Universität Freiburg
Institut für Informatik
Prof. Dr. Peter Fischer
Lecture XML and Databases
Winter Semester 2011/12

Due date/discussion: 10.2.2012

Solution Sheet 11

XML and Relational Databases

Exercise 1: XML to Relational Mapping

1.1.

a. Schema-based shredding using *exercise10-1-a.xml* and *exercise10-1-a.xsd*

Since there is no nesting, this is straight-forward and very similar to exercise 3. Mapping the doc element is useless for the queries.

Passenger(passportNumber, name, address)
Airport(airpId, name, tax)
Flight(flightId, date, departure, arrival, seats, source, destination)
Reservation(flightRef, passportNumber)

Filling the table is straightforward as the XML file is almost already formatted as a table.

b. Schema-based shredding using *exercise10-1-bcd.xml* and *exercise10-1-bcd.xsd*

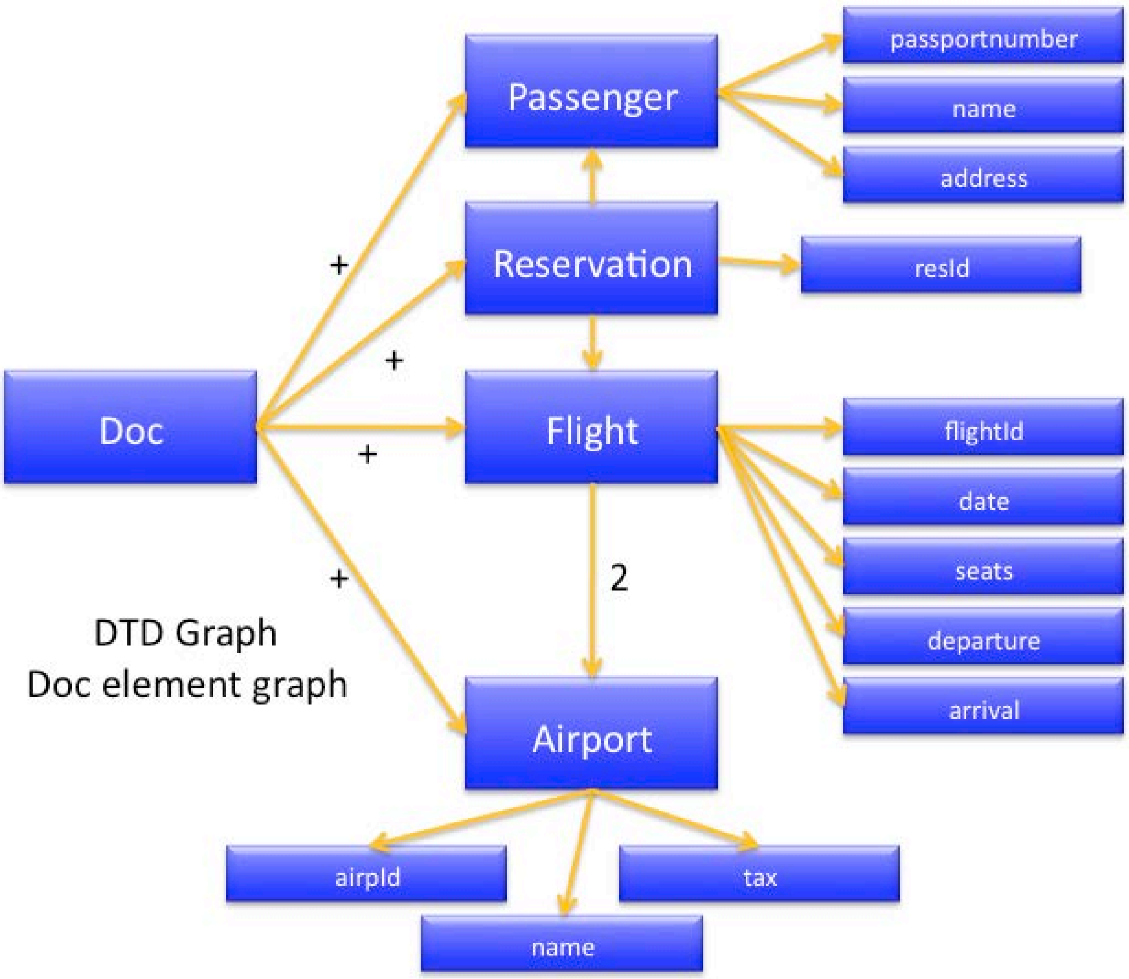
This time, flights and reservation no longer use foreign keys to reference airports, flights and passengers. Some shredding is needed.

