Native XML Databases

An Overview - based on material by Ronald Bourret
http://www.rpbourret.com

Native XML Database (NXD)

Native XML Databases are databases designed especially to store XML documents, preserving
• Document order
• Processing instructions
• Comments
• CDATA sections
• Entity usage

Aim to provide typical database characteristics such as
• Transactions
• Security
• Multi-User access
• Programmatic APIs
• Query Languages

Support XML document orientated queries
• “Get me all documents in which the third paragraph after the start of the section contains an italicised word”
Database Definition
[Elsmari and Navathe]

A database is a collection of related data

A database management system is a collection of programs that enables users to create and maintain a database.

The DBMS is hence a general-purpose software system that facilitates the process of defining, constructing and manipulating databases for various applications.

Building on this Database Definition

A XML database is a collection of related XML documents

A native XML database management system is a collection of programs that enables users to create and maintain a XML database. The DBMS is hence a general-purpose software system that facilitates the process of defining, constructing and manipulating XML databases for various applications.
Formal definition
[XML:DB initiative]

A native XML database:

- Defines an XML data model
  - Model must include elements, attributes, text, and document order
  - Examples are XPath, Info Set, DOM, and SAX
  - XQuery model will be de facto standard in future?
- Uses a document as its fundamental unit of (logical) storage
  - Equivalent structure in RDBMS is a row
- Can have any physical storage
  - Tables for SAX "objects" in relational database
  - DOM objects in object-oriented database
  - Binary file format optimized for XPath data model
  - Hashables of XPaths and values
  - Compressed, indexed XML documents in file system

Use Cases
Use case: Tabular data

Manufacturing data
Accounting data
Employee data
Scientific data

... Use a native XML DB only if:
  • You want to lose your job.
  • Relational databases excel at storing tabular data
  • Systems are mature, fast
  • Few reasons to change existing code or DBMS

Use case: Documents

Product documentation
Static Web pages
Presentations
Advertisements

... Use a native XML DB because:
  • Structure too irregular for relational model
  • Physical info (entities, CDATA, ...) is important
  • Need document-centric queries
    – Get first chapter with a paragraph containing “XML.”
    – Get headings of all chapters that contain figures
  • Want to retain document identity
Use case: Semi-structured data

Some structure, but not regular
- Structured data: white pages
- Semi-structured data: yellow pages

Examples
- Health data
- Result of EAI (Enterprise Application Integration)
- Geneological data

Use a native XML DB because:
- Difficult to store in a relational database
  - Use single, sparsely populated table or...
  - Many tables (one per subject area)
- Documents might contain arbitrary extensions

Use case: “Natural” format is XML

Only format is XML
- XSLT stylesheets

Data stored temporarily as XML
- Long-running transactions
- Enterprise application integration (EAI)
- Documents in a message queue

Schema-less documents
- No schema or schema not known

Use a native XML DB because:
- Want query language, security, transactions, etc.
- No reason to use an XML-enabled database
  - Natural format is XML
  - Mapping arbitrary documents at run-time is inefficient, error-prone
Use case: Large documents

Need entire document in memory
- DOM tree
- XSLT transformation
- etc.

Document too large for memory
Use a native XML DB because:
- In-memory tree lazily instantiated
  - Unused nodes kept on disk
- Tree size limited only by:
  - Disk size
  - Database address space

Use case: Archiving

Long-term document storage
Often required by law
- Pharmaceutical industry
- Contracts
- Financial transactions

Use a native XML DB because:
- Want query language, security, transactions, etc.
  - Better than file system or BLOBs
- Retains complete document
  - Retaining original document might be a problem
- May support versioning through updates or diffs
Features of NXDs

Document Collections
- Play a role similar to a directory in a file system
- E.g. define a sales order collection so queries limited over that set

Query Languages
- Standards XPath and XQuery from W3C becoming popular
- Proprietary

Updating and Deleting documents or fragments
- Quite often proprietary
- “Vendor-neutral” Xupdate standard being promoted by XML:DB.org

Transaction, Locking and Concurrency
- All NXDs support transactions but locking generally at level of “document”
- Depending on definition of “document” it can imply low level of multi user
- Fragment level locking a research issue

Application Programming Interfaces (APIs)
- ODBC-like interfaces common: connect to db; explore metadata; execute query; retrieve results
- Results can be returned in variety of ways such as XML string, DOM tree, SAX parser etc.
- “Vendor-neutral” API being promoted by XML:DB.org
Features of NXDs

Round Tripping
- Important for document centric applications obviously, but less so for data centric ones
- Store XML document in NXD and when you retrieve from NXD you get the “same” document
- Generally not possible in XML enabled traditional databases

Indexing
- All NXDs support the indexing of element and attribute values to enable the query engine to jump quickly to appropriate fragment of document

External Entity Storage
- No single answer to question “expand entity or not”
  - E.g. a reference to a CGI program that gets current weather
How does an NXDBMS organize the documents?

