
ORACLE®

Complex XML Schemas

Sam Idicula
Consulting Member of Tech. Staff, Oracle XML DB

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

What is an XML Schema ?

- W3C standard for defining the content of an XML document
  - Replaces the DTD format XML inherited from SGML
  - Strong data typing, 47 Scalar data types
  - Extensible type model
- An XML Schema is an XML document
  - Content is defined by W3C “Schema for Schemas”
- Typically authored using graphical tools such as Altova’s XMLSpy and Oracle’s Jdeveloper.

XML Schema Usage

- Primarily used for validation
  - An XML Schema validator can determine whether an XML document is a valid instance of an XML Schema.
- Explosion of XML schema based industry standards
  - Enable organizations to share information in common, well defined and verifiable formats
  - Financial Services : FpML, XBRL, SwiftML, FixML
  - Public Safety : NIEM
  - Authoring and Publishing : Dita, Docbook, Open XML
  - Geospatial : GML, KML
  - Healthcare : HL7
  - Oil and Gas : WitsML

XML Schema and Object Relational Storage

- Predicated on XML Schema
  - SQL Object model derived from the XML Schema
  - Lossless conversion between XML and SQL Objects
  - SQL Objects stored as XMLType and Nested tables
- XQuery operations compiled into the same relational algebra as SQL statements
  - Optimizer executes SQL and XQuery in the same way
- Light-Weight validation performed as part of the conversion to SQL object model
  - Full XML Schema validation via check constraint or trigger

XML Schema and Binary XML Storage

- XML stored in a post-parsed binary format in a secure file LOB.
- Optimized token management,
  - Set of tag names is known ipso facto
- Non-character data (Numbers, dates, etc) stored using native representations
- Query and update operations optimized based on knowledge of XML Schema
- Full Schema validation part of the Binary Encoding process

XML Schema Registration

- Must register XML schema with the database before using it for Binary or XML storage
- Compiled version of the XML Schema stored in the XML DB repository
  - Compiled version stored memory as an SGA object.
  - Shared by all database users

What makes schema registration complex?

- Mutually recursive dependencies (import/include) between multiple schemas
- Deep type hierarchy and/or large types
- Large number of subtypes for a particular complex type
- Cycles involving base and extension types
- Lot of substitution groups
- Large number of elements in a single substitution group

Complex Schema Examples

- NIEM: National Information Exchange Model
  - 115 base schemas and several extensions
  - Complex interdependencies between schemas
  - Over 2000 subtypes
  - Deep type hierarchy
- HL7 CDA: Healthcare - Clinical Doc Architecture
  - Over 100 schemas
  - Mutually recursive dependencies between schemas
  - Large number of substitution group elements
  - Large number of subtypes

