Can I Make XML Go Faster?

Presented to:

RMOUG
ROCKY MOUNTAIN ORACLE USERS GROUP

Presented by: John Jay King
King Training Resources
john@kingtraining.com

Download this paper from:
http://www.kingtraining.com

Copyright @ 2009, John Jay King

Objectives

- Understand XML document processing
- Understand how Oracle stores XML data
- Learn different factors impacting XML performance
- Be able to improve the performance of XML documents
- Choose the best Oracle XMLType for the performance of your application
- Be familiar with Oracle XMLType indexing options

Who Am I?

- John King – Partner, King Training Resources
- Providing training to Oracle and IT community for over 20 years
  - Databases: Oracle, DB2, SQL Server, more…
  - Languages: PL/SQL, Java, C#, COBOL, PL/I, more…
  - Operating Systems: Linux, Unix, Windows, z/OS
  - Tools: XML, HTML, JavaScript, more…
- Leader in Service Oriented Architecture (SOA) design and implementation
- Home is Centennial, Colorado – I like to hike and drive in the mountains

eXtensible Markup Language (XML)

- XML is a set of rules for defining tags to describe a document’s structure and parts
- XML is a "meta-markup" language, providing the syntax used to define the syntax and structure of a document, not the presentation or the format
- The XML specification is authored by the W3C (World Wide Web Consortium www.w3.org)
- “Markup" is from typesetting: publishers "markup" a document telling the typesetter how to format the page
- XML is "extensible" because no tags are predefined; organizations/industries define tags for “XML Language” to support business needs

Copyright @ 2009, John Jay King

Why XML?

- Is XML Cheaper? Smaller? Faster?
- Is XML more-capable?
- XML is STANDARD
  - Character strings are common to all operating systems and computer languages
  - XML standardizes the character string formats exchanged between programs and systems
- Savings attributed to XML are usually due to reduced complexity (Standards for Technology in Automotive Retail (STAR))

XML and Savings

- Savings attributed to XML are usually due to reduced complexity (Standards for Technology in Automotive Retail (STAR))
  - STAR was used to standardize interchange of parts and automobile information between automobile manufacturers, suppliers, dealers, and credit organizations
  - Millions in estimated savings due to standardization

XML Advantages

- XML is non-proprietary
- Elements and tags may be defined as needed allowing specialized languages
- Document templates, files, and database data can all be stored using an XML-described format
- Standardized formats make data easier to share
- Industry groups and companies use XML to build common tag sets and common data structures
- XML is frequently used by software vendors to specify configuration
- XML is used to describe data files used for: Electronic Data Interchange (EDI), word processing, and more

Copyright @ 2009, John Jay King

XML Uses

- Industry groups and companies use XML to build common tag sets and common data structures
- XML is frequently used by software vendors to specify configuration
- XML is used to describe data files used for: Electronic Data Interchange (EDI), word processing, and more

XML "Traits"

- XML is designed to be "human readable"
  - Verbose tag names are used to aid readability
  - Design "best practices" lead to use of sub-elements to maximize flexibility
  - Hierarchical nature of XML allows deep structures of elements, sub-elements, sub-sub-elements and so on
  - Tab characters and carriage-returns (end-of-line markers) are added to support readability
- Result: XML documents take up too much space!
  XML documents take too long to parse!

Nobody should need to read XML documents manually
(except when problem-solving)

Tag/Element Naming

- XML has specific rules for naming of Tags/Elements
- Element names begin with a letter or an underscore
- Element names may contain letters, underscores (_), numbers, hyphens (-), and colons (:)
- Start tags must match end tags exactly
- Names in XML are case-sensitive and may not contain blanks (officially there is no length limit)
- Names should not begin with "xml"

```
<name>Jones</lastname>            incorrect
<lastname>Jones</lastname>  correct

<last name>Jones</last name>      incorrect
<lastname>Jones</lastname>  correct

<lastname>Jones</lastName>  incorrect
<lastName>Jones</lastName>  correct
```

Attributes

- Descriptive Attributes are added to an element’s start tag using the name of the attribute followed by an equal sign, followed by the value of the attribute (surrounded by quotes or apostrophes)

```
<book isbn="0-13-960162-7" binding="perfect">
    <name>Learning XML</name>
    <author>Eric T Ray</author>
    <publisher>O&apos;Reilly</publisher>
</book>
```

- Attribute naming rules are the same as for element naming, attribute names must be unique within an element. Usually, attributes are used to provide information about the data in an element
- Attribute values must be enclosed by quotation marks (") or apostrophes (')

Elements and Attributes

```
<book isbn="0-13-960162-7" binding="perfect"
          topic="IT XML" name="Learning XML"
          author="Eric T Ray" publisher="O’Reilly" />
```

143 bytes

```
<book>
  <isbn>0-13-960162-7</isbn>
  <binding>perfect</binding>
  <topic>IT XML</topic> >
  <name>Learning XML</name>
  <author>Eric T Ray</author>
  <publisher>O’Reilly</publisher>
</book>
```

179 bytes

```
<book isbn="0-13-960162-7" binding="perfect“ topic="IT XML" >
  <name>Learning XML</name>
  <author>Eric T Ray</author>
  <publisher>O’Reilly</publisher>
</book>
```

154 bytes

Which is "Better"

- Are Elements better than Attributes or vice-versa?
  - Some feel attributes should be used for metadata or for unchangeable values (primary keys)
  - Some feel that elements are more flexible long-term
  - Some feel that attributes without too many elements reduces file sizes
  - Attributes may not be divided in any way
(Elements may have sub-Elements and Attributes)
  - Attributes may not be repeated within an element
(Elements may be repeated as specified by Schema)
- Choose the mechanism that best seems to fit the business rules of your system

"Well-Formed" XML

- Strict rules determine that XML is "well-formed":
  - Document should declare itself using an XML declaration **`<?xml version="1.0" encoding="UTF-8"?>`**
  - A single "root" element must completely contain all other elements in the document (one set of outer tags)
  - All elements that include data must have both start <name> and end </name> tags
  - Empty tags are marked using a slash before the close of the start tag and omitting the end tag <name/> (usually include attributes <name first=“Al” last=“Ono” />)
  - Tags may not overlap, but, may be nested
  - Attribute values enclosed in quotes (") or apostrophes (')
  - XML tools refuse to process non-”well-formed” documents

Copyright @ 2009, John Jay King

Being “Well-Formed” Costs!

- Software processing XML must make sure the document is “well-formed” before processing (a requirement of the XML standard)
  - XML document must be stored to make sure it is “well-formed” (large documents take up space)
  - Computer resources must be used to ensure that rules for “well-formed”-ness are enforced

Typical SOAP Invocation

- SOAP message received
  - – SOAP processor checks if message **well-formed**
  - – Process SOAP message and retrieve payload
  - – If payload is XML it is checked to be sure it is **well-formed** before being passed to program
  - – Pass payload to program
  - – Program's parser checks if document **well-formed**
  - – Program processes file
  - – Program generates XML result
  - – Parser checks to see if result is **well-formed** before passing to SOAP
  - – SOAP parser checks XML to make sure it is **well-formed**
  - – Process repeated on other end…

Copyright @ 2009, John Jay King

Schemas

- Being “well-formed” does not mean that the XML document being transmitted/received is correct!
- W3C has a method for validating XML documents called XML Schema (previously used “DTDs”)
- Schemas are well-formed XML documents themselves describing an XML document's format
- With Schemas, XML documents and their format descriptions use the same basic formatting rules (XML) perhaps making it easier to work with both
- Schemas are also useful as documentation tools, since they follow the rigid XML standard they are machine-readable!

