Storage and Retrieval of XML Data Using Relational Databases

Surajit Chaudhuri
Microsoft Research

Kyuseok Shim
KAIST

About this Tutorial

- Tutorial presented based solely on publicly available information
- Information is incomplete and could be inaccurate
- Our presentation reflects our understanding which may be erroneous
- Does not reflect views of our employers

What's the big deal about XML?

Disconnected Empires

- Before XML
  - No "universal" programming language
  - Various wire interoperation protocols based on RPC
    - Based on CORBA, DCOM
  - Corporate firewalls block DCOM, CORBA traffic
    - But allows HTTP in

XML: A Wire Protocol

- XML = A minimal wire representation for data and storage exchange
  - A low-level wire transfer format – like IP in networking
- Minimal level of standardization for distributed components to interoperate
  - Platform, language and vendor agnostic
  - Easy to understand and is extensible
- Data exchange enabled via XML transformations

SOAP

- Facilitate Platform independent distributed computing
- Protocol that defines a uniform way to perform RPC with
  - HTTP as wire protocol
  - Data encoded as XML

Example (SOAP)

POST /cgi-bin/purchase-book.cgi HTTP/1.1
Methodname: Purchasebook
InterfaceName:
soap:cdl:com.develop.demos.purchase_book
MessgeType: Call
Content-Type: text-xml-SOAP
<Purchasebook>
<ISBN>0201379369</ISBN>
</Purchasebook>

Core XML Technologies

- Validation of XML
  - Schema and namespaces
- XML API: Programmatic Access to XML
  - DOM, SAX
- Transformation Technology: For Data Exchange and Display
  - XSL, XSLT, XQuery

Implications of XML for Databases

- Data stored in SQL databases need to be published in XML for data exchange
  - Core requirement
  - Specification schemes for publishing needed
- Storage and retrieval of native XML for "document-centric" applications
  - Need to support XML API-s

Architectural Alternatives for Storing Native XML

- Relational Approach – exploit traditional benefits of databases
- Other Architectures (not discussed)
  - File Systems
  - Native semistructured store
    - e.g. eXcelon, LORE, NATIX

Rationale for Relational Representation for Native XML

- Relational and native XML data can co-exist
- RDBMS are scalable and offers many services
  - Transaction Management, Query optimization
- XML Schema/DTD can help guide relational representation
  - Incremental update and partial retrieval enabled
- LOB storage and indexing complements structured XML ("shredded") representation
  - Fast for storing and retrieving whole documents
  - Incremental update is difficult

Outline of Tutorial

- XML and Related Technologies
- Examples of XML Support in Commercial Systems
- Publication of Relational Data in XML
- Storage of Native XML in Relational Databases
- Final Thoughts

XML and Related Technologies

XML

- Tagged elements describe the semantics of the data
- An element may have attributes to provide additional information
- An element can contain a sequence of nested sub-elements
  - Sub-elements may themselves be tagged elements or character data

An XML Document

```
<?xml version="1.0"?>
<!DOCTYPE sigmodRecord SYSTEM "sigmodRecord.dtd">
<sigmodRecord>
<issue>
  <volume>1</volume>
  <number>1</number>
 <articles>
    <article>
      <title> XML Research Issues</title>
      <initPage>1</initPage>
      <endPage>5</endPage>
      <authors>
        <author AuthorPosition="00">Tom Hanks</author>
      </authors>
    </article>
    <article>
      ..
      </authors>
    </article>
  </articles>
</issue>
</sigmodRecord>
```


Document Type Definition (DTD)

- An XML document may have a DTD
- Grammar for describing the structure of XML document
- Terminology
  - well-formed: if tags are correctly closed
  - valid: if it has a DTD and conforms to it
  - For exchanges of data, validation is useful

A DTD Example (SIGMOD Record)

```
<?xml version="1.0"?>
<!– SIGMOD Record DTD -->
<!ELEMENT SigmodRecord (issue)* >
<!ELEMENT issue (volume,number,articles) >
<!ELEMENT volume (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT articles (article)* >
<!ELEMENT article (title, initPage, endPage, authors) >
<!ELEMENT title (#PCDATA)>
<!ELEMENT initPage (#PCDATA)>
<!ELEMENT endPage (#PCDATA)>
<!ELEMENT authors (author)* >
<!ELEMENT author (#PCDATA)>
<!ATTLIST author AuthorPosition CDATA #IMPLIED>
```


DTD Specification

- comma: sequence
- |: or
- (): grouping
- ?, *, +: zero or one, zero or more, one or more occurrences
- ANY: allows an arbitrary XML fragment to be nested within the element
- PCDATA: parsed character data.
- CDATA: character data.

XML Schema

- Schema
  - http://www.w3c.org/TR/xmlschema-i (i=0..2)
  - Specifies structure of XML documents
  - Datatypes for elements/attributes
    - string, int, float
    - Unordered set is also allowed
    - Derivation of types are allowed
  - Constraints/Keys
- Replaces DTDs
  - Removes syntactic distinctions between DTD and XML
  - Richer types compared to DTD

XML Schema Example

```
<xsd:element name="article" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="initPage" type="xsd:string"/>
       <xsd:element name="endPage" type="xsd:string"/>
      <xsd:element name="author" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

DTD

```
<!ELEMENT article (title, initPage, endPage, author) >
<!ELEMENT title (#PCDATA)>
<!ELEMENT initPage (#PCDATA)>
<!ELEMENT endPage (#PCDATA)>
<!ELEMENT author (#PCDATA)>
```

XML API: DOM

- Characteristics
  - Hierarchical (tree) object model for XML documents
  - Associate a list of children with every node (or text value)
  - Preserves the sequence of the elements in the XML document
- May be expensive to materialize for a large XML collection

XML API: SAX

- Characteristics
  - Event-driven: fire an event for every open tag/end tag
  - Does not require full parsing
  - Enables custom object model building
- Could be significantly faster for a simple object model

XSL

- Styling is rendering information for consumption
- XSL = A language to express styling ("Stylesheet language")
- Two components of a stylesheet
  - Transform: Source to a target tree using template rules expressed in XSLT
  - Format: Controls appearance

XSLT

- XPATH acts as the pattern language (more later)
- Primary goal is to transform XML vocabularies to XSL formatting vocabularies
  - But, often adequate for many transformation needs

XQuery: An XML Query Language

- Focused on expressing transformation
  - XPATH is a starting point
  - Unlike XSLT, no need to write stylesheet
- Make "database queries" easy to express (and hopefully to optimize)
- Strong Typing

XPATH

- [www.w3.org/TR/xpath]
- Used as a building block for
  - XSL Transformations (XSLT)
  - XQuery
- Syntax for tree navigation and node selection
  - Navigation is described using location paths

XPATH

- . : current node
- .. : parent of the current node
- / : root node, or a separator between steps in a path
- // : descendants of the current node
- @ : attributes of the current node
- * : "any" (node with unrestricted name)
- [ ] : a predicate for a given step
- [n] : the element with the given ordinal number from a list of elements

XPATH Example

- List the titles of articles in which the author has "Tom Hanks"

  //article[//author="Tom Hanks"]/title
- Find the titles of articles authored by "Tom Hanks" in volume 1.

  //issue[/volume="1"]/articles/article/[//author="Tom Hanks"]/title

XQuery

- Origin: Quilt [Chamberlin, Jobie, Florescu: WebDB 00]
- A functional language
  - query is an expression
  - expressions are recursively constructed
- Features
  - Includes XPATH as a sub-language
  - SQL-like FLWR expression
- Borrows features from many other languages: XQL, XML-QL, ML,..

XQuery

- FLWR expression
  - FOR/LET Clauses
    - Ordered list of tuples of bound variables
  - WHERE Clause
    - Pruned list of tuples of bound variables
  - RETURN Clause
    - Instance of XML Query data model

Example: XQuery

- List the titles of the articles authored by "Tom Hanks"

Query Expression

