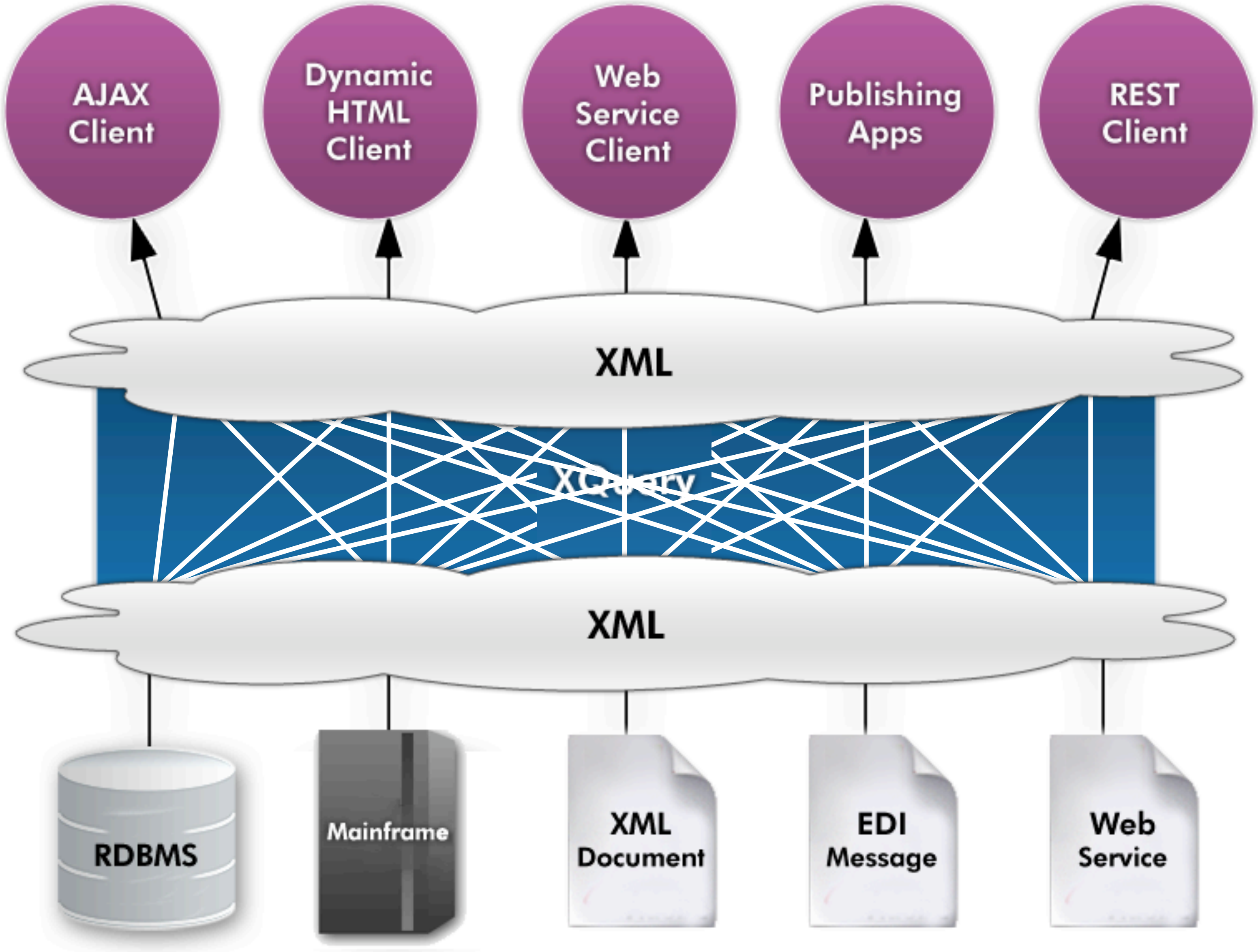
Data – XML and XQuery
A language that can combine and transform data

John de Longa
Solutions Architect
DataDirect technologies
john.de.longa@datadirect.com
Mobile +44 (0)7710 901501

Data integration through XML in the Enterprise



Why is DataDirect talking about XML and XQuery ?

DataDirect’s roots go back to the beginnings of Standards based connectivity.
Initially starting with ODBC then JDBC and more recently ADO.NET

So a byline for DataDirect is a Data Connectivity Standards Based Company

Over time XML has emerged as more that just a file format
XML is used in many integration roles for moving data from one application, computer or company to another.

Standards have evolved over time that have embraced XML

SQL/XML from the ISO/IEC standards committee
XPath from WC3 version 1 - 16 November 1999 version 2 - 23 January 2007
XSLT from W3C version 1 - 16 November 1999 version 2 - 23 January 2007
and more recently XQuery version 1 - become ratified - 23 January 2007

DataDirect have been active on the XQuery Working party

What is XQuery?

- W3C Query Language for XML
  - Native XML Programming Language
  - “The SQL for XML”
  - Designed to query, process, and create XML
- High level functionality
  - Find anything in an XML structure
  - Querying and combining data
  - Creating XML structures
  - Functions
  - User-defined function libraries

XQuery a Language and a Processor

- XQuery has two components of any implementation
  - The language syntax for a particular implementation
  - This is specified by the WC3
  - Certain aspect of the syntax is both optional and specific to the implementation.

  - The XQuery processor, processes the XQuery and communicates with the various data sources, these being XML files, Web Services, Relational data sources and non XML data sources via XML Converters.

