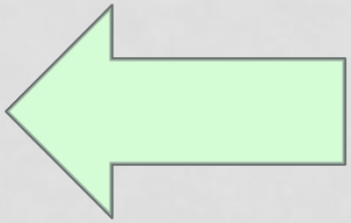
PROCESSING AND QUERYING XML

CS561-SPRING 2012
WPI, MOHAMED ELTABAKH

ROADMAP

- **Models for Parsing XML Documents**



- **XPath Language**
- **XQuery Language**
- **XML inside DBMSs**

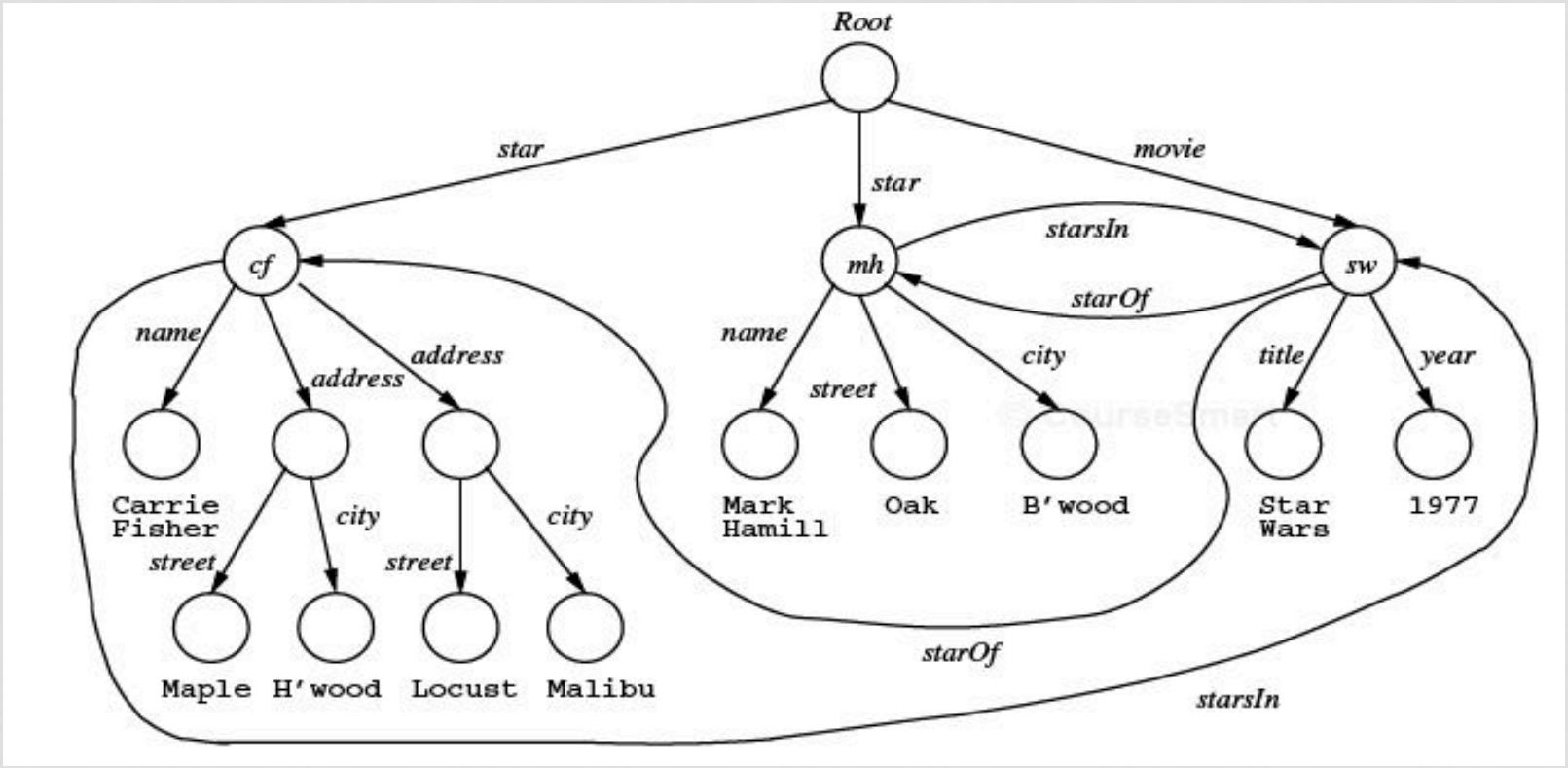
PROCESSING XML

- Non-validating parser:
  - checks that XML doc is syntactically well-formed
- Validating parser:
  - checks that XML doc is also valid w.r.t. a given DTD or Schema
- Parsing yields tree/object representation:
  - Document Object Model (DOM) API
-
- Or a stream of events (open/close tag, data):
  - Simple API for XML (SAX)

DOM STRUCTURE MODEL

- hierarchy of Node objects:
  - document, element, attribute, text, comment, ...
- language independent programming DOM API:
  - get... first/last child, prev/next sibling, childNodes
  - insertBefore, replace
  - getElementsByTagName
  - ...

