SQLX
Bringing XML and SQL Together
Jim Melton
Oracle Corp.
Part I — Getting Started

<table>
<thead>
<tr>
<th>Part I</th>
<th>Part II</th>
<th>Part III</th>
<th>Part IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Started</td>
<td>Mapping Concepts</td>
<td>SQL + XML</td>
<td>Finishing Up</td>
</tr>
</tbody>
</table>
SQLX Group

- A group of developers & standardizers with expertise in both XML and SQL
- Have the support of their employers (Bentley Transportation, Fidelity Investments, FileTek, IBM, Informix, Microsoft, NCR/Teradata, Oracle, PRC, Progress, SAIC, Sybase, Tandem, & still growing)
- *Not* an official consortium
- Operates like the SQLJ Group: Unofficially
SQLX Group

☑ Goals:
  • Develop complete specifications
  • Submit to other standards organizations (e.g., ANSI, ISO, perhaps even W3C)

☑ Non-goal: Do not self-publish “standards”

☑ Do not compete with W3C — complement work of W3C, ANSI, ISO, etc.
SQLX Group Activities

- Telecons at least monthly
- Face-to-Face meetings at least bi-monthly
- Continuous electronic discussions (email)
- Develop “base document(s)” to respond to new subproject in ISO for new part of SQL standard
ISO SQL Subproject

- Expected to be ISO/IEC 9075-14, SQL/XML
- Project proposed in September 2000
- Candidate areas of technology:
  - Representation of XML data in SQL and *vice versa*
  - Mapping between XML schemas and SQL schemas
  - Representation of SQL schemas in XML
  - Representation of SQL actions in XML
  - SQL-friendly syntax for querying XML
Current Status

- Candidate base document prepared
- Formal Working Draft published and enhanced
- First ISO ballot (FCD) expected to begin in April, 2002
Current Status

- Current contents comprise infrastructure — mappings of:
  - SQL identifiers to & from XML QNames
  - SQL types & values onto XML schema types & values
  - SQL tables onto XML schema types

- In Development:
  - Publishing functions (SQL data expressed as XML)
  - Possibly an XMLType for SQL
  - Possibly annotated XML schemas
XML Concepts

- “Tagged text” — Elements & attributes
- Tree-structured data
- Semi-structured data
  - Self-describing
  - Possible irregular structure
- Everything is textual — no binary data
- XML: hacemos sus datos más gordo
XML Concepts: Tagged Text

<book>
  <title>Querying XML</title>
  <author>Jim Melton</author>
  <author>Barbara Edelberg</author>
  <publ>MKP</publ>
  <date>2002</date>
</book>
XML Concepts: Tagged Text

- Everything is either an element, an attribute, or a value of one of those
- Inspired by “ordinary text”
  `<p>I am <b>not</b> going!</p>`
- But most useful for “structured text”
  `<step required="Y">
    <task>Insert</task>
    <part>screw</part> and
    <task>tighten</task>.
  </step>`
XML Concepts: Tree Structure

- Every element has exactly one "parent"...
- ...except "root", which has no parent
- Tree walking algorithms useful (breadth-first vs depth-first)
- Severe complication: attributes
  <book year="2001">
    <title>Querying XML</title>
  </book>
- What are the children of <book>?
XML Concepts: Tree Structure

```xml
<books>
  <book ISBN="1-67890-123-4">
    <author>...</author>
    <title>...</title>
  </book>
  <book ISBN="1-23456-789-0">
    <author>...</author>
    <title>...</title>
  </book>
</books>
```
XML Concepts: Tree Structure

- Simple...deceptively so:
  <step required="Y">
    <task>Insert</task>
    <part>screw</part> and
    <task>tighten</task>.
  </step>

- Mixed content: <step> has some children that are elements (<part> & <task>) and some that are text (and & .).