  - Some implementations require application server to be running before the XQuery processor can consume XQuery queries.
  - Some implementation do not require an application server, just a Java container.
  -

DataDirect TECHNOLOGIES

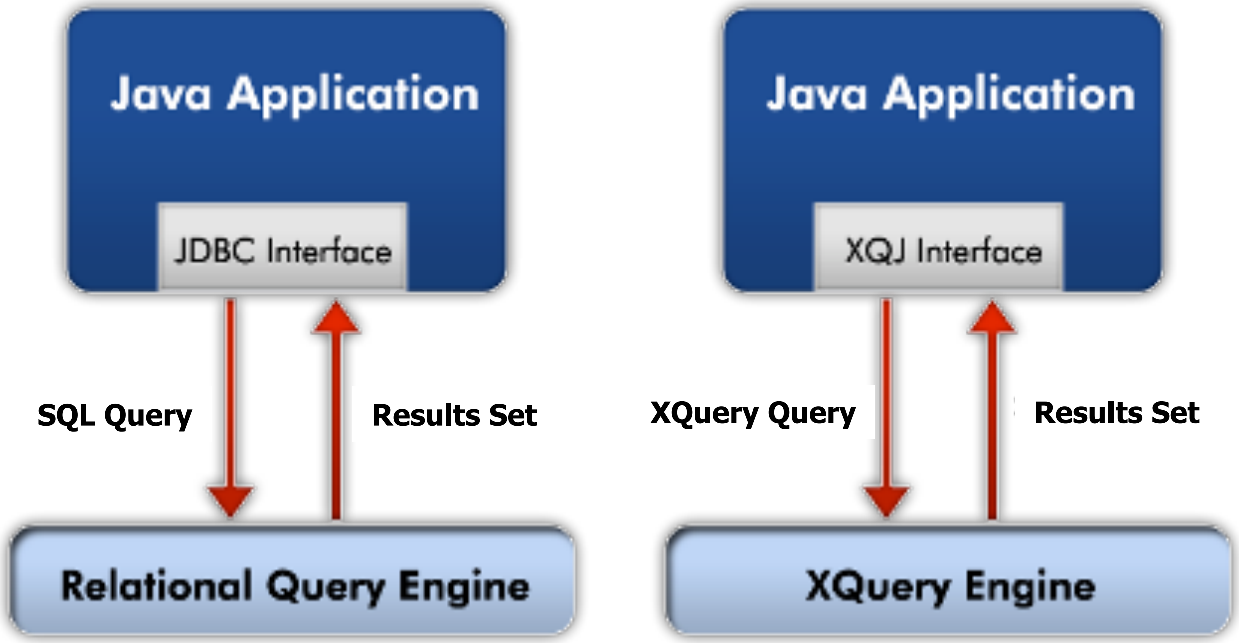
XQuery – DataDirect’s implementation

- XQuery is a language agnostic to platform
- DataDirect XQuery is a Java based implementation
- With DataDirect XQuery we ship an interface that allows Java applications to interact with our XQuery implimentation called
- XQJ XQuery API for Java JSR-000225
- DataDirect’s XQuery implementation supports querying relational databases and returning XML, accessing Web services and non XML data sources such as EDI, Flat files etc via XML Converters
- DataDirect’s XQuery does not require an specific application server stack.
- DataDirect’s XQuery is a pluggable component into a larger infrastructure

DataDirect TECHNOLOGIES

What is XQJ?

- XQJ is the API used for connecting a Java application to XQuery engine.
- Analogy to JDBC/SQL
  - JDBC is the API that passes SQL queries to the data sources.
  - XQJ is the API that passes XQuery queries to the data sources.
- Developed under Java community process (JSR 225).
- We are on the JSR 225 committee!



