Advanced databases and data models:

Theme2: Query languages

Lena Strömbäck

Today´s lecture

Different kinds of query languages
Languages
- Lorel
- XPath
- XQuery
- SPARQL

Transformation of data
- XSL
- FOR XML

Data guides – why and what

Semi-structured data - properties

Data model/data guide could be supportive or a hinder while querying

Data model/guide changes commonly

Object can change type/class

The distingsion between data and schema is blurred

Types of query languages

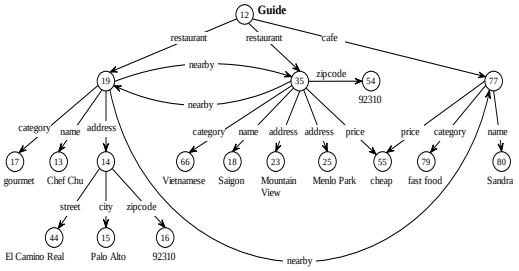
Typically:

Data	Unstructured	Semi-structured	Structured
Example	Text	XML, RDF, OEM	Relational database
Typical query language	Keywords	XPath, XQuery, Lorel …	SQL, ….
Typical user	Every day		Expert

However, this is changing… (more in coming lectures)

OEM example

Restaurant Guide



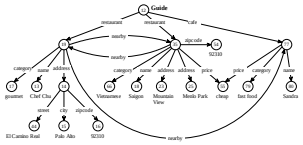
Lorel query language

Wildcards and variables

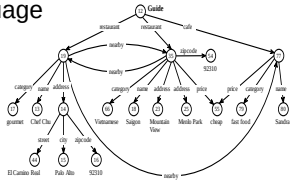
? - 0 or 1 path
+ - 1 or more paths
* - 0 or more paths
# - any path
% - 0 or more chars

- object variables
  select P from Guide.% P
  select A from #.address{A}
- path variables
  select Guide.#@P.name

Do your recognize this??



Lorel query language



1. Find all places to eat Vietnamese food
   select P
   from RestaurantGuide.% P
   where P.category grep "ietnamese"

2. Find the names and streets of all restaurants in Palo Alto
   select R.name, A.street
   from RestaurantGuide.restaurant{R}.address A
   where A.city = "Palo Alto"

3. Find all restaurants to eat with zipcode 92310
   select RestaurantGuide.restaurant
   where RestaurantGuide.restaurant(.address)?.zipcode = 92310

Exercise

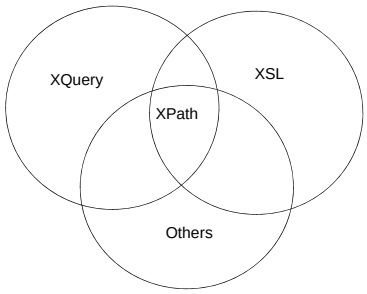
Using the data model from the previous question, formulate the following queries using Lorel:

find all the restaurants that are located in Linkoping

find the address (city and street) of the "Hamlet" restaurant

list the restaurants by city (equivalent of GROUP BY)

XPath/XQuery/XSLT



XPath

Objects:
- Nodes
- Atomicvalues
- Items

Relations:
- Parent
- Children
- Ancestor
- Descendants

```
<minimodel name="sugartransport" >
  <listOfCompartments>
    <compartment id="blood" name="inblood" />
    <compartment id="cell" name="musclecell" />
  </listOfCompartments>
  <listOfSpecies>
    <species id="sug1" name="sugarinblood" compartment="blood" />
    <species compartment="ins" id="insulin" name="blood"/>
    <species compartment="sug2" id="sugarincell" name="cell"/>
    <species compartment="en" id="energy" name="cell"/>
  </listOfSpecies>
  <listOfReactions>
    <reaction id="tocell" name="sugartocell">
      <listOfReactants>
        <speciesReference species="sug1"/>
        <speciesReference species="ins"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="sug2"/>
      </listOfProducts>
    </reaction>
    <reaction id="move" name="makemovement">
      <listOfReactants>
        <speciesReference species="sug2"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="en"/>
      </listOfProducts>
    </reaction>
  </listOfReactions>
</minimodel>
```

XPath

Node selection:
- minimodel
- /minimodel
- //species
- //species/@id
- //species/..

Predicates:
- //species[@id='ins']

Wildcards:
- //*
- //reaction/@*

Several paths:
- //species|//compartment

And a lot of functions for selecting ancestors etc.

```
<minimodel name="sugartransport" >
  <listOfCompartments>
    <compartment id="blood" name="inblood" />
    <compartment id="cell" name="musclecell" />
  </listOfCompartments>
  <listOfSpecies>
    <species id="sug1" name="sugarinblood" compartment="blood" />
    <species compartment="ins" id="insulin" name="blood"/>
    <species compartment="sug2" id="sugarincell" name="cell"/>
    <species compartment="en" id="energy" name="cell"/>
  </listOfSpecies>
  <listOfReactions>
    <reaction id="tocell" name="sugartocell">
      <listOfReactants>
        <speciesReference species="sug1"/>
        <speciesReference species="ins"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="sug2"/>
      </listOfProducts>
    </reaction>
    <reaction id="move" name="makemovement">
      <listOfReactants>
        <speciesReference species="sug2"/>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="en"/>
      </listOfProducts>
    </reaction>
  </listOfReactions>
</minimodel>
```

```
data(doc('minimodel.xml')//compartment[@id='blood']/@name)
```

XQuery

XML Query language

```
for $x in doc('minimodel.xml')//species
    where data($x/@compartment)="blood"
    return $x
```

XPath vs. XQuery

When do you need XQuery?

XQuery allows many variations

```
for $x in doc('minimodel.xml')//species,
    $y in doc('minimodel.xml')//compartment
where data($x/@compartment)=data($y/@id)
return
    <pair> {data($x/@name)} is inside {data($y/@name)} </pair>

for $x in doc('minimodel.xml')//species,
    $y in doc('minimodel.xml')//compartment
          [@id=data($x/@compartment)]
return
    <pair> {data($x/@name)} is inside {data($y/@name)} </pair>
```

XQuery functions

Declare local functions
Can be recursive!

```
declare function local:fak($n)
    {if ($n=1)
      then 1
      else ($n * local:fak($n - 1))
    };
<res>{local:fak(3)}</res>
```

XQuery functions

```
declare function local:next($r)
    { <next>
    {data($r/listOfProducts/speciesReference/@species)}
    {for $x in $r/listOfProducts/speciesReference,
         $y in doc('minimodel.xml')
                  //listOfReactants/speciesReference
                  [@species=data($x/@species)]
    return local:next($y/../..)}
    </next>};
for $x in doc('minimodel.xml')//species,
    $y in doc('minimodel.xml')
          //listOfReactants/speciesReference
          [@species=data($x/@id)]
return  <answer> {data($x/@id)} leads to
                    {local:next($y/../..)} </answer>
```

XQuery - tips

- The Xpath expressions can return many answers.
- Attribute values – use data()
- Start from something simple – build on it.

SPARQL – Query language for RDF

```
PREFIX books:   <http://example.org/book/>
PREFIX dc:    <http://purl.org/dc/elements/1.1/>
SELECT ?book ?title
WHERE   { ?book dc:title ?title }

PREFIX dc:    <http://purl.org/dc/elements/1.1/>
CONSTRUCT { $book dc:title $title }
WHERE   { $book dc:title $title }
```

Transformation: XSL

```
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
</cd>
....
```

Transform XML to another format

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by XMLSpy® -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title" /></td>
        <td><xsl:value-of select="artist" /></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Transformation: Relations -> XML

```
select xmlagg(xmlelement("speciesReference",
    xmlattributes(reactants.reactant as species))
from reactants where reaction.id=reactants.id)
```

```
<speciesReference  species="insulin">
```

Transformation: Relations -> XML

```
select xmlelement("reaction", xmlattributes(reaction.reaction as
    name, reaction.id),
      xmlelement("listOfReactants",
        (select xmlagg(xmlelement("speciesReference",
    xmlattributes(reactants.reactant as species))
         from reactants where reaction.id=reactants.id))
      xmlelement("listOfProducts",
        (select xmlagg(xmlelement("speciesReference",
    xmlattributes(products.product as species))
         from products  where reaction.id=products.id))
from reaction
```

Transformation: Relations -> XML

```
select speciesReference.reactant as species
from reactants speciesReference
where reaction.id = speciesReference.id
for XML auto, type
```

<speciesReference species="insulin">

Transformation: Relations -> XML

```
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
```

Lab exercises:

Construct queries for your relational and XML data.

Record what is hard or easy.

Answer questions, compare and write report.

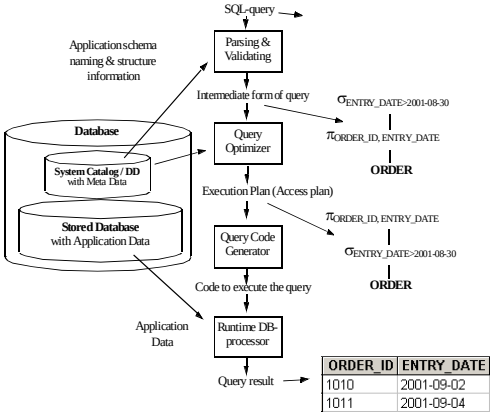
For +: XSLT and XML generation for XML

Types of query languages

Typically:

Data	Unstructured	Semi-structured	Structured
Example	Text	XML, RDF, OEM	Relational database
Typical query language	Keywords	XPath, XQuery, Lorel …	SQL, ….
Typical user	Every day		Expert

For efficient query processing,
the data structure is important.



