

XML (part 2)

Chapitre 2b: BD XML

Outline

1. Background
2. Native XML Databases: Tamino
3. Non-native XML Databases: Oracle9i (9.2)
 - 3.1. storing (no schema)
 - 3.2. querying
 - 3.3. updating
 - 3.4. storing (schema-based)
 - 3.5. mapping

1. Background

- **Problem:**
 - storage of (large sets of large) XML documents
- **Goal:**
 - find a storage system that supports both efficient search and updates
- **Several possibilities of storing XML documents**
 - in flat files
 - in relational (or object-oriented) databases
 - in XML databases

1. Background

■ Storing XML data in **flat files**:

Pros:

- natural storage mechanism (XML is primarily a file format)
- XML tools to access and query XML data in files

Cons:

- lacks data isolation, integrity checks, atomicity, concurrent access, security (that are available in RDBMS)

1. Background

- Storing XML data in **relational databases**:
 - Pros:
 - reuse of existing mature technology
 - relational databases are widely used
 - Convert relational DB to XML data: straightforward
R(A, B, C) with 2 tuples: {a1, b1, c1} and {a2, b2, c2}

```
<R>
  <tuple>
    <A>a1</A>
    <B>b1</B>
    <C>c1</C>
  </tuple>
  ...
</R>
```

Convert XML data to relational DB: not straightforward

1. Background

- ❑ Cons: mismatches between XML data and relational DB

XML

- Data in single hierarchical structure
- Elements can be nested
- Elements are ordered
- Schema optional
- Elements can be recursive
- Direct storage/retrieval of XML documents
- Query with XML standards

RDB

- Data in multiple tables
- Atomic cell values
- Attribute/tuple order not defined
- Schema required
- Little support for recursion
- Joins often necessary to retrieve decomposed XML documents
- Query with SQL retrofitted for XML

1. Background

❑ Approach 1: **Tree representation**

XML data modeled as a tree, stored in 2 relations

node(id, type, label, value)

child(child-id, parent-id, position)

- a tuple is inserted in *node* for each element and attribute

id: unique identifier

type: element or attribute

label: element name or attribute name

value: text value of element or attribute value (nullable)

- a tuple is inserted in *child* to record the parent element of each element and attribute

position: position of child among the children of the parent (order)

1. Background

- Pros:
 - all XML information represented
 - many XML queries can be translated into relational queries and executed within the DBMS
- Cons:
 - each element is broken in many pieces, leading to a large number of joins to reassemble an element

1. Background

example:

```
<bank-2>
  <account account-nb="A-101">
    <branch-name>Downtown</branch-name>
    <balance>500</balance>
  </account>
  <account account-nb="A-102">
    <branch-name>Perryridge</branch-name>
    <balance>400</balance>
  </account>
  <customer customer-id="C-10"
    accounts="A-101 A-102">
    <customer-name>Johnson</customer-name>
    <customer-city>Palo Alto</customer-city>
  </customer>
</bank-2>
```

- give the 2 relations for the XML document bank2.xml (next slide)

- translate the following query into SQL:
/bank-2/account/branch-name/text()

[illegible]

1. Background

- relations *Node* and *Child*:

Id	Type	Label	Value

Child-id	Parent-id	Position

1. Background

- ❑ Approach 2: **Map** XML data to relations
 - takes into account the DTD or XML schema
 - different approaches proposed in the literature (e.g. inlining)

1. Background

- Storing XML data in **XML databases**:
 - Native XML databases: XML is the basic data model
 - Native XML DBMS built from bottom up to easily store, retrieve, and query XML data
 - Supports XML DTDs or schemas
 - Supports XPath or other XML query languages (XQuery)
 - Non-native XML databases: XML model is built as a layer on top of the relational or object model

1. Background

- XML documents can be:
 - **data-centric:**
 - Regular structure
 - Order does not matter
 - Mixed content does not occur
 - **document-centric:**
 - Less regular structure
 - Order is significant
 - Mixed content occurs
 - **hybrid**

