Databases and Information Systems 1
- XML storage and XPath implementation -

Dr. Rita Hartel
Fakultät EIM, Institut für Informatik
Universität Paderborn
WS 2011 / 2012
XML extensions of relational databases

- data are already stored in relational database (e.g. Oracle)
- extra data type generates XML:

```xml
select
  <customer name = Customer.name>
    <order>
      Customer.order
    </order>
    <address>
      Customer.address
    </address>
  </customer>
from Customer
where ...
```

- data are already stored in relational database (e.g. Oracle)
- extra data type generates XML:
XML extensions of relational databases

load XML data into relational database

- requires mapping
  (defining how to store data in RDB)

- mapping uses annotations
  in DTD or XML Schema

- requires DTD or XML Schema
- for strongly structured XML only
XML columns in relational databases

extra columns of type XML-Document (e.g. Microsoft)

- arbitrarily structured XML documents storable
- search in imported XML documents possible
- XPath queries internally mapped to ordinary queries

<table>
<thead>
<tr>
<th>date</th>
<th>orderID</th>
<th>XML document regarding the order</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><img src="image-url" alt="XML Diagram" /></td>
</tr>
</tbody>
</table>

... ...
XML-storage in databases

hybrid XML-database system (e.g. DB2 Universal Server):

- relational data in relational database
- efficient query evaluation in relational part of database
- XML data stored separately
- imported XML documents can be stored
- both database parts can be joined in queries
- efficient search in imported XML documents
Implementing XML by relational databases

- many different approaches
- one approach is based on label, first-child (fc) and next-sibling(ns)

```
<order>
  <customer>Meier</customer>
  <item>pc500</item>
</order>
```
Implementing XML by relational databases

- many different approaches
- one approach is based on label, first-child (fc) and next-sibling(ns)

```
<order>
  <customer>Meier</customer>
  <item>pc500</item>
</order>
```

<table>
<thead>
<tr>
<th>ID</th>
<th>Label(ID)</th>
<th>ID</th>
<th>fc(ID)</th>
<th>ID</th>
<th>ns(ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>root</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>order</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>customer</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>„Meier“</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>item</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>„pc500“</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementing XML axes by fc and ns axes

following-sibling  = ns^+  
child             = fc / (ns)^*  
descendant        = child+ = fc / (fc | ns)^*  
preceding-sibling = following-sibling^{-1} = (ns^{-1})^+  
parent            = child^{-1} = (ns^{-1})^* / fc^{-1}  
ancestor          = descendant^{-1} = (fc^{-1} | ns^{-1})^* / fc^{-1} 

<table>
<thead>
<tr>
<th>ID</th>
<th>Label(ID)</th>
<th>ID</th>
<th>fc(ID)</th>
<th>ID</th>
<th>ns(ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>root</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>order</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>customer</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>„Meier“</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>item</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>„pc500“</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementing XML axes by fc and ns axes

- following = ancestor-or-self/ following-sibling/descendant-or-self
- preceding = following⁻¹
  = ancestor-or-self/preceding-sibling/descendant-or-self

<table>
<thead>
<tr>
<th>ID</th>
<th>Label(ID)</th>
<th>fc(ID)</th>
<th>ns(ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>root</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>order</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>customer</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>„Meier“</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>item</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>„pc500“</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>
Implementing XPath queries by fc and ns axes

//customer  →  \{ C | \exists R: Label(R)="root" ∧ C ∈ descendant(R) ∧ Label(C)="customer"\}

Answer: //customer = \{3\}.  \( R = 1 \)
Implementing XPath queries by fc and ns axes

//order[ item]/customer → \{R | \exists R, O, I: Label(R) = “root“ ∧

0 ∈ descendant(R) ∧ Label(O) = “order“ ∧

I ∈ child(O) ∧ Label(I) = “item“ ∧

C ∈ child(O) ∧ Label(C) = “customer“ \} → C = \{3\}

<table>
<thead>
<tr>
<th>ID</th>
<th>Label(ID)</th>
<th>ID</th>
<th>fc(ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>root</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>order</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>customer</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>„Meier“</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>item</td>
<td>ID</td>
<td>ns(ID)</td>
</tr>
<tr>
<td>6</td>
<td>„pc500“</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Exercise: Rewrite and Evaluate the following query

//*[customer=“Meier“]/following::order

```xml
<customer name="Meier">  
  <order>  
    <item name="pc500">  
      <item name="Reich">  
        <item name="pc700">  
          <item name="Arm">  
            <item name="pc100">  
        </item>  
      </item>  
    </item>  
  </order>  
</customer>
```

```xml
<customer name="Meier">  
  <order>  
    <item name="pc500">  
      <item name="Reich">  
        <item name="pc700">  
          <item name="Arm">  
            <item name="pc100">  
        </item>  
      </item>  
    </item>  
  </order>  
</customer>
```
Generating and storing a binary XML tree

How to transform an XML document such that we can answer core XPath queries with only fc, ns and label?

Transform XML document into Binary Simple SAX Events:

1. XML file
2. SAX Events
3. Simple SAX Events (convert to elements only)
4. Binary Simple SAX Events

(Details given on the following slides)
How to treat different node types?

escape text nodes:

```
<E>
  text
</E>
```

```
<E>
  <=>text>
  </=text>
</E>
```

escape attribute nodes:

```
<E a="value">
</E>
```

```
<E>
  <=>a>
  <=>value>
  </=value>
</=a>
</E>
```

+ escape root node, comments, PIs → only elements remain
From SAX to a Simple SAX-Parser-API (1)

```java
public void startElement( String namespaceURI, String localName,
    String qName, Attributes attrs) throws SAXException {
    simpleSax.startElement(qName);
    for(int i=0; i<atts.getLength(); i++) { // for each attribute
        simpleSax.startElement("@" + atts.getQName(i));
        simpleSax.startElement("=" + atts.getValue(i));
        simpleSax.endElement();
    }
    simpleSax.endElement();
}

public void endElement( String namespaceURI, String localName,
    String qName) throws SAXException {
    simpleSax.endElement();
}
```

...
From SAX to a Simple SAX-Parser-API (2)

```java
...

public void characters(char[] ch, int start, int length)
throws SAXException
{
    String text = new String(ch, start, length);
    text = text.trim();

    simpleSax.startElement("=" + text);
    simpleSax.endElement();
}

...
```
How to transform XML into a binary XML tree

1. Simplify → single node type (element nodes) only

2. generate binary tree and store binary tree

OR: generate and process binary SAX events (→ next slide)
From SAX events to binary SAX events

Pairs of SAX-Events

Location Step

end-element( _ )
start-element( a )

next-sibling :: a

start-element( _ )
start-element( a )

first-child :: a

end-element( _ )
end-element( _ )

(nothing)

start-element( _ )
end-element( _ )

parent :: *

end-element( _ )
start-element( a )
start-element( _ )
Conclusions: XPath implemented by fc and ns

- only the following base axes (and there inverse axes) or Simple SAX Events are needed:
  - Label( ID), fc( Parent), ns( Node)
- remaining XPath evaluation can be done by “concatenation” of these axes
- when storage format is changed, only implement Label, fc, ns
- If compressed XML supports implementation of Label, fc, ns, then core XPath can be evaluated
Summary and Conclusions

- XPath evaluation can be implemented, if XML storage format supports fc, ns, label

- XML storage in a relational database:
  - store relationships fc, ns, label in table(s)

- Pre-processing of the XML input:
  - XML document $\rightarrow$ SAX-Events $\rightarrow$ Simple SAX events $\rightarrow$ Binary Simple SAX Events
Summary