EXAMPLE OF DOM TREE



SAX: SIMPLE API FOR XML

- Event-based SAX API (Simple API for XML)
  - does not build a parse tree (reports events when encountering begin/end tags)
  - for (partially) parsing very large documents

DOM SUMMARY

- Object-Oriented approach to traverse the XML node tree
- Automatic processing of XML docs
- Operations for manipulating XML tree
- Manipulation & Updating of XML on client & server
- Database interoperability mechanism
- Memory-intensive

SAX SUMMARY

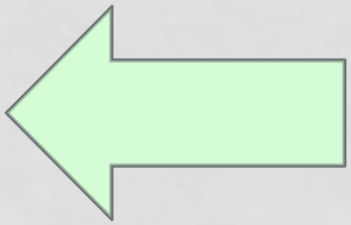
- **Pros:**
  - The whole file doesn't need to be loaded into memory
  - XML stream processing
  - Simple and fast
  - Allows you to ignore less interesting data

- **Cons:**
  - limited expressive power (query/update) when working on streams

  => application needs to build (some) parse-tree when necessary

ROADMAP

- **Models for Parsing XML Documents**
- **XPath Language**
- **XQuery Language**
- **XML inside DBMSs**



XPATH

- **Goal = Permit access some nodes from document**

- XPath main construct : Axis navigation
- Navigation step : axis + node-test + predicates
- Examples
  - descendant::book ➔ //book
  - child::author ➔ ./author or author
  - attribute::booktitle =“XML” ➔ @booktitle = “XML”

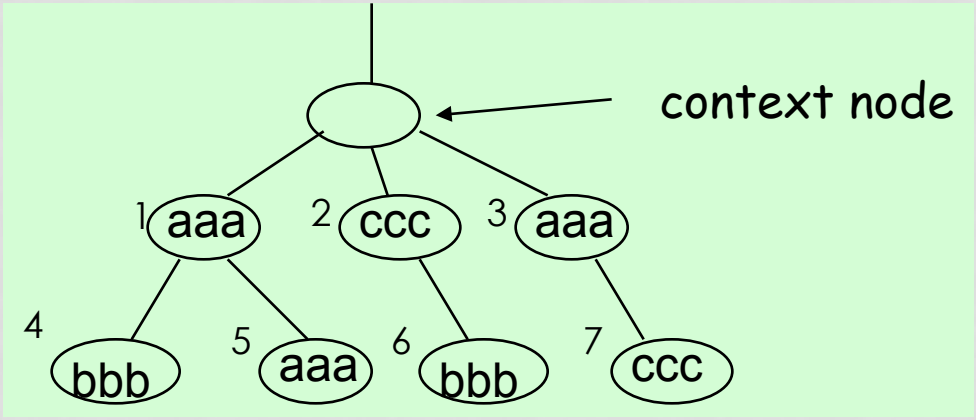
XPATH- CHILD AXIS NAVIGATION

- **/doc** -- all doc children of the root
- **./aaa** -- all **aaa** children of the context node (equivalent to **aaa**)
- **text()** -- all text children of context node
- **node()** -- all children of the context node (includes text and attribute nodes)
- **..** -- parent of the context node
- **.//** -- the context node and all its descendants
- **//** -- the root node and all its descendants
- ***** - all children of content node
- **//text()** -- all the text nodes in the document

XPATH EXAMPLE

- **//aaa**
  - Nodes 1, 3, 5
- **./aaa or aaa**
  - Nodes 1, 3

- **bbb**
  - None
- **//aaa/ccc**
  - Node 7



PREDICATES

- [2] or ./[2] -- the second child node of the context node
- chapter[5] -- the fifth chapter child of context node
- [last()] -- the last child node of the context node
- chapter[title="introduction"] -- the chapter children of the context node that have one or more title children whose string-value is "introduction" (string-value is concatenation of all text on descendant text nodes)
- Person[//firstname = "joe"] -- the person children of the context node that have in their descendants a firstname element with string-value "Joe"

MORE EXAMPLES

- **Find all books written by author 'Bob'**

//book[author/first-name= 'Bob']

- **Given a content bookstore node, find all books with price > $50**

book[@price > 50]