DataDirect TECHNOLOGIES

DataDirect Data Integration Suite

High performance

Scalable

RDBMS updates

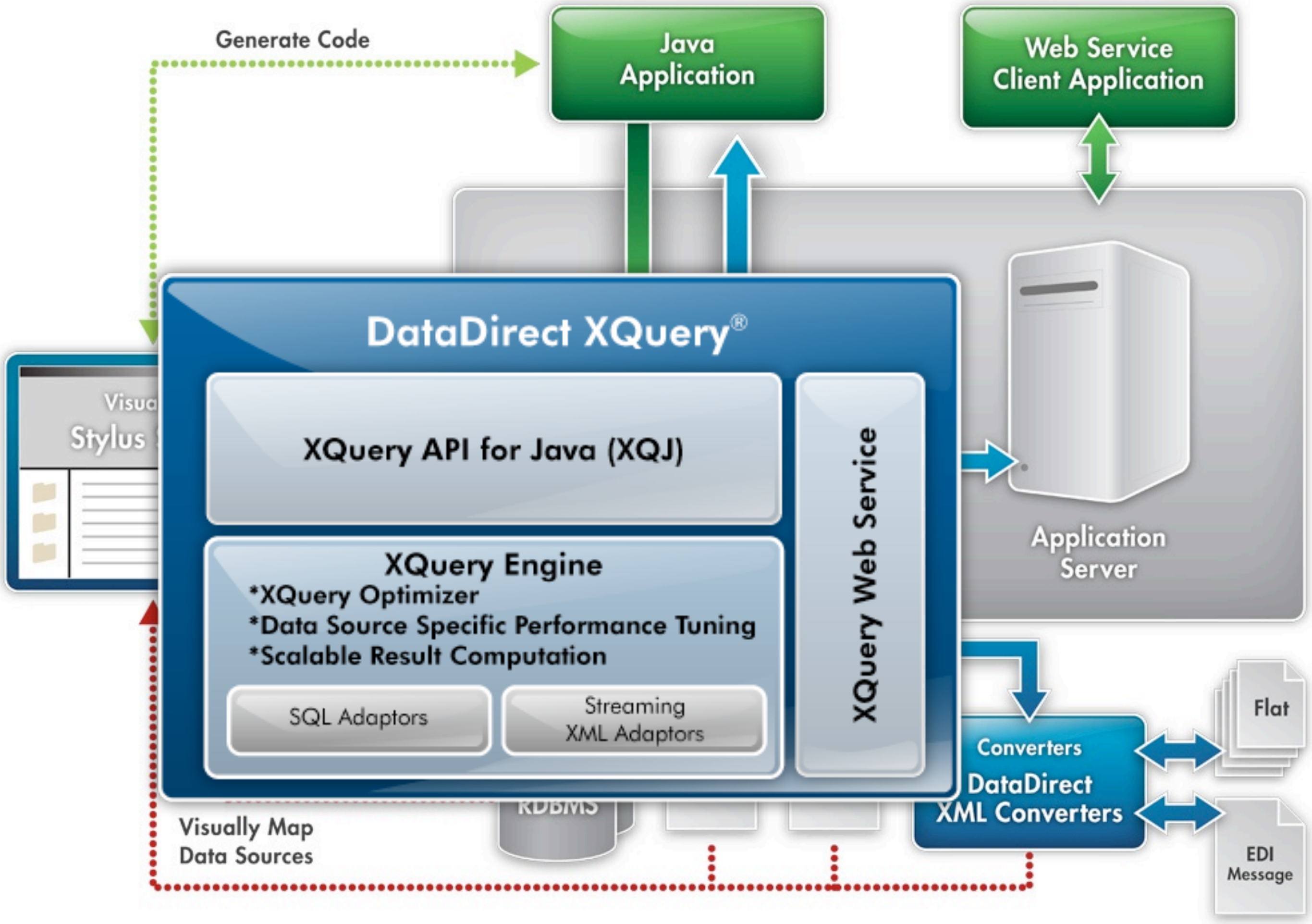
Embeddable

Plugs into any architect

Accesses
  almost
  any data
  source

No dependency
  on servers

Standards-based



Differences between XQuery and XSLT

XQuery has many SQL queries similarities, Querying a data source to return a subset of the data source being queries.

XQuery is designed to be scalable and to take advantage of Database functions such as indexes.

XSLT implementations are generally optimized when transforming a whole document and this is read into memory.

XQuery syntax is possibly easier to read than the equivalent XSLT code.

XQuery is generally more succinct than XSLT being 5 to 20 smaller. This makes the code required to achieve the same function is somewhat smaller that equivalent XSLT code, making it easier to embed in applications.

XQuery - Basics

As mentioned earlier XQuery has its roots in XPath

So simple XQuery can be

```
<root> Hello World </root>
<root> 5+8 </root>
<root> {5+8}</root>
```

A simple XQuery of an XML file can look very much like an XPath expression

```
doc("books.xml")/bookstore/book[price>30]/title
```

XQuery FLWOR Expression Syntax

XQuery’s main query language syntax rules are based around the FLWOR Expressions

FLWOR is an acronym for "For, Let, Where, Order by, Return".

In this example

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

The **for** clause selects all book elements under the bookstore element into a variable called $x.

The **where** clause selects only book elements with a price element with a value greater than 30.

The **order by** clause defines the sort-order. Will be sort by the title element.

The **return** clause specifies what should be returned. Here it returns the title elements.

XQuery – Basics a simple FLOWR statement

XQuery using a FLOWR statement accessing an XML doc

With Doc

and

Return

```
<order> {
    for $book in doc("file:///c:/xml2007/xmlfiles/books-
order1.xml")/order/book
    return
        <book>
            <title>{$book/title/text()}</title>
            <quantity>{data($book/@quantity)}</quantity>
            <ISBN>{$book/isbn/text()}</ISBN>
        </book>
} </order>
```

XQuery – Basics a simple FLOWR statement

XQuery using a FLOWR statement accessing an XML doc

Introducing a join of two files

With a let function,

Join in Let clause

And Return

```
<order> {
    for $book in doc("file:///c:/xml2007/xmlfiles/books-
order2.xml")/order/book
    return
        let $details := doc("file:///c:/xml2007/xmlfiles/books-
order2-1.xml")/details/book[@bookid=$book/@bookid]
        return
            <book>
                <title>{$details/title/text()}</title>
                <quantity>{data($book/@quantity)}</quantity>
                <ISBN>{$details/isbn/text()}</ISBN>
            </book>
} </order>
```

XQuery – A FLOWR statement accessing an EDI file

XQuery using a FLOWR statement accessing
an EDI File with DataDirect XML Converters

With a let function,

And Return

```
<order>
    {
    for $GROUP_28 in
doc('converter:EDI:long=yes?file:///c:/xml2007/order.edi')/EDIFACT/
ORDERS/GROUP_28
    return
    <book>
        <quantity>
            {$GROUP_28/QTY/QTY01-QuantityDetails/QTY0102-
Quantity/text()}
        </quantity>
        <ISBN>
            {$GROUP_28/LIN/LIN03-ItemNumberIdentification/LIN0301-
ItemIdentifier/text()}
        </ISBN>
    </book>
    }
</order>
```

XQuery – A FLOWR statement accessing a RDMS

XQuery using a FLOWR statement accessing a Database with DataDirect's implementation of a "Collection"

```
<order>
        {
        for $details in collection("Books.dbo.booksXML")/booksXML
        return
   <book>
        <title> {$details/title/text()} </title>
        <publisher> details/manufacturer/text()}</publisher>
        <publishing-date>{$details/releaseDate/text()}</publishing-date>
        </book> }
</order>
```

XQuery – A FLOWR statement updating a RDMS from an XML file

XQuery using a FLOWR statement accessing a Database with DataDirect's implementation of a "Collection"

With a let function,

And Return

```
for $book in
doc("file:///c:/xml2007/xmlfiles/fullOrder2.xml")/order/book
return
    ddtek:sql-insert("Books.dbo.orders", "isbn", $book/ISBN,
"quantity", $book/quantity)
```

DataDirect TECHNOLOGIES

XQuery – A FLOWR statement joining an EDI file and RDMS table

XQuery using a FLOWR statement joining an EDI file and Database table.