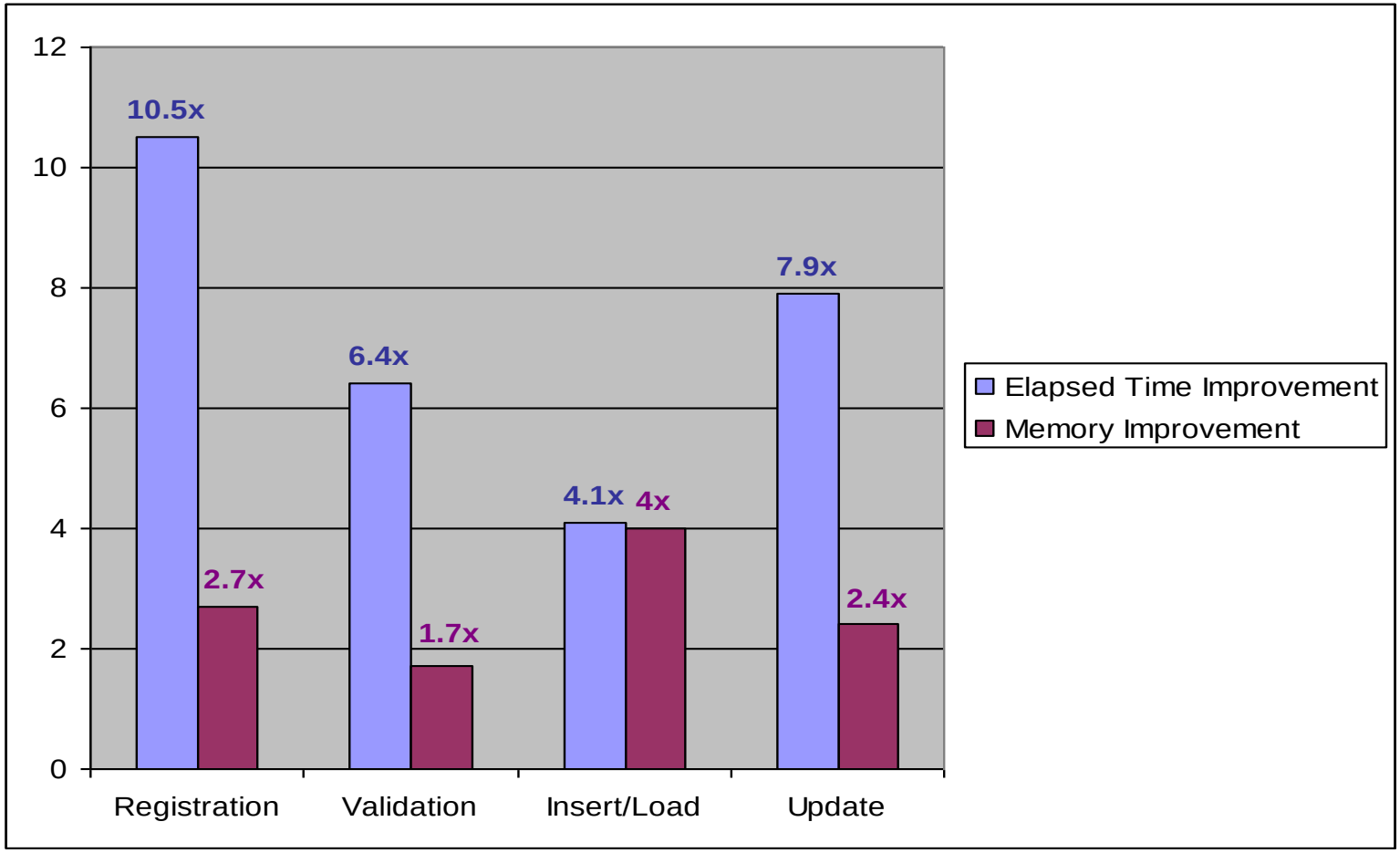
11gR2 Schema Enhancements

- Major improvements in both memory & time
- Much faster Registration and loading of Schemas
  - Speedup is higher for more complex schema sets
  - Upto 200x in some cases
- 2-3x reduction in shared memory usage
- Much less PGA usage during schema registration

11gR2 Schema Enhancements

- StoreVarrayAsTable annotation now honored when creating XMLType tables and columns manually
- Streaming Schema Validator Cache for Binary XML
  - Validation
  - DML: Insert & partial update
  - Significant improvements for small documents

11gR2 Performance



Schema Registration Tips

- Set aside sufficient shared_pool memory
  - Examples: NIEM needs about 1024M
- Register different schemas or sets of schemas in different PL/SQL blocks if possible
- If generating tables, set xdb:defaultTable="" as appropriate
- For O-R storage
  - Break up large types using xdb:SQLInline="false"
  - Use xdb:maintainDOM=false on appropriate elements to improve performance
    - No ordering, substitution groups etc

11.2.0.2.0 Enhancements

- Improvements in cycle detection with-in type hierarchy
  - Errors related to type cycles eliminated
  - Significant reductions in errors related to memory usage

11.2.0.2.0 Enhancements

- New package to manage schema annotation
  - DBMS_XMLSCHEMA_ANNOTATE*
  - Common annotations can be applied programmatically
  - Makes it easier to annotate XML Schemas
  - Makes it easier to manage annotations as schemas change
- New package to help manage object-relational storage
  - DBMS_XMLSTORAGE_MANAGE*
  - Simplifies renaming tables and creating indexes for object relational storage

* Packages need to installed manually

Object relational storage

- XML Schema simpleTypes map directly to SQL data types
  - Elements and attributes based on simple types become an attributes of a SQL object
  - Each simpleType requires a single column in the underlying storage table
  - XDB Annotation mechanism can be used to overide the default mapping between simple types and SQL Types
- SQL Object Types generated from XML Schema complexTypes
  - VARRAY Types generated for repeating elements
  - Elements based on complex types can require 100’s or 1000’s of columns in the underlying storage table

Object relational mapping

- SQL Object and Attribute names derived from XML names
- Name-mangling algorithm used to map from valid XML names to valid SQL names
  - Name mangling applies to SQL Attribute name, SQL Type names and Table names
  - XDB Schema annotation mechanism can be used to override name mangling.
- Annotations can be provided programmatically using

