XML Databases & Applications

N. H. N. D. de Silva

(Slides are based on the slides by Chris Wallace)
Lecture Plan

- Native XML databases
- "Students Online" - a faculty information system
- "Ways of Seeing" - a collaborative, linked-data project using model-driven code
- "The Secession Papers" and "Bristol Sailing Association" - documents with embedded data
- "Gloucester Road Story" - crowd-sourcing spatial/temporal data
- Workshop - Location-aware Sculpture web application
What is an XML database?

- Most RDBMS (Oracle, DB2, SQL Server) can store XML as a column in a table as a kind of text value, and can query it with some kind of XPath. Here we are interested in Native XML databases which store all content as XML.
  - Well-formed XML documents can be stored in the database without the need of a schema
  - Complex XML documents can be stored as single structures without the need to 'shred' them into tables
  - Documents in the database can be queried using XPath expressions
  - Applications can be developed in XQuery and XSLT which are integrated with the database

Wikipedia
What are the best-known XML databases?

- The market leader is MarkLogic - corporate, expensive, high performance
- Sausalito has an interesting application deployment framework
- The leader in open-source projects is eXist which we use.

There are many others. They provide varying facilities and additional capabilities, so it's not easy to move applications from one to another.
What is XQuery?

XQuery is a functional programming language based on XPath2 and designed for querying an XML database (as SQL is designed for querying Relational Databases). With the support of additional modules it can be used to develop full web applications. (like having SQL and PHP in one language)

- [W3C standard](#)
- 30,000 ft view
- [XQuery wikibook](#)
- Some example applications [on a UWE server](#) and [on an Amazon EC2 instance running Centrix](#)
Why did you choose eXist?

- eXist web site
- open Source - team of developers in Europe and the US
- implemented in Java for portability
- Multiple interfaces - REST, XML:RPC and Web Dav
- eXist provides addition function libraries to support a wide range of functions required for web applications
- Documents are actually stored in an efficient B+ tree structure
- Documents are indexed in different ways, including the Lucene free-text index
- Documents organised into collections
- eXist supports updates to the database as well as reading
- Non-XML resources such as XQuery, CSS, JavaScript, JPEG can be stored as Binary objects
Case study
Case study: Ways of Seeing

- "Ways of Seeing" by John Berger is a classic text in art criticism.
- It has about 150 paintings reproduced but in black-and-white and rather small.
- Paintings made before 1922 are not copyright so colour images are often online.
- This project is about making an online version of the book.
A typical entry in the index is:

111 Portrait of himself and Saskia by Rembrandt van Ryn. 1606-69 Pinakotek, Dresden

and there is a scan of that page on the right.

What do you think would be a suitable data model to store this data? (5 minutes)
An ER model for the book illustrations

This took me a couple of iterations. Initially I had data on which page the painting was on in the same Painting entity as the title and artist, but they really belong in two separate entities, one about the Painting, the other about the use of the painting in a book, Illustration. I wanted to say things about the Book too, so that became an entity, even if there is only one at present.

- Basic diagram
- Detailed diagram
- HTML
- raw XML
Model-driven application

- Developing this model is a good thing to do but it's better still if we can use it to drive the application.

- The application supports browsing the book and its illustrations, but also supports editing the data.

- We will use it to add a link to an online copy of one of the paintings.
Linking Entities

- All Entities have a generated key unique across the whole database (p-130, a-12) and all have the same attribute 'id'. Thus we only have to provide the id and the application can find it in a single index.

```
$wos:data//*[id=$id]
```

Where

```
declare variable $wos:data := collection("/db/apps/berger/data");
```
Linking Entities

The relationship between Artist and Painting is one-to-many.

How would you implement this relationship in a Relational Database?
eXist provides a REST interface which allows us to execute XPath expressions in the URL.

What will this query retrieve? `//*[id='p-16']`

How would you express that query in English?
We can use XPath to answer questions like

- what paintings has a-12 created?