With Doc and Collection

With Join in Where clause

And Return

```
<order>
 {
for $book in doc("file:///c:/xml2007/xmlfiles/books-
order1.xml")/order/book,
        $details in collection("Books.dbo.booksXML")/booksXML
    where $book/isbn = $details/isbn
    return
    <book>
        <title>{$book/title/text()}</title>
        <quantity>{data($book/@quantity)}</quantity>
        <ISBN>{$book/isbn/text()}</ISBN>
        <publisher>{$details/manufacturer/text()}</publisher>
        <publishing-date>{$details/releaseDate/text()}</publishing-
date>
    </book>
    }
</order>
```

Scalability

- When processing large files there is only so much memory in the simple container like Tomcat or Application Servers like JBoss
- To process XML files and Database Queries that run into the large Megabyte or Gigabyte range the XQuery implementation has to have optimizing processes
-
- Document Projection
  - Discards unwanted data before loading in to the JVM
- Streaming
  - Processes and starts writing the results set as soon as possible.

How are XML documents ‘typically’ queried?

- XQuery processor invokes XML Parser
- XML Parser generates ‘events’
- Events are captured by processor
- In-memory model of XML document is created
- Processor will ‘query’ this in-memory model
- Transformation of XML results creates new in-memory model

How are XML documents ‘typically’ queried?

- What does an “in-memory model” cost?
- There are many factors
  - XML vocabulary
  - Usage of namespaces
  - Indentation
  - Depth of XM document
  - Length of text nodes
  - Etc
- Compared to serialized XML
  - DOM consumes typically 10 to 15 times memory of XML file
  - Good processors today consume 5 to 7 time memory of XML file

Querying large XML documents
Performance and Scalability

- DataDirect supports
- XML Document Projection
- XML Streaming
- In-memory Indexing
- Streaming result construction

XML Document Projection

- Optimize the in-memory representation of documents
- How does it work?
  - Prepare time
    - analyze the query, determine which <u>structural</u> fragments of document are needed
  - Run time
    - document is completely parsed
    - only required fragments of document are instantiated
- How much improvement?
  - depends on query and document structure

XML Document Projection

- for $s in doc("portfolio.xml")//stock[ticker eq "EBAY"]
  return $s/name

-
  ```
  <portfolio>
          <user>Jonathan</user>
          <period>
                  <start>2003-01-01</start>
                  <end>2004-01-01</end>
          </period>
          <stocks>
                  <stock>
                          <ticker>AMZN</ticker>
                          <name>Amazon.com, Inc.</name>
                          <shares>3000.00</shares>
                          <minprice>18.86</minprice>
                          <maxprice>59.69</maxprice>
                  </stock>
                  <stock>
                          <ticker>EBAY</ticker>
                          <name>eBay Inc.</name>
                          <shares>4000.00</shares>
                          <minprice>33.51</minprice>
                          <maxprice>60.46</maxprice>
                  </stock>
  ...
  ```

XML Streaming

- The idea…
  - Processes document and query simultaneous
  - Discarding portions that are no longer needed
  - Consumer (your application) is in charge
    - Execute doesn’t do much
    - Consuming results triggers a ‘window’ of query execution
- Streaming Doesn’t always kick in!
  - Document can be queried only once
  - No reverse axis
  - Etc.
- XML Streaming and document projection are complementary

XML Streaming

-
  ```
  for $s in doc("portfolio.xml")//stock[ticker eq "EBAY"]
  return $s/name
  ```

-
  ```
  <portfolio>
          <user>Jonathan</user>
          <period>
                  <start>2003-01-01</start>
                  <end>2004-01-01</end>
          </period>
          <stocks>
                  <stock>
                          <ticker>AMZN</ticker>
                          <name>Amazon.com, Inc.</name>
                          <shares>3000.00</shares>
                          <minprice>18.86</minprice>
                          <maxprice>59.69</maxprice>
                  </stock>
                  <stock>
                          <ticker>EBAY</ticker>
                          <name>eBay Inc.</name>
                          <shares>4000.00</shares>
                          <minprice>33.51</minprice>
                          <maxprice>60.46</maxprice>
                  </stock>
  …
  ```

In-memory Indexing

- Joins are used frequently
  - Joins within single XML document
  - Join of multiple XML documents
  - XQuery grouping is done through joins!
- Typically, joins are performed through nested loops
  - Slow with large document sets
- Build in-memory index
  - Time required to build indexes is irrelevant compared to document parsing
  - Runtime improvements are huge for large data sets

DataDirect TECHNOLOGIES

Streaming result construction

- Large documents result in large results
  - Not always
  - Likely for transformations
  - Less likely for queries
- Compute results when needed
  - Compute results when requested by application
  - So called “pull based”
  - Results are really fine grained, up to the “XML tag level”
  - Query results are computed as needed

Supported input formats

- All discussed optimizations are supported with
  - fn:doc
  - fn:collection
  - fn:doc/collection with custom URI resolver
  - XQuery external variables
  - XQuery initial context item
  - Java External Functions

XMark

- Independent XQuery benchmark
- What do we measure?
  - Performance
    # execute/fetch cycles using null SAX handler
  - Memory consumption
- We’ll show results for
  - DataDirect XQuery 3.0 (DDXQ)
  - Popular open source XQuery implementation (OS)
- Default Java VM (64MB)
- XML Document from 25K up to 500 MB