DBMS_XMLSCHEMA_ANNOTATE.setSQLName()

DBMS_XMLSCHEMA_ANNOTATE.setSQLType()

DBMS_XMLSCHEMA_ANNOTATE.setDefaultTable()

DOM Fidelity

- Content and order of the nodes for a document stored in XDB is identical to the content and order of the nodes in the original document
  - Does not preserve insignificant whitespace
- Dom Fidelity maintains instance meta data
  - Overhead when inserting and retrieving XML
  - Additional storage requirements
- Metadata is tracked for each element based on a complexType
  - Managed using a Positional Descriptor attribute

The Positional Descriptor

- The Positional Descriptor contains information relating to the use of
  - Comments and Processing Instructions
  - Mixed content text() nodes
  - Location and Prefix in Namespace declarations
  - Empty Vs Missing Nodes and xsi:nil usage
  - Substitutable elements
  - Ordering of elements for a repeating choice
- The Positional Descriptor is named SYS_XDB$PD
  - The format is not documented
  - Stored as a (in-line) LOB.

Disabling DOM Fidelity

- DOM Fidelity is typically not required for most ETL type use cases
- Documents will still be valid per the XML Schema
- Disabling DOM Fidelity improves performance
  - Eliminates overhead associated with maintaining instance level metadata
  - Reduces storage requirements
  - Improves elapsed time for insert and retrieval operations

Disabling DOM Fidelity

- Ordering of members of a collection is preserved when repetition is specified at element level. Eg

  <element name="Foo" type="FooType" maxOccurs="10"/>

- Ordering of members is not preserved when repetition is specified at the 'model' level. E.g.

  <choice maxOccurs="unbounded>…

  <element "Foo" type="FooType>

  <element "Baa" type="BaaType">

  </choice>

Disabling DOM Fidelity

- DOM Fidelity is disabled on a Type by Type basis
  - Add annotation xdb:maintainDOM="false“ on complexType definition
  - Removes the PD attribute from the SQLType.
  - Causes the information managed in the PD to be discarded when ingesting the XML document
- Can be done programmatically using

  DBMS_XMLSCHEMA_ANNOTATE. disableMaintainDOM()

Using DBMS_XMLSCHEMA_ANNOTATE

DEMO

Nested Table storage

- Normalized (Master-Detail) storage model for repeating elements
  - Repeating elements mapped to SQL VARRAY object types
  - VARRAY is stored as a nested table
  - Each member of the VARRAY becomes a row in the nested table
  - Relationships between parent and child tracked using system generated Primary Key / Foreign Key
- Improves performance
  - Operations executed as SQL on the nested table
  - Enables indexing of repeating elements and attributes
- Multiple levels of nesting supported

Nested Table Storage

Specifying Nested Table storage model

- Collection Storage controlled by schema level annotation “xdb:storeVarrayAsTable”
  - xdb:storeVarrayAsTable=“false” : Use Lob based storage
  - Xdb:storeVarrayAsTable=“true” : Use Nested Tables
- Default setting
  - Pre 11gR1 : “false”
  - 11gR1 : “true”
- Scope
  - Pre 11gR2 : Annotation applies to XMLType tables generated during call to dbms_xmlschema.registerSchema()
  - 11gR2 : Annotation applies all XMLType tables and columns bound to the XML Schema

Nested tables with manual DDL (Pre 11gR2)

- In 11gR1 and earlier manual DDL must explicitly specify nested table storage when using create table
  - STORE VARRAY AS TABLE clause required for each collection
  - Create Table statements becomes very complex for deeply nested collections
- In 11gR2 nested tables will automatically be generated when using create table.
  - No need to specify STORE VARRAY AS TABLE
  - Nested tables are given system generated names
  - Nested tables can be renamed programmatically using **DBMS_XMLSTORAGE_MANAGE. renameCollectionTable()**

Use of the “Default” table

- In 11gR1 and earlier many applications relied on the “default” table to manage XML
  - Used nested-table storage model by default
  - Default Table is meant to be used with XML DB repository.
- In 11gR2 there is no difference between tables or columns created using create table and default tables
  - Do not specify default tables in the XML schema unless intent is to use the XML DB Repository.
  - Register XML Schemas with genTables false unless Out-Of-Line storage model is required
  - Use Create Table statement to create tables or columns of XMLType after registering XML schema