  //Painting[artist='a-12']

- what other paintings are in l-4

  //Painting[location='l-4']
Linking Entities

- This way of linking XML structures is similar to linking between objects in an object-oriented language or between records in a Relational Database.
- How is it different?
Derived data

- The data model supports derived data by having the XQuery expression stored in the model.

- **HTML** has a derived wikipage link, eg. Rembrandt. The XQuery fragment is

  \[
  \text{concat("http://en.wikipedia.org/wiki/", replace($entity/artist-name," ",","_"))}
  \]

- which uses XPath2 functions to create a **URL** from the artist name which may be the wikipage. $entity is a variable used in the XQuery script and used here a bit like self in Java.

- More complex XQuery is needed to calculate the age given the year of birth and death, and to count the **number of illustrations in a book**
One main advantage of a data-driven approach is that we can just change the model and the application will change too. For example, to add a link from Location to GoogleMap we make a new version of the model, add a new attribute which defines a URL. The form of the URL depends on whether the location has a latitude/longitude defined or not:

```xml
<attribute name="googlemap" type="uri" derived="true">
  <rule>
    concat("http://maps.google.com/maps?q=",
              if ($entity/latitude) then
                concat($entity/latitude,"",$entity/longitude,"($entity/location-name,")")
              else $entity/location-name)
  </rule>
</attribute>
```

- change the version number, and the new derived property is now present. If we don't like it, change it back
- Now we can use the link to find a location, get its lat/long and edit in the data to create a more accurate link.
- The code for this application is about 680 lines of code, of which about 50% is generic.

What are the pros and cons of a data-driven approach?
In eXist, XML documents are stored in a hierarchical file system. The way in which the total data is split into documents is an architectural decision. I have chosen to put all the data for an entity type in one document, but you could equally well have all the data created in a month in one document. Here, all the artists are stored in a single document which is updated in situation when the application is editing.

This XML structure is table-like and could be held in a RDMBS but:

- XML attributes are used to hold meta-data - who made this edit, (when the edit was made, what language)
- multiple elements of the same name can be held - every editor can make a comment - an RDMBS would need a separate table
- content can be a complex, nested document with multiple elements in a varied structure, such as an HTML page

Backup and restore are needed and the backup format is just XML. eXist functions allow XQuery scripts to do scheduled backups.
Case study

VIRGINIA IN THE CIVIL WAR: TEI
Case study: Virginia in the Civil War: TEI

- **Text Encoding Initiative** (TEI) is an XML format used in the humanities for representing literary and historical documents e.g. NZ Electronic Text Collection.
- The TEI schema is very complex but typically a project will choose a subset of the elements.
- The main task is document conversion - scanning, OCR, manual editing and additional markup for places, people and events.
- An XML database is ideal for storage and querying these documents. **Feb 13-May 1 1861**
- The University of Richmond in Virginia has a project on the Secession of Virginia during the Civil War.
The data is on 4 large, annotated documents + some reference files, stored in an eXist database. Feb 13-May 1 1861. The front-end is in PHP developed in Richmond, the back end in XQuery developed in Bristol.

part of a TEI document

The key problem is fast searching. We used Lucene a fulltext search engine integrated into eXist. Search results need to be sorted and paginated and extracted for display - all in XQuery

It was essential to develop unit tests to check the backend API was working after each change. The test framework is in XQuery and the test definitions are in XML.
Case study
Case Study: Bristol Sailing Association

- The members of my local sailing association go on about 30 sailing trips, mostly on charter boats. After a trip someone writes a brief report and these are published in a Word document which is put online on the current website.

  - e.g. May 2010

- Word is OK for personal documents but is not a good choice for a website Why not? Presenting data in this way means that basic data about a trip: the boat, the skipper, the crew, where and when the trip was - is buried in the text. That is where we want to read about it but it doesn't help to locate documents.

- For example, find all the trips which were skippered by Jeff Birkin. Why can't you just search for 'Jeff Birkin'?