- Some children are element nodes, one is an attribute node, some are text nodes.
XML Concepts: Tree Structure

- Every XML document has at least two representations:
  - Serialized — Meaning textual document form
  - Abstracted — Several possibilities
    - Infoset
    - PSVI
    - Query Data Model
    - DOM
    - Possibly others
The Information Set (Infoset)

- An XML data model
- Tree-structured
- Each node is a data structure representing a “piece” of the document
- Serves as the paradigm on which all W3C specs are built
Post-Schema Validation Infoset (PSVI)

- Builds upon Infoset
- Adds type information
  - Simple types
    - Built-in — both primitive and derived
    - User-defined — derived (by definition)
  - Complex types — all user-defined
XPath 2.0 and XQuery 1.0

Data Model

- Builds upon PSVI
- But is a superset of a subset — not all information in PSVI is used, but additional information is required
  - Loses most type facet information
  - Loses most constraints (e.g., ID/IDREF)
  - Adds sequences and other comparable information
Document Object Model (DOM)

- Not *truly* an object model — an API that implies an object model (a type of data model)
- Significantly different from Infoset, PSVI, and XPath/XQuery Data Model
- Emphasis is on access from application programs, not on modeling for specification purposes
XML Concepts: Semistructured

- Metadata not required, often not present
- Data includes identifying information
- Missing: semantics of identifying information!
- Irrelevant and unavailable information usually omitted
- More art than science?
XML Concepts: Textual Only

- What is type of 2002?
  `<book year="2002">…</book>`
- Without metadata, it is “just a string”
- System might infer that it is a number and permit numeric operations
- But we “know” that it’s part of a date…
- Use of Unicode solves some problems, may cause others (such as size of data)
XML Concepts: Metadata

- Well-formed documents
  - Properly nested structure
    `<b>do</b> <i>not</i> <b>go</b>` is prohibited
  - Balanced element tags
    `<doc><title>Africa</doc>` is prohibited
  - No data type information
  - Well-formed documents *do have* an Infoset
XML Concepts: Metadata

- **DTD** — Document Type Definition
  - Allows control of structure
  - Supports a few constraints (e.g., ID/IDREF)
  - Still no data type information
  - Documents that are valid against a DTD have an Infoset
XML Concepts: Metadata

- **XML Schema**
  - Fine control of document structure
  - Additional constraints, including on values
  - Significant data type information
  - Verbose and complex
  - Market competition: RELAX NG, Schematron, perhaps others
  - Major corporations support XML Schema
  - Documents valid against Schema have a PSVI
SQL Concepts

- Native format for data
- Completely regular structure
- Totally dependent on metadata

SQL: we make your data flat
XML: we make your data fat
SQLX: Bringing XML and SQL Together

Jim Melton

All data — even metadata — represented in tabular form:

<table>
<thead>
<tr>
<th>BOOKS</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLE</td>
<td>PUBLISHER</td>
<td>DATE</td>
</tr>
<tr>
<td>Querying XML</td>
<td>MKP</td>
<td>2002</td>
</tr>
<tr>
<td>SQL:1999</td>
<td>MKP</td>
<td>2001</td>
</tr>
</tbody>
</table>
SQL Concepts: Regular Structure

- Missing information cannot be completely omitted!

<table>
<thead>
<tr>
<th>BOOKS</th>
<th>PUBLISHER</th>
<th>DATE</th>
<th>etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Querying XML</td>
<td>MKP</td>
<td>2002</td>
<td></td>
</tr>
<tr>
<td>SQL:2003</td>
<td>MKP</td>
<td>(null)</td>
<td></td>
</tr>
</tbody>
</table>
SQL Concepts: Native Formats

- Numbers represented as SMALLINT, INTEGER, DECIMAL, FLOAT, etc.
- Dates and times represented as DATE, TIME, TIMESTAMP, INTERVAL
- Character strings *based on* Unicode; may be fixed-length or varying
- Boolean is BOOLEAN
- Binary string (BLOB) types, too…
SQL Concepts: Metadata

- Every SQL implementation — not just the standard — provides descriptions of tables and columns in each database
- Names, data types, constraints, etc.
- SQL provides closure — every statement operates on a table and produces a table, so intermediate results have implicit metadata
“Come Together…”
(The Beatles)

- SQL Identifiers
  - Regular: Letters, digits, underscore (plus ideographs, syllables)
  - Delimited: Any character at all…within double quotes
  - Limited in length (18, 30/31/32, 128 characters)

- XML QNames and NCNames
  - Strict limitations on characters
  - No limit on length
“Come Together…”

SQL Types

- SMALLINT, INTEGER, NUMERIC, DECIMAL, FLOAT, REAL, DOUBLE PRECISION
- CHARACTER, VARCHAR, CLOB
- BLOB
- DATE, TIME, TIMESTAMP, INTERVAL
- BOOLEAN
“Come Together…”

- **XML Types**
  - Well-formed and DTDs: strings, inference of numbers
  - Schema:
    - ✓ Numbers: decimal, float, double; 13 additional types derived from decimal
    - ✓ Strings: string; 12 additional derived types
    - ✓ Boolean: boolean
    - ✓ Datetime: duration and 8 more primitive types
    - ✓ Other: 5 additional primitive types
“Come Together...”