```
for $b IN document("sigmodRecord.xml")//article
where $b//author = "Tom Hanks"
return <title> $b/title.text() </title>
```

Query Result

```
<title>XML Research Issues</title>
```


Example: XQuery

- List the articles authored by "Tom Hanks".

Query Expression

```
<articles>
 {
   for $b IN document("sigmodRecord.xml")//article
   where $b//author = "Tom Hanks"
   return $b
 }
</articles>
```

Query Result

```
<articles>
  <article>
    <title>XML: Where are we heading for?</title>
    <initPage>6</initPage>
    <endPage>10</endPage>
    <authors>
      <author AuthorPosition="00">Tom Hanks</author>
    </authors>
  </article>
</articles>
```

Examples of XML Support in Commercial Systems

Microsoft SQL Server: Publishing

- Provide extensions on SQL to return results as XML document (FOR XML clause)
  - RAW mode: Converts each row in the SQL result into an XML without subelements
  - AUTO mode: Produce query result as nested XML document
  - EXPLICIT mode: the most general
    - Can control the shape of XML document
    - Expected nesting is explicitly specified as part of the query
- Generation of XML view using XDR
- Templates

RAW Mode

- Returns Flat XML documents
  - Each row in the query result becomes an XML element with the name row
  - Each non-NULL column value is mapped to an attribute (column name becomes the attribute name)
- Q:
```
select Customers.CustomerID, Orders.OrderID, Orders.OrderDate
from Customers, Orders
where Customers.CustomerID = Orders.CustomerID
order by Customers.CustomerID
for XML RAW
```
- A:
```
<row CustomerID="ALFKI"  OrderID="10643"  OrderDate="1997-08-25"/>
<row CustomerID="ANATR" OrderID="10308"  OrderDate="1996-09-18"/>
```


AUTO Mode

- Returns nested XML elements.
  - Each table in the FROM clause is represented as an XML element
  - Table-list determines element nesting
  - The columns listed in the SELECT clause are mapped to the attributes
  - When the ELEMENTS option is specified, the table columns are mapped to sub-elements

Example: AUTO Mode

```
select Customers.CustomerID, Orders.OrderID,
    Customers.ContactName
from Customers, Orders
where Customers.CustomerID = Orders.CustomerID
for XML AUTO
```

```
<Customers CustomerID="ALFKI" ContactName="Maria Ansers">
  <Orders OrderID="10643"/>
  <Orders OrderID="10692"/>
  <Orders OrderID="10702"/>
  <Orders OrderID="10835"/>
  <Orders OrderID="10952"/>
  <Orders OrderID="11011"/>
</Customers>
```


Example: AUTO Mode (2)

```
select Customers.CustomerID, Orders.OrderID,
    Customers.ContactName
from Customers, Orders
where Customers.CustomerID = Orders.CustomerID
for XML AUTO, ELEMENTS
```

```
<Customers>
  <CustomerID>ALFKI</CustomerID>
  <ContactName>Maria Ansers</ContactNAme>
  <Orders><OrderID>10643</OrderID></Orders>
  <Orders><OrderID>10835</OrderID></Orders>
  <Orders><OrderID>10952</OrderID></Orders>
  <Orders><OrderID>11011</OrderID></Orders>
</Customers>
```

EXPLICIT Mode

- Motivation: Flexible publishing of relational data
  - Unlike RAW, AUTO modes
- Steps
  - Define a SQL view to assemble relevant rows
  - The rowset (universal table) must have certain format
    - Special Columns: Tag and Parent
    - Column names
    - Row ordering

Explicit Mode: Universal Relation Format

- 1st column
  - Column name: Tag
  - Tag number of the current element
- 2nd column:
  - Column name: Parent
  - Tag number of the parent element
  - If the value is 0 or NULL, the row is placed on the top level
- Ordering of Rows: Parent must be followed immediately by its children

Example1: EXPLICIT Mode

```
select 1                          as Tag,
       NULL                       as Parent,
       Customers.CustomerID   as [Customer!1!CustomerID]
       NULL                       as [Order!2!OrderID]
from Customers

UNION ALL

select  2,
       1,
       Customers, Orders
where Customers.CustomerID = Orders.CustomerID
order by [Customer!1!CustomerID], [Order!2!OrderID]
for XML EXPLICIT
```


Example1: EXPLICIT Mode (2)

Tag	Parent	Customer!1!CustomerID	Order!2!OrderID
1	NULL	ALFKI	NULL
2	1	ALFKI	10643
2	1	ALFKI	10692
2	1	ALFKI	10702
2	1	ALFKI	11011
2	1	ALFKI	…
1	NULL	ANATR	NULL
2	1	ANATR	10308
2	1	ANATR	10625
2	1	ANATR	…

Example1: EXPLICIT Mode (3)

```
<Customer CustomerID="ALFKI">
  <Order OrderID="10643" />
  <Order OrderID="10692" />
  <Order OrderID="10702" />
  <Order OrderID="11011" />
</Customer>
<Customer CustomerID="ANATR">
  <Order OrderID="10308" />
  <Order OrderID="10625" />
</Customer>
```


Example2: EXPLICIT Mode (1)

```
select 1                        as Tag,
       NULL                     as Parent,
       Customers.CustomerID     as [Customer!1!CustomerID]
       NULL                     as [Order!2!OrderID!element]
from Customers

UNION ALL

select  2,
       1,
       Customers, Orders
where Customers.CustomerID = Orders.CustomerID
order by [Customer!1!CustomerID], [Order!2!OrderID!element]
for XML EXPLICIT
```

Example2: EXPLICIT Mode (2)

```
<Customer CustomerID="ALFKI">
   <Order> <OrderID>10643 </OrderID </Order>
   <Order>  <OrderID>10692</OrderID>
   <Order>  <OrderID>10702</OrderID>
</Customer>
<Customer CustomerID="ANATR">
   …….
</Customer>
```


Defining XML Views

- Annotated schema using XDR (XML-Data Reduced Data)
  - Description similar to DTD
  - Also includes mapping to SQL
- Views may be consumed by XML tools
  - Queried by XQuery
  - Processed by XSLT

Example: XML Views using XDR

```
<schema xmlns = ..
Xmlns:sql =  ..>
<ElementType name = "Customer"
sql:relation = "Customers">
<AttributeType name = "ID" />
<attribute type ="ID" Sql:field
  ="CustomerID"/>
</ElementType
</schema>
```


MS SQL Server: Support for Native XML

- Alternatives using OpenXML rowset provider
  - Edge Table: A graph representation of the XML (parent-child hierarchy)
    - Select * from OpenXML(@doc, '/ROOT/Customer')
  - Shredded Rowset*:* Derive custom rowset from XML
    - A Flag determines alternatives:
      - Columns = Attributes or Columns = Elements

Example: Shredded Rowset

Attribute-Centric Mapping

