document()`

≪•≫ *SQL and XSLT/XPath similarities*

- Output: the select clause (SQL) and template and value-of elements (XSLT).
- Retrieval: the from clause (SQL) and path expressions (XPath).
- Filtering: the where clause (SQL) and xsl:if and xsl:choose elements (XSLT) and path expressions (XPath).
- Statistical functions
- How about joins?

Example 4: Consider the join of the Results and the Students tables.

```
                Join of Results and Students
-------------------------------------------------------
Item  Id    Submitted    Mark  Id     First     Last
-------------------------------------------------------
  1   871   08-Sep-2005   80   871    Hans      Zupp
  1   862   07-Sep-2005   60   862    Bill      Board
  1   854   08-Sep-2005   70   854    Ann       Dover
  1   872   10-Sep-2005   55   872    Betty     Kahn
  1   868   06-Sep-2005   90   868    Will      Gambol
  1   869   09-Sep-2005   70   869    Rip       Orff
  2   871   21-Oct-2005   70   871    Hans      Zupp
  2   869   22-Oct-2005   80   869    Rip       Orff
  2   872   21-Oct-2005   65   872    Betty     Kahn
  2   862   22-Oct-2005   70   862    Bill      Board
  2   868   21-Oct-2005   75   868    Will      Gambol
  3   869         ?       95   869    Rip       Orff
  3   872         ?       45   872    Betty     Kahn
  3   862         ?       40   862    Bill      Board
  3   868         ?       50   868    Will      Gambol
  3   871         ?       60   871    Hans      Zupp
  3   854         ?       65   854    Ann       Dover
-------------------------------------------------------
```

This is achieved by means of the following SQL:

```
Select *
From   Results r, Students s
Where  r.Id = s.Id
```

An XML version of the join might look like this:

```
<query>
   <row>
      <Item>1</Item>
      <Id>871</Id>
      <Submitted>08-Sep-2005</Submitted>
      <Mark>80</Mark>
      <Id>871</Id>
      <First>Hans</First>
      <Last>Zupp</Last>
   </row>
   :
</query>
```

The output could be generated in number of ways. Here is one of these:

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.1">
<xsl:output indent="yes"/>

<xsl:template match="/">
<query>
<xsl:for-each select="//Result">
 <xsl:variable name="rId" select="Id"/>
  <row>
   <Item><xsl:value-of select="Item"/></Item>
   <Id><xsl:value-of select="$rId"/></Id>
   <Submitted><xsl:value-of select="Submitted"/></Submitted>
   <Mark><xsl:value-of select="Mark"/></Mark>
   <Id><xsl:value-of select="//Student[Id=$rId]/Id"/></Id>
   <First><xsl:value-of select="//Student[Id=$rId]/First"/></Fi
   <Last><xsl:value-of select="//Student[Id=$rId]/Last"/></Last
 </row>
</xsl:for-each>
</query>
</xsl:template>

</xsl:stylesheet>
```

Example 5: What if we want to know how many items of assessment each student submitted.

```
Select s.Id, count(*)
From Students s, Results r
Where s.Id = r.Id
Group by s.Id
union
Select Id, 0
From Students
Where Id not in (Select Id
                 From   Results)
```

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.1">
<xsl:output indent="yes"/>

<xsl:template match="/">
<query>
<xsl:for-each select="//Student">
 <xsl:variable name="sId" select="Id"/>
 <row><sid><xsl:value-of select="$sId"/></sid>
      <sas><xsl:value-of
                select="count(//Result[Id=$sId])"/></sas>
 </row>
</xsl:for-each>
</query>
</xsl:template>

</xsl:stylesheet>
```