The Problem:
How can SQL data be “published” in XML so that it retains the maximum amount of information, including identifiers, proper data type knowledge, and so forth?
Part II — Mapping Concepts

Part I
Getting Started

Part II
Mapping Concepts

Part III
SQL + XML

Part IV
Finishing Up
Mapping Concepts

- Mapping SQL Identifiers onto XML QNames
- Mapping XML QNames onto SQL Identifiers
- Mapping SQL Data Types onto XML Schema Types
- Mapping XML Schema Types onto SQL Data Types
- Mapping SQL Tables onto XML Documents
Mapping SQL identifiers onto XML QNames

- SQL’s “regular identifiers” are no problem:
  
  `employee` ⇒ `EMPLOYEE`
  
  (effectively up-cased by SQL)
  
  `<EMPLOYEE>...<EMPLOYEE>`

- SQL’s “delimited identifiers” more interesting:

  "Employee" ⇒ Employee
  
  `<Employee>...<Employee>`

  "Emp/Name® & ID;®" ⇒ ???
Mapping SQL identifiers onto XML QNames

- The key: Represent characters that are invalid in Qnames using “escape” notation: Ω ⇒ _x03A9_ (the Unicode value)
- "Emp/NameΩ & ID;®" ⇒ Emp_x002F_Name_x03A9__x0020__x0026__x0020_ID_x003B__x00AE_
- Fully reversible, of course
- Length apparently not a problem
Mapping SQL identifiers onto XML QNames

Q Why choose `_xnnnnn_`? Why not `\nnnnn` or `\xnnnnn` or `\unnnnn`?

Q Three reasons:

- Possible conflict with host languages (think about Embedded SQL)
- Most acceptable compromise to enough people
- Trailing underscore needed to distinguish 4-hexit from 6-hexit values
Mapping SQL identifiers onto XML QNames

- Names including colons (:) — might be used to indicate namespace, so cannot always escape
  - “Partially escaped” ⇒ Escape only *leading* colons
  - “Fully escaped” ⇒ Escape *all* colons
- Names beginning with “xml” (any case) — reserved by XML 1.0 spec
  - “Partially escaped” ⇒ Leave such names alone
  - “Fully escaped” ⇒ Prefix names with \_xFFFF\_ (one of Unicode’s *not-a-character* codepoints)
Mapping SQL identifiers onto XML QNames

- Names that include the sequence of characters “\_x” — nothing prevents users from writing the following SQL identifier: "Emp\_Name"

- Solution: Escape the “\_” in such cases: Emp\_Name
Mapping SQL identifiers onto XML QNames

- (Fully escaped) "Salary:FY2000" ⇒ Salary_x003A_FY2000
- "Work@home" ⇒ Work_x0040_home
- "Work_x0040_home" ⇒ Work_x005F_x0040_home
- "Work_home" ⇒ Work_home
- "@@" ⇒ _x0040__x0040_ /*Note double “_”*/
- xmlText ⇒ _xFFFF_FXMLTEXT
Mapping SQL identifiers onto XML QNames

- Unicode values
  - Basic Multilingual Plane (BMP) — always 16 bits, or 4 hexits
  - Beyond BMP — no more than 21 bits, could be done using 5 hexits, but 6 hexits chosen
  - Does *not* use two consecutive surrogate values — that is only used in UTF-16
Mapping XML QNames onto SQL identifiers

- Reversing the mapping…
  - Convert all sequences “\_xNNNN\_” and “\_xNNNNNNN\_” to the corresponding Unicode character
  - Current debate about 6-hexit versus 8-hexit
  - Repeat: Does not use Unicode surrogates!
  - Put double quotes around the result to make it into an SQL delimited identifier
Mapping XML QNames onto SQL identifiers

- Problems:
  - What about invalid escape values (\xFEFF does not correspond to any Unicode character) — map to implementation-defined substitute

- Choices:
  - Delimited identifier is sometimes overkill
  - Must map to existing SQL character set SQL_TEXT, or exception is raised
Data Type Mapping
Mapping SQL Data Types Onto XML Schema Types