```
Select * from
OpenXML(@doc, '/ROOT/Customer', 1)
WITH (CustomerID varchar(10)
      Contactname varchar(20))
```

Retrieval from Different levels

```
Select * from
OpenXML(@doc, '/ROOT/Customer/Order/OrderDetail', 2)
WITH (OrderDate datetime '../@OrderDate',
      ProdID int '@ProductID')
```


IBM DB2

- Publishing
  - Mapping from SQL to XML specified in Data Access Definition (DAD) files
- Storage of Native XML
  - Provides two primary storage and access methods for native XML
    - XML column and XML collection
  - Mapping specified using DAD files as well

Using DAD File for Publishing

- Defines the mapping to XML view
  - An XML document itself
  - Used also for native storage of XML (discussed later)
- Two ways to specify mapping
  - SQL Mapping
  - RDB_node Mapping
- Use dxxGenXML() Stored Procedure to generate XML
  - Input: DAD
  - Output: Output_Table (single-column)
  - Additional customization parameters

Example: DAD-SQL Mapping

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "dxx_install/dtd/dad.dtd">
<dad>
   <dtdid>dxx_install/dtd/getstart.dtd</dtdid>
   <validation>YES</validation>
   <Xcollection>
      <SQL_stmt>
         select o.order_key, customer_name, customer_email, p.part_key, color,
           quantity, price, tax, ship_id, date, mode
         from order_tab o, part_tab p,
               (select db2xml.generate_unique() as ship_id, date, mode, part_key
                 from ship_tab) as s
         where o.order_key = 1 and p.price > 20000 and p.order_key =
           o.order_key and  s.part_key = p.part_key
          order by order_key, part_key, ship_id
      </SQL_stmt>
```

Example: DAD-SQL Mapping (Contd.)

```
<prolog>?xml version="1.0"?</prolog>
<root_node>
<element_node name="Order">
   <attribute_node name="key">
      <column name="order_key"/>
   </attribute_node>
   <element_node name="Customer">
      <element_node name="Name">
         <text node>
            <column name="customer_name"/>
         </text node>
      </element_node>
   </element_node>
</element_node>
</root_node>
</Xcollection>
</dad>
```


Example: DAD-RDB_node Mapping

```
<DAD> <Xcollection> <root_node>
<element_node name = "Order">
<RDB_node>
    <table_name = "order_tab"/>
    <table_name = "part_tab"/>
    <table_name = "ship_tab"/>
    <condition>
    Order_tab.order_key = part_tab_order_key AND
    Part_tab.part_key = ship_tab.part_key
    </condition>
</RDB_node>
<attribute_node name ="key">
<RDB_node>
    <table_name= "order_tab"/>
    <column name = "order_key" />
</RDB_node> </attribute_node>
```

Example: DAD-RDB_NODE Mapping (2)

```
....
<element_node name = "ShipDate"
    <text_node>
        <RDB_node>
            <table_name= "ship_tab"/>
            <column name= "date" />
            <condition>
                Date > "1966-01-01"
            </condition>
        </RDB_node>
    </text_node>
</element_node>
...
</root_name>
</Xcollection>
</DAD>
```


IBM DB2: Storing Native XML in a Column

- Store DTD in DTD repository
- Register a XML column with type:
  - *XMLCLOB*: for large XML documents; *XMLVARCHAR*: for small XML documents, *XMLFile*: for XML documents stored outside DB2
- Create a DAD file
  - To specify any "side tables" for indexing
- Enable the XML column
  - Creates side tables as per DAD
  - Add necessary triggers and other meta information
- Insert XML document
  - Side tables get automatically updated

DAD for Side Table Specification

```
.. <dad>
   <dtdid>dxx_install/dtd/getstart.dtd</dtdid>
   <validation>YES</validation>
   <Xcolumn>
      <table name="order_side_tab"> </table>
      <table name="part_side_tab">
         <column name="price" type="decimal(10,2)"
          path="/Order/Part/ExtendedPrice" multi_occurrence="YES"/>
      </table>
      <table name="ship_side_tab">
         <column name="date" type="DATE"
                  path="/Order/Part/Shipment/ShipDate"
    multi_occurrence="YES"/>
      </table>
   </Xcolumn>
</dad>
```


Stored Procedures for Handling XML Columns

- Storage:Type conversion
  - XMLVarCharFromFile(),..
- Retrieval: Cast Functions
  - Varchar(XMLVarChar)
- Update
  - Cast functions or storage UDFs
  - Update(xmlobj, path,value)
- Selection using XPATH
  - WHERE Extractvarchar(order, '/order/Customer') LIKE "%IBM%'
  - Or, use side tables

Native XML Storage: XML Collection Option

- Allows to decompose an XML document into a set of DB2 tables
  - A subset of data requires frequent update
  - Strongly related to other relational data
  - Retrieve only a subset of data
- DAD is used to define the mapping of DTD to relational tables and columns
  - Using RDB_Node mapping in DAD
    - Need primary key for each table and column types
  - To decompose an incoming XML document, use dxxShredXML (uses DAD)
  - To compose a shredded XML, use dxxGenXML() as in publishing phase

Oracle

- XML SQL Utility (XSU) supports publishing and native storage/retrieval
- Publishing Relational Data
  - The desired hierarchy in XML output can be specified by object views
    - Exploits the object-relational support in Oracle
  - Mapping is *implicit* for the object table/view
    - Result can be in text or a DOM tree
- Native storage of XML
  - System Provided XMLTYPE with supporting methods
  - Shredding by mapping to an object table or view

Publishing Relational Data

- Mapping Rules
  - Columns map to top level elements
  - Scalar values map to a elements with text only content
  - Object types are mapped to elements with its attributes appearing as sub-elements. Collections map to lists of elements
  - Object references and referential constraints can be mapped to IDREFs in the XML document
- Steps (From [Bannerjee et al.])
  - Initialize
    - OracleXMLQuery qry = new OracleXMLQuery(conn, query)
  - Set document and row element names
    - qry.setRowsetTag("str"), qry.setRowTag("str")
  - Get XML
    - qry.getXMLString()

Example: Object of Publishing

```
CREATE TYPE EmployeeType AS OBJECT (
   EMPNO NUMBER,
   ENAME VARCHAR2(20),
   SALARY NUMBER );

CREATE TYPE EmployeeListType AS TABLE OF EmployeeType;

CREATE TABLE Dept (
   DEPTNO NUMBER,
   DEPTNAME VARCHAR2(20),
   EMPLIST EmployeeListType );
```

Example: Corresponding XML

```
<?xml version='1.0'?>
<ROWSET>
    <ROW num="1">
        <DEPTNO>100</DEPTNO>
        <DEPTNAME>Sports</DEPTNAME>
        <EMPLIST>
            <EMPLOYEE_TYPE num="1">
                <EMPNO>7369</EMPNO>
                <ENAME>John</ENAME>
                <SALARY>10000</SALARY>
            </EMPLOYEE_TYPE>
            <!-- additional employee types within the employee list -->
        </EMPLIST>
    </ROW>
</ROWSET>
```


Native Storage of XML using XMLType

- Oracle9i introduces XMLType as a new server datatype
  - Currently allows only CLOB storage natively
  - Enables Strong Typing and XPATH functionality
  - Unsuitable for piecewise updates
- Create an XMLType instance
  - Sys.XMLType.createXML('xml_as_string')
  - Use techniques for XML Publishing from database rows

