XML

- eXtensible Markup Language
  - Came out of document community
  - Simplified subset of: Standard Generalized Markup Language (SGML)
- De facto data exchange format
  - Self-describing (although beware of Semantic Heterogeniety)
  - Text (passes through firewalls, compresses well)

NOTE:
- Somewhat older paper
  - Different languages popular today
  - XPath, XQuery etc..
- Much work since then on this topic
Example XML: Parking Space Information

```xml
<State id="Pennsylvinia">
  <County id="Allegheny">
    <City id="Pittsburgh">
      <Neighborhood id="Oakland">
        <total-spaces>200</total-spaces>
        <Block id="1">
          <GPS>…</GPS>
          <pSpace id="1">
            <in-use>no</in-use>
            <metered>yes</metered>
          </pSpace>
          <pSpace id="2">
            ...
            </pSpace>
        </Block>
      </Neighborhood>
      <Neighborhood id="Shadyside">
        ...
        </Neighborhood>
    </City>
  </County>
</State>
```

Example XML Fragment for PSF
Example Queries

- Users issue queries against the document as a whole
  - Find all available parking spots in Oakland
    ```xml
    /State[@id="Pennsylvania"]/County[@id="Allegheny"]/City[@id="Pittsburgh"]/Neighborhood[@id="Oakland"]/Block/pSpace[in-use = "no"]
    ```
  
  - Find all blocks in Allegheny have more than 20 metered parking spots
    ```xml
    /State[@id="Pennsylvania"]/County[@id="Allegheny"]
    //Block[count(.//pSpace[metered = "yes"]) > 20]
    ```
  
  - Find the cheapest parking spot in Oakland Block 1
    ```xml
    /State[@id="Pennsylvania"]/County[@id="Allegheny"]/City[@id="Pittsburgh"]/Neighborhood[@id="Oakland"]/Block[@id='1']/pSpace[not(../pSpace/price > ./price)]
    ```

XML: Some Key Issues

- How to store?
  - Option 1: Using relational databases
    - Need to develop schemes to convert back and forth
    - Not easy to do... (XMLSchema, DTD help)
    - XML queries are naturally hierarchical → need a lot of joins
  
  - Option 2: Using a native XML database
    - Reverting back to hierarchical/network database models?
    - Some beginning to appear over last few years
  
- We will focus on Option 1 for this class
XML: Some Key Issues

- Query Languages
  - XPath: Identify a set of nodes in the document
  - XSLT: Transformation language
    - Fairly verbose... essentially a program that traverses the document
  - XQuery: The current standard
    - Personally, I think it is too complicated
    - Likely only a subset will be used/implemented in practice

- Schemas
  - DTD (Document Type Descriptor)
  - XMLSchema (current standard ??)

XML in RDBMS

- NOTE: Somewhat older paper... much work since then
- Key issues
  - Converting an XML document to relational
    - Called “shredding”
    - Uses the DTD Information
  - Processing queries
  - Converting the relational data back to XML
    - Essentially a query + some post-processing (maybe as a UDF)
**XML in RDBMS**

1. Simplify the DTD
   - The conversion can be a one-way process
   - No need to preserve exact structure in the relational schema
   - **Order is important in XML**
     - See a later paper: *Handling order when converting*

2. Create a set of tables
   - Simple option: Create a table for each element
     - Too many tables; a lot of joins needed later
     - Can think of that as denormalizing
     - Should try *inlining* as much as possible

---

**XML in RDBMS: Example**

```
<book>
  <booktitle>The Selfish Gene</booktitle>
  <author id="dawkins">
    <name>
      <firstname>Richard</firstname>
      <lastname>Dawkins</lastname>
    </name>
    <address>
      <city>Timbuktu</city>
      <zip>99999</zip>
    </address>
  </author>
</book>
```

*Figure 1*

```
<book>
  <booktitle>The Selfish Gene</booktitle>
  <author id="dawkins">
    <name>
      <firstname>Richard</firstname>
      <lastname>Dawkins</lastname>
    </name>
    <address>
      <city>Timbuktu</city>
      <zip>99999</zip>
    </address>
  </author>
</book>
```

*Figure 2*
XML in RDBMS: DTD Graph

Figure 8

XML in RDBMS: Schema without inlining

```sql
booktitle (booktitleID: integer, booktitle: string)
article (articleID: integer, article.article.contactauthor.authorid: string, article.title: string)
article.author (article.authorID: integer, article.author.name.firstname: string, article.author.name.lastname: string, article.author.address: string, article.author.authorid: string)
contactauthor (contactauthorID: integer, contactauthor.authorid: string)
title (titleID: integer, title: string)
editor (editorID: integer, editor.parentID: integer, editor.name: string)
author (authorID: integer, author.name.firstname: string, author.name.lastname: string, author.address: string, author.authorid: string)
name (nameID: integer, name.firstname: string, name.lastname: string)
firstname (firstnameID: integer, firstname: string)
lastname (lastnameID: integer, lastname: string)
address (addressID: integer, address: string)
```

Figure 10
XML in RDBMS: Schema with aggressive inlining

article (articleID: integer, article.contactauthor.isroot: boolean, article.contactauthor.authorid: string)

Figure 11

XML in RDBMS: Query Conversion

- Left: Original Query (in XML-QL and Lorel syntax),
- Right: Converted query

WHERE <book>
  <booktitle> The Selfish Gene </booktitle>
  <author>
    <name>
      <firstname> $f </firstname>
      <lastname> $l </lastname>
    </name>
  </author>
</book> IN * CONFORMING TO pubs.dtd
CONSTRUCT <result> $f $l </result>

Select Y.name.firstname,
  Y.name.lastname
From   book X, X.author Y
Where  X.booktitle = "Databases"

Select A."author.name.firstname",
  A."author.name.lastname"
From   author A, book B
Where   B.bookID = A.parentID
        AND A.parentCODE = 0
        AND B."book.booktitle" = "The Selfish Gene"

Figure 18
4.2 Converting Simple Recursive Path Expressions to SQL

Consider the following XML-QL query that requires...

```
WHERE <*.monograph>
    <editor.(monograph.editor)>
    <name>$n</name>
</>
    <title>Subclass Cirripedia</title>
</>
IN * CONFORMING TO pubs.dtd
CONSTRUCT <result>$n</result>
```

```
Select Y.name
From  *.monograph X, X.editor.(monograph.editor)* Y
Where X.title = "Subclass Cirripedia"
```

```
With Q1 (monographID, name) AS
(Select X.monographID, X."editor.name"
From monograph X
Where X.title = "Subclass Cirripedia"
UNION ALL
Select Z.monographID, Z."editor.name"
From Q1 Y, monograph Z
Where Y.monographID = Z.parentID AND
    Z.parentCODE = 0
)
Select A.name
From Q1 A
```

Note: (right) is a recursive query
- The query (WITH part) creates a table (Q1) and refers to it in the FROM clause

XML in RDBMS: Converting results to XML

- Some result construction can be done using SQL
- More complex ones require a post-processing step
  - Can be done using a user-defined function or embedded SQL or like
  - Can use "group by" etc to create the appropriate sets
Limitations of RDBMS

- Simple XML queries required too many joins or unions
- No support for sets
  - XML data usually set-valued
- No support for untyped references
  - IDREF is not typed, so storing it is problematic
- No text indices
- Need flexible comparison operators
  - XML treats everything as string
- More powerful recursion
  - SQL3 (latest version) allows recursion
  - Not very commonly used
  - Somewhat hard to reason about