Simplifying Nested Table usage in 11gR1

- If you need nested table storage in 11gR1 or earlier use 11gR2 and DBMS_METADATA package
  - Register the XML schema in 11gR2
  - Execute the create table statement in 11gR2
  - Capture the actual DDL for the table using DBMS_METADATA().
  - Edit to remove 11gR2 specific options
  - Register XML Schema in 11gR1
  - Execute DDL in 11gR1

B-Tree Indexing on Object-Relational Storage

- Indexing singletons is easy
  - Create an index using XMLCast(XMLQuery)
  - Confirm Index creation was re-written by checking that there is no entry for the index in user_ind_expressions
- Indexing collections is more complex
  - Collections are stored in nested tables
  - Indexes need to be created directly on the nested table
- Both types of Indexes can be created using xpath expressions using

  DBMS_XMLSTORAGE_MANAGE. xpath2TabColMapping()

Mapping non-repeating element

```
select  DBMS_XMLSTORAGE_MANAGE.XPATH2TABCOLMAPPING
        (
          USER,
          'PURCHASEORDER ',
          NULL,
          '/PurchaseOrder/Reference ',
          ' '
         ) from dual
```

```
<Result>
  <Mapping TableName="PURCHASEORDER" ColumnName="SYS_NC00008$"/>
</Result>
```

- Package returns the name of the collection table and the internal column name for the specified node specified by the XPath
- This name can be used in DDL operations such as create index.

Mapping repeating elements

```
select  DBMS_XMLSTORAGE_MANAGE.XPATH2TABCOLMAPPING
        (
          USER,
          'PURCHASEORDER ',
          NULL,
          '/PurchaseOrder/LineItems/LineItem/Part ',
          ' '
         ) from dual
```

```
<Result>
  <Mapping TableName="LINEITEMS_TABLE" ColumnName="SYS_NC00008$"/>
</Result>
```

- Package returns the name of the collection table and the internal column name for the specified node specified by the XPath
- This name can be used in DDL operations such as create index.

Using DBMS_XMLSTORAGE_MANAGE

DEMO

Schema Ordering

- Registration requires XML schema be valid
  - All external references (include / import) must be resolvable
  - Schemas must be registered in order
  - ‘Force’ mode used to manage cyclic dependencies
- Ordering is not always obvious
  - Need to traverse the set of include / import elements in all of the related XML schemas
  - Need to detect cyclic dependancies
- Common for include / import to use relative URLs

Calculating Schema Ordering

- Utility package XDB_ANALYZE_XMLSCHEMA can help with ordering schemas
  - Method schemaOrderingScript() will create a SQL script that will register a set of related XML schemas in the correct order.
  - Arguments
    - The path to an XML DB repository folder that contains the set of XML schemas to be registered
    - The relative path to the XML Schema that contains the definition of the root element
    - Any modifier which needs to be applied to the relative path to generate the correct value for the schema location hint
  - Minimizes the number of ‘force’ operations required to successfully register the set of XML schemas

Schema Ordering and Type Analysis

DEMO : FPML V5.0

Type Compilation and Analysis

- Complex complexTypes can result in SQL Objects that require 100's or 1000's of columns
- A table can only have 1000 column
- Need to break the SQL Objects into more manageable units
- Move elements out-of-line
  - Creates an XMLType table which contains a fragment
  - Enables a degree of normalization of the XML Schema
  - The number of columns required by the parent type is reduced by the number of columns moved out-of-line

Type analysis

- Needs to be looked at holistically
  - Cannot work on a schema by schema basis as types that extend types will change the number of columns required
  - Need to register all XML Schemas then perform compilation and analysis
  - Calculate the set of annotations required to register the XML schema.

Type analysis

- Generate a set of scripts that will register the set of XML Schemas
  - Uses DBMS_XMLSCHEMA_ANNOTATE package
  - May take a few minutes to execute all the scripts
- Script can be generated programmatically using package XDB_ANALYZE_XMLSCHEMA

Schema Registration

DEMO : FPML V5.0

Schema Evolution & Other Considerations

- Evolution: Changes in schema to which all documents (new & old) should conform
- Use InPlaceEvolve for most common backward compatible changes
- CopyEvolve for other changes
  - Involves data copy => time-consuming process
- When is schema-optimized persistence not useful?
  - Too many schema changes (especially backward-incompatible)
  - Insert & Full-retrieval overheads negate schema benefits
  - Alternative: Non-schema-based Binary XML storage & XML Index

SOFTWARE. HARDWARE. COMPLETE.