2. Tamino

■ Native XML DBMS:

DBMS built and designed for the handling of XML,
not a DBMS for an arbitrary data model with an XML
layer on top

- it will not make a difference at the user level, but a fundamental difference is inside the system
- no mapping of XML to another data model, avoiding thus 'impedance mismatch'
 - no limitations in functionalities
 - better performance

2. Tamino

■ Tamino XML Server

- ❑ handles data-centric and document-centric documents uniformly
- ❑ is designed to process XML documents efficiently regardless of their structure
- ❑ can store other types of data (e.g. images, sound files)
- ❑ has a complete DBMS built in: transactions, security, multiuser access
- ❑ has querying facilities: full-text retrieval, XPath, XQuery

2. Tamino

- **Tamino DB**

- consists of multiple collections. Each document resides in only one collection.

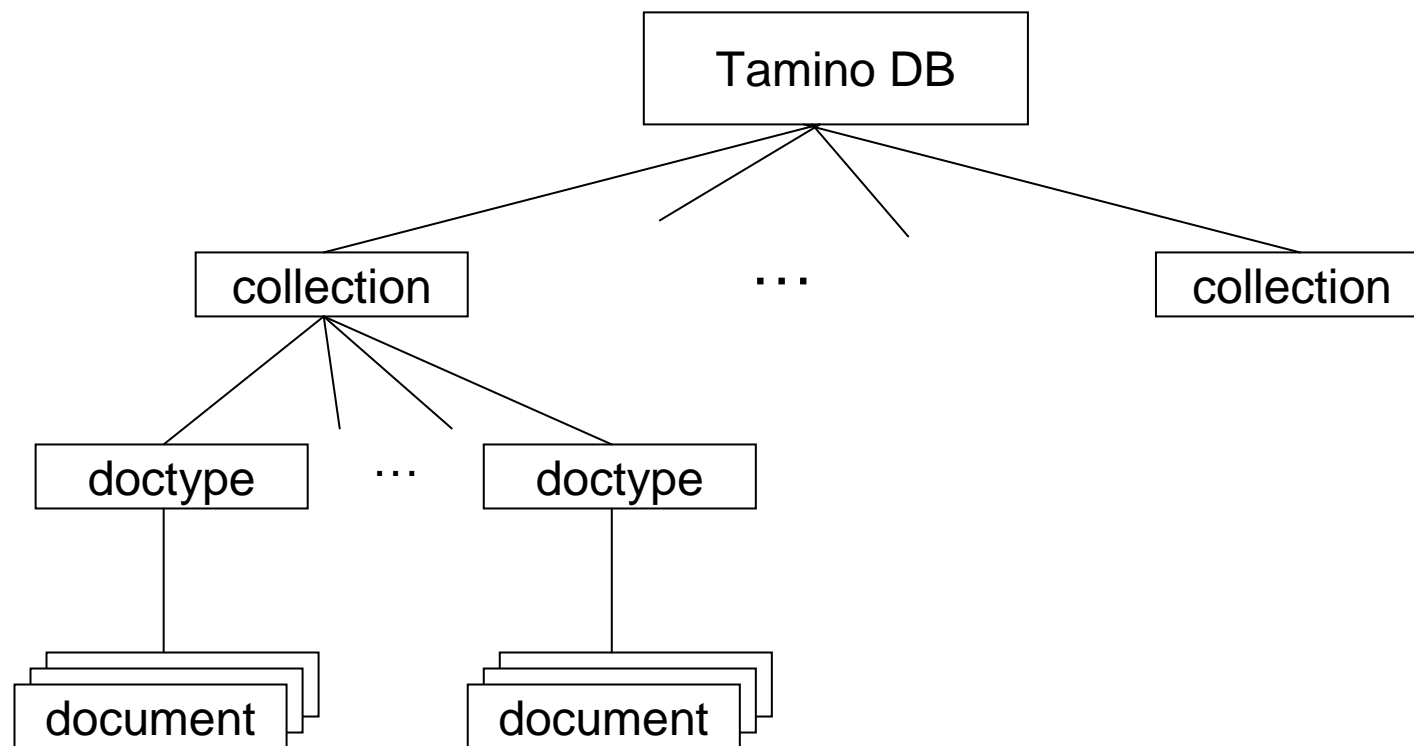
- **Collection**

- is a container to group documents together
 - has an associated set of schemas

- **Doctype**

- identifies one of the global elements declared in the schema as root element
 - within a collection, a document is stored as member of exactly one doctype

2. Tamino



2. Tamino

■ Schema:

- Tamino has an advanced schema editor, and conversion tools (DTD to XML Schema)
- Tamino validates incoming documents against their associated schema
- XML Schema: *processContents* attribute to control the behavior of the validation (strict, lax, skip)
- Additional Tamino option: *open content*

The document is validated against the schema. If information items are found that are not described in the schema, they are accepted nevertheless.

2. Tamino

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="City">
        <tsd:collection name="mycollection"></tsd:collection>
        <tsd:doctype name="City">
          <tsd:logical>
            <tsd:content>open</tsd:content>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="City"> ... </xs:element>
</xs:schema>
```

2. Tamino

- ❑ Element *xs:annotation*

- child of *xs:schema*, allows applications to add their annotations to an XML schema without compromising the interoperability of the schema
- Tamino uses this feature adding its information below the *xs:appinfo* child
 - ❑ element names are from a Tamino namespace (tsd)