Since schema-based shredding is based on DTDs, we use the following equivalent DTD:

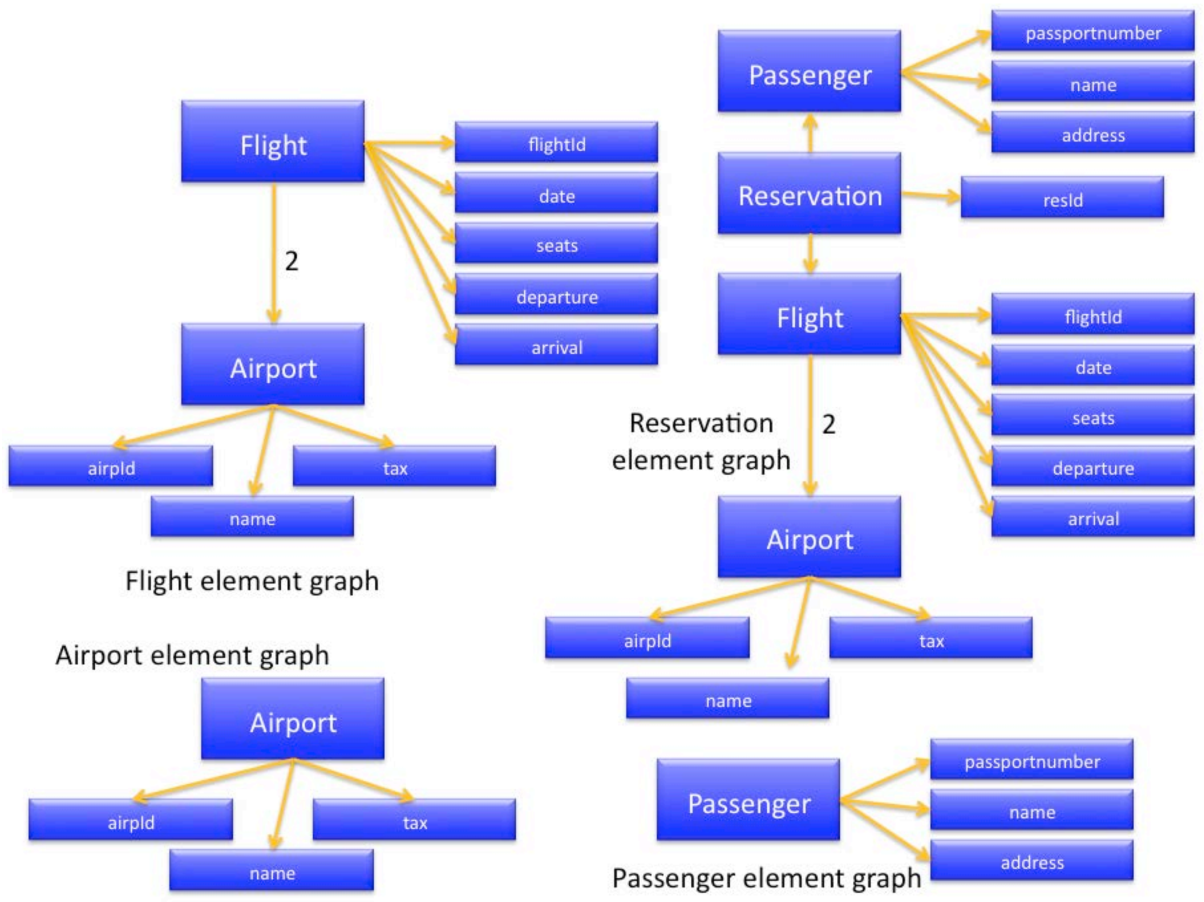
```
<!ELEMENT Doc(Airport+, Passenger+, Flight+, Reservation+ >
<!ELEMENT Passenger(passportNumber, name, address) >
<!ELEMENT Airport(airpId, name, tax) >
<!ELEMENT Flight(flightId, date, departure, arrival, seats,
                    Airport, Airport) >
<!ELEMENT Reservation(resId, Flight, Passenger)>
<!ELEMENT passportNumber #PCDATA>
<!ELEMENT name #PCDATA>
<!ELEMENT address #PCDATA>
<!ATTLIST Airport airpId CDATA>
<!ELEMENT tax #PCDATA>
<!ATTLIST Flight flightId CDATA>
<!ELEMENT date #PCDATA>
```

```
<!ELEMENT departure #PCDATA>
<!ELEMENT arrival #PCDATA>
<!ELEMENT seats #PCDATA>
<!ELEMENT resId #PCDATA>
```

The DTD Graph looks as follows:



Now, for each element, an element graph is done, which can be seen as an expansion of the DTD Graph starting at this element. Since there is no recursion, it is simply the subgraph starting at this node. We only show the graphs for Flight, Reservation, Airport and Passenger as the others are trivial (one single node).