Copyright @ 2009, John Jay King

XML Schema Syntax, page 1

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="myStudents">
      <xs:complexType>
             <xs:sequence>
                <xs:element ref="class"
                          maxOccurs="unbounded"/>
             </xs:sequence>
      </xs:complexType>
  </xs:element>
  <xs:element name="class">
      <xs:complexType>
             <xs:sequence>
                   <xs:element ref="title"/>
                   <xs:element ref="numberdays"/>
                   <xs:element ref="scheduledClass"
                           maxOccurs="unbounded"/>
             </xs:sequence>
      </xs:complexType>
  </xs:element>
```

XML Schema Syntax, page 2

```
<xs:element name="title" type="xs:string"/>
   <xs:element name="numberdays" type="xs:byte"/>
       ...
   <xs:element name="classcode">
       <xs:simpleType>
             <xs:restriction base="xs:short">
                    <xs:enumeration value="1504"/>
                    <xs:enumeration value="1508"/>
                    <xs:enumeration value="1511"/>
             </xs:restriction>
       </xs:simpleType>
   ...
</xs:schema>
```

Using XML Schemas

- XML Schemas are sometimes referenced from an XML document’s root element:

```
<?xml version="1.0" standalone="no" ?>
<?xml-stylesheet href="myStudents.css" type="text/css" ?>
<myStudents
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance“
    xsi:noNamespaceSchemaLocation="myStudents.xsd">
    <class>
           <title>Introduction to XML</title>
    ...
```

XPath

- XML Path Language (XPath) is designed to provide quick and easy access to any node in our document's hierarchy
- Having lots of information in XML is of little use if we cannot get to data when it is needed, XPath to the rescue!
- XPath provides a mechanism to address any element or attribute
- XPath is a World Wide Web Consortium (W3C) standard: **`http://www.w3.org/Style/XSL`**

Copyright @ 2009, John Jay King

XPATH Locations

- XPath nodes are very similar to DOM (Document Object Model) nodes; being sometimes-complex webs of nested elements and attributes
- An XPath address points to a specific end-point (position) in the hierarchy, or, to a specific node
  - Location paths may be absolute or relative:
    - Absolute paths start at the root node and move out from there
    - Relative paths start at a preselected spot called the context node (usually both easier and faster); context node is the current location
  - Each step in a location path is divided into three parts:
    - Axis, describes the direction of travel through the nodes
    - Node Test, to select the desired nodes
    - Predicates, optional tests to eliminate some XML from output

Copyright @ 2009, John Jay King

XML Performance

- The size of XML documents impacts performance if many documents are being transmitted/stored
- XML Processing for “well-formed” testing cannot be avoided
- XML Processing for schema validation is optional in most XML environments and may be avoided if unnecessary (careful!)

Copyright @ 2009, John Jay King

XML optimization

- Techniques exist to minimize the size of XML documents
- Smaller XML documents reduce:
  - Bandwidth requirements
  - Memory space requirements
  - XML Parsing time
  - Storage requirements
- XML optimization techniques and savings apply to just about any XML document

Why Optimize XML?

- XML has become the "lingua franca" of Information Technology data transfer used as:
  - EDI (Electronic Data Interchange) messages
  - Web Service messages
  - Transaction messages
  - Configuration data
- XML processing causes performance issues in systems passing many XML documents or very large XML documents

Keep “True” to XML Standard

- XML optimization should not corrupt the integrity of the XML document
- XML optimization techniques reduce the size of XML documents while keeping the “human readable” goal of XML
(though admittedly less-readable)
- Software and hardware products are available that will convert XML documents to proprietary binary forms to speed processing; eliminating the advantages of standardized XML
(not discussed further here…)

Copyright @ 2009, John Jay King

XML Optimization Techniques

- XML Optimization Techniques include:
  - Elimination of whitespace
  - Elimination of comments
  - Careful use of CDATA sections
  - Reducing name lengths for: Element, Attribute, and Namespace
  - “Flattening” Structure
  - Careful use of Schema validation

Eliminating Whitespace

- While human readability is a stated goal of XML
- In most cases XML is transmitted between computer programs and humans need to read it only when something goes wrong
- Eliminate extra blank spaces
- Eliminate indentation
- Eliminate tab characters
- Eliminate carriage-return line-feed characters

Copyright @ 2009, John Jay King

Whitespace Reduction

- Document with carriage returns and tabs

```
<book>
   <isbn>0-13-960162-7</isbn>
   <binding>perfect</binding>
   <topic>IT XML</topic>
   <name>Learning XML</name>
   <author>Eric T Ray</author>
   <publisher>O’Reilly</publisher>
</book>
```

- Document with whitespace reduced (saves 8 chars at least)

```
<book><isbn>0-13-960162-7</isbn><binding>perfect</binding>
<topic>IT XML</topic><name>Learning XML</name><author>Eric T
   Ray</author><publisher>O’Reilly</publisher></book>
```

Comments

- XML comments are useful when describing an XML document to a human
- Many good XML document designers include comments explaining the roles of Elements and Attributes
- Most comments are “whitespace” in production use and may be eliminated

CDATA Sections

- XML allows documents to include text that might contain XML formatting characters
- CDATA sections are simply passed by XML processing without any attempt to check them for “well-formed”-ness
- CDATA sections should be as short as possible (may be tricky)

Reduce XML Content

- XML documents being used as Web Service messages or transactions sometimes contain information that is redundant at best and potentially incorrect
- For example if a purchase document includes:
  - Item ID, Item Description, Item Size, and Item weight in addition to quantity purchased, price, and date
  - Chances are Item ID may be used by the programs on either end of the transaction to look up Item specifics
- Consider eliminating redundant Elements and Attributes

Copyright @ 2009, John Jay King

Reducing Name Sizes

- Since human readability is a stated object of XML; names tend to be lengthy
- Element names are repeated in Start <name> and End <//name> tags
- Namespace prefixes are repeated each time they are referenced <myNamespace:myElement>
- In most systems XML documents are “seen” only be computer programs; why not make names as short as possible?
- Consider reducing names to 1-3 character meaningful mnemonics so that the document may still be read in the event of problems

Name Shrinking

- Names at original length

```
<book>
  <isbn>0-13-960162-7</isbn>
  <binding>perfect</binding>
  <topic>IT XML</topic>
  <name>Learning XML</name>
  <author>Eric T Ray</author>
  <publisher>O’Reilly</publisher>
</book>
```

- Abbreviated names (reduced 34 characters)

