Advanced databases and data models:
Theme3: Efficient storage of XML

Lena Strömbäck

Today's lecture
Native XML management
Shredding
Hybrid solutions
SQL/XML
HShreX
Efficiency

XML as a data model

XML is richer than the relational model
  Tree structure,
  Order
...
Vary from highly structured to unstructured
  Database export
...
Annotated text documents
Can contain links to other type of entities

What does this mean for efficient storage?
Native XML databases

- Defines a (logical) model for an XML document
- Elements, attributes, PCDATA, document order.
- The XML document is the logical unit of storage
- Can have any physical underlying storage model
- Often: grouping documents, collections
- Query model: XPath and in most cases XQuery

Examples:
- eXist http://exist-db.org/
- MarkLogic http://www.marklogic.com

Storage possibilities for XML

<table>
<thead>
<tr>
<th>Mapping layer</th>
<th>Native XML DBMS</th>
<th>Relational DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage design</td>
<td>Loading XML Docs</td>
<td>Loading SQL Tuples</td>
</tr>
<tr>
<td>Quering XQuery</td>
<td>XML Results</td>
<td>Relational Results</td>
</tr>
<tr>
<td>XML schema</td>
<td>XQuery</td>
<td>SQL</td>
</tr>
<tr>
<td>Docs</td>
<td>XML</td>
<td>Mapping schema</td>
</tr>
<tr>
<td>XML schema</td>
<td>SQL</td>
<td></td>
</tr>
</tbody>
</table>
| Related work

Shredding of XML into relational storage

- Bohannon et al. 2002, (ICDE),
- Florescu and Cossman 1999 (IEEE Data Eng),
- Georgiadis and Vassalos 2007 (SIGMOD),
- Grust et al. (2007) (SIMOD),
- Mlynkova 2009 (DEXA)
Hybrid XML Storage

XML data
XQuery
XML results
Mapping layer
XML data
SQL/XML
XML/Relational results

Hybrid DBMS

New possibilities…

Species:

<table>
<thead>
<tr>
<th>Species</th>
<th>Reaction</th>
<th>Kinetic Law</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reactants:

<table>
<thead>
<tr>
<th>Species</th>
<th>Reaction</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Products:

<table>
<thead>
<tr>
<th>Species</th>
<th>Reaction</th>
<th>Annotation</th>
</tr>
</thead>
</table>

XML type

CREATE TABLE:
create table sbml_data (sbml_doc XML);
create table reaction (id varchar(100) not null, name varchar(250), math XML, annotation XML, primary key(id))

Hybrid storage and SQL/XML

Hybrid XML storage
Beyer et al.
Hua Liu et al.
Rys et al. (2005) (SIGMOD)

Proposed as SQL standard
Available in: DB2, Microsoft SQL Server, Oracle
More limited versions in other databases….
Querying

```sql
select sbml_doc.query('/sbml/model/listOfSpecies/species[@id = "C2"]')
from sbml_data;
```

```xml
<species id="C2" name="cdc2k" compartment="cell">
  <annotation> … </annotation>
</species>
```

Querying

```sql
select reactome_doc.query("for $react in /model/listOfReactions/reaction
return <path>
  <from> {data($react/listOfReactants/speciesReference/@species)}
  </from>
  <via>{( $ @ )}</via>
  <to> {data($react/listOfProducts/speciesReference/@species)} </to>
</path>")
from reactome_data;
```

```xml
<path><from>M</from><via>Reaction1</via><to>C2</to></path>
<path><from>M</from><via>Reaction1</via><to>YP</to></path>
...
```

Transforming data to relations

```sql
insert into reaction(id, name, annotation, formula)
select r.value('@id','varchar(100)') as id,
       r.value('@name','varchar(250)') as name,
       r.value('annotation','xml') as annotation,
       r.value("

transform data to XML

```sql
select reaction.reaction as name, reaction.id,
       (select (select speciesReference.reactant as species from reactants
               speciesReference where reaction.id = speciesReference.id
               for XML auto, type)
      from emptyXML as listOfReactants for XML auto, type),
       (select (select speciesReference.product as species from products
               speciesReference where reaction.id = speciesReference.id
               for XML auto, type)
      from emptyXML as listOfProducts for XML auto, type) ,
       from reaction
       for XML auto, type
```
Syntax differs between DBMS's (DB2)

```xml
select xml_doc.query(
    'for $y in /model/listOfReactions/reaction/listOfModifiers/modifierSpeciesReference,
    $z in /model/listOfSpecies/species[@id = $y/@species]
    return <modifier> {$y/@species} {$y/../../@id} {$z/@compartment} </modifier>
')
from reactome_data;
```

Efficiency:
Increasing query complexity

Efficiency:
Combining representations

Efficiency:
Return the result as XML
Higher complexity

Numerous alternatives for XML storage
How to achieve models that are efficient and easy to use
Dependent on application, XML data and query load

What are we doing?

Efficiency studies
Guidelines
Tool development

Guidelines:

Keep together what naturally belong together
Do not shred parts of the XML where the schema allows large variation
Take variations of the actual data into account
Shred elements that are critical for performance
Prefer the representation that is required for query results
Avoid shredding where future versions of the schema is likely to change.
Avoid shredding if parallel versions of data is to be kept

HShreX – a tool for evaluation

Extension of an old tool Shrex to allow hybrid storage
<table>
<thead>
<tr>
<th>Person</th>
<th>1</th>
<th>0</th>
<th>Lena Lektor</th>
<th>Lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>School</td>
<td>0-0</td>
<td>0-0</td>
<td>Ludvig Skolan</td>
<td>Student</td>
</tr>
</tbody>
</table>

**Working with HShreX:**

