# Advanced topics

# “Special” databases

- Biological data
- Geographic data – GIS
- Movies
- etc.

- New types of queries
- New ways of indexing data
- Storing/retrieval issues (e.g. large sizes, streaming, real-time, etc.)

# Examples

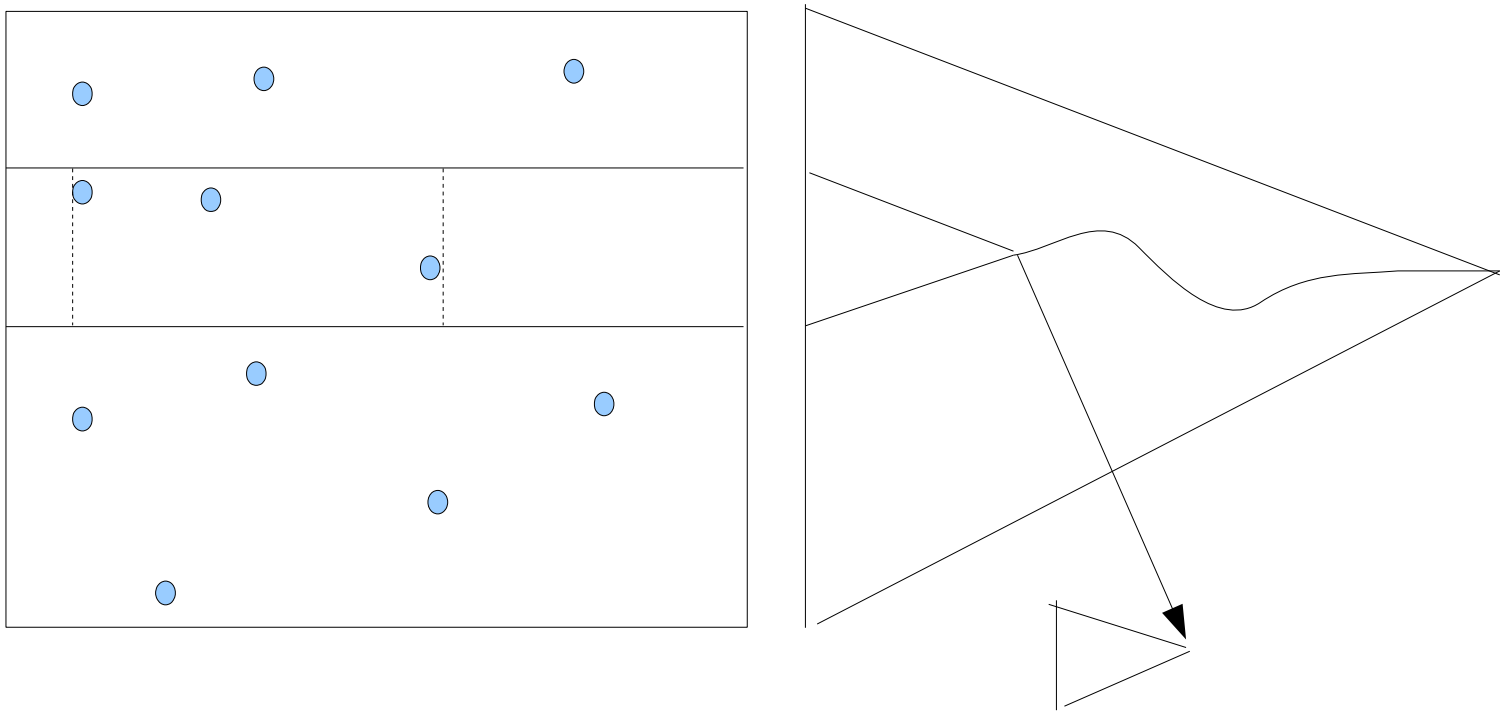
- Biological data
  - refinement of “like” queries: find sequences that are “related”

```
Query: 1   MSVMYKKILYPTDFSETAEIALKHVKAFKTLKAEEVILLHVIDEREIKKRDIFSLLLGVA 60
           M  M++K+L+PTDFSE A  A++  +    ++  EVILLHVIDE  +++     L+ G +
Sbjct: 1   MIFMFRKVLFPTDFSEGAYRAVEVFEKRNKMEVGEVILLHVIDEGTLEE-----LMDGYS 55
```

- Spatial/geographic data (GIS)
  - find all Home Depot stores within 15 miles of Baltimore
  - find a point in Maryland that's farther than 15 miles from the nearest Lowes and is densely populated
  - find all cities within lat/lon square: 39.00 N, 40.00 N, 76.00W, 77.00W.
  - special/spatial index: R-tree

# R-tree

- Binary search tree on Y-coordinate
- Each internal node contains search structure on X-coordinate for all points with Y coordinates in the corresponding subtree



# OLAP

- On-line Analytical Processing
- Why ?
  - Exploratory analysis
    - Interactive
    - Different queries than typical SQL queries
  - Data CUBE
    - A summary structure used for this purpose
      - E.g. *give me total sales by zipcode; now show me total sales by customer employment category*
    - Much much faster than using SQL queries against the raw data
      - The tables are *huge*
- Applications:
  - Sales reporting, Marketing, Forecasting etc etc

# Cross Tabulation of *sales* by *item-name* and *color*

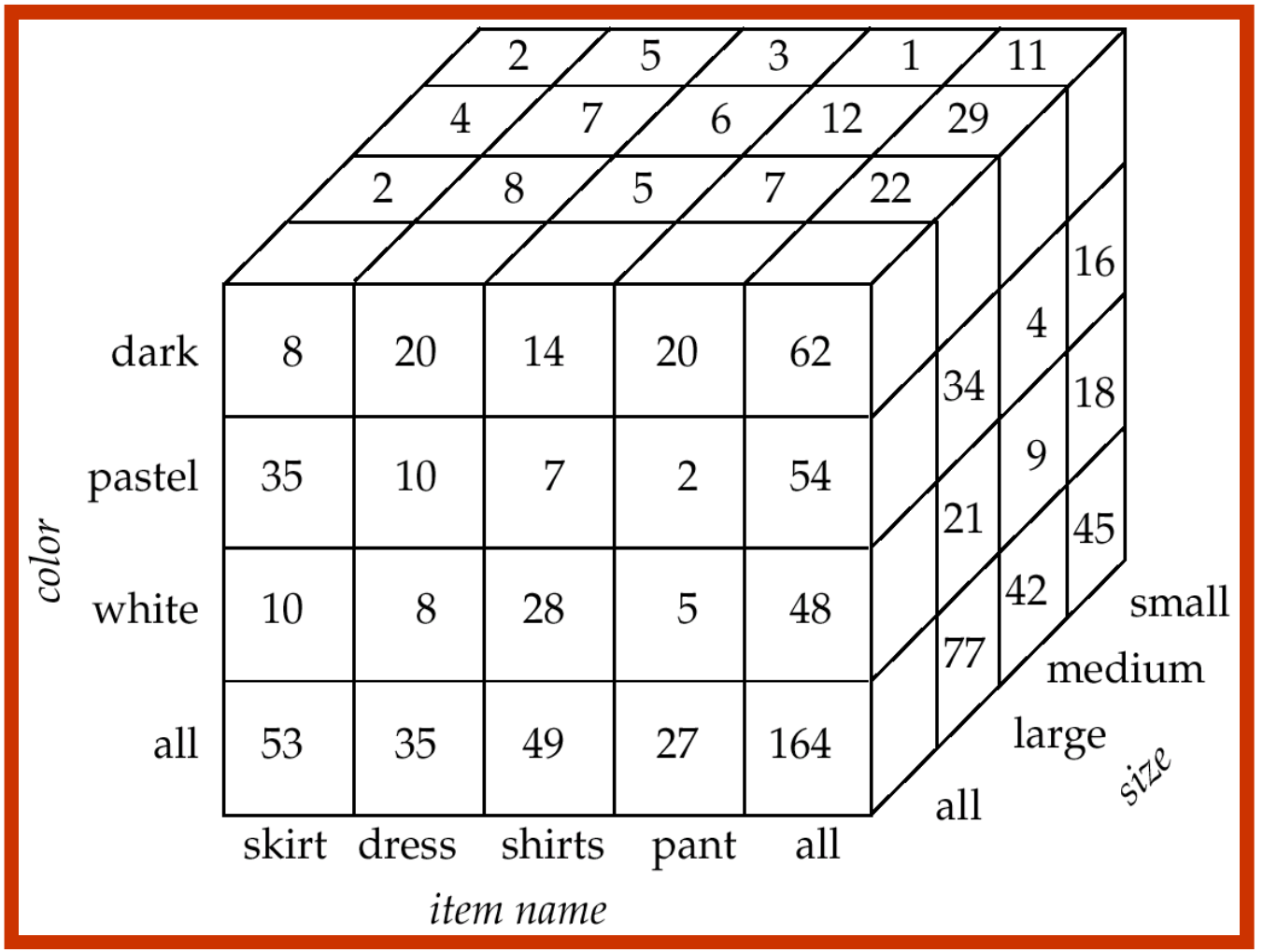
*size*: **all**

| *item-name* | *color* | | | |
|---|---|---|---|---|
| | dark | pastel | white | Total |
| skirt | 8 | 35 | 10 | 53 |
| dress | 20 | 10 | 5 | 35 |
| shirt | 14 | 7 | 28 | 49 |
| pant | 20 | 2 | 5 | 27 |
| Total | 62 | 54 | 48 | 164 |

- The table above is an example of a **cross-tabulation** (**cross-tab**), also referred to as a **pivot-table**.
  - Values for one of the dimension attributes form the row headers
  - Values for another dimension attribute form the column headers
  - Other dimension attributes are listed on top
  - Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.

# Data Cube

- A **data cube** is a multidimensional generalization of a cross-tab
- Can have *n* dimensions; we show 3 below
- Cross-tabs can be used as views on a data cube



# Data federation

- E.g. biological data:
  - VectorBase – organisms that carry human disease (e.g. mosquito)
  - Flybase – fruit flies
  - InsectBase???
- Federation -combining multiple databases into a single virtual database
- Has many issues:
  - schema translation?
  - common vocabulary? (e.g. ontologies, semantic web)
  - privacy/security
  - performance
- Non-biological: SkyServer/SkyQuery (Sloan Digital Sky Survey)

# Data warehouses

- Brute-force solution to federation:
  - download all databases
  - convert them to a common schema
  - provide a common interface
- Problems:
  - data storage & duplication
  - hard to keep up to date
  - performance (single point of entry/ failure)
- Examples:
  - GenBank (US biological data repository)
  - Ensembl (EU biological data repository)

# Data Mining

- Searching for patterns in data
  - Typically done in data warehouses
- Association Rules:
  - When a customer buys X, she also typically buys Y
  - Use ?
    - Move X and Y together in supermarkets
  - A customer buys a lot of shirts
    - Send him a catalogue of shirts
  - Patterns are not always obvious
    - Classic example: It was observed that men tend to buy *beer* and *diapers* together (may be an urban legend)
- Other types of mining
  - Classification
  - Decision Trees

# XML

- Extensible Markup Language
- Derived from SGML (Standard Generalized Markup Language)
  - Similar to HTML, but HTML is not *extensible*
    - Extensible == can add new tags etc
- Emerging as the *wire format (data interchange format)*

# XML

```
<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>
```

# Attributes

- Elements can have **attributes**

```
<account acct-type = “checking” >
    <account-number> A-102 </account-number>
    <branch-name> Perryridge </branch-name>
    <balance> 400 </balance>
</account>
```

- Attributes are specified by *name*=*value* pairs inside the starting tag of an element
- An element may have several attributes, but each attribute name can only occur once
  - <account acct-type = “checking” monthly-fee=“5”>

# Attributes Vs. Subelements

- Distinction between subelement and attribute
  - In the context of documents, attributes are part of markup, while subelement contents are part of the basic document contents
  - In the context of data representation, the difference is unclear and may be confusing
    - Same information can be represented in two ways
      - <account  account-number = “A-101”>  …. </account>
      - <account>
        <account-number>A-101</account-number> …
        </account>
  - Suggestion: use attributes for identifiers of elements, and use subelements for contents

# Namespaces

- XML data has to be exchanged between organizations
- Same tag name may have different meaning in different organizations, causing confusion
- Specifying a unique string as an element name avoids confusion
- Better solution: use  unique-name:element-name
- Avoid using long unique names all over document by using XML Namespaces

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

# Document Type Definition (DTD)

- The type of an XML document can be specified using a DTD
- DTD constraints structure of XML data
  - What elements can occur
  - What attributes can/must an element have
  - What subelements can/must occur inside each element, and how many times.
- DTD does not constrain data types
  - All values represented as strings in XML
- DTD syntax
  - <!ELEMENT element (subelements-specification) >
  - <!ATTLIST element (attributes) >
- Also – XML Schema (not covered -read in book & online)

# Bank DTD

```
<!DOCTYPE bank [
    <!ELEMENT bank ( ( account | customer | 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)>
]>
```

# IDs and IDREFs

- An element can have at most one attribute of type ID
- The ID attribute value of each element in an XML document must be distinct
  - Thus the ID attribute value is an object identifier
- An attribute of type IDREF must contain the ID value of an element in the same document

# 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,
                                    custome-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

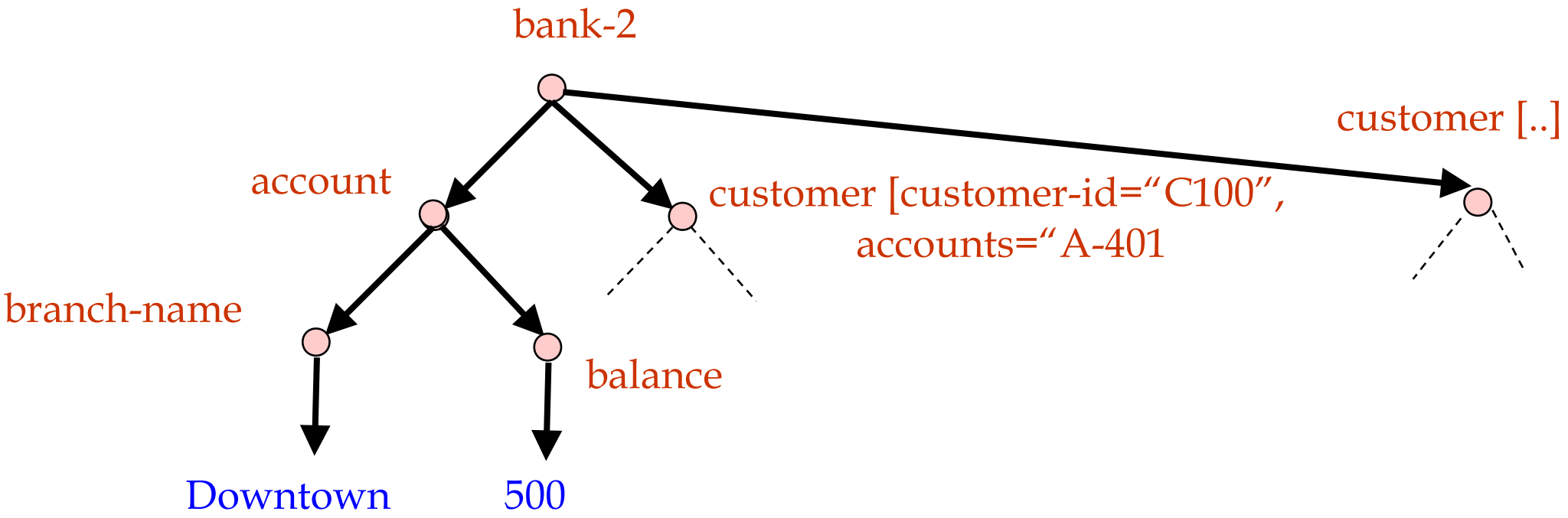
```
<bank-2>
    <account account-number="A-401" owners="C100 C102">
        <branch-name> Downtown </branch-name>
        <balance>        500 </balance>
    </account>
    <customer customer-id="C100" accounts="A-401">
        <customer-name>Joe        </customer-name>
        <customer-street> Monroe  </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>
```