- XML Namespaces
  - Used to “protect” names (e.g., tag & attribute)
  - Identifier ⇒ URI
  - XML Schema:
    - `<xsd>` ⇒ http://www.w3.org/2000/08/XMLSchema
    - `<xsi>` ⇒ http://www.w3.org/1999/XMLSchema-instance
  - SQLX:
Mapping SQL Data Types Onto XML Schema Types

- No “SQL-centric” mappings
  - Would be of little value since XML does not have “type templates” (perhaps in the future)
  - SQL’s `CHARACTER(10)` ⇒ XML’s `sqlxml:CHAR`, but no way to specify length in data type name (no parameterized types)
  - Use XML’s `xsd:string` and `xsd:length` facet, which is XML-centric
Mapping SQL Data Types Onto XML Schema Types

- **Philosophy:** Map each SQL predefined type to its closest analog in XML Schema, using XML Schema facets to capture as much as possible of the SQL type’s semantics.

- **But make it possible to determine the initial SQL type from the XML Schema type — How?**
Mapping SQL Data Types Onto XML Schema Types

- Annotations
  - Every aspect of SQL type system has an annotation, even if redundant with non-annotation information present in XML type

<table>
<thead>
<tr>
<th>name</th>
<th>length</th>
<th>maxLength</th>
</tr>
</thead>
<tbody>
<tr>
<td>characterSetName</td>
<td>collation</td>
<td>precision</td>
</tr>
<tr>
<td>maxExponent</td>
<td>minExponent</td>
<td>scale</td>
</tr>
<tr>
<td>leadingPrecision</td>
<td>userPrecision</td>
<td></td>
</tr>
</tbody>
</table>
Mapping SQL Data Types Onto XML Schema Types

- Character String Types
  - CHARACTER, CHAR
  - CHARACTER VARYING, VARCHAR
  - CHARACTER LARGE OBJECT, CLOB
  - NATIONAL variants
  - Map to `xsd:string` using these annotations: `name`, `characterSetName`, `collation`, and either `length` or `maxLength` (as appropriate)
Mapping SQL Data Types Onto XML Schema Types

- Character String Types
  - `CHAR(10)` CHARACTER SET LATIN1 COLLATION ESPAÑOL
    - `<xsd:simpleType>`
      - `<xsd:restriction base="xsd:string">`
        - `<xsd:length value="10"/>
        - `<xsd:annotation>`
          - `<sqlxml:sqltype name="CHAR" length="10" characterSetName="LATIN1" collation="ESPAÑOL"/>`
          - `<xsd:restriction>`
            - `<xsd:annotation>`
              - `<xsd:restriction>`
                - `<xsd:annotation>`
                  - `<xsd:restriction>`
Mapping SQL Data Types Onto XML Schema Types

- Binary String Types
  - BLOB(1000)
  - `<xsd:simpleType>`
  - `<xsd:restriction base="sqlxml:binaryhex">`
  - `<xsd:maxLength value="2000"/>
  - `<xsd:annotation>`
  - `<sqlxml:sqltype name="BLOB" maxLength="1000"/>`
  - `</xsd:annotation>`
  - `</xsd:restriction>`
  - `</xsd:simpleType>`
Mapping SQL Data Types Onto XML Schema Types

- NUMERIC and DECIMAL
- DECIMAL(8, 2) where implementation uses 9

```
• <xsd:simpleType>
  <xsd:restriction base="xsd:decimal">
    <xsd:precision value="9"/>
    <xsd:scale value="2"/>
    <xsd:annotation>
      <sqlxml:sqltype name="DECIMAL"
        userPrecision="8" scale="2"/>
    </xsd:annotation>
  </xsd:restriction>
</xsd:simpleType>
```
Mapping SQL Data Types Onto XML Schema Types

- **INTEGER** and **SMALLINT**
- **SMALLINT** (implementation uses 16-bit two's-complement binary integers)

```xml
<xsd:simpleType>
  <xsd:restriction base="xsd:integer">
    <xsd:maxInclusive value="32767"/>
    <xsd:minInclusive value="-32768"/>
    <xsd:annotation>
      <sqlxml:sqtype name="SMALLINT"/>
    </xsd:annotation>
  </xsd:restriction>
</xsd:simpleType>
```
Mapping SQL Data Types Onto XML Schema Types