 - ❑ name of the schema, name of the collection, name of the doctype. For each doctype, open or closed content can be specified

2. Tamino

■ Indexing

- ❑ Indexes are defined in a Tamino schema using `xs:appinfo`
- ❑ 3 types of indexes:
 - **standard index**: value-based index
 - ❑ fast lookup when searching for elements or attributes having certain values
 - ❑ ex: index on price attribute of book element (find all books with a price less than 50)
 - ❑ type-aware: indexes on numerical values support numerical order, on textual values support lexicographic order

2. Tamino

- **text index:**
 - ❑ for efficient text retrieval functionality
 - ❑ words contained in an element or attribute are indexed, s.t. search for words within the content of an element or attribute accelerated
 - ❑ Text indexes defined not only on leaf elements. It is possible to text-index a subtree
- **structure index:**
 - ❑ keeps info. about all paths in any instance of a specific doctype
 - ❑ accelerates query execution

3. Oracle9i

■ Oracle XML DB

- is a set of built-in storage and retrieval technologies for XML
- fully absorbs the W3C XML data model into Oracle9i database and provides new standard access methods for navigating and querying XML
- can be used to store, query, update, transform, or otherwise process XML, while at the same time providing SQL access to the same XML data

■ Several approaches to store an XML document in Oracle9i:

- as rows in one or more tables
 - DBMS not aware that it is managing XML content
- using a CLOB column (Character Large Object)
 - DBMS not aware that it is managing XML content
- using the **XMLType** datatype
 - 2 options: store in an XMLType column, or in an XMLType table
 - DBMS is aware that it is managing XML content, so it can use features to process XML content efficiently

3.1. Oracle9i – storing (no schema)

- **Storing a non schema-based XML document** (in an XMLType table):

- create the table

create table MyXMLtable of XMLTYPE;

- load the doc., convert it into an XMLType instance (using the XMLTYPE constructor) and insert it as a row into the table

- create a directory object:

- create a directory to put the xml files (in the OS)
- create directory MyXMLdir as '<location_of_xml_files>';

- create the getDocument function in PL/SQL (uses the directory object)

3.1. Oracle9i – storing (no schema)