```
<bk>
  <isbn>0-13-960162-7</isbn>
  <bdg>perfect</bdg>
  <top>IT XML</top>
  <nam>Learning XML</nam>
  <auth>Eric T Ray</auth>
  <pub>O’Reilly</pub>
</bk>
```

XML Structures

- XML Documents with a structure of Elements, Sub-Elements, and Sub-Sub-Elements are said to be “vertical”

```
<book>
  <isbn>0-13-960162-7</isbn>
  <binding>perfect</binding>
  <topic>IT XML</topic>
  <name>Learning XML</name>
  <author>Eric T Ray</author>
  <publisher>O’Reilly</publisher>
</book>
```

- XML Document structures using Attributes rather than Sub-Elements are said to be “horizontal”

```
<book isbn="0-13-960162-7" binding="perfect"
      topic="IT XML" name="Learning XML"
      author="Eric T Ray" publisher="O’Reilly" />
```

“Flattening” Structure

- XML Documents with a structure of Elements, Sub-Elements, and Sub-Sub-Elements are said to be “vertical”
- XML Documents with a structure using Attributes rather than Sub-Elements are said to be “horizontal”
- If Elements are optional or limited to one occurrence; Attributes may be used rather than Sub-Elements without changing the intent of the XML designer (“flattening” the vertical document)
- Using XML “Empty Elements” the ending Element tag is eliminated:

```
<name first=”Al” last=”Orr” />
```

- Eliminating unnecessary Sub-Elements (vertical depth) reduces the number of tags required

Schema Validation

- Careful use of Schema validation can reduce processing time
  - Scheme validation is optional
  - If an XML document is being used in a closed-loop system; validation may be redundant and wasteful
  - Complex Schema validations take longer than simple Schema validations
- Make sure that the complex Schema testing is really necessary

Optimize XML Processing

- The Parser used to process XML can have a dramatic impact on performance
- As usual in IT there is no “one-size-fits-all” solution
- Many types of Parsers are available, the most common are:
  - DOM
  - SAX
  - JDOM
  - StAX

XML Processors

- Software that reads/uses XML is called an "XML Processor"
- Many web browsers, XML editors, and software products are XML processors
- Some features often provided in XML Processors:
  - Parser Translate XML markup & data into tokens
  - Event Switcher Sorts/routes tokens to event handler or Call-back procedure
  - Call-back procedures Responds to events and adds nodes to "tree"
  - Tree representation Persistent hierarchy of XML document, may allow manipulation
  - Tree processor Code that processes the XML tree

XML Parsing

- Parsers are the fundamental part of any XML processor
- Parsers provide several useful purposes, they are used to:
  - Read XML data
  - Translate the data into recognizable tokens (the stream of characters is separated into instructions or hierarchical information)
  - Assemble data into a hierarchy

XML Parsers are Strict!

- All documents must be "well-formed"
  - Start tags must have End tags (or be empty)
  - Start and End tag names must match exactly (case too!)
  - Ambiguous names are not allowed
- By standard, XML parsers are not allowed to "fix" things
- Any error aborts the parsing operation
- Strict parsing means that a successfully parsed document is predictable and reliable

Parser Software

- Many parsers are available, including:
  - Xerces (Java, C++, Perl)
  - JAXP (Java)
  - IBM XML Parser for Java
  - Oracle XML Parser (Java)
  - Microsoft MSXML (C++, C#, JavaScript, VB, .NET, Java, Perl, Python)
  - IBM Enterprise COBOL
  - XML::Parser (Perl)
  - IBM Alphaworks XML for C++
  - Xparse (Python)
- Get a more-complete list of available parsers at: http://wdvl.com/Software/XML/parsers.html

Parser APIs

- Programs use XML via Application Programming Interfaces (APIs)
  - Low-level APIs Allow the programmer to deal directly with the XML document and its data
    - DOM, SAX, and JDOM are the most commonly-used low-level APIs today
    - JAXP (Java API for XML Programming) is also popular
  - High-level APIs Provide a simpler interface that calls one of the lower-level APIs “under-the-covers”
  - High-level APIs tend to be easier to develop with but usually add processing costs (no free-lunch!)
    - XML data binding (JAXB) is an example of a high-level interface

Low-Level XML APIs

- DOM, SAX, JDOM, and JAXP all offer low-level Application Programming interfaces:
  - DOM (Document Object Model) has been around for many years and is frequently used
  - SAX (Simple API for XML) offers the most basic Java-specific features
  - JDOM (Java Document Object Model) is a Java-specific API tailored specifically to the needs of Java programmers
  - JAXP (Java API for XML Programming) is really a higher-level API designed to take some of the complexity out of using DOM, SAX, or JDOM

Copyright @ 2009, John Jay King

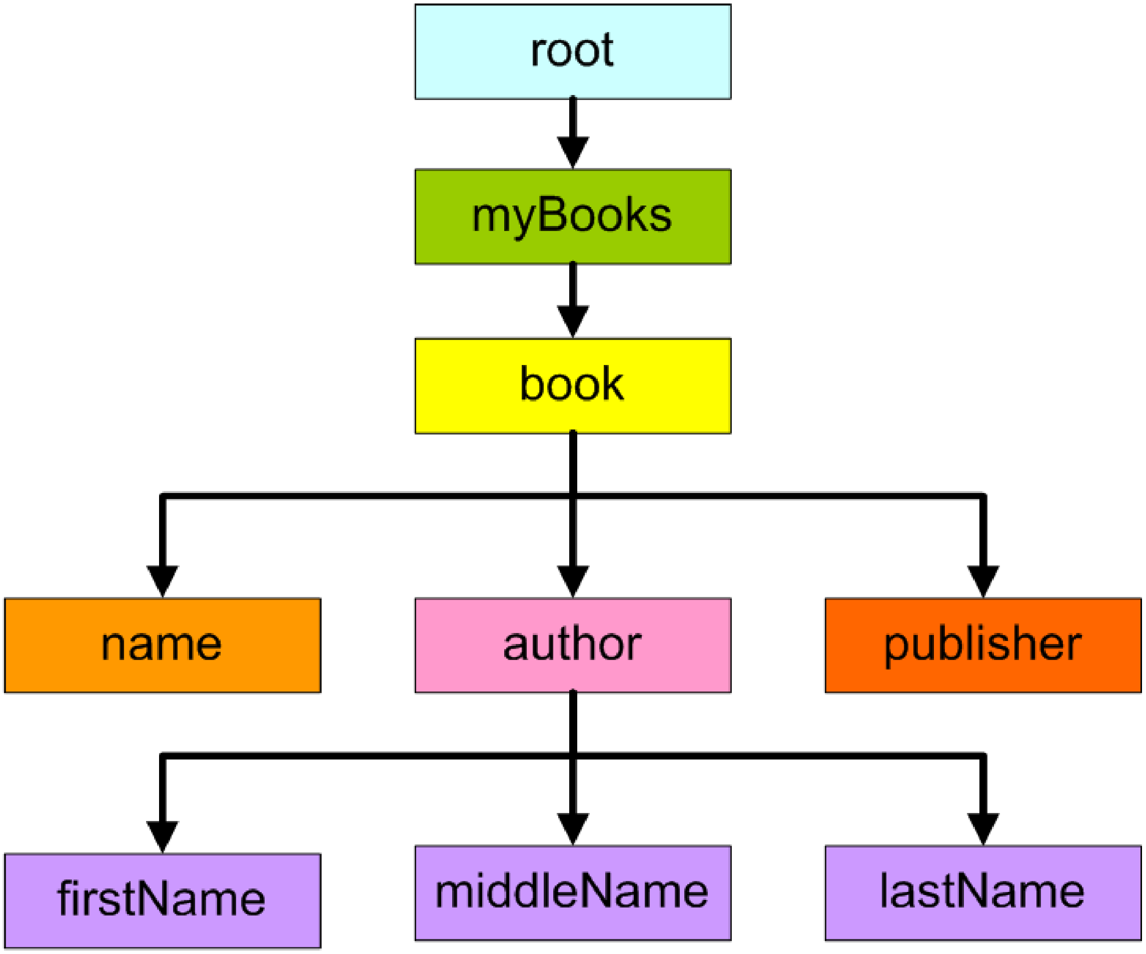
SAX, DOM, JDOM

- The three most commonly used Java-XML APIs today are SAX, DOM, and JDOM:
  - SAX allows the quickest possible processing of an XML document since data is read and written in a continuous stream; however, it is up to the programmer to make some sense of what is being processed
  - DOM reads an XML document into memory and creates a hierarchical "tree" structure that may be referenced in a reasonably simple fashion, DOM has been available for quite some time and is sometimes criticized for not being Java-like
  - JDOM is the brainchild of a Java programmer who wanted the convenience of DOM but also wanted a pure-Java mechanism
  - Other APIs exist, these are simply the most common

DOM

- DOM is a recommendation of the W3C
- DOM creates an object-tree that is very useful for parsing and processing XML's hierarchical data
- DOM is both platform and language agnostic and is heavily used in Java, C++, and JavaScript environments
- DOM parsers read XML documents and organize the data in memory into a “tree” structure of objects
- DOM then uses the “tree” for processing
  - Tree has a root that encompasses the entire document
  - Programs may navigate the branches of the tree
  - Nodes of the tree may be read and/or modified

DOM "Tree"



Copyright @ 2009, John Jay King

Simple API for XML (SAX)

- SAX is the basic mechanism for Java-XML programming (also IBM Enterprise COBOL)
- SAX reads an XML document and passes the document's elements one at a time to the program
- SAX uses “event-based” parsing where values are read and presented to the program using a method created by the program
- When using SAX at least two Java classes are involved:
  - Controlling class   Uses “Content Handler” to process XML document
  - Content Handler   Reads and processes XML data
- SAX might also involve an Event Handler class and/or an Error Handler class

SAX Content Handler

- Content handlers implement the ContentHandler interface and must include code for “events”:

– setDocumentLocator	Get object for finding SAX events
– startDocument	Begin XML document
– processingInstruction	Examine PI's (except xml PI)
– startPrefixMapping	Map prefix to namespace
– startElement	Start XML element
– characters	Process element characters
– endElement	End of XML element
– endPrefixMapping	Stop mapping prefix to namespace
– ignorableWhitespace	Return contiguous whitespace
– skippedEntity	Return name of skipped entity
– endDocument	End of XML document

SAX Parsing of “myBooks.xml”

- SAX parsers read a document as events:

```
start document
  start element: myBooks
    start element: book
      start element: name
        text: "Learning XML"
      end element: name
      start element: author
        start element: lastName
          text: "Ray"
        end element: lastName
        start element: firstName
          text: "Eric"
        end element: firstName
        start element: middleName
          text: "T"
        end element: middleName
      end element: author
      start element: publisher
        text "O'Reilly"
      end element: publisher
    end element: book
  ... <more “book” elements> …
  end element: myBooks
end document
```

JAXP

- JAXP (Java API for XML Parsing) was released by Sun in an attempt to make Java XML parsing simpler through abstraction
- Many programmers do not realize a new parser is being used since Sun's parser is downloaded when people download JAXP
- JAXP is really an API and is not a parser
- JAXP makes it easier to use the existing SAX, DOM, and JDOM APIs

Copyright @ 2009, John Jay King

JAXP and SAX

- JAXP works with SAX using the SAXParser class
- SAXParser objects are created using a SAXParserFactory object
- SAXParser and SAXParserFactory objects use the available SAX parser to accomplish their work
- Generic factory methods are used to initiate validation and namespace recognition
- Generic methods are used to test current settings
- The parse() method allows specification of a DefaultHandler object or HandlerBase object to handle document events

Copyright @ 2009, John Jay King

JAXP and DOM

- JAXP with DOM is pretty much the same as using JAXP with SAX only using slightly different class names
- The DocumentBuilderFactory and DocumentBuilder classes seen earlier are used to generically access the available DOM parser

JAXB

- JAXB (Java Architecture for XML Binding) provides a “bridge” between XML and Java, it uses APIs like SAX and DOM it does not replace them
- JAXB maps XML to Java objects
- JAXB uses a DTD at the same level of the XML
  - Maps XML to Java data objects
  - JAXB acts as a code-generator using a DTD or Schema to generate Java class code mapping XML to internal data objects
  - JAXB uses a DTD or Schema to “bind” to a set of generated classes
  - The binding schema is an XML-based binding language

Benefits of JAXB

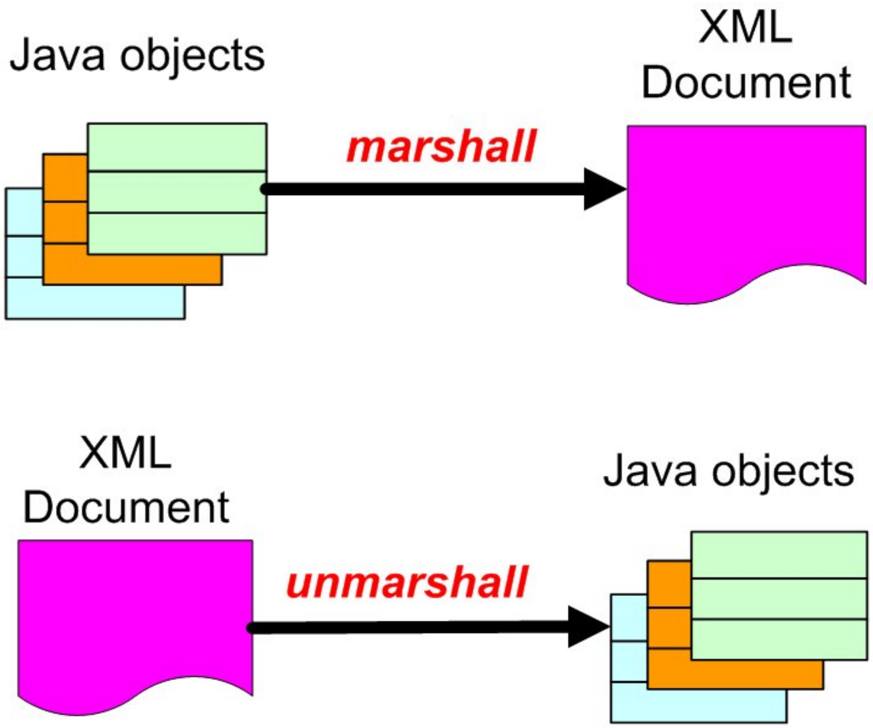
- Code is much simpler “hiding” the complexity of XML parsing
- SAX and DOM are generic XML parsers and parse any well-formed XML
- JAXB creates a SAX or DOM parser that is specific to your DTD or Schema and parses only valid XML
- JAXB produces a “tree” in memory specific to the Elements and Attributes defined by your DTD or Schema
- In early releases JAXB worked with DTD only

How JAXB Works

- JAXB is based upon data binding which includes:
  - Java source file and class generation
    Using utility program using DTD or Schema as input to produce Java class definitions matching document
  - Unmarshalling
    Taking data from XML document into Java objects
  - Marshalling
    Moving data from Java objects into XML document
  - Binding Schemas
    Rules for generating Java classes

Copyright @ 2009, John Jay King

Marshalling/Unmarshalling



Java objects

marshall

XML Document

XML Document

unmarshall

Java objects

StAX

- Before StAX programmers had only two likely choices when processing XML documents:
  - Reading/writing streamed XML content (SAX)
  - Building the entire document in memory before processing (DOM)
- Both SAX and DOM have drawbacks:
  - SAX programs must carefully track progress through the document creating miscellaneous structures as needed
  - DOM stores the entire document in memory using a generic "tree" structure that is easy to use but bulky
- Streaming APIs like SAX are called "push" APIs; they shovel data to the program whether the program is ready or not based upon events

Using StAX

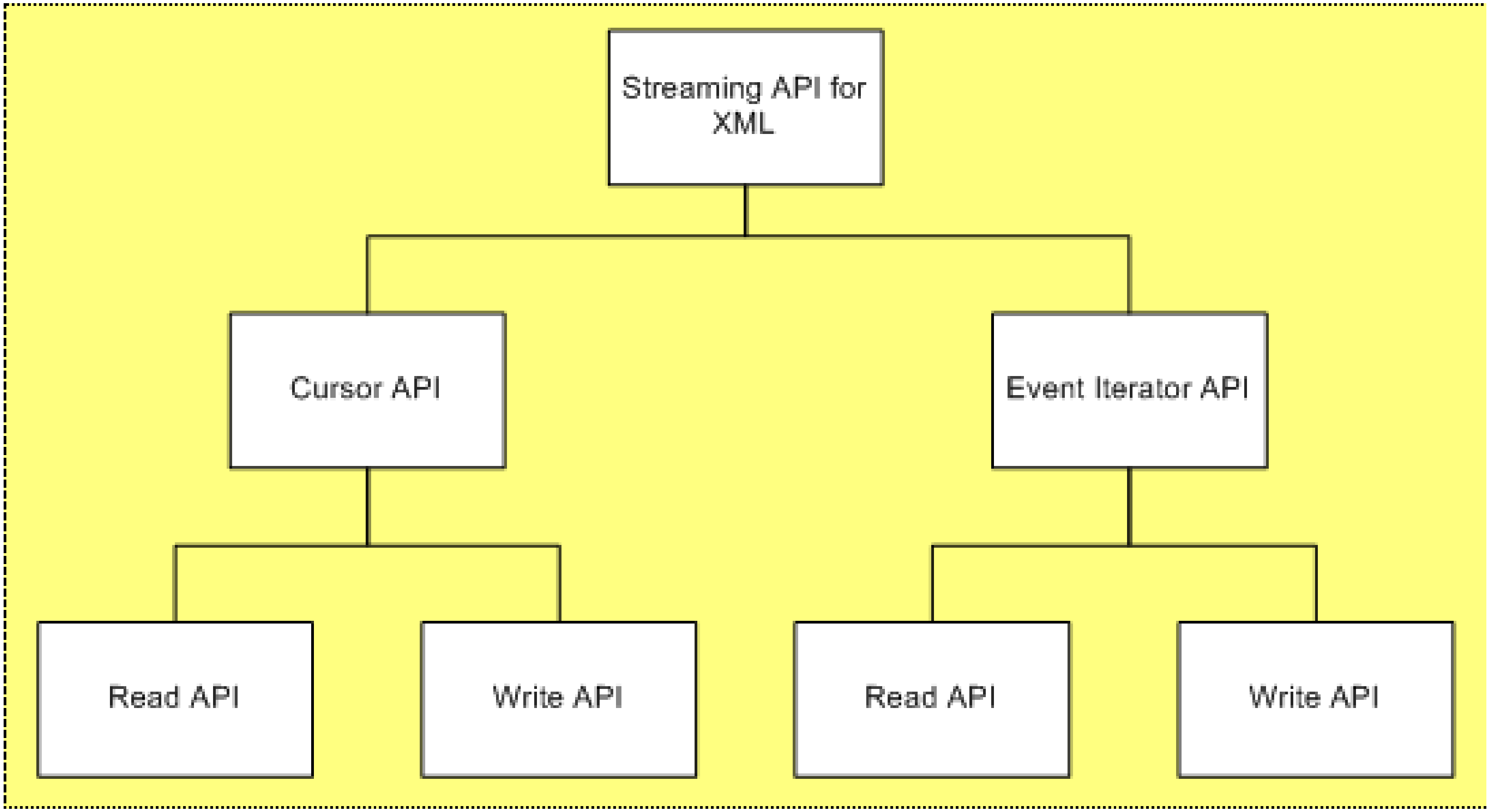
- StAX is even simpler than SAX, it is a "pull" type API that allows the programmer to control parsing via an iterator-based API and stream of events
- StAX provides both a low-level cursor API and a higher-level event-iterator API
  - Cursor-based API (based upon XMLStreamWriter); best for creating a document from application data
  - Event-based API (based upon XMLEventWriter); best when creating a new document based upon multiple existing documents

What is a “Pull-Type” API?

- StAX is a newer style of API, a pull API
- Pull APIs are streaming APIs and very fast, but are also memory efficient
- The program is in control asking the parser for the next part of the document the program wishes to process

  (the program pulls data from the document)

Dual Methods of StAX



StAX Cursor API

- The cursor-based API uses a virtual cursor to process the XML document
- When reading, the XMLStreamReader is created using an XMLInputFactory
- The API has a built-in iterator using hasNext() and next() methods to walk throught the document