- `FLOAT(p)`
- `REAL`
- `DOUBLE PRECISION`

- `REAL` (if implementation uses IEEE floating point)

```
• <xsd:simpleType>
  <xsd:restriction base="xsd:float">
    <xsd:annotation>
      <sqlxml:sqltype name="REAL" precision="24" minExponent="-149" maxExponent="104"/>
    </xsd:annotation>
  </xsd:restriction>
</xsd:simpleType>
```
Mapping SQL Data Types Onto XML Schema Types

 BOOLEAN

  • <xsd:simpleType>
  <xsd:restriction base="xsd:boolean"/>
  <xsd:annotation>
    <sqlxml:sqltype name="BOOLEAN"/>
  </xsd:annotation>
  </xsd:restriction>
  </xsd:simpleType>
Mapping SQL Data Types Onto XML Schema Types

**DATE**

- SQL omits time part, XML has it; use the pattern facet to prohibit the time part of XML Schema date type

- `<xsd:simpleType>
  <xsd:restriction base="xsd:date">
    <xsd:pattern value="\p{Nd}{4}-\p{Nd}{2}-\p{Nd}{2}"/>
    <xsd:annotation>
      <sqlxml:sqltype name="DATE"/>
    </xsd:annotation>
  </xsd:restriction>
</xsd:simpleType>"
Mapping SQL Data Types Onto XML Schema Types

- **TIME and TIME WITH TIME ZONE**
- **TIME (2)**
  - `<xsd:simpleType>`
    - `<xsd:restriction base="xsd:time">`
    - `<xsd:pattern value="\p{Nd}{2}:\p{Nd}{2}:\p{Nd}{2}\p{Nd}{2}"/>
    - `<xsd:annotation>`
      - `<sqlxml:sqltype name="TIME" scale="2"/>

```xml
<welcome>
  <from>Jim Melton & Oracle Corp.</from>
  <subject>SQLX: Bringing XML and SQL Together</subject>
</welcome>
```
Mapping SQL Data Types Onto XML Schema Types

- INTERVAL YEAR(4) TO MONTH
  - `<xsd:simpleType>
    `<xsd:restriction base="xsd:timeDuration">
      `<xsd:pattern value="-?P\p{Nd}{1,4}Y\p{Nd}{2}M"/>
      `<xsd:annotation>
        `<sqlxml:sqltype name="INTERVAL YEAR TO MONTH" leadingPrecision="4"/>
        `</xsd:annotation>
    `<xsd:restriction>
    `<xsd:simpleType>`
Mapping SQL Data Types Onto XML Schema Types

- Not included in this presentation:
  - Distinct types
  - Structured types
  - Row types
  - Array types
  - Reference types

- They are *In Progress*, but may not make first version
Mapping XML Schema Types Onto SQL Data Types

- In Progress, but several participants have recently expressed the opinion that this is not required for SQL/XML
- To Be Determined!
Value and Table Mapping
Mapping SQL values onto XML values

- Dictated by SQL data type mappings onto XML schema types
- Minor exceptions
  - Numeric types — should a decimal point be required, allowed, or prohibited if no fraction digits?
  - SQLX response — prohibit for compatibility with SQL CAST rules (“shortest literal”)

Copyright © 2001 Oracle Corp.
Mapping SQL values onto XML values

- One more potential issue
  - In a character string value, should less-than signs (<), ampersands (&), and other “special characters” be mapped directly?
  - Or should they be mapped to the corresponding “entity” representation (e.g., “&lt;” or “&amp;”)
  - SQLX response: still under consideration, may be supported through “parameterization”
Mapping SQL Tables Onto XML Schema Types

- One dimension has (at least!) two approaches
  - Map SQL columns to XML *elements*
  - Map SQL columns to XML *attributes*

- Another dimension has two approaches
  - Each row placed in element with table name
  - Each row placed in `<row>` element

- This is somewhat speculative at present—discussions still in progress!
Mapping SQL Tables Onto XML Schema Types

- Mapping SQL Columns as XML elements
  - CREATE TABLE Employees (EmpName CHARACTER(20), Salary NUMERIC(8,2), HireDate DATE)
  - <Employees>
    <EmpName>...</EmpName>
    <Salary>...</Salary>
    <HireDate>...</HireDate>
  </Employees>
  - Using XML Schema, each element has well-defined data type
Mapping SQL Tables Onto XML Schema Types