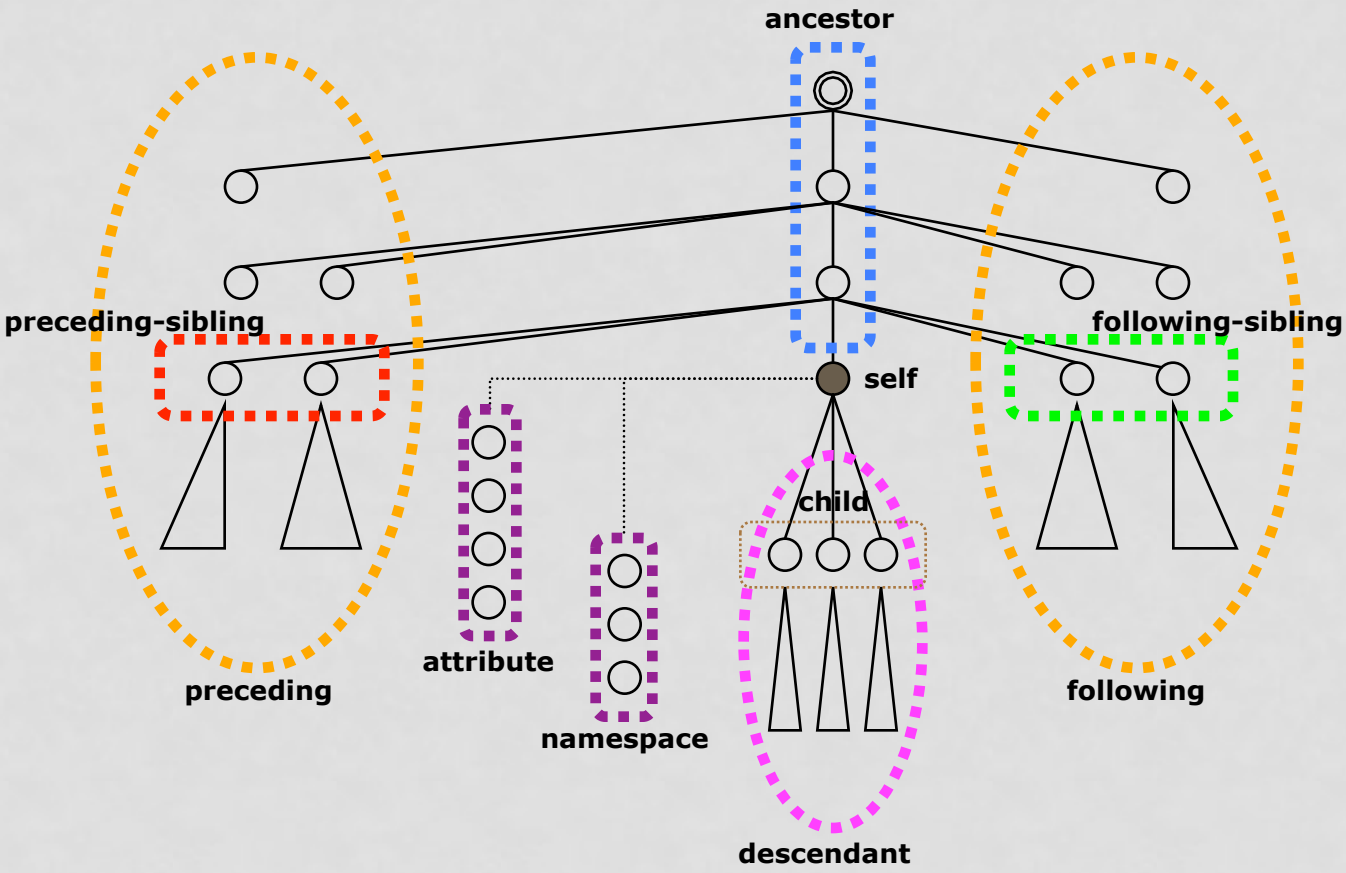
- **Find all books where the *category* equals the bookstore *specialty*?**

//book[../bookstore/@specialty= @category]

```
<Doc>
  <bookstore specialty="science">
      <book category="science" price = 10>
          <author>
              <first-name>Bob</first-name>
          </author>
      </book>
       …
  </bookstore>
   ….
  <bookstore specialty="sport">
      <book>
      </book>
  </bookstore>
</Doc>
```

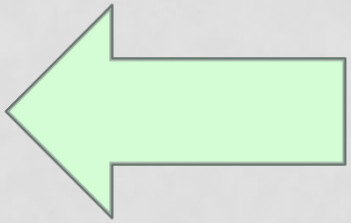
AXIS NAVIGATION

- XPath has several axes: ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self, following, following-sibling, namespace, parent, preceding, preceding-sibling, self



ROADMAP

- **Models for Parsing XML Documents**
- **XPath Language**
- **XQuery Language**



- **XML inside DBMSs**

FLWR ("FLOWER") EXPRESSIONS

FOR ...