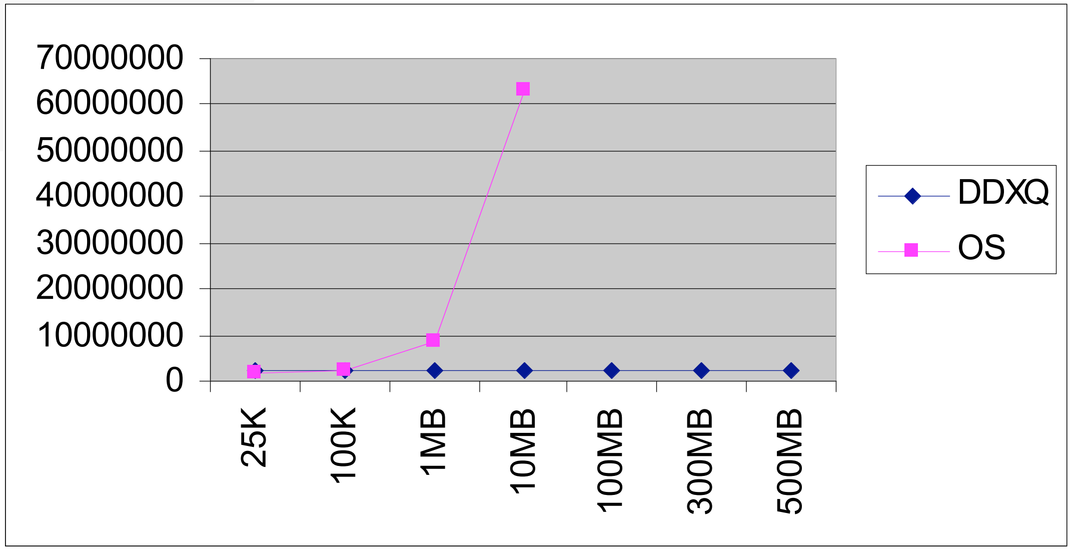
XMark - 3 queries…

```
(:doc – not standard XMark:)
doc('xmark.xml')

(:Q1:)
for $b in
  doc('xmark.xml')/site/people/person[@id='person0']
return $b/name/text()
(:Q8:)
for $p in doc('xmark.xml')/site/people/person
let $a := for $t in doc('xmark.xml')
                             /site/closed_auctions/closed_auction
            where  $t/buyer/@person = $p/@id
           return $t
return
  <item person='{$p/name/text()}'>{count($a)}</item>
```

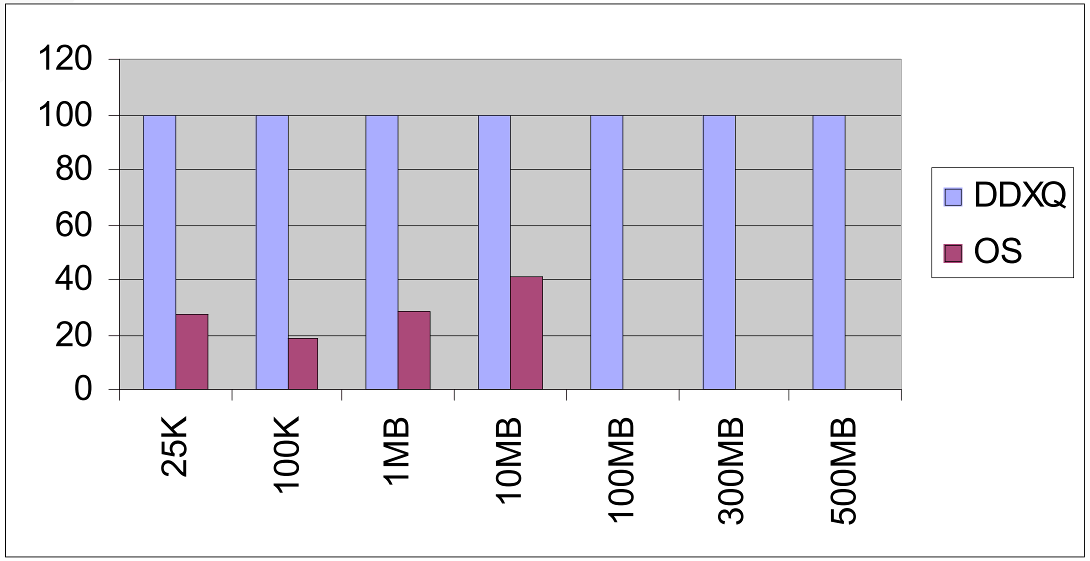
XMark - doc

```
(:doc – not standard XMark:)
doc('xmark.xml')
```



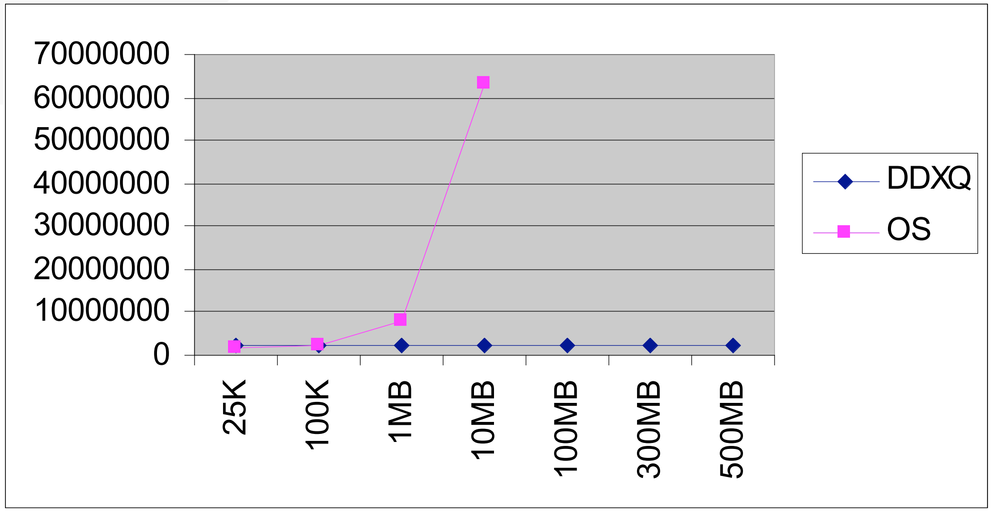
XMark - doc

```
(:doc – not standard XMark:)
doc('xmark.xml')
```



