XML Examples

Bank Information

- Basic structure:

```
<bank>
        <account>
            <account_number> A-101     </account_number>
            <branch_name>      Downtown </branch_name>
            <balance>              500         </balance>
        </account> …
        <customer>
            <customer_name> Johnson</customer_name>
            <customer_street>Alma</customer_street>
            <customer_city>Surrey</customer_city>
        </customer>…
        <depositor>
            <account_number> A-101     </account_number>
            <customer_name> Johnson </customer_name>
        </depositor>…
</bank>
```

Alternative Structure using Nested Elements

```
<bank-1>
    <customer>
        <customer_name> Hayes </customer_name>
        <customer_street> Main </customer_street>
        <customer_city>    Harrison </customer_city>
        <account>
            <account_number> A-102 </account_number>
            <branch_name>     Perryridge </branch_name>
            <balance>             400 </balance>
        </account>
        <account>
            …
        </account>
    </customer>
      .
      .
</bank-1>
```

Alternative Structure using Attributes

- <account acct-type = “checking” >
  <account_number> A-102 </account_number>
  <branch_name> Perryridge </branch_name>
  <balance> 400 </balance>
  </account>
- An element may have several attributes, but each attribute name can only occur once

  <account acct-type = “checking” monthly-fee=“5”>
- Same information can be represented in two ways
  - <account account_number = “A-101”> …. </account>
  - <account>
    <account_number>A-101</account_number> …
    </account>
- In general: use attributes for identifiers of elements, and use subelements for contents
  - Or: attributes for properties of the element and subelements for subparts.

Namespaces

- Might have something like:

```
<bank Xmlns:FB='http://www.FirstBank.com'>
      …
       <FB:branch>
            <FB:branchname>Downtown</FB:branchname>
            <FB:branchcity>    Brooklyn   </FB:branchcity>
       </FB:branch>
      …
</bank>
```

Bank DTD

```
<!DOCTYPE bank [
    <!ELEMENT bank ( ( account l customer l depositor)+)>
    <!ELEMENT account (account_number branch_name balance)>
    <! ELEMENT customer(customer_name customer_street
                                                    customer_city)>
    <! ELEMENT depositor (customer_name account_number)>
    <! ELEMENT account_number (#PCDATA)>
    <! ELEMENT branch_name (#PCDATA)>
    <! ELEMENT balance(#PCDATA)>
    <! ELEMENT customer_name(#PCDATA)>
    <! ELEMENT customer_street(#PCDATA)>
    <! ELEMENT customer_city(#PCDATA)>
]>
```

Bank DTD with Attributes

- Bank DTD with ID and IDREF attribute types.

```
<!DOCTYPE bank-2[ …
    <!ELEMENT account (branch, balance)>
    <!ATTLIST account
              account_number ID      # REQUIRED
              owners         IDREFS # REQUIRED>
     <!ELEMENT customer(customer_name, customer_street,
                                       customer_city)>
     <!ATTLIST customer
              customer_id    ID      # REQUIRED
              accounts       IDREFS # REQUIRED>
     … declarations for branch, balance, customer_name,
                        customer_street and customer_city
]>
```

XML data with ID and IDREF attributes

```
<bank-2>
    <account account_number="A-401" owners="C100 C102">
        <branch_name> Downtown </branch_name>
        <balance>        500 </balance>
    </account>
    <customer customer_id="C100" accounts="A-401">
        <customer_name>Joe        </customer_name>
        <customer_street> Monroe  </customer_street>
        <customer_city>    Madison</customer_city>
    </customer>
    <customer customer_id="C102" accounts="A-401 A-402">
        <customer_name> Mary     </customer_name>
        <customer_street> Erin       </customer_street>
        <customer_city>    Newark </customer_city>
    </customer>
</bank-2>
```

XML Schema Version of Bank DTD

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>
<xs:element name="bank" type="BankType"/>
<xs:element name="account">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="account_number" type="xs:string"/>
            <xs:element name="branch_name"    type="xs:string"/>
            <xs:element name="balance"        type="xs:decimal"/>
        </xs:squence>
    </xs:complexType>
</xs:element>
….. definitions of customer and depositor ….
<xs:complexType name="BankType">
    <xs:sequence>
        <xs:element ref="account"   minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="customer" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="depositor" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

More features of XML Schema

- Key constraint: “account numbers form a key for account elements under the root bank element”:

```
<xs:key name = “accountKey”>
        <xs:selector xpath = “/bank/account”/>
        <xs:field xpath = “account_number”/>
</xs:key>
```

- Foreign key constraint from depositor to account:

```
<xs:keyref name = “depositorAccountKey” refer=“accountKey”>
        <xs:selector xpath = “/bank/account”/>
        <xs:field xpath = “account_number”/>
</xs:keyref>
```

XPath

- E.g. /bank-2/customer/customer_name evaluated on the bank-2 data returns

  <customer_name>Joe</customer_name>

  <customer_name>Mary</customer_name>

XPath (Cont.)

- Selection predicates:
  - /bank-2/account[balance > 400]
    - returns account elements with a balance value greater than 400
  - /bank-2/account[balance]
    - returns account elements containing a balance subelement
  - /bank-2/account[balance > 400]/@account_number
    - returns the account numbers of accounts with balance > 400

XQuery

- XQuery uses a
  for … let … where … order by …return …
  syntax:

  for ⇔ SQL from
  where ⇔ SQL where
  order by ⇔ SQL order by
  return ⇔ SQL select

  'let' allows temporary variables, and has no equivalent in SQL

FLWR Syntax in XQuery

- Find all accounts with balance > 400, with each result enclosed in an <account_number> .. </account_number> tag

```
for     $x in /bank-2/account
let     $acctno := $x/@account_number
where $x/balance > 400
return <account_number> { $acctno } </account_number>
```

  - Items in the **return** clause are XML text unless enclosed in {}, in which case they are evaluated

- Query can also be written as:

```
for $x in /bank-2/account[balance>400]
return     <account_number>
                    { $x/@account_number }
           </account_number>
```

Joins

- Joins can be specified in a manner very similar to SQL

```
for $a  in  /bank/account,
    $c  in  /bank/customer,
    $d  in  /bank/depositor
where   $a/account_number = $d/account_number
    and $c/customer_name = $d/customer_name
return <cust_acct> { $c $a } </cust_acct>
```

- The same query can be expressed with the selections specified as XPath selections:

```
for $a in /bank/account
    $c in /bank/customer
    $d in /bank/depositor[
                account_number = $a/account_number and
                customer_name  = $c/customer_name]
return <cust_acct> { $c $a } </cust_acct>
```

Nested Queries

- The following query converts data from the flat structure for bank information into a nested structure.

```
<bank-1> {
    for $c in /bank/customer
    return
        <customer>
          { $c/* }
          { for $d in /bank/depositor[customer_name = $c/customer_name],
                $a in /bank/account[account_number=$d/account_number]
          return $a }
        </customer>
} </bank-1>
```

- $c/* denotes all the children of the node to which $c is bound, without the enclosing top-level tag

Sorting in XQuery

- To return customers sorted by name

```
for $c in /bank/customer
order by $c/customer_name
return <customer> { $c/* } </customer>
```

- Can sort at multiple levels of nesting (sort by customer_name, and by account_number within each customer)

```
<bank-1> {
    for $c in /bank/customer
    order by $c/customer_name
    return
        <customer>
            { $c/* }
            { for $d in /bank/depositor[customer_name=$c/customer_name],
                  $a in /bank/account[account_number=$d/account_number] }
            order by $a/account_number
            return <account> $a/* </account>
        </customer>
} </bank-1>
```