Selecting/Extracting XMLType Instance

- Extract()
  - Obtains the node/nodes from the document identified by the XPath (subset of) expression
  - Use getStringVal() or getNumberVal() to get scalar content

```
Select p.poDoc.extract('/PO/PONO/text()').getnumberval()
From po_xml_tab p;
```

- ExistsNode()
  - Checks if the given XPath evaluates in at least a single XML element or text node
  - Useful for filtering as well as for functional indexes

```
Select e.poDoc.extract('//PONO/text()').getNumberVal() as pono
From po_xml_tab e
Where e.podoc.existsnode('/Po/PONO') = 1 and poid > 1
```

- Functions to convert from XMLType to other types

Indexing XMLType

- Functional Indexes using ExistsNode or Extract
  - Create Index city_index ON po_xml_tab (poDoc.extract('//PONO/text()').getNumberVal())
- Text Indexes
  - Supports CONTAINS
  - CONTAINS has been extended using INPATH
    - <text query> INPATH(<path expression>)

Storage of Native XML in Structured Fashion

- Shredded storage
  - Can be mapped to an object table if "mapping rules" (discussed earlier) are satisfied

    String xmldoc = "..."

    OracleXMLSave sav = new OracleXMLSave(conn, tablename)

    Sav.**insertXML(xmldoc)**
- Decompose using custom XSLT

Commercial Systems: Other Aspects of XML Support

- Enabling specification for custom publishing
  - XSLT support for rendering
- Updates, Bulkloading
- HTTP Access , Security
- Use of File systems
- ...

Publication of Relational Data in XML

Approaches to Publication

- Publishing without full support for views
  - Published XML cannot be composed with XQuery without materialization
    - [Shanmugasundaram et al. VLDB 00]
- Publishing with support for views
  - SilkRoute
    - [Fernandez, Tan, Suciu: WWW 00]
    - [Fernandez, Morishima, Suciu: SIGMOD 01]
  - XPERANTO
    - [Shanmugasundaram et. al.: VLDB 01]

Publishing without support for Views

- [Shanmugasundaram et al.: VLDB 00]
  - Explores different execution plans for generating the contents of XML documents
- Conclusion
  - Constructing XML document inside the relational engine best for performance
    - When the result fits in main memory, Unsorted outer union approach
    - Otherwise,Sorted outer union approach
  - Need extensions to SQL

Example: Desired XML

```
<article id = "a1">
  <title> XML Research Issues</title>
  <authors>
    <author>Tom Hanks</author>
  </authors>
</author>
<article id = "a2">
  <title>Storage and Retrieval of XML Data Using Relational DBMS</title>
  <authors>
    <author>Surajit Chaudhuri</author>
    <author>Kyuseok Shim</author>
  </authors>
</author>
```

Example: Query Syntax

```
select art.name, ARTICLE(art.id, art.title,
          (select XMLAGG(AUTH(auth.ID, auth.name))
            from authors auth
           where art.ID=auth.IDfrom))
from  articles art
```

- XML element is created by calling ARTICLE() constructor
- XMLAGG() concatenates the XML fragments
- XMLAGG aggregate function needs to work on ordered inputs.

Example: Defining User Defined Function

```
define XML constructor ARTICLE(
                artId:integer, titl:varchar(20), authorList:xml) AS {

          <article id = $artId>
               <title> $titl</name>
               <authors> $accList</authors>
          </article>
}
```

Implementation Alternatives

- Early Tagging and Early Structuring
  - Stored Procedure Approach
    - Process queries for each nested structure
    - A fixed join ordering and join technique
  - CLOB Approach
    - Need to support XML constructors and XML aggregation functions
    - Represents the XML fragments generated by constructors as Character Large Objects (CLOBs)
    - Correlated or De-correlated

Implementation Alternatives

- Late Tagging and Late Structuring
  - Redundant Relation Approach
    - Represents a hierarchical structure as a single table
    - Number of tuples grows as product of relations – inefficient!
  - Unsorted Outer Union Approach
    - Computes each path from root level table to a leaf level table and outer union them
    - A separate tuple describing an ancestor is needed if ancestor has no children
    - NULL is used to fill the columns of other children
  - Hash-based tagging is used

Implementation Alternatives

- Late Tagging and Early Structuring
  - Sorted Outer Union Approach
    - Sort on its id fields
    - The children of a parent nodes are grouped together after the parent
    - Tuples having NULL values in sort field are placed before tuples having non-NULL values
    - **Scales well for large data sets**
  - Tagging requires only memory to keep the parent ids of the last tuple seen

Publishing with support for Views

- Provide XML views over relational data
- Allow queries on these views using an XML query language
- Steps
  - **Query composition with view definition**
  - Query optimization
  - Various structuring and tagging techniques
  - Generation of XML

Query Composition

- XML view is not necessarily materialized
  - Materializing the entire XML view incurs unnecessary computations
- Retrieve only the required fragment of relational data
  - Push computation into relational engine as much as possible

SilkRoute

- XML view is defined using a declarative RXL query language
  - Allows complex structure and arbitrary levels of nesting
  - Do not address the ordering issue in XML document
- Accept XML-QL queries over the defined view
- XML-QL queries are composed with the view
  - Generates another RXL query

XML-QL

- [Deutch, Fernandez, Florescu, Suciu: WWW 99]
- Similar structure to SQL: "WHERE-IN-CONSTRUCT"
  - WHERE – specify XML pattern to search
  - IN – specify data sources to search
  - CONSTRUCT – specify the format of XML result
- Support nested query
- Can express selection, projection, join, grouping
- Can construct deeply nested XML elements

Example: Sample Relational Schema

Issues(ID, volume, number, articleID)
Articles(ID, title, initPage, endPage, authorID)
Authors(ID, name, authorPosition)

Example: Desired XML Output

```
<articles>
  <article id = "a1">
    <title> XML Research Issues</title>
    <authors>
      <author>Tom Hanks</author>
    </authors>
  </author>
  <article id = "a2">
    <title>Storage and Retrieval of XML Data Using Relational DBMS</title>
    <authors>
      <author>Surajit Chaudhuri</author>
      <author>Kyuseok Shim</author>
    </authors>
  </author>
<articles>
```


Example: View Definition using RXL

- Allow nested and block structures
- A left-outer join
- Skolem function is used to group elements

```
construct
 <articles ID=Articles()>
    from articles $a
    construct
    {  <article ID=Article($a.ID)>
         <title>$a.title</title>
         <authors>
         {  from authors $b
            where $a.authorID = $b.ID
            construct
            <author>$b.name</author>
         }
         </authors>
       </article>
    }
 </articles>
```

XML-QL Query on the View

```
construct
  <result> {
    where <articles>
              <article>
                <title> $t </title>
                <authors>
                  <author>
                    <name $n</name>
                  </author>
                </authors>
              </article>
            <articles> in "sigmodRecord.xml",
            $n = "Tom Hanks"
     construct
       <title> $t </title>
  } </result>
```


Composed RXL query

```
construct
  <result>
  {
      from articles $a, authors $b
      where $a.authorID = $b.ID,
              $b.name ="Tom Hanks"
     construct
        <title> $t </title>
  } </result>
```