```
create or replace function getDocument(filename varchar2)
  return clob  authid current_user is
  xfile bfile;
  xclob clob;
begin
  xfile := bfilename('MyXMLdir', filename);
  dbms_lob.open(xfile);
  dbms_lob.createtemporary(xclob,TRUE,dbms_lob.session);
  dbms_lob.loadfromfile(xclob,xfile,dbms_lob.getlength(xfile));
  dbms_lob.close(xfile);
  return xclob;
end;
/
show errors
```

3.1. Oracle9i – storing (no schema)

- ❑ insert statement (calls the getDocument function):
insert into MyXMLtable
values (XMLTYPE(getDocument('PurchaseOrder.xml')));

- ❑ see the stored document:

```
set long 20000      -- SQL*PLus environment variable  
select value(t) from MyXMLTable t;
```

- ❑ query the dictionary:

- ❑ select table_name from user_xml_tables;
- ❑ select * from all_directories;

3.1. Oracle9i – storing (no schema)

```
<?xml version="1.0" ?>
<PurchaseOrder>
  <Reference>ADAMS-
20011127121040988PST</Reference>
  <Actions>
    <Action>
      <User>SCOTT</User>
      <Date>2002-03-31</Date>
    </Action>
  </Actions>
  <Reject/>
  <Requestor>Julie P. Adams</Requestor>
  <User>ADAMS</User>
  <CostCenter>R20</CostCenter>
  <ShippingInstructions>
    <name>Julie P. Adams</name>
    <address>Redwood Shores, CA 94065</address>
    <telephone>650 506 7300</telephone>
  </ShippingInstructions>
  <SpecialInstructions>Ground</SpecialInstructions>
</PurchaseOrder>
```

```
<LinItems>
  <LinItem ItemNumber="1">
    <Description>The Ruling Class</Description>
    <Part Id="715515012423" UnitPrice="39.95"
      Quantity="2"/>
  </LinItem>
  <LinItem ItemNumber="2">
    <Description>Diabolique</Description>
    <Part Id="037429135020" UnitPrice="29.95"
      Quantity="3"/>
  </LinItem>
  <LinItem ItemNumber="3">
    <Description>8 1/2</Description>
    <Part Id="037429135624" UnitPrice="39.95"
      Quantity="4"/>
  </LinItem>
</LinItems>
</PurchaseOrder>
```

PurchaseOrder.xml

3.2. Oracle9i - querying

- Querying an XML document using XPath

- **existsNode** function:

- evaluates whether or not a given document contains a node which matches an XPath expression (returns 1/0)

- select existsNode(value(x), '/PurchaseOrder/Reference')
from MyXMLtable x;

- most commonly used in the WHERE clause of SELECT, UPDATE, or DELETE statements

- select count(*)
from MyXMLtable x
where existsNode(value(x), '/PurchaseOrder[User="ADAMS"]') = 1;

- delete from MyXMLtable x
where existsNode(value(x), '/PurchaseOrder[User="ADAMS"]') = 1;

3.2. Oracle9i - querying

- ❑ **extractValue** function:

- returns the content of a text element or attribute value associated with an XPath expression (returns a scalar data type)

- ❑ select extractValue(value(x), '/PurchaseOrder/Reference')
from MyXMLtable x;

-- This returns:

-- EXTRACTVALUE(VALUE(X),'/PURCHASEORDER/REFERENCE')

-- ADAMS-20011127121040988PST

- gives an error if the XPath expression returns:

- ❑ more than one node
 - ❑ a node which is neither a text element node nor an attribute node

3.2. Oracle9i - querying

- can be used in the WHERE clause of SELECT, UPDATE or DELETE statements. This makes possible to perform joins between XMLType tables and other relational tables
- select extractValue(value(x), '/PurchaseOrder/Reference')
from MyXMLtable x, EMP e
where extractValue(value(x), '/PurchaseOrder/User') = e.Ename
and e.EmpNo = 7876;

3.2. Oracle9i - querying

- ❑ **extract** function:

- is used when the XPath expression returns a collection of nodes
- returns an instance of XMLType, which can be a document or a document fragment (document with more than one root node)

- ❑ set long 20000