# Querying and Transforming XML Data

- Standard XML querying/translation languages
  - XPath
    - Simple language consisting of path expressions
    - Forms a basic component of the next two
  - XSLT
    - Simple language designed for translation from XML to XML and XML to HTML
  - XQuery
    - An XML query language with a rich set of features

# Tree Model of XML Data

- Query and transformation languages are based on a tree model of XML data



# XPath

- /bank-2/customer/customer-name

  <customer-name>Joe</customer-name>
  <customer-name>Mary</customer-name>

- /bank-2/customer/customer-name/text( )

  Joe
  Mary

- /bank-2/account[balance > 400]
  - returns account elements with a balance value greater than 400
- /bank-2/account[balance > 400]/@account-number
  - returns the account numbers of those accounts with balance > 400

# Functions in XPath

- /bank-2/account[customer/count() > 2]
  - Returns accounts with > 2 customers
- Boolean connectives and and or and function not() can be used in predicates
- IDREFs can be referenced using function id()
  - E.g. /bank-2/account/id(@owner)
    - returns all customers referred to from the owners attribute of account elements.

# More XPath Features

- "//" can be used to skip multiple levels of nodes
  - E.g. /bank-2//customer-name
    - finds any customer-name element *anywhere* under the /bank-2 element, regardless of the element in which it is contained.
- Wild-cards
  - /bank-2/*/customer-name
  - Match any *element* name

# XSLT

- A stylesheet stores formatting options for a document, usually separately from document
  - E.g. HTML style sheet may specify font colors and sizes for headings, etc.
- The XML Stylesheet Language (XSL) was originally designed for generating HTML from XML
- XSLT is a general-purpose transformation language
  - Can translate XML to XML, and XML to HTML
- XSLT transformations are expressed using rules called templates
  - Templates combine selection using XPath with construction of results

# XSLT Templates

- Example of XSLT template with **match** and **select** part

```
<xsl:template match="/bank-2/customer">
    <xsl:value-of select="customer-name"/>
</xsl:template>
<xsl:template match="*"/>
```

- The match attribute of xsl:template specifies a pattern in XPath
- Elements in the XML document matching the pattern are processed by the actions within the xsl:template element
  - xsl:value-of selects (outputs) specified values (here, customer-name)
- For elements that do not match any template
  - Attributes and text contents are output as is
  - Templates are recursively applied on subelements
- The <xsl:template match="*"/> template matches all elements that do not match any other template
  - Used to ensure that their contents do not get output.

# Creating XML Output

- Any text or tag in the XSL stylesheet that is not in the xsl namespace is output as is
- E.g. to wrap results in new XML elements.

```
<xsl:template match="/bank-2/customer">
  <customer>
    <xsl:value-of select="customer-name"/>
  </customer>
</xsl:template>
<xsl:template match="*"/>
```

  - Example output:

```
<customer> Joe   </customer>
<customer> Mary </customer>
```

# XQuery

- XQuery is a general purpose query language for XML data
- Currently being standardized by the World Wide Web Consortium (W3C)
- Alpha version of XQuery engine available free from Microsoft
- XQuery is derived from the Quilt query language, which itself borrows from SQL, XQL and XML-QL
- XQuery uses a

  **for … let … where .. result** …

  syntax

  **for** ⇔ SQL from

  **where** ⇔ SQL where

  **result** ⇔ SQL select

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

# FLWR Syntax in XQuery

- For clause uses XPath expressions, and variable in for clause ranges over values in the set returned by XPath
- Simple FLWR expression 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>
```

- Let clause not really needed in this query, and selection can be done In XPath. Query can be written as:

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

# Joins

- Joins are 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>
```

# XML: Summary

- Becoming the standard for data exchange
- Many details still need to be worked out !!
- Active area of research…
  - Especially optimization/implementation

Worst...idea...ever!