Conceptual representation:

- A **composite** collection comprised of collections and XML documents.
- A **simple** (nested) collection.
- A **root** collection representing the whole database.

*Collections* help us in housekeeping the contents of our database.

What other benefits do we get from collections?

- A simple (nested) collection.
- A root collection representing the whole database.

How does an NXDBMS organize the documents?

The collections concept can be further elaborated, since every root collection (database instance) must be contained in the global system root collection.

The system root collection is under the control of the Collections Manager.

Now a NXDBMS can host unlimited number of distinct XML database instances.
How does an NXDBMS organize the documents?

Benefits of *collections*:

- A collection of XML documents allows us to **access** (query or modify) **multiple documents**.

- The collections **do not require a “schema for collections”**, namely, they do not have to obey any particular rule or constraint.
  
  - The relational model is based on relationships and on referential integrity. Constraints are prevalent.
  
  - Here, the schema existence is optional. This is a good or bad property?
    
    Answer: both, it depends.
    
    - **bad**, because there is **reduced**, if not existed at all, **referential integrity** among the constructs.
    
    - **good**, because eventually we managed to **free our data** from the integrity constraints. This allows us to manipulate our documents with increased flexibility.

How does an NXDBMS organize the documents

Benefits of *collections* (cont’d):

- Restrict and create indexes on the same collection to increase performance.

- Frequently joined documents should belong to the same collection.

- Demystifies and simplifies the mechanism of opening a connection to an XML database.
  
  - In relational systems the JDBC API opens a connection to a database using an address like: “jdbc:odbc:COMPANY_DB”.
  
  - Similarly, the XML:DB API opens a connection to a collection using an address like: “xmldb:mysystemdb:///COMPANY_DB”.
How does an NXDBMS organize the documents?

From the previous figure we conclude that a database can be everywhere, since even the simplest collection of documents, by definition, comprises a database.

Hence we can depict the following tree of databases or tree of collections, where each node represents an XML database, that is essentially contained within its parent node.

- **Level 0:** system
- **Level 1:** physical databases
- Deeper levels: virtual databases

**There are databases everywhere!**

The granularity increases as we go deeper in the tree.

- The indexes are created bottom-up but, of course, the databases are created top-down.

- An index that is created on a child collection is also available to its parent collection (and to all of its ancestors).
  - This is good, because a specific collection "knows" and applies those indexes that are the best for this collection.
  - Each upper level does not have to know about the lower level indexes (index-isolation). It only has to create its own indexes. For what purpose are these indexes created? See below.

- As we move up to the upper levels of the collections-tree the only need to create a new index is for joining those XML documents that belong to the descendant collections.

Storage: The actual information, the fundamental unit of storage (the XML documents) are contained in the leaf collections. The Storage Manager, in collaboration with the Collections Manager, has to store only these lower level collections. All the others are virtual.
How does an NXDBMS organize the documents?

Indexing

Classic file organisation techniques used for indexing structure
- E.g. inverted file

Different indexes can be kept, indexing
- Values
- Parent of an element
- Paths (sequences of elements)
- Etc.
Example NXD: eXist

**eXist**

- **Developer:** Wolfgang Meier
- **URL:** [http://exist.sourceforge.net](http://exist.sourceforge.net)
- **License:** Open Source
- **Database type:** Flexible

### Supports

- XPath
- XML:DB API
- Xupdate
- Form of Triggering

---

**Implementation and Performance**
A “text based” native XML database is one that stores XML as text
- File in File system, CLOB in RDB or...
- May need to parse documents at run time
- Uses indexes to avoid extra parsing, increase search speed

A “model based” native XML database is one that builds an internal object model (when loaded/parsed) from the document and stores this model
- Performance of Model based NXDs which use other databases (e.g. relational) for storage will vary greatly depending on design
- Most products that use a proprietary storage format optimised for their model will have performance similar to text based NXD
- Uses indexes to speed searches
Model-based databases

Proprietary
- Tamino, Xindice, Neocore, Ipedo, XStream DB, XYZFind, Infonyte, Virtuoso, Coherity, Luci, TeraText, Sekaiju, Cerisent, DOM-Safe, XDBM, ...