```
select extract(value(x),  
                '/PurchaseOrder/LineItems/LineItem/Description')  
from MyXMLtable x;
```

-- This returns:

-- EXTRACT(VALUE(X),'/PURCHASEORDER/LINEITEMS/LINEITEM/DESCRIPTION')

-- <Description>The Ruling Class</Description>

-- <Description>Diabolique</Description>

-- <Description>8 1/2</Description>

3.2. Oracle9i - querying

□ select extract(value(x), '/PurchaseOrder/LineItems/LineItem[1]')
from MyXMLtable x;

--This returns:

-- EXTRACT(VALUE(X),'/PURCHASEORDER/LINEITEMS/LINEITEM[1]')

-- -----

-- <LineItem ItemNumber="1">

-- <Description>The Ruling Class</Description>

-- <Part Id="715515012423" UnitPrice="39.95" Quantity="2"/>

-- </LineItem>

3.3. Oracle9i - updating

■ Updating an XML document

use **updateXML** function, to update a text node, an attribute value, or a subtree.

- the target is identified using an XPath expression, followed by a value expression. The pair <xpath, value_expr> can be repeated

- the value expression is:

- an XMLType instance if xpath returns an element node
- any scalar datatype if xpath returns a text node or attribute node

- update MyXMLtable t

set value(t) = updateXML(value(t),

 '/PurchaseOrder/Reference/text()'

 'MILLER-200203311200000000PST')

where existsNode(value(t),

 '/PurchaseOrder[Reference="ADAMS-20011127121040988PST"]') = 1;

3.3. Oracle9i - updating

```
□ update MyXMLtable t
  set value(t) = updateXML(value(t),
    '/PurchaseOrder/LineItems/LineItem[2]',
    xmltype('<LineItem ItemNumber="4">
      <Description>Andrei Rublev</Description>
      <Part Id="715515009928"
        UnitPrice="39.95" Quantity="2"/>
    </LineItem>'))
  where existsNode(value(t), '/PurchaseOrder[Reference="MILLER-
    200203311200000000PST"]') = 1;
```

3.3. Oracle9i - updating

- updateXML() and NULL values:
 - setting an attribute to NULL removes the attribute
 - setting an element to NULL makes the attributes and children of the element disappear, and the element becomes empty
 - update MyXMLtable t
 set value(t) = updateXML(value(t),
 '/PurchaseOrder/Reference', null)
 where existsNode(value(t), '/PurchaseOrder[Reference="ADAMS-20011127121040988PST"]') = 1;

3.4. Oracle9i – storing (schema-based)

- Oracle XML DB supports XML Schema in 2 ways:
 - automatic validation of instance documents when they are inserted or updated
 - definition of storage models: unstructured and structured storage
- **Storing a schema-based XML document** (in an XMLType table):
 - **register the XML schema**
 - using the method registerSchema in the PL/SQL package DBMS_XMLSCHEMA
 - provide the URL which will be used in the root element of the instance document, and the schema document

```
exec dbms_xmlschema.registerSchema(  
    'PurchaseOrder.xsd', getDocument('PurchaseOrder.xsd'));
```

3.4. Oracle9i – storing (schema-based)

- registerSchema causes Oracle XML DB to
 - parse and validate the XML schema
 - create a set of entries in Oracle Data Dictionary (for the schema)
 - create SQL object definitions based on complexTypes defined in the schema
- create the table
 - create an XMLType table which can only contain documents that conform to the specified schema (URL), with the given root element

```
create table XML_PURCHASEORDER of XMLTYPE  
  XMLSCHEMA "PurchaseOrder.xsd"  
  ELEMENT "PurchaseOrder";
```