Heuristic optimisation

Idéa: Do selection and projection first, join as late as possible

Algorithm:

- **Break up conjunctive select into cascades**
- **Move down select as far as possible in the tree**
- **Rearrange select operations – most restrictive first**
- **Convert cross product to join with the appropriate join condition from a selection**
- **Move down project operations as far as possible in the tree**
- **Identify subtrees that can be executed by a single algorithm**

SQL

Incompleteness – NULL – Inner and outer join.

Name	Phone	Address
Ludvig	12345	NULL
Lena	23456	NULL

Incompleteness of XQuery

(a) node ids (they can be replaced by node variables);
(b) node labels (they can be replaced by wildcards);
(c) precise vertical relationship between nodes (we can use descendant edges in addition to child edges);
(d) precise horizontal relationship between nodes (using younger-sibling edges instead of next-sibling).

Data Guides

A structural summary over a databank that is used as a dynamic schema

Is used in query formulation and optimization

Is often created a posteriori

Properties:
concise
accurate
convenient

Data Guides - definitions

Label path: sequence of labels L1.L2. … .Ln

Data path: alternating sequence of labels and oid:s L1.o1.L2.o2. … .Ln.on

Data path *d* is an instance of label path *l* if the sequences of labels are identical in *l* and *d*.

Data Guides - definition

A data guide for object *s* is an object *d* such that every label path of *s* has exact one data path instance in *d*, and each label path in *d* is a label path of *s*.

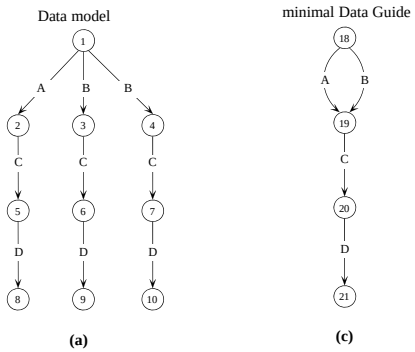
Data Guides

A databank can have several data guides

Minimal data guides

the smallest data guides

Data Guides - example



Minimal Data Guides

Concise

May be hard to maintain

Example: child node for 10 with label E

Strong Data Guides

Intuitively:

”label paths that reach the same set of objects in the data model = label paths that reach the same objects in the data guide”

Strong Data Guides - definitions

An object *o* can be reached from *s* via *l* if there is a data path of *s* that is an instance of *l* and that has *o* as last oid

(L1.o1.L2.o2. … Ln.o)

The target set for label path *l* in object *s* is the set of objects that can be reached from *s* via *l*. *Notation: T(s,l)*

L(s,l): set of label paths of *s* that have the same target set in *s* as *l*.

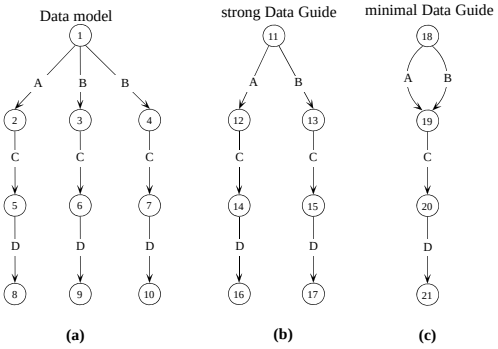
Strong Data Guides - definitions

Definition:

d is a strong data guide for *s* if
for all label paths *l* of *s* it holds that *L(s,l) = L(d,l)*

There is a 1-1-mapping between target sets in the data model and nodes in a strong data guide.

Data Guides - example



Strong Data Guides - algorithm

Implementation:

Traverse data model depth-first.

Each time you find a new target set for label path *l*, create a new object in the data guide.

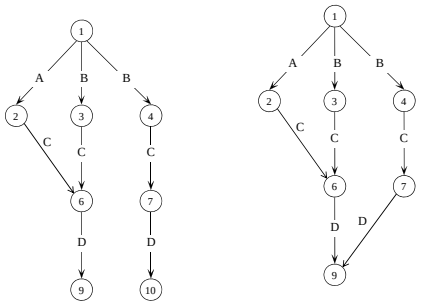
If the target set is already represented in the data guide, do not create a new object, but link to the existing object.

Strong Data Guides - use

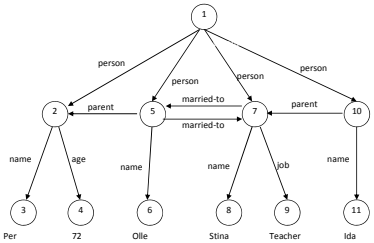
Easier to maintain

Used as path index for query optimization

Construct the strong data guide



Exercise: Construct the minimal and strong data guides.



Exercise

Draw the strong Data Guide for the restaurant guide data model below.