Relational
- Xfinity, eXist, Sybase, DBDOM

Object-oriented
- eXcelon, X-Hive, Ozone/Prowler, 4Suite, Birdstep

Performance

No public benchmark data?
Native vendors say native is faster
Relational vendors say relational is faster
Native XML databases rely heavily on indexing
- Slows update times
Some guesses are possible
Whole documents and fragments (Text-based databases)

Should be very fast
- Data is contiguous on disk
- Retrieval requires index lookup and single disk read

Whole documents and fragments (Model-based databases)

Databases with proprietary stores should be fast
- Can use physical pointers between nodes
Databases built on other DBs may be fast or slow
- Depends on underlying database and implementation
Unindexed data

Slow for model-based databases
- Must read many elements, not just particular type
- Comparisons may be slower due to converting text

Very slow for text-based databases
- Must parse document as well as comparing values

Unindexed data: Optimization

/Order[Date="29.10.00"]
- Only need to search children of Order
- Schema may help locate Date element

//Order[Date="29.10.00"]
- Schema may help locate Order and Date elements
- Without schema, must search entire document
Query return types

Text-based databases
- Very fast returning strings
- Slow returning DOM or SAX due to parsing

Model-based databases
- Probably similar to XML-enabled databases

Queries not following storage hierarchy

The big question ...
Slower than hierarchical queries
Pointers to parents may help model-based DBs

Get the dates and numbers of all sales orders for part "A-10"
1. Index lookup for part “A-10”
2. Follow pointers or index to Order
3. Search children for Number, Date
Code Examples

XML:DB API

Database-neutral API from XML:DB Initiative
Similar to JDBC, OLE DB, etc.

- Implemented by database drivers
- Methods to connect to database, execute queries, etc.
- Uses separate query language(s)
JDBC example: INSERT

// Instantiate the Sun JDBC-ODBC bridge driver. The driver registers
// itself with the driver manager.
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

// Get a connection to the Orders database and turn auto-commit off.
Connection conn = DriverManager.getConnection("jdbc:odbc:Orders",
   "rpbourret",
   "my_password");
conn.setAutoCommit(false);

// Get a Statement.
Statement s = conn.createStatement();

JDBC example: INSERT (cont.)

// Execute INSERT statements to insert new rows into the
// Orders and Items tables.

s.executeUpdate("INSERT INTO Orders " +
   "VALUES ('1234', 'Gallagher Co.',
   #10/29/00#)");

s.executeUpdate("INSERT INTO Items " +
   "VALUES ('1234', 'A-10', 12)");

s.executeUpdate("INSERT INTO Items " +
   "VALUES ('1234', 'B-43', 600)");

// Commit the transaction. Close the statement and the
// connection.

conn.commit();
s.close();
conn.close();
XML:DB example: INSERT

// Get the database (driver) for dbXML/Xindice. Register it
// explicitly with the database manager.
Class c = Class.forName("org.dbxml.client.xmldb.DatabaseImpl");
Database db = (Database) c.newInstance();
DatabaseManager.registerDatabase(db);

// Get the Orders collection.
Collection coll =
    DatabaseManager.getCollection("xmldb:dbxml:///db/Orders",
    "rpbourret",
    "my_password");

// Get a transaction service. XML:DB is designed to use "services",
// such as transaction, collection management, and query services.
TransactionService txn =
    (TransactionService)coll.getService("TransactionService", "1.0");

XML:DB example: INSERT (cont.)

// Start a new transaction.
txn.begin();

// Create a new (empty) XMLResource in the Orders collection.
XMLResource res = (XMLResource)coll.createResource("Order1234",
"XMLResource");

// Read the contents of the Order1234.xml file into a string.
int b;
FileReader r = new FileReader("Order1234.xml");
StringWriter w = new StringWriter();
while ((b = r.read()) != -1)
{
    w.write(b);
}
XML:DB example: INSERT (cont.)

```java
// Set the contents of the new resource to the string.
res.setContent(w.toString());

// Commit the transaction.
txn.commit();

// Close the collection and deregister the database.
coll.close();
DatabaseManager.deregisterDatabase(db);
```

---

JDBC example: SELECT

```java
// Instantiate the Sun JDBC-ODBC bridge driver. The driver registers
// itself with the driver manager.
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

// Get a connection to the Orders database.
Connection conn =
   DriverManager.getConnection("jdbc:odbc:Orders",
   "rpbourret",
   "my_password");

// Execute a SELECT statement to get all orders from 10/29/00.
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("SELECT Number, Date, Customer " +
   "FROM Orders WHERE
   Date=#10/29/00#");
```
JDBC example: SELECT (cont.)

```java
// Get each row in the result set and print it.

while (rs.next())
{
    System.out.println("Order: " + rs.getString(1));
    System.out.println("Customer: " + rs.getString(2));
    System.out.println("Date: " + rs.getDate(3));
    System.out.println();
}

// Close the result set, the statement, and the connection.

rs.close();
s.close();
conn.close();
```