- getText() and other methods may be used to obtain information from the current element, attribute, or text
- When writing with the Cursor API the XMLStreamWriter provides methods used to write elements, attributes, and text to the XML file
- XMLStreamWriter is created using the XMLOutputFactory

StAX Event Iterator API

- The event iterator API reads and writes XML data too
- A new factory, XMLEventFactory builds events to be processed
- Once again a built-in iterator using hasNext() and nextEvent() methods is available to process various events
- When writing, the XMLEventFactory creates events that may be added to the output with the XMLEventWriter object

Why Pull Parsing?

- Pull Parsing is the wave of the future for many applications
- Stax grew out of the need to read and write XML in an efficient manner in the context of XML Binding and Web Services
- XML Pull Parsing is touted as a high performance alternative to DOM for XML parsing that is easier to use than SAX
- SAX is a push API and used more widely than any other push API currently in use
- So far many pull APIs have been created and only recently has the industry coalesced around a single one (StAX)
- Choosing between tree oriented (DOM), streaming push (SAX), and pull (StAX) parsers it is important to understand the limits and advantages of each

Choosing Parser API

- Clearly there are many issues involved in processing XML data; choose the parser which best suits your situation
  - DOM Small-medium size documents; easy to use, supported widely
  - SAX Longer documents, more difficult to use, supported widely
  - StAX Longer documents, can be difficult to use, not available in all environments
  - JAXB Use SAX or DOM “under the covers”, requires Schema/DTD, Java-only, easier programming
  - JAXP Use SAX or DOM “under the covers”, easier than DOM/SAX/StAX not as easy as JAXB, Java-only

Oracle XML DB

- Oracle's XML support is provided as XML DB:
  - W3C (Worldwide Web Consortium) XML compliance
  - XMLType is an Oracle-defined datatype storing XML data
    - Unstructured (CLOB underneath)
    - Structured (“Shredded” into relational columns and rows)
    - Binary XMLType (new with Oracle 11g)
  - The XML parser is part of the database
  - Oracle provides several XML-oriented SQL functions to support XML, some support the emerging ISO/ANSI SQLX initiative
  - Check the reference manual for complete information: "XML DB Developer’s Guide"

Copyright @ 2009, John Jay King

XMLType Datatype

- XMLType may be used to represent a document or document fragment in SQL
- XMLType has several built-in member functions to operate on XML content
- XMLType may be used in PL/SQL as variables, return values, and parameters
- XMLType APIs are provided for both PL/SQL and Java programming
- XMLType is also supported on the client via FTP, HTTP, and WebDav

XMLType Functions

- XMLType member functions include:

– createXML()	Create XMLType instance
– existsNode()	Checks if XPath can find valid nodes
– extract()	Uses XPath to return XML fragment
– isFragment()	Checks if document is a fragment
– getClobVal()	Gets document as a CLOB
– getStringVal()	Gets value as a string
– getNumberVal()	Gets numeric value as a number
– isSchemaBased	Returns 1 if schema based (0 if not)
– isSchemaValid	True if XMLType is valid
– schemaValidate	Validates XMLType using Schema
– Transform	Apply XSL Stylesheet to XMLType
– XMLType	Constructs an XMLType instance from CLOB, VARCHAR2 or object

Copyright @ 2009, John Jay King

ISO-ANSI SQL/XML (SQLX)

- SQL/XML is an ISO-ANSI working draft for XML-Related Specifications (aka. SQLX)
- SQLX defines how SQL may be used with XML
- SQLX functions are used to generate XML from existing relational (and object relational) tables
- SQLX standard functions supported by Oracle:
  - XMLAgg()
  - XMLAttribute()
  - XMLCast ()
  - XMLComment ()
  - XMLConcat()
  - XMLElement()
  - XMLExists ()
  - XMLForest()
  - XMLParse ()
  - XMLPI ()
  - XMLQuery ()
  - XMLSerialize ()

Copyright @ 2009, John Jay King

Oracle SQL/XML Extensions

- XMLCdata — Generate cdata section from specified expression
- XMLColAttVal — Create series of XML fragments using an element name of "column" and column names and values as attributes
- XMLDiff — Compare two XML documents and return difference(s) as a document
- XMLPATCH — Patches XMLType using second XMLType
- XMLRoot — Generate XML identification line (PI)
- XMLSequence — Creates Varray of XMLType instances
- SYS_XMLGEN — Convert specified database row and column into an XML document
- SYS_XMLAGG — Generate single XML document from aggregate of XML data

Other XML Functions

- APPENDCHILDXML
- DELETEXML
- DEPTH
- EXTRACT (XML)
- EXISTSNODE
- EXTRACTVALUE
- INSERTCHILDXML
- INSERTXMLBEFORE
- PATH
- SYS_DBURIGEN
- SYS_XMLAGG
- SYS_XMLGEN
- UPDATEXML
- XMLTransform

Most-Used XML Functions, 1

- XMLElement is used to define Elements