XMark – Q1

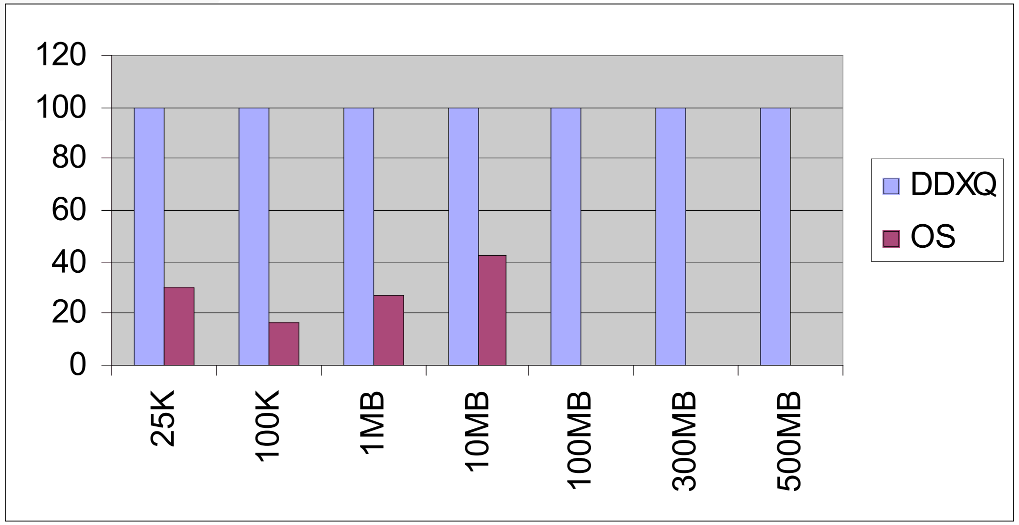
```
for $b in doc('xmark.xml')/site/people/person[@id='person0']
return $b/name/text()
```



DataDirect TECHNOLOGIES

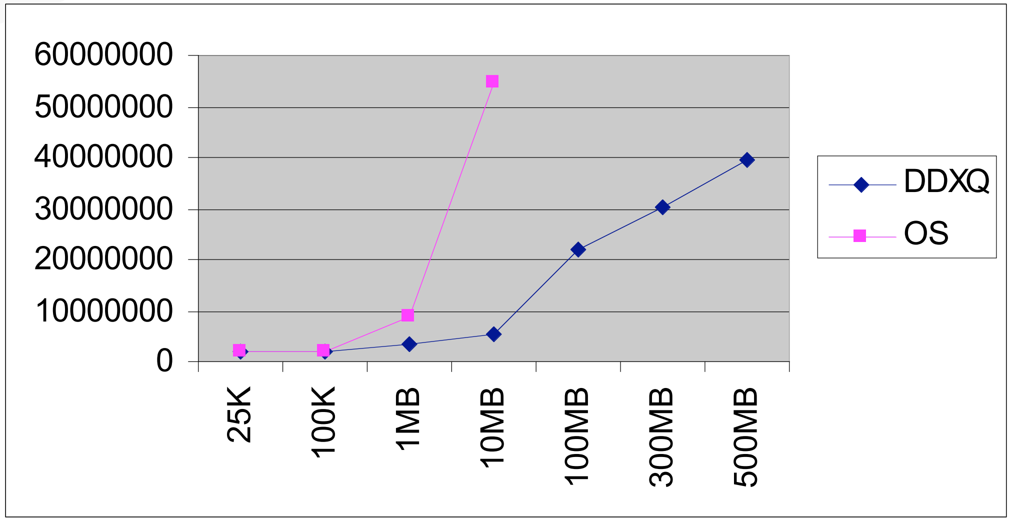
XMark – Q1

```
for $b in doc('xmark.xml')/site/people/person[@id='person0']
return $b/name/text()
```



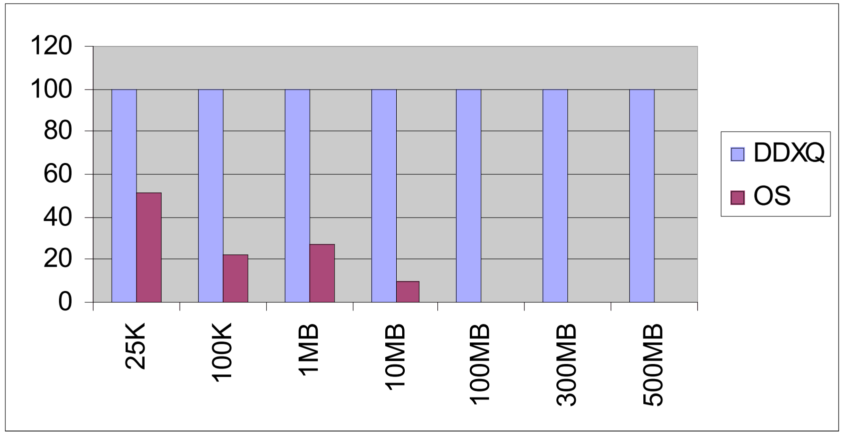
XMark – Q8

```
for $p in doc('xmark.xml')/site/people/person
let $a := for $t in doc('xmark.xml')/site/closed_auctions/closed_auction
          where  $t/buyer/@person = $p/@id
          return $t
return <item person='{$p/name/text()}'>{count($a)}</item>
```