For each of these element graphs, a relation is created for the root of the element graph. All descendants are inlined, provided they do not have a backpointer edge (recursion) and they do not occur more than once. If they do, a separate relation is made.
Attributes in the relations are named by the path from the root of the element graph. For each relation, an ID is created.

We also add an attribute "position" to keep track of document order.

So that the schema looks as follows:

For Doc:

Doc(DocID)

Doc.Passenger(Doc.PassengerID, Doc.Passenger.parentID, Doc.Passenger.passportNumber, Doc.Passenger.name, Doc.Passenger.address, position)

Doc.Airport(Doc.Airport.airpId, Doc.Airport.parentID, Doc.Airport.name, Doc.Airport.address, position)

Doc.Flight(Doc.FlightID, Doc.Flight.flightID, Doc.Flight.date, Doc.Flight.seats, Doc.Flight.departure, Doc.Flight.arrival, position)

Doc.Flight.Airport(Doc.Flight.AirportID, Doc.Flight.Airport.parentID, Doc.Flight.Airport.airpID, Doc.Flight.Airport.name, Doc.Flight.Airport.tax, position)

Doc.Reservation(Doc.ReservationID, Doc.Reservation.parentID, Doc.Reservation.resId, Doc.Reservation.Passenger.passportnumber, Doc.Reservation.Passenger.name, Doc.Reservation.Passenger.address, Doc.Reservation.Flight.flightID, Doc.Reservation.Flight.date, Doc.Reservation.Flight.seats, Doc.Reservation.Flight.departure, Doc.Reservation.Flight.arrival, position)

Doc.Reservation.Flight.Airport(Doc.Reservation.Flight.AirportID, Doc.Reservation.Flight.Airport.parentID, Doc.Reservation.Flight.Airport.airpId, Doc.Reservation.Flight.Airport.name, Doc.Reservation.Flight.Airport.tax, position)

For Reservation:

Reservation(ReservationID, Reservation.parentID, Reservation.resId, Reservation.Passenger.passportnumber, Reservation.Passenger.name, Reservation.Passenger.address, Reservation.Flight.flightID, Reservation.Flight.date, Reservation.Flight.seats, Reservation.Flight.departure, Reservation.Flight.arrival)

Reservation.Flight.Airport(Reservation.Flight.AirportID, Reservation.Flight.Airport.parentID, Reservation.Flight.Airport.airpId, Reservation.Flight.Airport.name, Reservation.Flight.Airport.tax, position)

For Flight:

Flight(FlightID, Flight.flightID, Flight.date, Flight.seats, Flight.departure, Flight.arrival)

Flight.Airport(Flight.AirportID, Flight.Airport.parentID, Flight.Airport.airpID, Flight.Airport.name, Flight.Airport.tax, position)

For Passenger:

Passenger(PassengerID, Passenger.parentID, Passenger.passportNumber, Passenger.name, Passenger.address)

For Airport:

Airport(Airport.airpId, Airport.parentID, Airport.name, Airport.address)

For the other elements:

```
name(nameID, name)
address(addressID, address)
…
```

With our document, the root of which is "Doc", we only have to fill tables beginning with "Doc".
Filling Doc, Doc.Airport and Doc.Passenger is straight forward (like in a.)
Filling Doc.Flight is done in the same way, ignoring the Airport children, which are put in Doc.Flight.Airport in the same way.
Filling Doc.Reservation is done in the same way, ignoring the Airport descendents, which are put in Doc.Reservation.Flight.Airport.

c. Edge approach using *exercise10-1-bcd.xml*

This is also independent from the schema used, as it is a generic approach. The effort to create the database schema is therefore minimal, however the queries will be more complicated.

One edge table:

```
Edge(Ordinal, Source, Label, Target)
```

Several value tables, one for each type:

```
ValueString(Id, Value)
ValueInteger(Id, Value)
ValueDate(Id, Value)
ValueTime(Id, Value)
ValueFloat(Id, Value)
```

Filling the table is done by:
- attributing a unique ID to each node in the document.
- filling Edge with all edges of the document (Source is the source ID, Target is the target ID, Label is the label of the edge, Ordinal its rank among its siblings)
- filling Value with all text node values (Id is the text node ID, Value its value)

d. Tree encoding using *exercise10-1-bcd.xml*

This is also independent from the schema used, as it is a generic approach.

```
Tree(pre, size, level, kind, prop, frag)
```

Filling the table is done by:
- attributing a unique integer ID to every node and attribute value (which are not nodes!) in the document
- filling Tree with all nodes and attribute values (pre is its integer iD, size is its number of descendents, level its level in the XML tree, kind its kind (element, attribute, text...),

prop is its name if it's an element or an attribute node, or its value if it is a text node or an attribute value, frag is the document ID).