Query Composition

- Construct the view tree consisting of a global template and a set of Datalog rules
  - Obtain the global template by merging templates from all construct clause
  - Two template are merged if and only if they share the same Skolem function
  - Construct one Datalog rule for each Skolem function
  - Datalog rules are non-recursive

Query Composition

- The composed query is the union of all possible matches
  - Construct one block for each match
  - Perform minimization to eliminate redundancies for each block (could be expensive)

Query Processing

- Sorted outer-union approach
  - Construct one large SQL query from XML query and XML view
  - The SQL query consists of several left-outer joins, which are combined in outer unions
- Fully partitioned strategy
  - Construct multiple SQL queries without outer joins and unions
  - Each result is sorted to permit merging and tagging

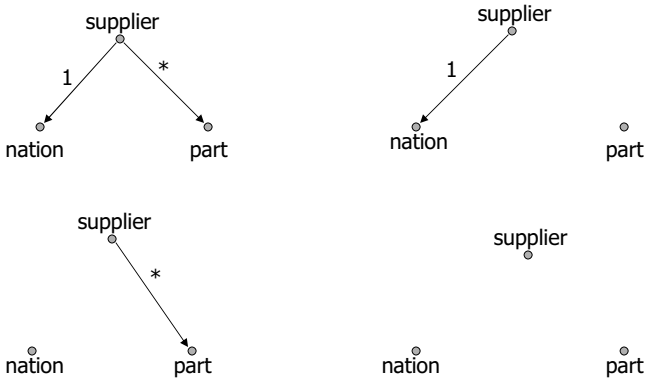
Query Processing

- The optimal plan lies between two extreme strategies
  - Need to optimize!
  - The number of possible translations of an RXL query into SQL is 2^E where E is the number of edges in the view tree
  - SilkRoute uses heuristic to choose a good plan

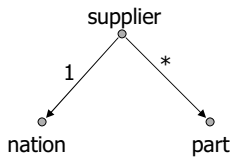
Example

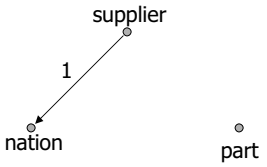
```
From supplier $s
Construct
  <supplier>
    <name>$s.name</name>
    {  from nation $n
       where $s.nationkey = $n.nationkey
       construct
         <nation>$n.name</nation> }
    {  from PartSupp $ps, Part $p
       where $s.suppkey = $ps.suppkey,
             $ps.partkey = $p.partkey
       construct
         <part><name>$p.name</name></part> }
  </supplier>
```


Execution Plans



Transformed SQLs






```
select s.superkey, n.name, Q.partkey, Q.name
from supplier s, nation n
where s.nationkey = n.nationkey
left outer join
   (select ps.suppkey as suppkey, p.name as pname
    from PartSupp ps, Part p
    where ps.partkey = p.partkey
   ) as Q
on s.suppkey = Q.suppkey
order by s.suppkey
```

```
select s.superkey, n.name
from supplier s, nation n
where s.nationkey = n.nationkey
order by s.suppkey

select s.suppkey, p.name
from supplier s, Part p, PartSupp  ps
where s.suppkey = ps.suppkey and
        ps.partkey = p.partkey
order by s.suppkey
```


XPERANTO

- Automatically creates a default XML view
  - Top-level elements correspond to table names
  - Row elements are nested under the table elements
  - Within a row element, column names appear as tags and column values appears as text
- Then, transform the default view into desired XML format
  - XQuery language is used to express the desired XML view

XPERANTO

- XML Query Graph Model (XQGM) is used as an internal representation
  - XQGM is a generalization of QGM model used for DB2
  - Enables translation of XQuery queries to SQL
  - Exploits XML query algebra
- Removes all XML navigation operators
  - Thus avoids construction of unnecessary intermediate XML fragments
  - Composition rules are used for elimination
- Pushes down join and selections into relational engine
  - Query Decorrelation
  - Tagger Pull-up

Example: Desired XML

```
<article id = "a1">
  <title> XML Research Issues</title>
  <authors>
    <author>Tom Hanks</author>
  </authors>
</article>
<article id = "a2">
  <title>Storage and Retrieval of XML Data Using Relational DBMS</title>
  <authors>
    <author>Surajit Chaudhuri</author>
    <author>Kyuseok Shim</author>
  </authors>
</article>
```

Example: Default XML View

```
<db>
  <Issues>
    <row>
      <ID>1</ID><volume>1</volume><number>1</number><articleID>a1</articleID>
    </row>
  </Issues>
  <Articles>
    <row>
      <ID>a1</ID><title>XML Research Issues</title>
      <initPage>1</initPage><endPage>5</endPage><authurID>1</authorID>
    </row>
    .....
  </Articles>
  <Authors>
    <row>
      <ID>1</ID><name>Tom Hanks</name><authorPosition>1</authorPosition>
    </row>
    .....
  <Authors>
</db>
```


Example: XML View Definition and Query

```
Create view articles as (
  for $article in view("default")/Articles/row
  return
     <article id = $article/ID>
       <title>$article/title></title>
       <authors>
         for $auth in view("default")/Authors/row
         where $article/authorID = $auth/ID
         return
             <author>$auth/name</author>
       </authors>
     </article>
```

Query

```
for $articles in view("articles")
where $articles/article/authors/author/text() = "Tom Hanks"
return $articles/article/title
```

Storage of Native XML in Relational Databases

How to store Native XML Data?

- Generic Mapping
  - No user-defined mapping, no DTD, no use of data
- User provides mapping to relational tables (discussed for commercial products)
- Infer mapping from DTD or XML Schema
- Analyze XML data and query workload

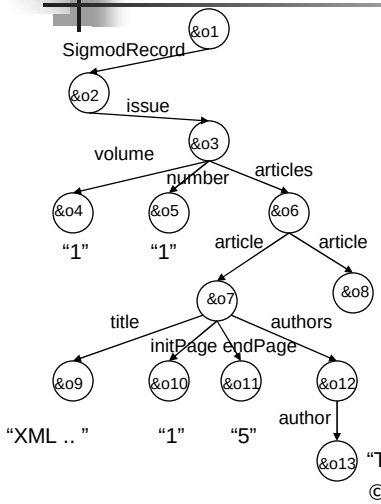
Generic Mapping

- [Florescu, Kossmann: IEEE Data Eng. Bulletin 99]
- Alternative ways to store edges for graphs
  - Edge Approach
    - Stores all edges in a single table
  - Binary Approach
    - Group all edges with the same label into one table
  - Universal Table
    - Result of a full outer join of all binary tables
    - Many fields with null
    - A lot of redundancy

Generic Mapping (2)

- Alternative ways to map values
  - Separate Value Table
  - Inlining
    - Keep columns for all types together
    - Thus, there could be many null values
- Binary approach + inlining shows the best overall
  - Most of data in a single edge table is irrelevant for a query
  - Universal table approach performs poorly for query with large results
  - Binary approach processes only relevant data
  - Inlining saves the cost of the joins with the Value tables