XMark – Q8

```
for $p in doc('xmark.xml')/site/people/person
let $a := for $t in doc('xmark.xml')/site/closed_auctions/closed_auction
          where  $t/buyer/@person = $p/@id
          return $t
return <item person='{$p/name/text()}'>{count($a)}</item>
```



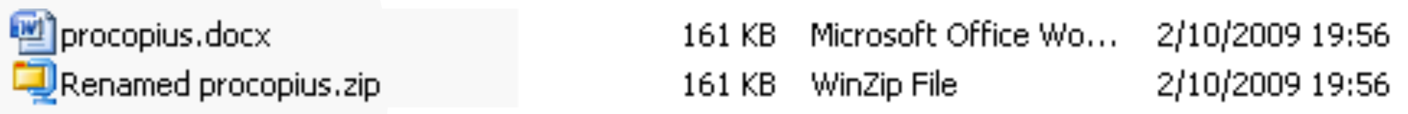
Office Documents are based on XML

- Microsoft Office 7.0 supports OpenXMLFormat
- XQuery queries can be pointed at a Document
- Large documents can be queried because of Document projection and Streaming.

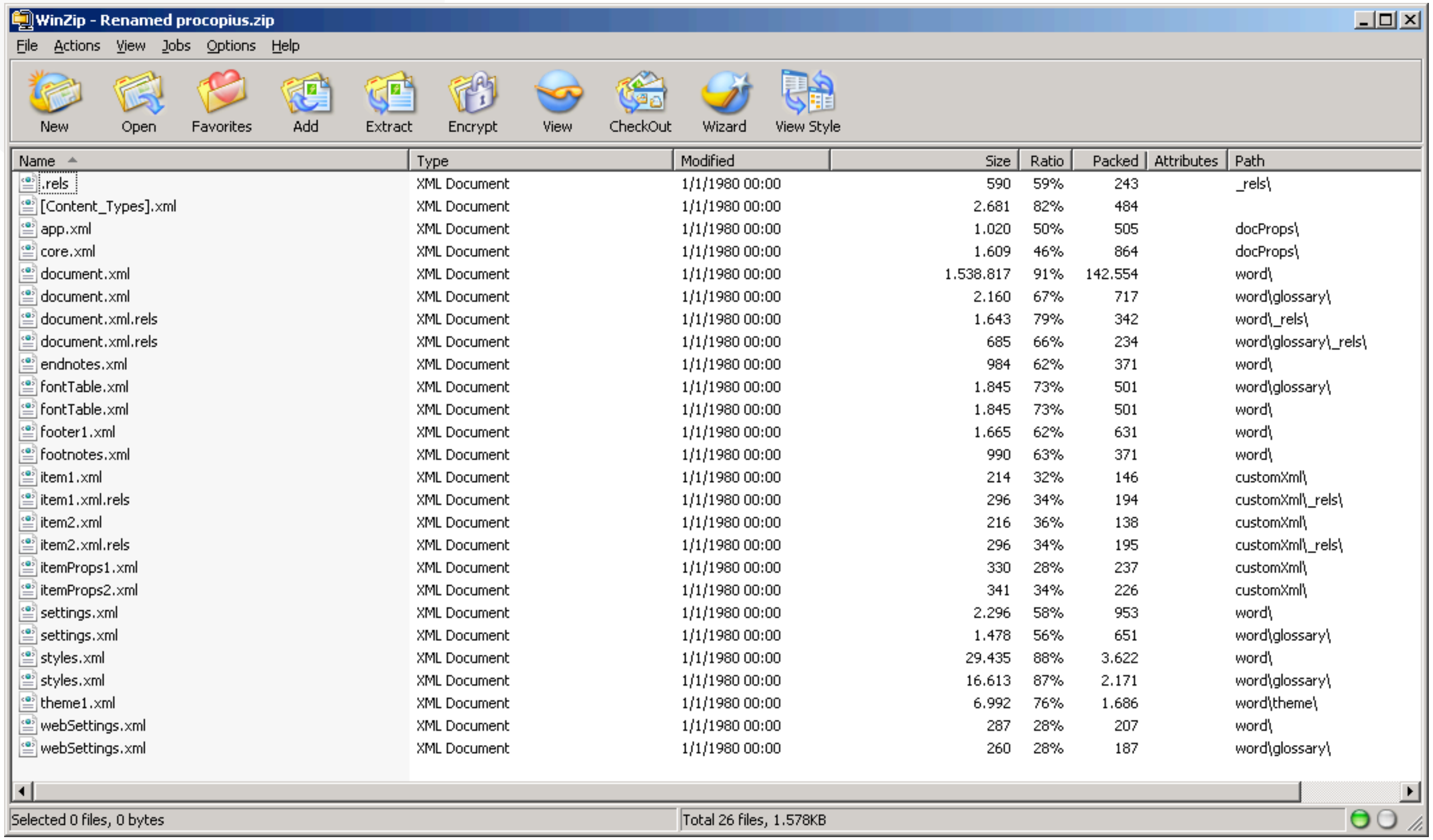
DataDirect TECHNOLOGIES

Microsoft Office 7

To see visually an XML structure of a Word Doc – Rename it!