```
XMLElement("MyElementName",valueExp)
```

  - MyElementName may be any valid XML name
  - valueExp may be a literal, column name, or expression providing the value for the element (May be nested)

- XMLAttributes is used to define Element Attributes; it should be used inside XMLElement and precede any SubElements for the chosen Element

```
XMLAttributes("MyAttributeName",valueExp)
```

  - MyAttributeName may be any valid XML name
  - valueExp may be a literal, column name, or expression providing the value for the element

Most-Used XML Functions, 2

- XMLForest works like nested XMLElements

```
XMLForest(valExp1,valExp2 AS "MyElement2")
```

  - valExp1 may be a literal, column name, or expression providing the value for the element
  - valExp2 may be a literal, column name, or expression providing the value for the element
  - MyElement2 may be any valid XML name
- XMLAgg aggregates calls to XMLElement, XMLAttribute, and XMLForest (and others) to create an XML document
- Column name used if Element and/or Attribute not explicitly named

XML Schema Support

- XML schemas may be used to automatically create tables and types, or, to validate updates and inserts
- XML schemas may be used as the basis for XMLType tables and columns (but, schemas are ***not*** required to store XMLType data)
- XML schemas must be registered in the database
- Once registered, XML schemas may be referenced using URL notation
- Registered XML schemas may be used to map XML documents to structured or unstructured database storage

Registering Schema

- Schemas must be created and tested (use an appropriate XML editor), then, register them with DBMS_XMLSCHEMA