```
create table XML_PURCHASEORDER of XMLTYPE  
  ELEMENT "PurchaseOrder.xsd#PurchaseOrder";
```

3.4. Oracle9i – storing (schema-based)

- query the dictionary:
 - ❑ select * from user_xml_schemas;
 - ❑ select * from user_xml_tables;
- ❑ **insert the document**
 - if the document is not valid, error.

insert into XML_PURCHASEORDER
values (xmltype(getDocument('PurchaseOrder.xml')));
 - Oracle XML DB performs minimal (some) validation.
Full instance validation can be enabled using:
 - ❑ **CHECK constraint:**

alter table XML_PURCHASEORDER
add constraint VALID_PurchaseOrder
check (XMLIsValid(sys_nc_rowinfo\$)=1);

3.4. Oracle9i – storing (schema-based)

- ❑ **BEFORE INSERT trigger:**

```
create trigger VALIDATE_PURCHASEORDER  
before insert on XML_PURCHASEORDER  
for each row  
declare  
    XMLDATA xmltype;  
begin  
    XMLDATA := :new.sys_nc_rowinfo$;  
    xmltype.schemavalidate(XMLDATA);  
end;  
/
```

- ❑ **CHECK constraint:** simpler to code, indicates whether or not the instance doc. is valid, does not give info. about why a doc. is invalid (unlike the trigger)

3.4. Oracle9i – storing (schema-based)

■ Structured or Unstructured Storage

- ❑ decide to store an XMLType column/table using unstructured or structured storage

- ❑ **unstructured storage:**

- ❑ document stored as a whole document (like a file)
 - ❑ content stored using the CLOB datatype

- structured storage:**

- ❑ document ‘broken up’ (decomposed) into object-relational tables
 - ❑ content stored as collection of SQL objects

- ❑ if no schema: only unstructured storage

if schema-based: choice (by default structured storage,
otherwise use STORE AS clause)

3.4. Oracle9i – storing (schema-based)

```
create table XML_PURCHASEORDER of XMLTYPE  
  XMLTYPE store as CLOB  
  XMLSCHEMA "PurchaseOrder.xsd"  
  ELEMENT "PurchaseOrder";
```

- ❑ Changing XMLType storage from structured to CLOB, and vice-versa
 - is possible using database IMPORT and EXPORT
 - does not affect your application code

3.4. Oracle9i – storing (schema-based)

	Unstructured storage	Structured storage
Performance: Storage & retrieval speed	Higher rates of ingestion & retrieval (avoids overhead associated with parsing and recomposing during storage and retrieval operations).	Slight overhead.
Performance: Operation speed	Slower than for structured storage.	XPath expressions are translated to SQL statements (query rewrite) that operate directly on the SQL objects. Significant performance improvements compared to unstructured storage.
Memory usage	Oracle XML DB parses the entire XML doc. and loads it in an in-memory DOM structure before any validation, or XPath operation.	Minimizes memory usage and optimizes performance of DOM-based operations.

3.4. Oracle9i – storing (schema-based)

	Unstructured storage	Structured storage
Update processing	Any update operation (updateXML) results in the entire CLOB being re-written.	Can update individual elements, attributes without rewriting the entire document. updateXML operation rewritten to SQL UPDATE statement.
Indexing	Can use B*Tree indexes (based on functional evaluation of XPath expressions), and Oracle Text inverted list indexes.	Can use B*Tree indexes and Oracle Text inverted list indexes.
Space needed	Can be large.	Reduces storage space (not necessary to store tag names).
Data integrity	--	Possible to define database integrity constraints.

3.4. Oracle9i – storing (schema-based)

	Unstructured storage	Structured storage
Tuning	None.	You can annotate XML schema, for fine grain control over SQL objects (e.g. how collections are managed, partitioning of tables).
Flexibility	Appropriate choice when the XML doc. contains highly variable content. Very flexible when schemas change.	Limited flexibility for schema changes.

3.4. Oracle9i – storing (schema-based)

■ DOM fidelity

- is preserved if the DOM generated from the stored representation of the XML document is identical to a DOM generated from the original document (no information loss)
- XML document contains additional info.
 - explicit, e.g. comments
 - implicit, e.g. ordering of child elements
- Oracle XML DB can preserve DOM fidelity with structured storage
 - adds a system binary attribute *SYS_XDBPD\$* to each created object type (stores additional info.)
 - if DOM fidelity not required, suppress *SYS_XDBPD\$* in the schema definition by setting the attribute *maintainDOM="false"*

3.4. Oracle9i – storing (schema-based)

- Structured storage makes it possible to define integrity constraints, e.g. unique and referential constraints
 - ❑ alter table XML_PURCHASEORDER
add constraint REFERENCE_IS_UNIQUE
unique (extractValue('/PurchaseOrder/Reference'));
 - ❑ alter table XML_PURCHASEORDER
add constraint USER_IS_VALID
foreign key extractValue('/PurchaseOrder/User') references
EMP(ENAME);
- Use CLOB storage for XMLType in the following cases:
 - ❑ you need to store XML as a whole document in the DB, and retrieve it as a whole document
 - ❑ you do not need to perform piece-wise updates on XML documents

3.5. Oracle9i – mapping

- Schema registration causes the creation of SQL object types

XML schema po.xsd:

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="PurchaseOrderType">
    <xs:sequence>
      <xs:element name="PONum" type="xs:decimal"/>
      <xs:element name="Company">
        <xs:simpleType>
          <xs:restriction base="string"><xs:maxLength value="50"/></xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Item" maxOccurs="1000">
        ...
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="PurchaseOrder" type="PurchaseOrderType"/>
</xs:schema>
```