- What do you think would be a suitable data model to store this data?
Trip Data model

- Document with annotations
- No attempt at referential integrity. E.g. any crew member can be added without checking against a reference table of Members.
- Document with annotations + reference tables
- Is Referential Integrity always necessary?
The new website treats Trip reports as documents. They are extracted by first saving the word document as a docx file (that’s Word XML) and then loading this into the XML database. Trip reports can then be extracted using XQuery by selecting paragraphs in the document.

Here is the trip report for ‘Dumbledore’s trip in April 2010 on the web and here as an XML document.

Note that the meta-data has been added into the document as attributes of an element which surrounds the relevant text.

This looks messy but we can search for Trips using XPath.

What do you think this query will return?

```xml
//Trip[.//skipper/@name='Jeff Birkin']
```

or

```xml
//skipper[@name='Jeff Birkin']/ancestor::Trip
```

How about finding ‘Dumbledore's trips? Answer

What about the names of the vessels that Gordon Ogden has skippered? Answer Better
The wonders of jQuery and GoogleMap API

Editing the annotations on the trip reports needs a different kind of editor. The user needs to select some text, select which annotation to apply, fill in any supporting attributes and submit the change. The page then needs to be updated instantly.

BSA site
Case study
Another project involves the crowd sourcing of historical data of occupancy of shops in Gloucester Road. The Gloucester Road Story.

It's based on a principle of inquiry stated in Rudyard Kipling's poem 'The Elephant child' but much older (14th Century)

Si sapiens fore vis sex servus qui tibi mando
Quid dicas et ubi, de quo, cur, quomodo, quando.
(If you wish to be wise I commend to you six servants,
Ask what, where, about what, why, how, when.)
I've interpreted 'how' as 'How do you know'. Here's one example of a failed business:

The Road Model
A location-aware webpage - client side

- Having this location-based data makes it obvious that we need a location aware website, so that as you walk down the street, you can view the history of each shop.

- This turns out to be not so difficult. If you look at the source of the page, there is only an empty div with id="info". JavaScript and a server XQuery script is going to populate the page dynamically.

  ```javascript
  $(document).ready(function() {
    if (navigator.geolocation) {
      navigator.geolocation.watchPosition(get_premise_latlong, errorFunction, {maximumAge:60000});
    } else {
      get_premise_number(200);
    }
  });

  function get_premise_latlong(position) {
    var lat = position.coords.latitude;
    var long = position.coords.longitude;
    var url = "xq/mobile.xq?lat="+lat+'&long='+long;
    // alert (url);
    $('#info').load(url);
  }

  - the lat and long are passed to the server script and the result replaces the div.
A location-aware webpage - server side

- On the server, an XQuery script accesses the lat and long URL parameters, finds the nearest premise and then generates an HTML div to return to the client.

- The function to find the premise nearest to the user from the set of all premises, $gl:premises:

```xml
declare variable $glm:range := xs:double(0.00001);
declare function glm:nearest-premise($lat as xs:double,$long as xs:double) as element(premise)? {
  (for $premise in $gl:premises[latitude][longitude] (:only geo-coded premises :) 
  let $dlat := xs:double($premise/latitude) - $lat 
  let $dlon := xs:double($premise/longitude) - $long 
  let $distance := $dlat * $dlat + $dlon * $dlon 
  where $distance < $glm:range (: only if within range of user :) 
  order by $distance 
  return 
  $premise 
 )[1] (:get the first i.e. nearest if any :) 
};
```

- Since I don't have an Android or iPhone, I haven't been able to test this on the ground but it has been tested in Opera. The utility of the application depends on the accuracy of the user's geo-location and the accuracy of the address geo-coding which used Google API.
SUMMARY

- XML databases are one of many NoSQL variants
- Any well-formed XML document can be stored and queried with XPath
- Complete applications can be developed in XQuery and an XML database
- XQuery is a functional language - no assignment- so recursion is used a lot
- XML databases are ideal for complex documents but can also do typical RDBMS data models and configuration files
- Development in XQuery and eXist is fast and fun
Workshop

- Developing a location-aware sculpture site.

- Research, Analysis and design for a location-aware Bristol Sculpture web application.