The results of the this program are as follows:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <query>
3.    <row>
4.       <sid>871</sid>
5.       <sas>3</sas>
6.    </row>
7.    <row>
8.       <sid>862</sid>
9.       <sas>3</sas>
10.   </row>
11.   <row>
12.      <sid>869</sid>
13.      <sas>3</sas>
14.   </row>
15.   <row>
16.      <sid>854</sid>
17.      <sas>2</sas>
18.   </row>
19.   <row>
20.      <sid>831</sid>
21.      <sas>0</sas>
22.   </row>
23.   <row>
24.      <sid>872</sid>
25.      <sas>3</sas>
26.   </row>
27.   <row>
28.      <sid>868</sid>
29.      <sas>3</sas>
30.   </row>
31. </query>
```

So, not only can XSLT accomplish a join, it can do a grouping operation and throw in a union operation too.

Example 6: Here is a join performed in a somewhat more "conventional" style:

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.1">
<xsl:output indent="yes"/>

<xsl:template match="/">
<join>
<xsl:for-each select="//Result">
 <xsl:variable name="rId" select="Id"/>
 <xsl:variable name="rItem" select="Item"/>
 <xsl:for-each select="//AssItem">
  <xsl:variable name="aItem" select="Item"/>
  <xsl:if test="$rItem=$aItem">
   <row><rid><xsl:value-of select="$rId"/></rid>
        <ritem><xsl:value-of select="$rItem"/></ritem>
        <aitem><xsl:value-of select="$aItem"/></aitem></row>
  </xsl:if>
 </xsl:for-each>
</xsl:for-each>
</join>
</xsl:template>

</xsl:stylesheet>
```

In database terms, this is a "nested loop" join.

```
<?xml version="1.0" encoding="utf-8"?>
<join>
   <row>
      <rid>871</rid>
      <ritem>1</ritem>
      <aitem>1</aitem>
   </row>
   :
   <row>
      <rid>862</rid>
      <ritem>3</ritem>
      <aitem>3</aitem>
   </row>
   :
</join>
```

Variables in XSLT

xsl:variable

- By means of this element, we may declare a local or global variable, and give it a value.
- The `name` attribute is used to name the variable.
- The `select` attribute may be used to provide a value for the variable.
- This attribute is optional. If missing, the content of the element is used instead, i.e., whatever appears between the `<xsl:variable>` and `</xsl:variable>` tags.

Why use a variable?

- To avoid repeating some common expression.
- To capture context-sensitive information.
- To hold a temporary tree structure.

Issues with XSLT variables

- As in any programming language, a variable is useful for calculating a value used in several different places.
- Unlike variables in most programming languages, XSLT variables cannot be updated.
- Once they are given initial value, they retain that value until they go out of scope.

Example 7: We can perform a join and store the results in a variable:

```
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.1">
<xsl:output indent="yes"/>

<xsl:template match="/">
<xsl:variable name="temp">
 <xsl:for-each select="//Result">
  <row>
   <Id><xsl:value-of select="Id"/></Id>
   <Item><xsl:value-of select="Item"/></Item>
   <Mark><xsl:value-of select="Mark"/></Mark>
   <Val>
   <xsl:value-of
   select="Mark*//AssItem[Item=current()/Item]/Weight div 100"/
   </Val>
  </row>
 </xsl:for-each>
</xsl:variable>

<FinalResults>
<xsl:for-each select="//Student">
 <Student>
 <Id><xsl:value-of select="Id"/></Id>
 <First><xsl:value-of select="First"/></First>
 <Last><xsl:value-of select="Last"/></Last>
 <Total>
  <xsl:value-of select="sum($temp/row[Id=current()/Id]/Val)"/>
 </Total>
 </Student>
</xsl:for-each>
</FinalResults>

</xsl:template>

</xsl:stylesheet>
```

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <FinalResults>
3.    <Student>
4.       <Id>871</Id>
5.       <First>Hans</First>
6.       <Last>Zupp</Last>
7.       <Total>65</Total>
8.    </Student>
9.    <Student>
10.      <Id>862</Id>
11.      <First>Bill</First>
12.      <Last>Board </Last>
13.      <Total>51</Total>
14.   </Student>
15.   <Student>
16.      <Id>869</Id>
17.      <First>Rip</First>
18.      <Last>Orff</Last>
19.      <Total>88</Total>
20.   </Student>
21.   <Student>
22.      <Id>854</Id>
23.      <First>Ann</First>
24.      <Last>Dover</Last>
25.      <Total>46</Total>
26.   </Student>
27.   <Student>
28.      <Id>831</Id>
29.      <First>Hans</First>
30.      <Last>Orff</Last>
31.      <Total>0</Total>
32.   </Student>
33.   <Student>
34.      <Id>872</Id>
35.      <First>Betty</First>
36.      <Last>Kahn</Last>
37.      <Total>52</Total>
38.   </Student>
39.   <Student>
40.      <Id>868</Id>
41.      <First>Will</First>
42.      <Last>Gambol</Last>
43.      <Total>61.5</Total>
44.   </Student>
45. </FinalResults>
```

There are some things that SQL can do that XPath cannot accomplish easily.

```
Select   S.Id, max(S.First), max(S.Last),
         sum(R.Mark*A.Weight/100)
From     Students S, Results R, Assess A
Where    S.Id = R.Id
and      R.Item = A.Item
Group by S.Id
Order by 4 desc

-------------------------------------------------------------
Id   max(S.First)  max(S.Last)   sum(R.Mark*A.Weight/100)
-------------------------------------------------------------
869      Rip          Orff                  88
871      Hans         Zupp                  65
868      Will         Gambol                62
872      Betty        Kahn                  52
862      Bill         Board                 51
854      Ann          Dover                 46
-------------------------------------------------------------
```

For example, student 869's mark is calculated as follows:

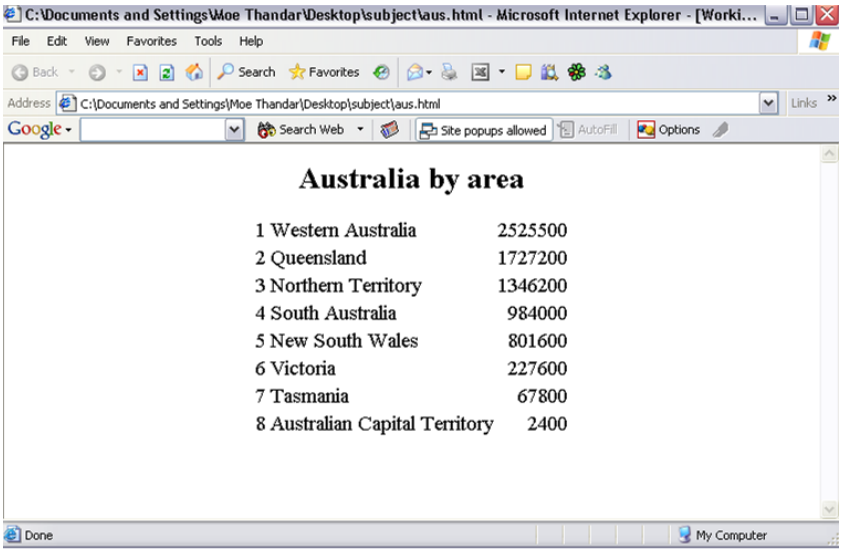
```
Id     Item  Mark   Weight   Mark*Weight/100
-----------------------------------------
869     1     70      10     70*10/100  =  7
869     2     80      30     80*30/100  = 24
869     3     95      60     95*60/100  = 57
                                         ---
                                          88
```

Sorting

Example 8: List the states and territories of Australia and their respective areas. Return the largest first.