3.5. Oracle9i – mapping

SQL object types:

```
create type "Itemxxx_T" as object
(
  ...
);
create type "Itemxxx_COLL" as varray(1000) of "Itemxxx_T";
create type "PurchaseOrderTypexxx_T" as object
(
  SYS_XDBPD$ XDB.XDB$RAW_LIST_T
  PONum number,
  Company varchar2(50),
  Item Itemxxx_COLL
);
```

3.5. Oracle9i – mapping

Query the dictionary:

- select table_name from user_object_tables;

```
TABLE_NAME
-----
PurchaseOrder579_TAB
XML_PURCHASEORDER
```

- desc xml_purchaseorder

```
Name                                     Null?  Type
-----
TABLE of SYS.XMLTYPE(XMLSchema "po.xsd" Element "PurchaseOrder") STORAGE Object-relational TYPE
"PurchaseOrderType576_T"
```

- desc "PurchaseOrderType576_T"

```
"PurchaseOrderType576_T" is NOT FINAL
Name                                     Null?  Type
-----
SYS_XDBPD$                             XDB.XDB$RAW_LIST_T
PONum                                  NUMBER
Company                               VARCHAR2(50)
Item                                  Item578_COLL
```

- select * from user_types;

3.5. Oracle9i – mapping

XML document po.xml:

```
<?xml version="1.0" ?>
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="po.xsd">
  <PONum>1001</PONum>
  <Company>Oracle Corp</Company>
  <Item>
    <Part>9i Doc Set</Part>
    <Price>2550</Price>
  </Item>
  <Item>
    <Part>8i Doc Set</Part>
    <Price>1050</Price>
  </Item>
</PurchaseOrder>
```

select p.xmldata."Company" from xml_purchaseorder p;

XMLDATA.Company

Oracle Corp

select p.xmldata.* from xml_purchaseorder p;

XMLDATA(SYS_XDBPD\$, PONum, Company, Item(SYS_XDBPD\$, Part, Price))

PurchaseOrderType576_T(XDB\$RAW_LIST_T('010...364'), 1001, 'Oracle Corp', Item578_COLL(Item577_T(XDB\$RAW_L
IST_T('010...000'), '9i Doc Set', 2550), Item577_T(XDB\$RAW_LIST_T('010...000'), '8i Doc Set', 1050)))

3.5. Oracle9i – mapping

- Oracle XML DB uses system-generated names for SQL object types.