LET...

WHERE...

RETURN...

XQuery is more powerful and expressive that XPath

XQUERY

Find the titles of all books published after 1995:

```
FOR $x  IN  document("bib.xml")/bib/book
WHERE $x/year > 1995
RETURN $x/title
```

How does result look like?

XQUERY

Find the titles of all books published after 1995:

```
FOR $x  IN  document("bib.xml")/bib/book
WHERE $x/year > 1995
RETURN $x/title
```

Result:

```
<title> abc </title>
<title> def </title>
<title> ghi </title>
```

XQUERY EXAMPLE

```
FOR $a IN   (document("bib.xml")
                  /bib/book[publisher="Morgan Kaufmann"]/author)
RETURN <result>
              $a,
              FOR $t IN /bib/book[author=$a]/title
              RETURN $t
          </result>
```

XQUERY EXAMPLE

For each author of a book by Morgan Kaufmann,
list all books she published:

```
FOR $a IN   (document("bib.xml")
                  /bib/book[publisher="Morgan Kaufmann"]/author)
RETURN <result>
            $a,
            FOR $t IN /bib/book[author=$a]/title
            RETURN $t
        </result>
```

What is query result ?

XQUERY

Result:

```
<result>
      <author>Jones</author>
      <title> abc </title>
      <title> def </title>
</result>
<result>
      <author>Jones</author>
      <title> abc </title>
      <title> def </title>
</result>

<result>
      <author> Smith </author>
      <title> ghi </title>
</result>
```

How to eliminate duplicates?

XQUERY EXAMPLE: DUPLICATES

For each author of a book by Morgan Kaufmann, list all books she published:

```
FOR $a IN distinct(document("bib.xml")
                   /bib/book[publisher="Morgan Kaufmann"]/author)
RETURN <result>
            $a,
            FOR $t IN /bib/book[author=$a]/title
            RETURN $t
        </result>
```

distinct = a function that eliminates duplicates

EXAMPLE XQUERY RESULT

Result:

```
<result>
        <author>Jones</author>
        <title> abc </title>
        <title> def </title>
</result>

<result>
        <author> Smith </author>
        <title> ghi </title>
</result>
```

XQUERY

- FOR $x in expr
  - binds $x to each element in the list expr
  - Useful for iteration over some input list

- LET $x = expr
  - binds $x to the entire list expr
  - Useful for common subexpressions and for grouping and aggregations

XQUERY WITH LET CLAUSE

```
<big_publishers>
      FOR $p IN  distinct(document("bib.xml")//publisher)
      LET $b := document("bib.xml")/book[publisher = $p]
      WHERE count($b) > 100
      RETURN $p
</big_publishers>
```

count = a (aggregate) function that returns number of elements

XQUERY

Find books whose price is larger than average:

```
LET $a = avg(document("bib.xml")/bib/book/@price)
FOR $b in document("bib.xml")/bib/book
WHERE $b/@price > $a
RETURN $b
```

FOR vs. LET

FOR

- Binds *node variables* → iteration

LET

- Binds *collection variables* → one value

FOR vs. LET

```
FOR $x IN document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:

```
<result> <book>...</book></result>
<result> <book>...</book></result>
<result> <book>...</book></result>
...
```

```
LET $x := document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:

```
<result> <book>...</book>
         <book>...</book>
         <book>...</book>
         ...
</result>
```

COLLECTIONS IN XQUERY

- Ordered and unordered collections
  - /bib/book/author = an ordered collection
  - distinct(/bib/book/author) = an unordered collection
- LET $a = /bib/book → $a is a collection
- $b/author → a collection (several authors...)

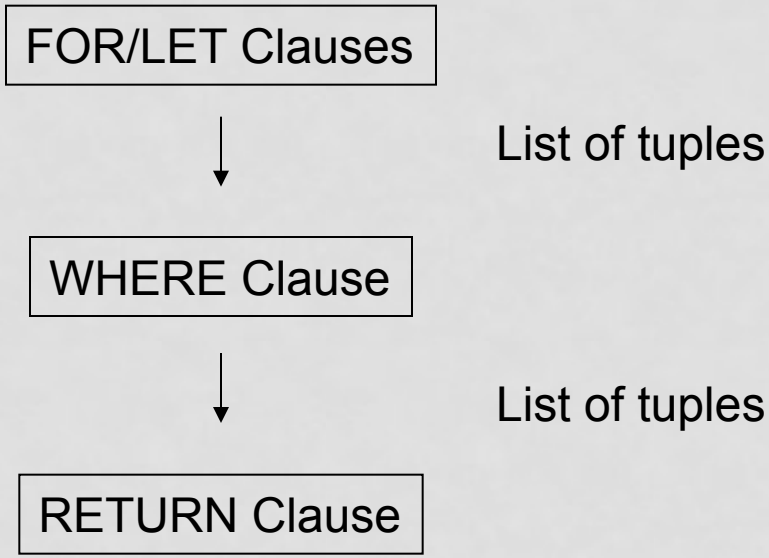
```
RETURN <result> $b/author </result>
```