- Mapping SQL Columns as XML *attributes*
  - `CREATE TABLE Employees (`
    - `EmpName    CHARACTER(20),`
    - `Salary     NUMERIC(8,2),`
    - `HireDate   DATE )`
  - `<Employees EmpName="..." Salary="..." HireDate="..." /></Employees>

- Data types of attributes can still be well-defined, but cannot be complex types — even with XML Schema!
Mapping SQL Tables Onto XML Schema Types

• Mapping columns to elements:
  
  ```xml
  <xsd:complexType name="EMPLOYEES">
    <xsd:sequence>
      <xsd:element name="EMPNAME">
        <xsd:simpleType>
          <xsd:restriction base="string">
            <xsd:length value="20"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  ```

  ...
Mapping SQL Tables Onto XML Schema Types

```xml
<xsd:element name="SALARY">
  <xsd:simpleType>
    <xsd:restriction base="decimal">
      <xsd:precision value="8"/>
      <xsd:scale value="2"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Mapping SQL TablesOnto
XML Schema Types

```xml
<xsd:element name="HIREDATE">
  <xsd:simpleType>
    <xsd:restriction base="date">
      <xsd:pattern value="\p{Nd}{4}-\p{Nd}{2}-\p{Nd}{2}"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```
Mapping SQL Tables Onto XML Schema Types

```xml
<xsd:element name="EMPLOYEES">
  <xsd:sequence>
    <xsd:element name="EMPNAME" type="EMPLOYEES"/>
  </xsd:sequence>
</xsd:element>
```
Mapping SQL Tables Onto XML Schema Types

- Mapping columns to attributes:
  ```xml
  <xsd:element name="EMPLOYEES">
    <xsd:complexType>
      <xsd:attribute name="EMPNAME" type="string" use="required"/>
      ...
    </xsd:complexType>
  </xsd:element>
  ```
XML Schema Types

Mapping SQL Tables Onto XML Schema Types

• ...

<xsd:attribute name="SALARY" type="decimal" use="required"/>

<xsd:attribute name="HIREDATE" type="date" use="required"/>

SQLX: Bringing XML and SQL Together

Jim Melton

Copyright © 2001 Oracle Corp.
Mapping SQL Tables Onto XML Schema Types

- More cumbersome to give attributes a type other than a “built-in” XML Schema type
- Must use additional types that are referenced in attribute definition
- Impossible to support SQL’s ROW, ARRAY, or MULTISET types in attributes
Mapping SQL Tables Onto XML Documents

- General philosophy:
  - Root element for an SQL schema
  - One subordinate element per table
  - Subordinate elements for each column

- Unresolved issue: Should rows of tables be specially identified? Two approaches…
Mapping SQL Tables Onto XML Documents

- **Approach 1**: Identify all tables, identify each row:

```xml
<myschema>
  <employees>
    <row>
      <name>Piattini</name>
      <salary>1000000</salary>
      <hiredate>2000-01-01</hiredate>
    </row>
  </employees>
</myschema>
```
Mapping SQL Tables Onto XML Documents

<row>
  <name>Marcos</name>
  <salary>1500000</salary>
  <hiredate>1999-01-01</hiredate>
</row>

...</row>

</employees>

SQLX: Bringing XML and SQL Together
Jim Melton

Copyright © 2001 Oracle Corp.
Mapping SQL Tables Onto XML Documents

- If there are no departments, then the `<departments/>` element is empty — but present!
Mapping SQL Tables Onto XML Documents

- Approach 2: Identify all rows from one table with a tag associated with the table

```xml
<myschema>
  <employees>
    <name>Piattini</name>
    <salary>1000000</salary>
    <hiredate>2000-01-01</hiredate>
  </employees>
</myschema>
```
Mapping SQL Tables Onto XML Documents

<employees>
  <name>Marcos</name>
  <salary>1500000</salary>
  <hiredate>1999-01-01</hiredate>
</employees>

Note that there are no elements concerning departments! Hard to tell whether there is even a table for departments, or if it’s just empty.
Mapping SQL Tables Onto XML Documents

Open Issue: Representation of SQL nulls

• Alternative 1: Use xsi:nil="true" attribute
  
  <emp><name>Robledo</name>
  
  <hiredate xsi:nil="true"/>
  
  <salary>250000</salary>

• