select-from where

where-construct syntax instead of the more familiar

of the query will be constructed

It has templates which show how the output XML data

- From input XML data

- Uses path expressions and patterns to extract data

- From Chapter 4

- Combines XML syntax with query language techniques

5.1 XML-QL

CH. 5 Query Languages for XML
For each binding, the query processor will construct $\forall$
possible ways with the data and bind the variables $T$ and $A$;
the query processor will match the pattern in all
construct $\forall A$;
where
Example:
constructing new XML data:
Consider the following XML data:
The answer to the previous query is:
Assume <prince> tag in <book> element is optional;

Process optional elements with nested quarters.
So, the result looks like:
Suppose we want to retrieve each author and all book titles she/he has published.

The bibliographic data has several authors for each book.
where book</book> in "abc.xmi"

open $book

</book>

<book>

construct $book</book>

publish</name>Morgan Kaufmann

</book>

<book id="item"/>

<book id="item"/>

$book is bound to element $book

construct $book

element-as $book in "abc.xmi"

</book></book>

<book><book>publisher</name>Morgan Kaufmann

<book>

element-as keyword:

to element.

XML-QL provides syntactic sugar that allows one to bind

variables in XML-QL as bound to element content rather

binding elements and contents
construct <result> $<\text{result}>
</result></book>
</book>

The XML-QL processor will translate the query into

$C$ is bound to content within $<\text{book}>/book>$

construct <result> $<\text{result}>
content-as $C$ in "abc.xml"
</result></book>

where $<\text{book}>/book>$

content-as keyword: allows one to bind to content
Notice an attribute value in the XML document becomes

```xml
<result>
  <book language="French">
    Get all book titles in French.
  </book>
</result>
```

Query Attributes: 
get all authors who have published at least two books.

By using the same variable in two matchings, we can express "Joins" joining elements by value:
Here there are two tag variables: $f$ and $g$.

```xml
</> Xml <$"> Construct title>
{author, editor} in
"mmm. a. d. /db. xml",
</>
</> Xml <$"> Year
</>
Year1996
</>
</>
where <$"> Title>
End all publications published in 1996 with Smith as an
Tag Variables:
```
...
construct result in "www.a.b.c".xm1

where (part|subpart|component|piece)<$>$

matches brand ford at any level in the xml graph.

</p>brand<ford>+*'

So, indicate a tag at any level (is abbreviated to *)

$*$

</p>result$>$ in "www.a.b.c".xm1

</p>brand<ford></p>$<$name $<*$>

where a tag is permitted. For example:

The wild card matches any tag and can appear anywhere
XSL: XML Style Sheet Language
The result will have the form:

- Recursively, this process ends at the third level, the second

Applying the templates on the child nodes: <book>

The template rule is applied; it matches the root node:

- The first rule is applied; it matches the root node.

(Here it is qid followed by any label followed by title)

When a match attribute is missing, the template matches any node.

```
<result />
<xsl:template/>
</xsl:template>
</result>
```

The following XSL program returns all titles:

```xml
<xsl:template>
  <xsl:variable name="/title">
    <xsl:apply-templates select="*"/>
  </xsl:variable>
</xsl:template>
```
3 comments and one &lt;p&gt;&lt;/p&gt; element.

From XSL's view point the root node for above data has 4 children:

```
&lt;-- comment 3 &gt;
 &lt;comment 3&gt;
 &lt;br&gt;
 &lt;comment 2 &gt;
 &lt;comment 1 &gt;
```

This is to accommodate processing statements before the top node

node is on top of &lt;p&gt; node /
XSLT does not allow variables; a substitution:

- node, subelement, and a language attribute.
  - matches a paper element in bib but only if it has a year.

  bib/paper/year and publisher/name and language
  of paper.

- paper/year matches a year element which appears as a subelement.

  paper/year matches a paper element which has a year subelement.

Non-linear patterns using notation:
XSL instructions:

The template consists of XSL (or HTML) code along with

So, $book/*/title$ can be abbreviated to title

XSL processor automatically proceeds to each pattern:

```xml
<xsl:template
  name="template"
  match="pattern"
>
    General Form:

    XSL Template Rules:
```
is equivalent to

```xml
<xs:element name="%" match="%"
<xs:element name="%" match="%"
```

creates a new element

... `xs:element name="%" match="%"`

evales to the string content of the current node.
the name of the output node.
name() returns the name of the current node which we use as

```xml
<xt:template>
<xt:element/>
</xt:template>
<xt:template name="name()"
<xt:template match="*" match="*"
```

which copies all top-level statements from the input file.

A situation where `xt:element` is truly useful is the following:
 Continued:

```xml
<xsl:template match="bib">
  <TABLE>
    <TBODY>
      <xsl:apply-templates/>
    </TBODY>
  </TABLE>
</xsl:template>

<HTML>
  <HEAD>
    <TITLE>Bibliography Entries</TITLE>
  </HEAD>
  <BODY>
    <TABLE>
      <TBODY>
        <TR><TD> t1 </TD> <TD> a1 </TD> <TD> a2 </TD> </TR>
        <TR><TD> t2 </TD> <TD> a3 </TD> <TD> a4 </TD> </TR>
        <TR><TD> t3 </TD> <TD> a5 </TD> <TD> a6 </TD> <TD> a7 </TD> </TR>
      </TBODY>
    </TABLE>
  </BODY>
</HTML>
```