Returns:

```
<result> <author>...</author>
         <author>...</author>
         <author>...</author>
         ...
</result>
```

XQUERY SUMMARY

FOR-LET-WHERE-RETURN = FLWR



SORTING IN XQUERY

```
<publisher_list>
   FOR $p IN distinct(document("bib.xml")//publisher)
   RETURN <publisher> <name> $p/text() </name> ,
                  FOR $b IN document("bib.xml")//book[publisher = $p]
                  RETURN <book>
                                    $b/title ,
                                    $b/@price
                               </book> SORTBY (price DESCENDING)
              </publisher> SORTBY (name)
</publisher_list>
```

IF-THEN-ELSE

```
FOR $h IN //holding

RETURN <holding>

                $h/title,

                IF $h/@type = "Journal"

                        THEN $h/editor

                ELSE $h/author

            </holding> SORTBY (title)
```

EXISTENTIAL QUANTIFIERS

```
FOR $b IN //book
WHERE SOME $p IN $b//para SATISFIES
   contains($p, "sailing")
   AND contains($p, "windsurfing")
RETURN $b/title
```

UNIVERSAL QUANTIFIERS

```
FOR $b IN //book
WHERE EVERY $p IN $b//para SATISFIES
   contains($p, "sailing")
RETURN $b/title
```

EXAMPLE: XML SOURCE DOCUMENTS

Invoice.xml

```
<Invoice_Document>
  <invoice>
      <account_number>2 </account_number>
      <carrier>AT&T</carrier>
      <total>$0.25</total>
  </invoice>
 <invoice>
      <account_number>1 </account_number>
      <carrier>Sprint</carrier>
      <total>$1.20</total>
  </invoice>
 <invoice>
      <account_number>1 </account_number>
      <total>$0.75</total>
  </invoice>
<auditor> maria </auditor>
</Invoice_Document>
```

Customer.xml

```
<Customer_Document>
  <customer>
      <account>1 </account>
      <name>Tom </name>
   </customer >
   <customer>
      <account>2 </account>
      <name>George </name>
   </customer >
</Customer _Document>
```