```xml
<xs:element name="families">
  <xs:complexType name="familyType">
    <!-- Family data here -->
  </xs:complexType>
</xs:element>
<xs:element name="families_family">
  <xs:complexType name="familyType">
    <!-- Family data here -->
  </xs:complexType>
</xs:element>
<xs:element name="families_family_child">
  <xs:complexType name="familyType">
    <!-- Family data here -->
  </xs:complexType>
</xs:element>
</xs:schema>
```

**Working with HShreX:**

```xml
<xs:element name="families">
  <xs:complexType name="familyType">
    <!-- Family data here -->
  </xs:complexType>
</xs:element>
<xs:element name="families_family">
  <xs:complexType name="familyType">
    <!-- Family data here -->
  </xs:complexType>
</xs:element>
<xs:element name="families_family_child">
  <xs:complexType name="familyType">
    <!-- Family data here -->
  </xs:complexType>
</xs:element>
</xs:schema>
```
HShrex Demo

Use case: Modelling

<table>
<thead>
<tr>
<th>Data model</th>
<th>Native</th>
<th>Mixed</th>
<th>Shredded</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBML</td>
<td>1</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8</td>
<td>121</td>
</tr>
<tr>
<td>UniProt</td>
<td>1</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>12</td>
<td>121</td>
</tr>
</tbody>
</table>

Annotations

maptoxml
ignore
withparenttable
outline
tablename
fieldname
sqltype

Use case: Querying

Mixed:

```sql
select m.species
from reaction r, modifier m
where r.shrex_id = m.shrex_pid and
  r.name = React1;
```

Shredded:

```sql
select accession,
m.species
from sbml_model_listOfReactions_reaction r,
     sbml_model_listOfReactions_reaction_l
listOfModifiers_modifiersSpeciesReference p,
     sbml_model_listOfReactions_reaction_l
listOfModifiers lm
where m.shrex_pid = lm.shrex_id and
  lm.shrex_pid = r.shrex_id and
  r.name = React1';
```

Native:

```sql
select entry.query('data(//uniprot:accession)')
from uniprot_entry
where entry.exist(//uniprot:subcellularLocation[uniprot:location/text()="Cytoplasm"]')=1;
```

Mixed:

```sql
select accession
from uniprot_entry_accession, entry,
     uniprot_entry_comment
where
  comment.exist(//uniprot:subcellularLocation[uniprot:location/text()="Cytoplasm"]')=1
  and uniprot_entry_comment.shrex_pid = entry.shrex_id
  and uniprot_entry_accession.shrex_pid = entry.shrex_id;
```
Use case: Efficiency

Lab exercises

Use HShreX for performance studies.

Three suggested datasets:
- SBML (Reactome)
- SBML (Biomodels)
- PSI MI
- Michigan Benchmark

If own data – check with the lab assistant

For + select own of several tasks

NoSQL – non relational databases

Examples:
- **Document store**: CouchDB, ApacheDB
- **XML database**: Marklogic Server, eXist
- **Graph**: AllegroGraph, Neo4j
- **Object database**: GemStone/S
- **Key/Value store on disk**: BigTable
- **Eventually consistent key-value store**: Cassandra
- **Ordered Key-value store**: Berkeley DB
- **Tabular**: BigTable, HyperTable, Hbase
- **Tuple store**: Apache River

Neo

Neo4j is a **graph database**. It is an embedded, disk-based, fully transactional Java persistence engine that stores data structured in graphs rather than in tables.

Linköping related company.

Interesting for semi-structured data.
The Neo Persistence Engine

- **Primitives:** nodes, relationships, properties
- **Features:**
  - ACID transaction
  - Durable persistence
  - Transaction recovery
- **Implementation:** Java

Representing XML in Neo: Basic solution

Yahoo!: A web of concepts

- Concept: Things of interest to the users of the web.
- Concept represented as:
  - Id
  - meta-data (attributes with values)
- Goals: Concept centric data organization
- What to support:
  - Nested structure?
  - Provenance, versions uncertainty?
  - Relations between concepts?
What do we search for?

Individual concepts: 60-70%  Sets of concepts 10-20%

Attributes of a concept:
   Rather small correlation (restaurant menu 3%)

Aggregation: 59% of users click on more than one URL.

Concepts vs. Browsing: Follow paths of how user browsed.
   Easy to find patterns of what users commonly visit.