To specify specific names, include the attributes `SQLName` and `SQLType` in the XML schema definition prior to registering the schema.

- **SQLName**: specify the name of the attribute within the SQL type that maps to this XML element
- **SQLType**: specify the name of the SQL type that maps to this XML element
- these attributes specified within attribute and element declarations
- these attributes belong to the namespace <http://xmlns.oracle.com/xdb>

```
<element name="Item" maxOccurs="1000"
      xdb:SQLName="ForItem" xdb:SQLType="Item_T">
```

3.5. Oracle9i – mapping

- Mapping of types:
 - ❑ default mapping using DBMS_XMLSCHEMA
 - ❑ can use your own mapping
 - specify the URL for the XML schema used for the mapping
(in the create table statement)

- Mapping attributes to SQL
 - ❑ SQL type and info. (length and precision) derived from the simpleType on which the attribute is based

3.5. Oracle9i – mapping

■ Mapping simpleType to SQL

- XML primitive type is mapped to the closest SQL datatype, e.g.
 - decimal, positiveInteger, float, mapped to NUMBER
 - string mapped to VARCHAR2(n) if n specified (n < 4000), else VARCHAR2(4000)
 - time mapped to TIMESTAMP, date mapped to DATE
 - boolean mapped to RAW(1)
RAW(size): Raw binary data of length size bytes. Maximum size is 2000 bytes.
- XML enumeration type mapped to an object type with a single RAW(n) attribute (n determined by the possible values)

3.5. Oracle9i – mapping

- Mapping complexType to SQL
 - a complexType is mapped to an object type containing attributes for:
 - each of the XML attributes
 - each of the subelements
 - the datatype of the object attribute is determined by the simpleType or complexType of the subelement
 - a subelement declared with *maxOccurs* > 1 is mapped to a collection SQL attribute (VARRAY (default) or nested table if *maintainOrder="false"*)

3.5. Oracle9i – mapping

- ❑ handling inheritance

- for a complexType extending another complexType:
 - ❑ its SQL type is specified as subtype of the other SQL type
 - ❑ all complexTypes can be extended and restricted by other types (unless *final* attribute is specified), so all SQL object types are created as *not final* types
 - ❑ only additional attributes and subelements declared in the sub-complexType are added as attributes to the sub-object type
- for a complexType restricting another complexType:
 - ❑ its SQL type is specified as subtype of the other SQL type (without additional attributes)
 - ❑ SQL does not support restriction of object types

3.5. Oracle9i – mapping

- ```
<xs:complexType name="Address" xdb:SQLType="ADDR_T">
 <xs:sequence>
 <xs:element name="street" type="xs:string"/>
 <xs:element name="city" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="USAddress" xdb:SQLType="USADDR_T">
 <xs:complexContent>
 <xs:extension base="Address">
 <xs:sequence>
 <xs:element name="zip" type="xs:string"/>
 </xs:sequence>
 </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

```
create type ADDR_T as object (
 SYS_XDBPD$ XDB.XDB$RAW_LIST_T,
 street varchar2(4000),
 city varchar2(4000)) not final;
create type USADDR_T under ADDR_T (
 zip varchar2(4000)) not final;
```

---

## 3.5. Oracle9i – mapping

- ❑ *complexType:simpleContent* (text-only with attributes) is mapped to SQL type with:
  - attributes corresponding to the XML attributes
  - an extra SYS\_XDBBODY attribute of type VARCHAR2(4000) corresponding to the body value
- ❑ *complexType:Any* is mapped to VARCHAR2 attribute

---

# References

- Database System Concepts, by A. Silberschatz, H. Korth, S. Sudarshan, 4th edition, McGraw-Hill, 2001
- XML Data Management, by A. Chaudhri, A. Rashid, R. Zicari (eds), Addison Wesley, 2003
- Oracle9i XML Database Developer's Guide - Oracle XML DB, Release 2 (9.2), March 2002