XQUERY EXAMPLE

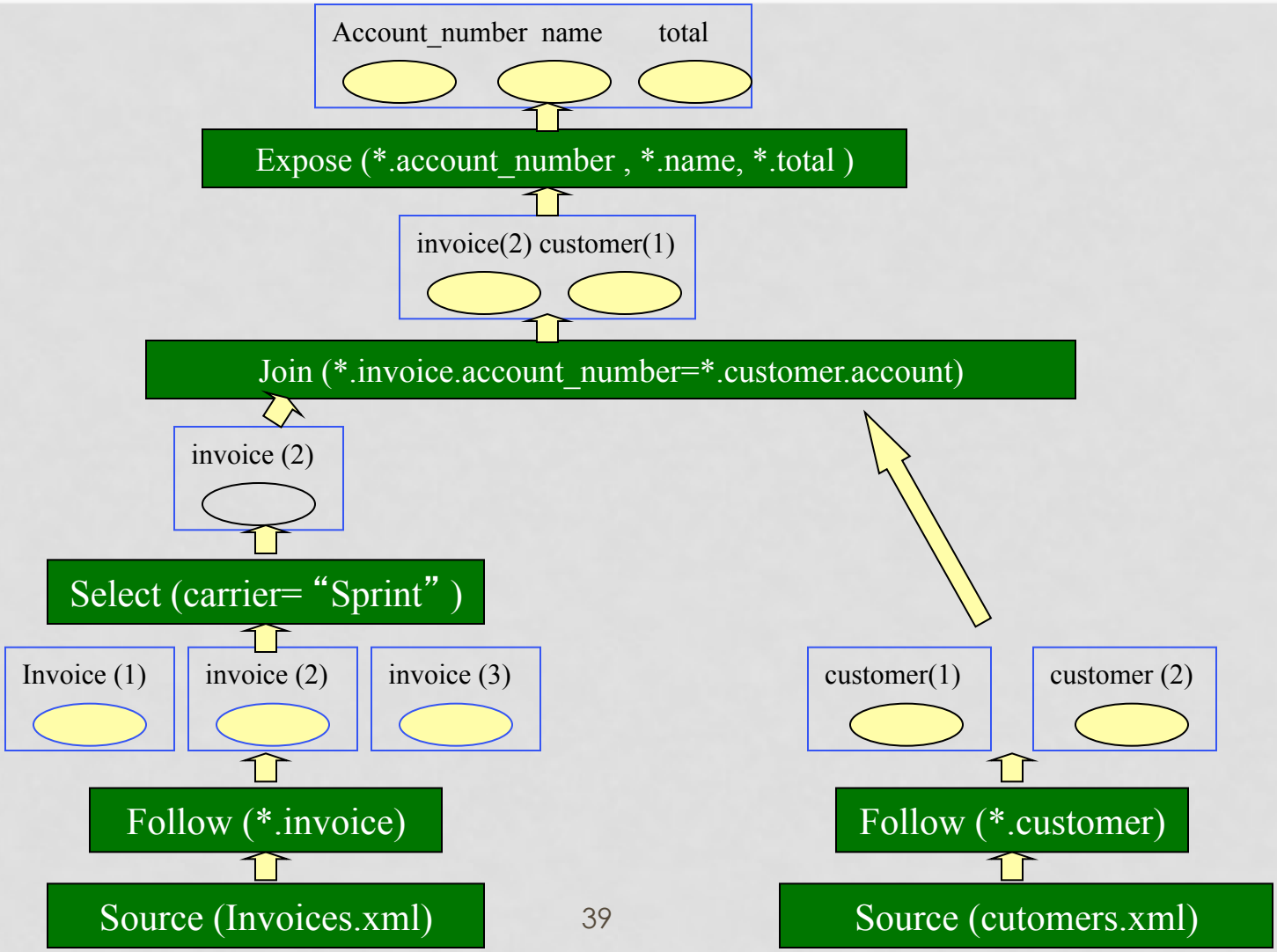
List account number, customer name, and invoice total for all invoices that have carrier = "Sprint".

```
FOR $i in (invoices.xml)//invoice,
        $c in (customers.xml)//customer
WHERE $i/carrier = "Sprint" and
          $i/account_number= $c/account
RETURN
              <Sprint_invoices>
                 $i/account_number,
                 $c/name,
                 $i/total
              </Sprint_invoices>
```

EXAMPLE: XQUERY OUTPUT

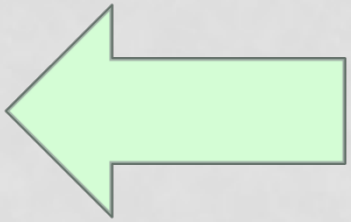
```
<Sprint_Invoice>
    <account_number>1 </account_number>
    <name>Tom </name>
    <total>$1.20</total>
 </Sprint_Invoice >
```

ALGEBRA TREE EXECUTION



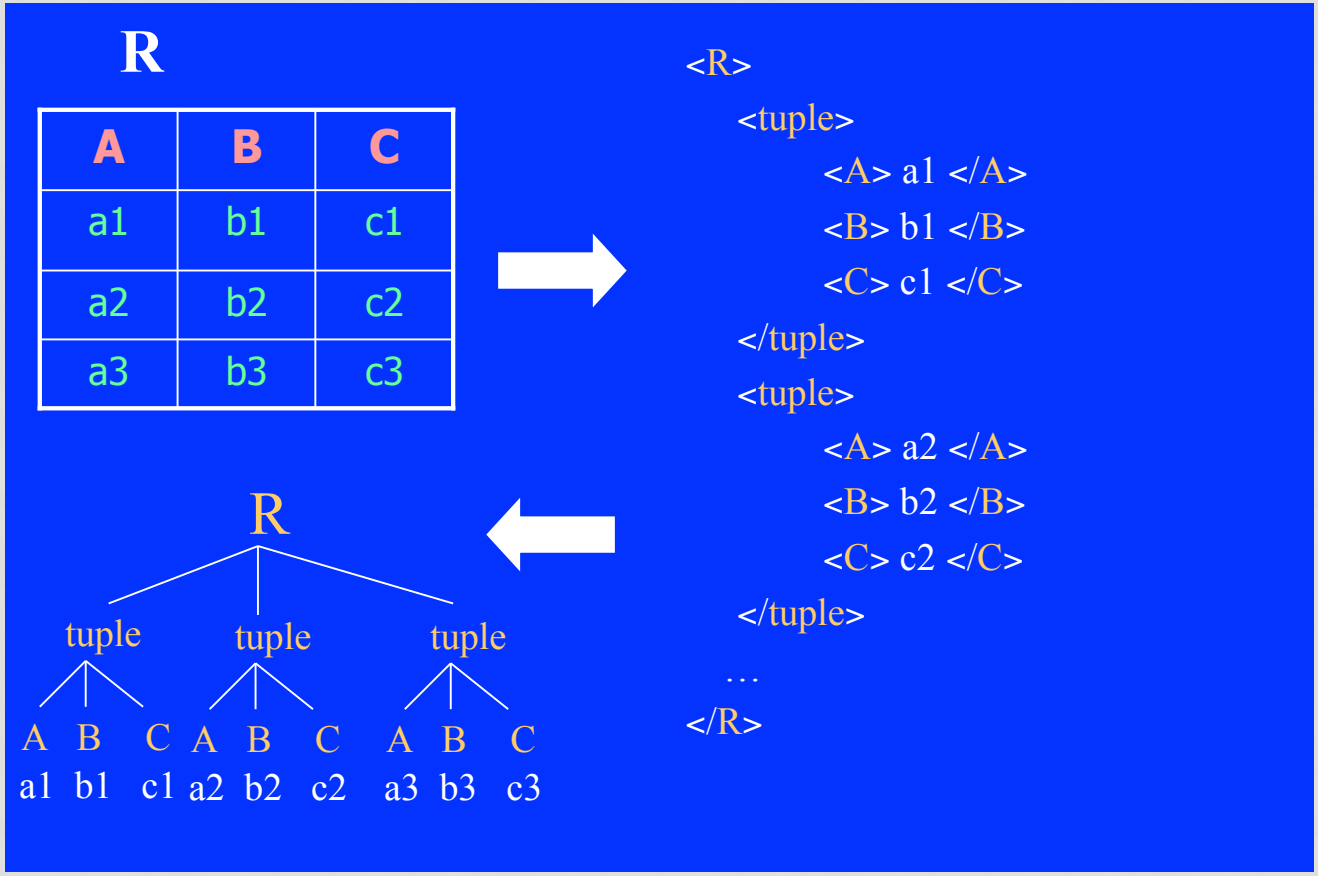
ROADMAP

- **Models for Parsing XML Documents**
- **XPath Language**
- **XQuery Language**
- **XML inside DBMSs**



XML vs. TABLES

Before providing a native support for XML, documents is translated back and forth to relational tables



GENERATING XML FROM RELATIONAL DATA

Step 1 : Set up the database connection

```
// Create an instance of the JDBC driver so that it has
// a chance to register itself
   Class.forName(sun.jdbc.odbc.JdbcOdbcDriver).newInstance();

// Create a new database connection.
  Connection con =
        DriverManager.getConnection(jdbc:odbc:myData, "", "");

// Create a statement object that we can execute queries with
  Statement stmt = con.createStatement();
```

GENERATING XML (CONT'D)

Step 2 : Execute the JDBC query

```
String query = "Select Name, Age from Customers";
ResultSet rs = stmt.executeQuery(query);
```

GENERATING XML (CONT'D)

Step 3 : Create the XML!

```
StringBuffer xml = "<?xml version= '1.0' ?
  ><myDatabase><customers>";

while (rs.next()) {
  xml.append("<custRec><custName>");
      xml.append(rs.getString("Name"));
      xml.append("</custName><custAge>");
      xml.append(rs.getInt("Age"));
  xml.append("</custAge></custRec>");
}
xml.append("</customers></myDatabase>");
```

STORING XML IN RELATIONAL TABLES

Step 1 : Set up the parser

```
StringReader stringReader = new StringRead(xmlString);
InputSource inputSource = new InputSource(stringReader);

DOMParser domParser = new DOMParser();
domParser.parse(inputSource);
Document document = domParser.getDocument();
```

STORING XML (CONT'D)

Step 2 : Read values from parsed XML document

```
NodeList nameList = doc.getElementsByTagName("custName");
NodeList ageList = doc.getElementsByTagName("custAge");
```

STORING XML (CONT'D)

Step 3 : Set up database connection

```
Class.forName(sun.jdbc.odbc.JdbcOdbcDriver).newInstance();

Connection con = DriverManager.getConnection
                          (jdbc:odbc:myDataBase, "", "");

Statement stmt = con.createStatement();
```

STORING XML (CONT'D)