procopius.docx	161 KB	Microsoft Office Wo...	2/10/2009 19:56
Renamed procopius.zip	161 KB	WinZip File	2/10/2009 19:56


WinZip - Renamed procopius.zip
File Actions View Jobs Options Help
New Open Favorites Add Extract Encrypt View CheckOut Wizard View Style

Name	Type	Modified	Size	Ratio	Packed	Attributes	Path
.rels	XML Document	1/1/1980 00:00	590	59%	243		_rels\
[Content_Types].xml	XML Document	1/1/1980 00:00	2.681	82%	484		
app.xml	XML Document	1/1/1980 00:00	1.020	50%	505		docProps\
core.xml	XML Document	1/1/1980 00:00	1.609	46%	864		docProps\
document.xml	XML Document	1/1/1980 00:00	1.538.817	91%	142.554		word\
document.xml	XML Document	1/1/1980 00:00	2.160	67%	717		word\glossary\
document.xml.rels	XML Document	1/1/1980 00:00	1.643	79%	342		word_rels\
document.xml.rels	XML Document	1/1/1980 00:00	685	66%	234		word\glossary_rels\
endnotes.xml	XML Document	1/1/1980 00:00	984	62%	371		word\
fontTable.xml	XML Document	1/1/1980 00:00	1.845	73%	501		word\glossary\
fontTable.xml	XML Document	1/1/1980 00:00	1.845	73%	501		word\
footer1.xml	XML Document	1/1/1980 00:00	1.665	62%	631		word\
footnotes.xml	XML Document	1/1/1980 00:00	990	63%	371		word\
item1.xml	XML Document	1/1/1980 00:00	214	32%	146		customXml\
item1.xml.rels	XML Document	1/1/1980 00:00	296	34%	194		customXml_rels\
item2.xml	XML Document	1/1/1980 00:00	216	36%	138		customXml\
item2.xml.rels	XML Document	1/1/1980 00:00	296	34%	195		customXml_rels\
itemProps1.xml	XML Document	1/1/1980 00:00	330	28%	237		customXml\
itemProps2.xml	XML Document	1/1/1980 00:00	341	34%	226		customXml\
settings.xml	XML Document	1/1/1980 00:00	2.296	58%	953		word\
settings.xml	XML Document	1/1/1980 00:00	1.478	56%	651		word\glossary\
styles.xml	XML Document	1/1/1980 00:00	29.435	88%	3.622		word\
styles.xml	XML Document	1/1/1980 00:00	16.613	87%	2.171		word\glossary\
theme1.xml	XML Document	1/1/1980 00:00	6.992	76%	1.686		word\theme\
webSettings.xml	XML Document	1/1/1980 00:00	287	28%	207		word\
webSettings.xml	XML Document	1/1/1980 00:00	260	28%	187		word\glossary\

Selected 0 files, 0 bytes

Total 26 files, 1.578KB

Office 7 Documents are XML based

Microsoft supports the OpenXMLFormat

```
declare namespace w =
  "http://schemas.openxmlformats.org/wordprocessingml/2006/main";
declare namespace cp =
"http://schemas.openxmlformats.org/package/2006/metadata/core-properties";
declare namespace dc = "http://purl.org/dc/elements/1.1/";
declare variable $doc_props :=
  doc('jar:file:///c:/xml2007/xmlfiles/procopius.docx!/docProps/core.xml');

for $book in doc("file:///c:/xml2007/xmlfiles/books-order6.xml")/order/book
    where $book/isbn = $doc_props/cp:coreProperties/cp:keywords/text()
    return
    <book>
        <title>{$book/title/text()}</title>
        <quantity>{data($book/@quantity)}</quantity>
        <ISBN>{$book/isbn/text()}</ISBN>
        <Abstract>{$doc_props/cp:coreProperties/dc:description/
```

Office 7 Doc are XML based - XQuery and Streaming

- Streaming example using a Word Doc of all Shakespeare that is actually a 16 Mb file - an Open XML document

```
declare variable $TITLE := 'Much Ado about Nothing';
declare variable $ACT := 'ACT IV';

<html>
  <body>{
    for $SPEECH in doc("file:///c:/xml2007/xmlfiles/shakespeare.xml")-
/SHAKESPEARE/PLAY[TITLE eq $TITLE]/ACT[TITLE eq $ACT]/SCENE/SPEECH
    return (
      <h3>{$SPEECH/SPEAKER}</h3>,
      for $line in $SPEECH/LINE
      return
        ( <i>{$line}</i>, <br/> )
    )
  }</body>
</html>
```

DataDirect TECHNOLOGIES

Useful Links

- XQuery and DataDirect Data Integration Suite links
- XQuery information www.XQuery.com
- Examples & Tutorials, XQuery Tutorial, tips & tricks, XQJ Tutorial
- XML Converters www.xmlconverters.com
- EDI conversions, Custom conversions
- DataDirect Data Integration suite
- http://www.datadirect.com/products/data-integration/ddis/index.ssp
- A highly technical blog
- http://www.xml-connection.com
- Introduction to XQuery for SQL Developers
- http://www.xml-connection.com/2008/06/xquery-for-sql-programmer-introduction.html
- XQuery your office documents
- http://www.xml-connection.com/2007/09/xquery-your-office-documents.html
- Integrating non-SQL Data, for Example LDAP
- http://www.xml-connection.com/2008/08/accessing-ldap-directory-services.html
- Plugin for Eclipse http://www.xquery.com/xml_tools/
- A Good Book on XQuery
- http://www.amazon.com/XQuery-Priscilla-Walmsley/dp/0596006349

DataDirect TECHNOLOGIES

Questions on XQuery ?