```
1. <xsl:stylesheet
2.      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3.      version="1.0">
4.
5. <xsl:template match="Australia">
6.   <TITLE>Australia by area</TITLE>
7.   <CENTER><H2>Australia by area</H2><TABLE>
8.   <xsl:apply-templates select="division">
9.     <xsl:sort
10.         select="area"
11.         order="descending"
12.         data-type="number"/>
13.  </xsl:apply-templates>
14.  </TABLE></CENTER>
15. </xsl:template>
16.
17. <xsl:template match="division">
18.  <TR>
19.  <TD ALIGN="right"><xsl:value-of select="position()"/></TD>
20.  <TD><xsl:value-of select="name"/></TD>
21.  <TD ALIGN="right"><xsl:value-of select="area"/></TD>
22.  </TR>
23. </xsl:template>
24.
25. </xsl:stylesheet>
```

The resulting HTML output will be displayed as follows:



xsl:sort

The XSLT `sort` element controls the order in which selected nodes are processed:

- It must be a child of either an `apply-templates` or a `for-each` element.
- The `select` attribute determines the data that is to be sorted.
- The `order` attribute determines whether the data is to be sorted into "ascending" or "descending" order.
- The `data-type` attribute determines whether each item of data is to be treated as a "number" or as a piece of "text".

Ask yourself

If the `data-type` was set to "`text`", what would be the first three states or territories produced?

The use of parameters:

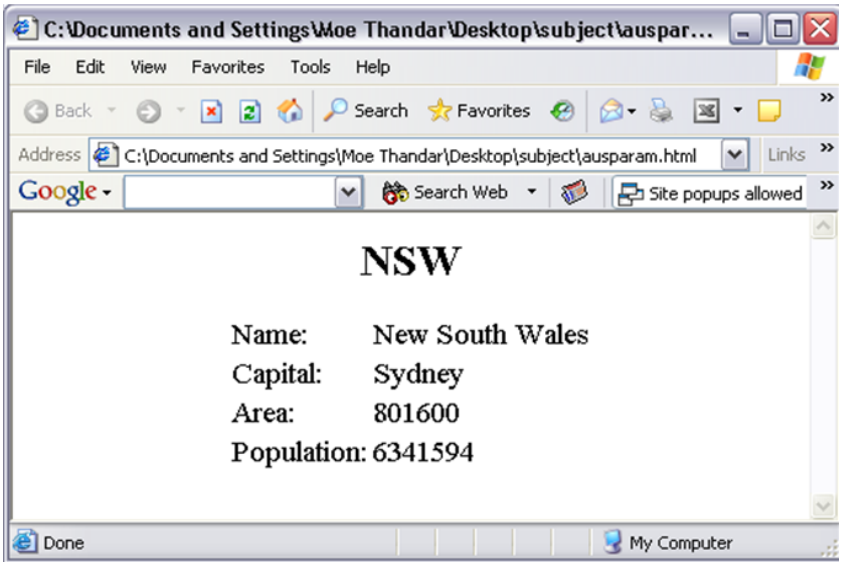
Example 9: Suppose we wanted to be able to find out, on demand, details of a selected state or territory.

```
1. <xsl:stylesheet
2.      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3.      version="1.0">
4.
5. <xsl:param name="short"/>
6.
7. <xsl:template match="Australia">
8.   <TITLE>All about
9.      <xsl:value-of select="//division/name[@abbr=$short]"/>
10.  </TITLE>
11.  <CENTER>
12.  <H2><xsl:value-of select="$short"/></H2>
13.  <TABLE>
14.  <xsl:apply-templates select="division[name[@abbr=$short]]"/>
15.  </TABLE>
16.  </CENTER>
17. </xsl:template>
18.
19. <xsl:template match="division">
20.  <TR><TD>Name:</TD><TD><xsl:value-of select="name"/></TD></TR>
21.  <TR><TD>Capital:</TD><TD><xsl:value-of select="cap"/></TD></T
22.  <TR><TD>Area:</TD><TD><xsl:value-of select="area"/></TD></TR>
23.  <TR><TD>Population:</TD><TD><xsl:value-of select="pop"/></TD>
24. </xsl:template>
25.
26. </xsl:stylesheet>
```

Instant saxon command

For the output generated below:

```
saxon -o ausparam.html aus.xml ausparam.xsl short=NSW
```



xsl:param

Summary

- XSLT and SQL
- XSLT variables
- Sorting in XSLT
- XSLT parameters
- XSLT functions and elements