```
begin
  dbms_xmlschema.registerschema('myBooks.xsd',
    '<?xml version="1.0" encoding="UTF-8"?>
     <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  elementFormDefault="qualified">
        <xs:element name="myBooks">
             <xs:complexType>
                <xs:sequence>
                   <xs:element ref="book"
                       maxOccurs="unbounded"/>
        **** rest of schema definition ****
     </xs:schema>',true,true,false,false);
end;
/
```

Using XMLType

- Three XMLType storage mechanisms are available:
  - Unstructured (CLOB underneath)
  - Structured (“Shredded” into relational columns and rows)
  - Binary XMLType (new with Oracle 11g)
- XMLType may be used to represent a document or document fragment in SQL

Unstructured XMLType

- Unstructured XML Type data is stored AS-IS without any change to the data
- Internally the data is stored as CLOB

Structured XMLType

- XML data is “shredded”
  - XML data separated and stored as database columns and rows on input
  - XML data “unshredded” (reassembled) for output
- Whitespace and formatting is removed but DOM fidelity is maintained (data is not altered)
- Requires XML Schema
  (must be registered before use)

Binary XMLType (11g on)

- Several improvements over unstructured storage including:
  - More efficient database storage
  - Piece-wise updating
  - Indexing
  - Fragment extraction
- XML data stored "as-is" (whitespace/formatting unaltered)
- May use XML Schema before or after initial creation but does not require XML Schema

Testing XML Performance

- Over the following pages are several examples using the Oracle “SH” Sales sample schema to produce an XML document containing product orders (one XML document per order)
  - Query uses SQL XML functions to select database data and create XML output
  - Query joins the SH.SALES and SH.PRODUCT tables
  - The query was limited to return only 29,999 rows (rownum < 30000)

Test Data Creation

- Three sets of test data were created:
  - Data with Long Element and Sub-Element names
  - Data with Abbreviated Element and Sub-Element names
  - Data with Sub-Elements converted to Attributes

Long Names

```
select xmlroot(xmlelement("sales",xmlelement("product",
  xmlattributes(cust_id as "custId",
                pr.prod_id as "prodId"),
  xmlelement("prodName",prod_name),
  xmlelement("timeId",
              to_char(time_id,'yyyy.-hh24:mi.ss')),
  xmlelement("supplierId",supplier_id),
  xmlelement("category",prod_category),
  xmlelement("categoryId",prod_category_id),
  xmlelement("price",prod_list_price),
  xmlelement("qtySold",quantity_sold),
  xmlelement("gross",amount_sold))))
      from sh.sales sa, sh.products pr
      where sa.prod_id = pr.prod_id
        and sa.channel_id in (3)
        and sa.promo_id in (999)
        and extract (year from sa.time_id) = 2000
        and rownum < 30000
```

Sample Long Data

- Data below is formatted; test data eliminated whitespace, tabs, indentation, and crlf characters

```
<?xml version="1.0" encoding="UTF-8"?>
<sales>
  <product custId="564" prodId="13">
    <prodName>5MP Telephoto Digital Camera</prodName>
    <timeId>2000.-00:00.00</timeId>
    <supplierId>1</supplierId>
    <category>Photo</category>
    <categoryId>204</categoryId>
    <price>899.99</price>
    <qtySold>1</qtySold>
    <gross>1075.12</gross>
  </product>
</sales>
```

313 Chars

Short Names

```
select xmlroot(xmlelement("sa",xmlelement("pr",
   xmlattributes(cust_id as "cid",
                pr.prod_id as "pid"),
   xmlelement("nm",prod_name),
   xmlelement("tid",
              to_char(time_id,'yyyy.-hh24:mi.ss')),
   xmlelement("sid",supplier_id),
   xmlelement("ct",prod_category),
   xmlelement("ctId",prod_category_id),
   xmlelement("pr",prod_list_price),
   xmlelement("qty",quantity_sold),
   xmlelement("gr",amount_sold))))
from sh.sales sa, sh.products pr
      where sa.prod_id = pr.prod_id
        and sa.channel_id in (3)
        and sa.promo_id in (999)
        and extract (year from sa.time_id) = 2000
        and rownum < 30000
```

Sample Short Data

- Data below is formatted; test data eliminated whitespace, tabs, indentation, and crlf characters

```
<?xml version="1.0" encoding="UTF-8"?>
  <sa>
    <pr cid="564" pid="13">
    <nm>5MP Telephoto Digital Camera</nm>
    <tid>2000.-00:00.00</tid>
    <sid>1</sid>
    <ct>Photo</ct>
    <ctId>204</ctId>
    <pr>899.99</pr>
    <qty>1</qty>
    <gr>1075.12</gr>
  </pr>
</sa>
```

215 Chars

Sub-Elements as Attributes

```
select xmlroot(xmlelement("sa",
   xmlattributes( cust_id as "cid",
      pr.prod_id as "pid",
      prod_name as "nm",
      to_char(time_id,'yyyy.-hh24:mi.ss') as "tid",
      supplier_id as "sid",
      prod_category as "cat",
      prod_category_id as "ctId",
      prod_list_price as "pr",
      quantity_sold as "qty",
      amount_sold "gr"))))
from sh.sales sa, sh.products pr
      where sa.prod_id = pr.prod_id
        and sa.channel_id in (3)
        and sa.promo_id in (999)
        and extract (year from sa.time_id) = 2000
        and rownum < 30000
```

Sample Attribute Data

- Data below is formatted; test data eliminated whitespace, tabs, indentation, and crlf characters

```
<?xml version="1.0" encoding="UTF-8"?>
<sa>
   <pa cid="564" pid="13"
     nm="5MP Telephoto Digital Camera"
     tid="2000.-00:00.00" sid="1" cat="Photo" ctId="204"
      pr="899.99“ qty="1" gr="1075.12">
  </pa>
</sa>
```

179 Chars

Total Sample Size

- For each type (Long, Short, Attribute) of output 29,999 XML documents were generated
- I/O costs and times were similar in all three cases
- Output sizes varied immensely:

– Long	39.4 MB transmitted to client
– Short	28.9 MB transmitted to client
– Attribute	27,3 MB transmitted to client

(not really a meaningful number except to illustrate difference in return quantities)

Unstructured Data

- For each type of data an Unstructured table using the XML documents as a data item were created
  - Each row uses a sequence number to create a key
  - Each row has one XML document as a column

```
create table xmlperfLong1 (
    sales_id number(5) not null primary key,
    sales  XMLTYPE )
    tablespace xmlperfLong1_space;
```

Unstructured Insert

- For each type of data the queries shown earlier were used in an “INSERT … AS SELECT …”

```
insert into xmlperfLong1
  select
    -- “Long”, “Short”, and “Attribute” query
      from sh.sales sa, sh.products pr
      where sa.prod_id = pr.prod_id
        and sa.channel_id in (3)
        and sa.promo_id in (999)
        and extract (year from sa.time_id) = 2000
        and rownum < 30000;
```

Unstructured Performance

- Here are some basic numbers from the creation of Unstructured data using SQL*Plus Autotrace output (my DBA friends are groaning here…), timing, and tablespace size

Long	Short	Attribute
70715 recursive calls 109735 db block gets 9293 consistent gets 1023 physical reads	70616 recursive calls 106453 db block gets 8865 consistent gets 1008 physical reads	70439 recursive calls 105908 db block gets 8679 consistent gets 1009 physical reads
Time: 00:00:29.12	Time: 00:00:23.06	Time: 00:00:21.93
14,286,848 bytes	11,141,120 bytes	20,185,088 bytes

Structured Data

- For each type of data an Structured table using the XML documents as a data item were created
  - Each row uses a sequence number to create a key
  - Each row has one XML document as a column using a schema

```
create table xmlperfLong1U (
   sales_id number(5) not null primary key,
   sales  XMLTYPE )
   xmltype column sales store as object relational
       xmlschema "xmlperfLong.xsd"
       element "sales"
   tablespace xmlperfLong1U_space;
```

Structured Insert

- For each type of data the queries shown earlier were used in an “INSERT … AS SELECT …”