A Edge Table with Separate Value Tables



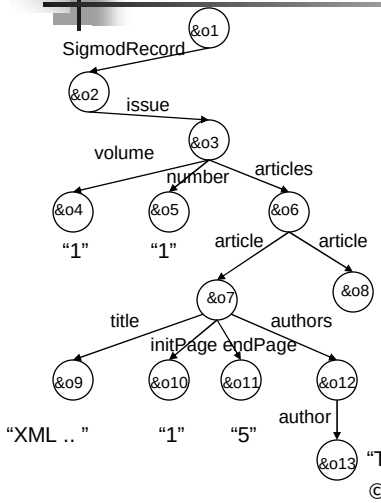
Edge Table

Source	Ordinal	Tag	Flag	Target
&o1	1	sigmodRecod	ref	&o2
&o2	1	issue	ref	&o3
&o3	1	volume	string	&o4
&o3	2	number	string	&o5
&o3	3	articles	ref	&o6
&o6	1	article	ref	&o7
&o6	2	article	ref	&o8
&o7	1	title	string	&o9
&o7	2	initPage	string	&o10

Vstring

Node	Value
&o4	1
&o5	1
&o9	XML Research Issues
&o10	1
&o11	5
&o13	Tom Hanks

Binary Approach and Inlining Values



Aauthor

Source	Ordinal	intValue	stringValue	Target
&o12	1	null	Tom Hanks	&o13

Aarticle

Source	Ordinal	intValue	stringValue	Target
&o6	1	null	null	&o7
&o6	2	null	null	&o8

XML Query Translation to SQL

- A single edge table
  - For every projection and selection attributes, perform self joins
  - Simple path expressions: self-joins
  - Kleene closures (*): recursive SQL
- Normalized Tables
  - Use SilkRoute or XPERANTO
  - Limited for certain class of XML queries

A Single Edge Table

- [Tian, DeWitt, Chen, Zhang: Wisconsin TR 00]

SourceID	Ordinal	Tag	TargetID	Data
1	1	sigmodRecord	2	""
2	1	issue	3	""
3	1	volume	4	"1"
3	2	number	5	"1"
3	3	articles	6	""
6	1	article	7	""
6	2	article	8	""
7	1	title	9	XML Research Issues
7	2	initPage	10	"1"
7	3	endPage	11	"5"
7	4	authors	12	""
12	1	author	13	"Tom Hanks"

Query Transformation

XQuery: for $b IN document("sigmodRecord.xml")/sigmodRecord/issue
where $b/volume = "1"
return $b/articles/article/title

SQL: select t6.value
from edges t1, edges t2, edges t3, edges t4, edges t5, edges t6
where t1.TargetID = t2.SourceID and t2.TargetID = t3.sourceID
and t2.TargetID = t4.sourceID and t4.TargetID = t5.sourceID
and t5.TargetID = t6.sourceID
and t1.tag = "sigmodRecord" and t2.tag = "issue"
and t3,tag = "volume" and t3.value = "1"
and t4.tag = "articles" and t5.tag = "article" and t6.tag = "title"

Infers Mapping from DTD

- [Shanmugasundaram et al. 99]
- Use DTD to generate a relational schema
  - DTD graph is generated from DTD
  - Element graph is generated by depth first traversal of the DTD graph for each element
- Viable Approaches
  - Shared
  - Hybrid

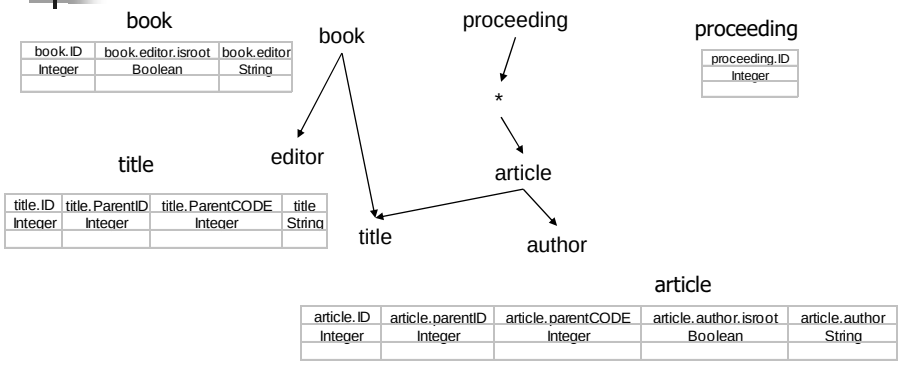
Shared and Hybrid Inlining Technique

- Shared Inlining
  - An element node is represented in exactly one relation
  - Relations are created for all elements having in-degree greater than one in DTD graph
    - Nodes with an in-degree of one are inlined
    - Nodes with an in-degree of zero are made into separate relations
  - Nodes below a * node are made into separate relations
  - Of mutually recursive elements all having in-degree one, one of them is made a separate relation
- Hybrid Inlining
  - Additionally inlines elements with in-degree greater than one that are not recursive or reached through a "*" node

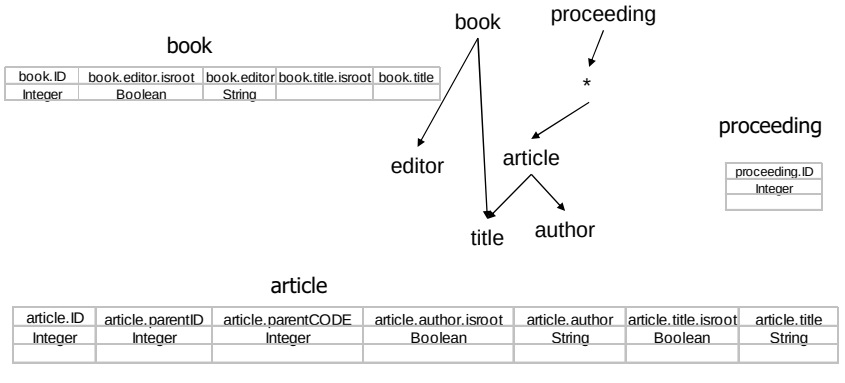
Example: DTD (SIGMOD Record)

```
<?xml version="1.0"?>
<!-- DTD -->
<!ELEMENT proceeding (article)* >
<!ELEMENT article (title, author) >
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT book (editor, title) >
<!ELEMENT editor (#PCDATA)>
```

Example: Shared Inlining Technique



Example: Hybrid Inlining Technique



STORED: Analyze XML Data

- [Deutsch, Fernandez, Suciu: SIGMOD'99]
- Semistructured data into relational data
- Integrate both relational and overflow systems
- Use data mining algorithm to find out frequent subtrees
  - There is no notion of DTD in semistructured data
- Overflow mapping is used to insure no loss of information
  - Overflow objects or object parts are stored in a separate semistructured data object repository

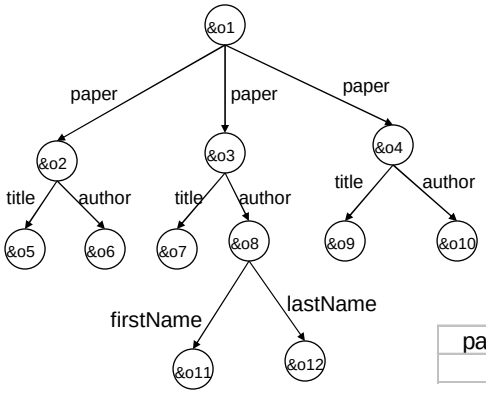
STORED: Overview of Technique

- Compute minimal path prefixes
  - Generates all prefixes $l_1 \ldots l_k$ with min support
- Discover frequent subtrees (patterns)
  - Use WL's algorithm
- Select K_0 patterns
  - Greedily choose a subset of K_0 patterns
- Select required attributes
- Generate mapping of K_0 patterns using STORED query languages
- Generate overflow mapping

Example

- Assume
  - The semistructured data has either
    - One author with string type
    - Or author having first name and last name
  - One author type appears frequently
- One relation for frequent structure and overflow relations G1 and G2 are generated

Example: Relational Schema



paper

ID	title	author
2		
3		
4		

G1

paperID
3

G2

source	tag	dest
3	firstName	4
3	lastName	5

Structural Summary and Indexing for XML

Structural Summary and Index

- Structural summary only
  - DTD
  - XML Schema
  - Representative Objects
- Structural summary and Index
  - Strong Dataguide
  - T-index: 1-index, 2-index
- Index only
  - Extensions of Inverted index
  - Access Support Relations
  - ToXin
  - Index Fabric

Why Structural Summary?

- Can help users to formulate meaningful queries
- Allows query processor to restrict the search space to only relevant portions of XML data
- Can be used to design an efficient relational representations for XML data
  - XML document may not always have an accompanying DTD or XML Schema
- Can be stored as a single edge table representation

Representative Objects

- [Nestorov, Ullman, Wiener, Chawathe: ICDE'97]
- Provides a concise representation of the inherent schema of semistructured data
- Full Representative Objects (FRO)
  - Describe the global structure of the data
  - Similar to Dataguide
- Degree-K Representative Objects
  - Describe the local aspects of the data by considering only paths of length K
  - Take less space than FRO
  - Construction time may be less than that of FRO

XTRACT

- [Garofalakis, Gionis, Rastogi, Seshadri, Shim: SIGMOD 00]
- Infers concise and semantically meaningful DTDs for XML documents
- Generalization
  - Generate zero or more candidate DTDs by replacing patters in the data with meta-characters like *
  - e.g. abab => (ab)*, bbbe => b*e
- Factorization
  - Factors common subexpressions from the candidate DTDs
  - e.g. b*d | b*e => b* (d | e)
- Minimum Description Length (MDL) Principle
  - MDL choose the minimum cost candidate DTD

Strong Dataguide

- [Goldman, Widom: VLDB 97]
- Summary of path structure from the root
  - Concise: describes every label path in the data once
  - Accurate: every label path in the data appears in the data guide
- Representation of the equivalent classes based on a deterministic automata (DFA)

Strong Dataguide

- All label paths reaching the same node in dataguide belong to the same equivalence class
- Construction is equivalent to the conversion of non-deterministic automata (NFA) into deterministic automata (DFA)
- Linear time for tree structured data
- Exponential time for graph structured data
- The size may be exponential in database size

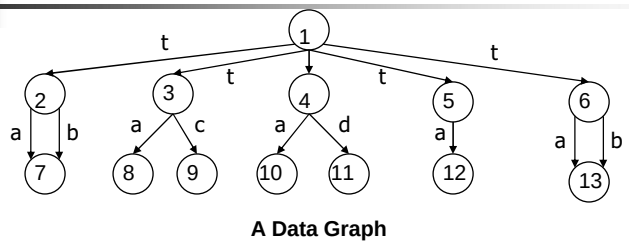
T-index

- [Milo and Suciu: ICDT 99]
- 1-index
  - To support queries of s_0 P v_1
    - s_0 : root
    - P: a regular expression
    - E.g. s0//title
  - Summary of path structure from the root
  - Representation of the equivalent classes based on a non-deterministic automata using bisimulation
  - Extents are disjoint
  - More compact size than dataguides
    - The size is at most linear in database size

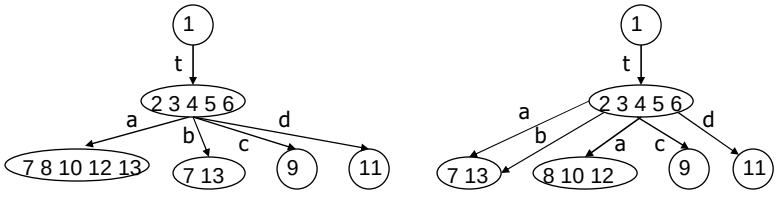
T-index

- 2-index
  - To support queries of $s_0\ v_1\ P\ v_2$
    - E.g. s0//articles//title
  - Summary of path structure between two arbitrary nodes in database
  - Intended to find pairs of nodes matching some arbitrary path expression
- T-index
  - To support queries of $s_0\ v_1\ P_1\ v_2\ P_2\ v_3\ P_3\ \ldots\ P_n v_n$

Exmaple



A Data Graph



Strong Dataguide **1-index**

Access Support Relations

- [Kemper, Moerkotte: IS 92]
- To support join along arbitrary reference chains between two object instances
- Not straight-forward to use with XML document without schema information
  - They are based on the paths in the schema
- Materialize access paths of arbitrary length

ToXin

- [Rizzolo: MS Thesis 01]
- Supports navigation of forward and backward traversals
  - Strong dataguide and 1-index support only forward traversal from the root
- Supports not only regular path expressions but also predicates over values
  - Access support relations and T-index store only predefined subsets of paths
- The index size is linear w.r.t. the size of database
  - Strong dataguide has exponential growth

Extension to Inverted Index

- Can be implemented as an edge table in RDBMS
- Assign a range to each element
- Containment is used to decide ancestor-descendant relationship
- [Chun Zhang at al. :SIGMOD'01]
  - Text words: T-index - (docno, wordno, level)
  - Elements: E-inde - (docno, begin:end, level)
- [Quanzhong Li and Bongki Moon: VLDB'01]
  - Each element in XML has a pair of numbers <order, size>
  - If x is a parent of y,
    - order(x) < order(y)
    - order(y) + size(y) <= order(x) + size(x)

Example: T-index and E-index

```
<articles>
  <article>
    <title> XML Research
   Issues</title>
    <authors>
      <author>Tom
   Hanks</author>
    </authors>
  </article>
<articles>
```

E-index

<articles>	(1, 1:15,0)
<article>	(1, 2:14, 1)
<title>	(1, 3:7, 2)
<authors>	(1, 8: 13, 2)
<author>	(1, 9:12, 3)

T-index

XML	(1, 4, 3)
Research	(1, 5, 3)
Tom	(1, 10, 4)
Hanks	(1, 11, 4)

Example

- XPath Expression: articles//author
  - The inverted list of articles and author are retrieved
  - The containment is checked for both lists

```
select *
from ELEMENTS e1, ELEMENTS e2
where e1.term = 'articles' and e2.term = 'author' and
        e1.docno = e2.docno and
        e1.begin < 2.begin and e2.end < e1.end
```


Index Fabric

- [Cooper at al.: VLDB'01]
- Encodes paths in XML data as strings
- Insert these strings into an efficient index for strings
- The index block and XML data are both stored in relational database system
- Evaluation of queries encode the desired path traversal as a search key string and perform a lookup
- Use Patricia trie to index a large number of strings

Final Thoughts

- XML enables data exchange and interoperability
- Efficient and flexible publishing of relational information will continue to be important
  - More work on query processing infrastructure as application needs evolve
  - Middleware v.s. server
- Native XML storage in the crossroads
  - Document Storage, LOB and structured search all play roles
  - Current support in RDBMS not yet sophisticated
  - Applications need to frame performance criteria
- Eventually performance yardsticks and benchmarks will be necessary

References

- Serge Abiteboul, Sophie Cluet, Tova Milo: Querying and Updating the File. VLDB 1993
- Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, Janet L. Wiener: The Lorel Query Language for Semistructured Data. Int. J. on Digital Libraries 1(1), 1997
- S. Abiteboul, P.Buneman, D. Suciu: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann, 1999
- D. Barbosa, A. Barta, A. Mendelzon, G. Mihaila, F. Rizzolo, P. Rodriguez-Gianolli: ToX - The Toronto XML Engine, International Workshop on Information Integration on the Web, Rio de Janeiro, 2001.
- Elisa Bertino, Won Kim: Indexing Techniques for Queries on Nested Objects. TKDE 1(2), 1989
- Ronald Bourret, XML and Databases, http://www.rpbourret.com/xml/XMLAndDatabases.htm
- Ronald Bourret, Mapping DTDs to Databases, http://www.rpbourret.com/xml/
- Michael J. Carey, Jerry Kiernan, Jayavel Shanmugasundaram, Eugene J. Shekita, Subbu N. Subramanian: XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents. VLDB 2000
- Donald D. Chamberlin, Jonathan Robie, Daniela Florescu: Quilt: An XML Query Language for Heterogeneous Data Sources. WebDB 2000
- Qiming Chen, Yahiko Kambayashi: Nested Relation Based Database Knowledge Representation. SIGMOD Conference 1991
- Vassilis Christophides, Sophie Cluet, Jérôme Siméon: On Wrapping Query Languages and Efficient XML Integration. SIGMOD Conference 2000: 141-152

References

- Brian Cooper, Neal Sample, Michael J. Franklin, Gisli R. Hjaltason, Moshe Shadmon: A fast index for semistructured data. VLDB 2001
- Alin Deutsch, Mary F. Fernandez, Dan Suciu: Storing Semistructured Data with STORED. SIGMOD Conference 1999
- Alin Deutsch, Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, Dan Suciu: A Query Language for XML. WWW, 1999
- Mary F. Fernandez, Dan Suciu: Optimizing Regular Path Expressions Using Graph Schemas. ICDE 1998
- Mary F. Fernandez, Wang Chiew Tan, Dan Suciu: SilkRoute: trading between relations and XML. WWW 2000
- Mary F. Fernandez, Atsuyuki Morishima, Dan Suciu: Efficient Evaluation of XML Middle-ware Queries. SIGMOD 2001
- Minos N. Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri, Kyuseok Shim: XTRACT: A System for Extracting Document Type Descriptors from XML Documents. SIGMOD Conference 2000
- Roy Goldman, Jennifer Widom: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. VLDB 1997
- Roy Goldman, Jason McHugh, Jennifer Widom: From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. WebDB, 1999

References

- Carl-Christian Kanne, Guido Moerkotte: Efficient Storage of XML Data . Technical Report 8/99, University of Mannheim, 1999
- Alfons Kemper, Guido Moerkotte: Access Support Relations: An Indexing Method for Object Bases. IS 17(2)
- Quanzhong Li, Bongki Moon: Indexing and querying XML data for regular path expressions. VLDB 2001
- Hartmut Liefke, Dan Suciu: XMILL: An Efficient Compressor for XML Data. SIGMOD Conference 2000
- Jason McHugh, Jennifer Widom: Query Optimization for XML. VLDB 1999
- Tova Milo, Dan Suciu: Index Structures for Path Expressions. ICDT 1999
- Svetlozar Nestorov, Jeffrey D. Ullman, Janet L. Wiener, Sudarshan S. Chawathe: Representative Objects: Concise Representations of Semistructured, Hierarchial Data. ICDE 1997
- F. Rizzolo, A. Mendelzon: Indexing XML Data with ToXin, Fourth International Workshop on the Web and Databases, Santa Barbara, CA. 2001
- Michael Rys: Bringing the Internet to Your Database: Using SQLServer 2000 and XML to Build Loosely-Coupled Systems. ICDE 2001
- Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, Jeffrey F. Naughton: Relational Databases for Querying XML Documents: Limitations and Opportunities. VLDB 1999

References

- Jayavel Shanmugasundaram, Eugene J. Shekita, Rimon Barr, Michael J. Carey, Bruce G. Lindsay, Hamid Pirahesh, Berthold Reinwald: Efficiently Publishing Relational Data as XML Documents. VLDB 2000
- Jayavel Shanmugasundaram, Jerry Kiernan, Eugene J. Shekita, Catalina Fan, John Funderburk: Efficiently Publishing Relational Data as XML Documents. VLDB 2001
- Feng Tian, David J. DeWitt, Jianjun Chen, and Chun Zhang: The Design and Performance Evaluation of Various XML Storage Strategies, Technical report, University of Wisconsin
- F. Rizzolo. ToXin: An Indexing Scheme for XML Data. M.Sc. Thesis, Department of Computer Science, University of Toronto, Canada. January 2001.
- Chun Zhang, Jeffrey F. Naughton, David J. DeWitt, Qiong Luo, Guy M. Lohman: On Supporting Containment Queries in Relational Database Management Systems. SIGMOD 2001
- Justin Zobel, James A. Thom, Ron Sacks-Davis: Efficiency of Nested Relational Document Database Systems. VLDB 1991

References (W3C)

- W3C Recommendation. Extensible Markup Language (XML) 1.0 (Second Edition) In http://www.w3.org/TR/REC-xml. 2000
- W3C Recommendation. Namespaces in XML In http://www.w3.org/TR/REC-xml-names. 1999
- W3C Recommendation. XML Path Language (XPath) 1.0. In http://www.w3.org/TR/xpath. 1999
- W3C XML representation of a relational database In http://www.w3.org/XML/RDB.html
- W3C Recommendation. XML Shcema Part 0: Primer In http://www.w3.org/TR/xnlschema-0. 2001
- W3C Recommendation. XML Shcema Part 1: Structure In http://www.w3.org/TR/xnlschema-1. 2001
- W3C Recommendation. XML Shcema Part 1: Datatypes In http://www.w3.org/TR/xnlschema-2. 2001
- W3C Recommendation. XSL Transformations (XSLT) 1.0. In http://www.w3.org/TR/xslt. 1999
- W3C Working Draft. XQuery 1.0: An XML Query Language In http://www.w3.org/TR/xquery. 2001

References (Products)

- Sandeepan Banerjee, Vishu Krishnamurthy, Muralidhar Krishnaprasad, Ravi Murthy: Oracle8i - The XML Enabled Data Management System. ICDE 2000
- Michael Rys: Bringing the Internet to Your Database: Using SQLServer 2000 and XML to Build Loosely-Coupled Systems. ICDE 2001
- Josephine M. Cheng, Jane Xu: XML and DB2. ICDE 2000
- Ronald Bourret: XML Database Products: In http://www.rpbourret.com/xml/XMLDatabaseProds.htm, July 2001
- Michael Rys: Bringing the Internet to Your Database: Using SQLServer 2000 and XML to Build Loosely-Coupled Systems. ICDE 2001: 465-472
- Josephine M. Cheng, Jane Xu: XML and DB2. ICDE 2000: 569-573
- eXcelon: Extensible Information Server White Paper. eXcelon Corporation, 2001
- IBM DB2 Universal Database for AS/400 XML Extender Administration and Programming Version 7. 2001
- Microsoft SQL Server Books Online
- Oracle8i Application Developer's Guide – XML Release 3 (8.1.7). 2000