1.2.

a. Schema-based shredding using *exercise10-1-a.xml* and *exercise10-1-a.xsd*

```
SELECT a.name FROM Reservation r, Flight f, Passenger p, Airport a
WHERE r.passRef = p.passportnumber
      AND p.name='Santa Claus'
      AND r.flightRef = f.flightId
      AND f.destination = a.airpId
```

b. Schema-based shredding using *exercise10-1-bcd.xml* and *exercise10-1-bcd.xsd*

```
SELECT a.name from Doc.Reservation r, Doc.Reservation.Flight.Airport a
WHERE a.Doc.Reservation.Flight.Airport.parentID = r.Doc.ReservationID
      AND r.Doc.Reservation.Passenger.name="Santa Claus"
      AND a.position = "2"
```

c. Edge approach using *exercise10-1-bcd.xml*

This is much more complicated, as we have to manually encode the navigation edge by edge and nest three queries:

```
Passenger_Edges
SELECT P
FROM Edge P, Edge N, ValueString Vpname
WHERE P.Label = "Passenger"
AND P.Target = N.Source
AND N.Label = "name"
AND N.Target = Vpname.Id
AND Vpname.Value = “Santa Claus”

Reservations_Edges
SELECT R
FROM Edge R, (Passenger_Edges) Passenger,
WHERE R.Name = "Reservation"
AND R.Target = Passenger.Source

Destination_Values:
SELECT Vaname.Value
FROM Edge F,
     Edge G,
     (Reservation_Edges) Reservation,
     ValueString Vaname
WHERE F.Source = Reservation.Target
AND F.Name = “Flight”
AND F.Target = G.Source
AND G.Name = "Airport"
AND G.Ordinal =  "2"
AND G.Target = Vaname.Id
```

The resulting query is:

```
SELECT Vaname.Value
FROM Edge F,
     Edge G,
     (SELECT R
      FROM Edge R,
           (SELECT P
            FROM Edge P, Edge N, ValueString Vpname
            WHERE P.Label = "Passenger"
            AND P.Target = N.Source
            AND N.Label = "name"
            AND N.Target = Vpname.Id
            AND Vpname.Value = “Santa Claus”
           ),
      WHERE R.Name = "Reservation"
      AND R.Target = P.Source
     ),
     ValueString Vaname
WHERE F.Source = R.Target
AND F.Name = “Flight”
AND F.Target = G.Source
AND G.Name = "Airport"
AND G.Ordinal =  "1"
AND G.Target = Vaname.Id
```

d. Tree encoding using *exercise10-1-bcd.xml*

We use the following predicate to test whether a is a child of b:

```
(a.pre > b.pre AND a.pre <= b.pre + b.size AND a.level = b.level+1)
```

Then the query is straightforward (we navigate through the tree):

```
SELECT dn.prop
FROM Tree p,
     Tree pn,
     Tree r,
     Tree f,
     Tree d,
     Tree dn
WHERE pn.prop="Santa Claus"
AND pn.kind="text"
AND (pn.pre > p.pre AND pn.pre <= p.pre + p.size AND pn.level = p.level+1)
AND p.prop="Passenger"
AND p.kind="element"
AND (p.pre > r.pre AND p.pre <= r.pre + r.size AND p.level = r.level+1)
AND r.prop="Reservation"
AND r.kind="element"
AND (f.pre > r.pre AND f.pre <= r.pre + r.size AND f.level = r.level+1)
AND f.prop="Flight"
AND f.kind="element"
AND (d.pre > f.pre AND d.pre <= f.pre + f.size AND d.level = f.level+1)
AND d.prop="Airport"
AND d.kind="element"
AND d.pre = (SELECT MAX(c.pre)
             FROM Tree c
             WHERE c.prop="Airport"
             AND c.kind="element"
             AND (c.pre > f.pre
                      AND c.pre <= f.pre + f.size and c.level = f.level+1)
            )
AND dn.kind="text
AND (dn.pre > d.pre AND dn.pre <= d.pre + d.size and dn.level = d.level+1)
```