XML:DB example: SELECT

```java
// Get the database (driver) for dbXML/Xindice. Register it
// explicitly with the database manager.

Class c = Class.forName("org.dbxml.client.xmldb.DatabaseImpl");
Database db = (Database) c.newInstance();
DatabaseManager.registerDatabase(db);

// Get the Orders collection.

Collection coll =
    DatabaseManager.getCollection("xmldb:dbxml:///db/Orders",
        "rpbourret",
        "my_password");

// Get an XPath query service over the Orders collection. XML:DB
// is designed to use a variety of "services", such as different
// query engines.

XPathQueryService service =
    (XPathQueryService)coll.getService("XPathQueryService", "1.0");
```
// Execute an XPath query to get all orders from 29.10.00.
ResourceSet docs = service.query("/Order[Date="29.10.00"]");

// Iterate over the returned documents and print each one. We do
// not need to know any metadata (such as data types) here. This
// is because metadata is contained in the XML document itself.
ResourceIterator iterator = docs.getIterator();
while (iterator.hasMoreResources())
{
    XMLResource doc = (XMLResource)iterator.nextResource();
    System.out.println((String)doc.getContent());
    System.out.println();
}

// Close the collection and deregister the database.
coll.close();
DatabaseManager.deregisterDatabase(db);
Updating

Xquery does not support updates yet
How to update and check whether schema violated or not?

- XML doc in File System: overwrite document, and reparse
- XML doc in RDB: use update support of SQL, and use integrity constraints of DB to maintain mappings
- XML doc in NXD: use DOM or proprietary APIs or Xupdate and use proprietary mechanism of NXD to validate

XUpdate

Emerged from XML:DB initiative
An update is represented by an xupdate: modifications element in an XML document

Child elements include

- Xupdate:insert-before
- Xupdate:insert-after
- Xupdate:append
- Xupdate:update
- Xupdate:remove
- Xupdate:rename
- Xupdate:variable
- Xupdate:value-of
- Xupdate:if
Example

```xml
<xupdate:insert-after
    select="/person[contains(name,'Gill')]/phone />
    <xupdate:element name="address">
        <town>Dublin</town>
    </xupdate:element>
</xupdate:insert-after>

Result would be creation of element
<xupdate:element name="address">
    <town>Dublin</town>
</xupdate:element>
```

Example...

**INPUT**
```xml
<?xml version="1.0"?>
<addresses version="1.0">
    <address id="1">
        <fullname>AndreasLaux</fullname>
        <born day='1' month='12' year='1978'/>
        <town>Leipzig</town>
        <country>Germany</country>
    </address>
</addresses>
```

**XUPDATE**
```xml
<?xml version="1.0"?>
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate">
    <xupdate:insert-after select="/addresses/address[1]">
        <xupdate:element name="address">
            <xupdate:attribute name="id">2</xupdate:attribute>
            <fullname>Lars Martin</fullname>
            <born day='2' month='12' year='1974'/>
            <town>Leizig</town>
            <country>Germany</country>
        </xupdate:element>
    </xupdate:insert-after>
</xupdate:modifications>
```
Example…

```xml
<?xml version="1.0"?>
<addresses version="1.0">
  <address id="1">
    <fullname>Andreas Laux</fullname>
    <born day='1' month='12' year='1978'/>
    <town>Leipzig</town>
    <country>Germany</country>
  </address>
</addresses>
```

**XUPDATE**

```xml
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
xmlns:xupdate="http://www.xmldb.org/xupdate">
  <xupdate:insert-after select="/addresses/address[1]" >
    <xupdate:element name="address">
      <xupdate:attribute name="id">2</xupdate:attribute>
      <fullname>Lars Martin</fullname>
      <born day='2' month='12' year='1974'/>
      <town>Leizig</town>
      <country>Germany</country>
    </xupdate:element>
  </xupdate:insert-after>
</xupdate:modifications>
```

**RESULTS**

```xml
<?xml version="1.0"?>
<addresses version="1.0">
  <address id="1">
    <fullname>Andreas Laux</fullname>
    <born day='1' month='12' year='1978'/>
    <town>Leipzig</town>
    <country>Germany</country>
  </address>
  <address id="2">
    <fullname>Lars Martin</fullname>
    <born day='2' month='12' year='1974'/>
    <town>Leizig</town>
    <country>Germany</country>
  </address>
</addresses>
```

---

**Summary**

Useful characterisation of XML documents effects choice of technologies to use

- **Data centric**
- **Document centric**

**Data centric technologies**

- Middleware, XML enabled databases, XML data binding

**Document centric technologies**

- Content Management Systems

**Technologies suitable for both**

- XML Servers, NXDs

**Native XML databases**

- Attempt to provide normal database features
REQUIRED READING:
“XML and Databases” Ronald Bourret
• http://www.rpbourret.com/xml/XMLAndDatabases.htm