Step 4 : Insert data using appropriate JDBC update query

```
String sql = "INSERT INTO Customers (Name, Age) VALUES (?,?)";

PreparedStatement pstmt = conn.prepareStatement(sql);

int size = nameList.getLength();

for (int i = 0; i < size; i++) {
        pstmt. setString(1, nameList.item(i).getFirstChild
  ().getNodeValue());
        pstmt.setInt(2, ageList.item(i).getFirstChild
  ().getNodeValue());
        pstmt.executeUpdate(sql);
}
```

USE CASE: ORACLE 10g

- New Data type "**XMLType**" to store XML documents

- **Before XMLType**
  - Either parse the documents and store it in relational tables
  - Or, store the documents in CLOB, BLOB

```
Creating a Table with an XMLType Column
CREATE TABLE mytable1 (key_column VARCHAR2(10) PRIMARY KEY,
                       xml_column XMLType);


Creating a Table of XMLType
CREATE TABLE mytable2 OF XMLType;


Creating a Table of XMLType with schema
CREATE TABLE mytable3 OF XMLType  XMLSCHEMA "....";
```

INSERTING XML INTO ORACLE

Many ways…

Inserting XML Content into an XMLType Table using SQL

```
CREATE DIRECTORY xmldir AS path_to_folder_containing_XML_file';
INSERT INTO mytable2 VALUES (XMLType(bfilename('XMLDIR', 'purchaseOrder.xml'),
                                nls_charset_id('AL32UTF8')));
```

Inserting XML Content into an XMLType Table using Java/DOM

```
public void doInsert(Connection conn, Document doc)
throws Exception
{
   String SQLTEXT = "INSERT INTO purchaseorder VALUES (?)";
   XMLType xml = null;
   xml = XMLType.createXML(conn,doc);
   OraclePreparedStatement sqlStatement = null;
   sqlStatement = (OraclePreparedStatement) conn.prepareStatement(SQLTEXT);
   sqlStatement.setObject(1,xml);
   sqlStatement.execute();
}
```

ORACLE XPATH OPERATIONS/ FUNCTIONS

```
SELECT warehouse_id, warehouse_name,
ExtractValue(warehouse_spec, '/Warehouse/Building/Owner') "Prop.Owner"
FROM warehouses
WHERE ExistsNode(warehouse_spec, '/Warehouse/Building/Owner') = 1;
```

```
Update warehouse
SET warehouse_spec = AppendChildXML(warehouse_spec,
'Warehouse/Building', XMLType('<Owner>Grandco</Owner>')
Where ExtractValue(warehouse_spec, '//Warehouse/Building') = 'Rented';
```

```
UPDATE warehouses
SET warehouse_spec = InsertChildXML(warehouse_spec,
'/Warehouse/Building', 'Owner', XMLType('<Owner>LesserCo</Owner>'))
WHERE warehouse_id = 3;
```

MORE XPATH EXAMPLES

```
SELECT XMLDIFF(
XMLTYPE('<?xml version="1.0"
<bk:book xmlns:bk="http://nosuchsite.com">
  <bk:tr>
    <bk:td>
      <bk:chapter>
        Chapter 1.
      </bk:chapter>
    </bk:td>
    <bk:td>
      <bk:chapter>
         Chapter 2.
       </bk:chapter>
    </bk:td>
  </bk:tr>
</bk:book>'),
XMLTYPE('<?xml version="1.0"
<bk:book xmlns:bk="http://nosuchsite.com">
  <bk:tr>
    <bk:td>
      <bk:chapter>
        Chapter 1.
      </bk:chapter>
    </bk:td>
    <bk:td/>
  </bk:tr>
</bk:book>'))
FROM DUAL;
```

ORACLE XQUERY

```
CREATE TABLE person_data (
person_id   NUMBER(3),
person_data XMLTYPE);
```

```
INSERT INTO person_data
(person_id, person_data)
VALUES
(1, XMLTYPE('
 <PDRecord>
   <PDName>Daniel Morgan</PDName>
   <PDDOB>12/1/1951</PDDOB>
   <PDEmail>damorgan@u.washington.edu</PDEmail>
 </PDRecord>')
);

INSERT INTO person_data
(person_id, person_data)
VALUES
(2, XMLTYPE('
 <PDRecord>
   <PDName>Jack Cline</PDName>
   <PDDOB>5/17/1949</PDDOB>
   <PDEmail>damorgan@u.washington.edu</PDEmail>
 </PDRecord>')
);
```

```
SELECT person_id, XMLQuery(
'for $i in /PDRecord
 where $i /PDName = "Daniel Morgan"
 order by $i/PDName
 return $i/PDName'
passing by value person_data
RETURNING CONTENT) XMLData
FROM person_data;
```

WHAT WE COVERED

- **Models for Parsing XML Documents**
- **XPath Language**
- **XQuery Language**
- **XML inside DBMSs**