```
insert into xmlperfLong1U
  select
    -- “Long”, “Short”, and “Attribute” query
      from sh.sales sa, sh.products pr
      where sa.prod_id = pr.prod_id
        and sa.channel_id in (3)
        and sa.promo_id in (999)
        and extract (year from sa.time_id) = 2000
        and rownum < 30000;
```

Structured Performance

- Here are some basic numbers from the creation of Structured data using SQL*Plus Autotrace output (my DBA friends are groaning here…), timing, and tablespace size

Long	Short	Attribute
70394 recursive calls 103296 db block gets 8404 consistent gets 1054 physical reads	70316 recursive calls 103305 db block gets 8385 consistent gets 1042 physical reads	70228 recursive calls 103312 db block gets 8385 consistent gets 1034 physical reads
Time: 00:11:25:53	Time: 00:11:26.06	Time: 00:11:27:39
8,388,608 bytes	8,388,608 bytes	8,388,608 bytes

Binary Data

- For each type of data a Binary table using the XML documents as a data item were created
  - Each row uses a sequence number to create a key
  - Each row has one XML document as a column

```
create table xmlperfLong1U (
  sales_id number(5) not null primary key,
  sales  XMLTYPE )
  xmltype column sales store as binary xml
     xmlschema "xmlperfLong.xsd"
     element "sales"
  tablespace xmlperfLong1U_space;
```

Copyright @ 2009, John Jay King

Binary Insert

- For each type of data the queries shown earlier were used in an “INSERT … AS SELECT …”

```
insert into xmlperfAttr1U
  select
    -- “Long”, “Short”, and “Attribute” query
      from sh.sales sa, sh.products pr
      where sa.prod_id = pr.prod_id
        and sa.channel_id in (3)
        and sa.promo_id in (999)
        and extract (year from sa.time_id) = 2000
        and rownum < 30000;
```

Binary Performance

- Here are some basic numbers from the creation of Binary data using SQL*Plus Autotrace output (my DBA friends are groaning here…), timing, and tablespace size

Long	Short	Attribute
71180 recursive calls 103074 db block gets 278786 consistent gets 1063 physical reads	71401 recursive calls 103220 db block gets 278835 consistent gets 1105 physical reads	71386 recursive calls 103604 db block gets 278889 consistent gets 1105 physical reads
Time: 00:05:05:54	Time: 00:05:06.56	Time: 00:04:56:23
6,946,816 bytes	7,995,392 bytes	7,995,392 bytes

Indexing XML Content

- XML data may be indexed to increase efficiency like other Oracle tables
- Unstructured XML data (CLOB storage)
  - XML & text aware indexing and searching with Oracle Text
- Structured XML data (Object-Relational storage)
  - Automatic query rewrite enables all existing indexes types
- Binary XML data
  - Use XMLIndex type or standard index

```
create index book_author on myBooks
   (books.extract('/myBooks/book/author.text()').getStringVal())
```

11g XMLIndex

- Oracle 11g introduces a new index type for XMLType called XMLIndex
- XMLIndex can improve performance of XPath-based predicates and fragment extraction
- XMLIndex is a (logical) domain index consisting of underlying physical table(s) and secondary indexes (replaces CTXSYS.CTXXPath; Oracle recommends replacing any CTXXPath indexes with XMLIndex)
- Supported by PL/SQL DBMS_XMLINDEX package

Copyright @ 2009, John Jay King

Test Platform

- When tuning performance there are few absolutes; all tests shown in these notes were performed under the following conditions:
  - Hardware: x86, 4GB RAM, Toshiba Tecra M9
  - Software: Microsoft Windows XP Pro,
SQL*Plus,
Oracle 11.1.0.6 Enterprise Edition
  - Machine had no other significant programs consuming resources or disk activity
  - Only 30,000 (29,999) rows of test data were processed

"Your Mileage May Vary…"

- Caution! Your results may be different than mine
- Please confirm any "improvements" you make by testing thoroughly in the actual runtime environment

Copyright @ 2009, John Jay King

Conclusion

- Can I Make XML Go Faster? -- Yes!
  - Eliminate non-essential whitespace including tabs and carriage-return line-feed characters
  - Eliminate non-essential Elements and Attributes
  - Reduce size of Element and Attribute Names
  - Consider “flattening” document by converting Elements into Attributes where practical
  - Choose appropriate Oracle XMLType
    - Unstructured
    - Structured
    - Binary
  - Index XMLType data

Copyright @ 2009, John Jay King

RMOUG

ROCKY MOUNTAIN ORACLE USERS GROUP

Training Days 2010

Save the dates!

February 17-18 2010!

Collaborate 2010

OAUG
oracle applications users group

Quest
International Users Group

IOUG
independent oracle users group

COLLABORATE 10

Technology and Applications Forum for the Oracle Community

Save the Date: ??? 2010

Las Vegas, Nevada!

ODTUG KALEIDOSCOPE 2009

- Web Architecture
- Web and Java
- Web and Not Java
- Database Server
- Best Practices
- Oracle Tools
- Essbase
- Hyperion
- BEA
- Third Party Tools
- Professional Development
- Application Express
- Business Intelligence and Data Warehousing

www.odtugkaleidoscope.com

Please fill out session Evaluations

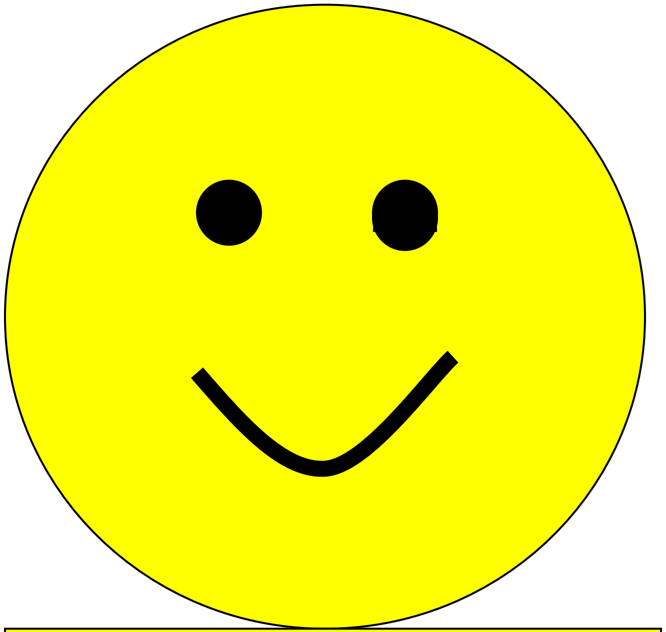
Can I Make XML Go Faster?

To contact the author:

John King

King Training Resources

6341 South Williams Street
Littleton, CO 80121-2627 USA
1.800.252.0652 - 1.303.798.5727
Email: john@kingtraining.com



Thanks for your attention!

Today's slides are on the web:

http://www.kingtraining.com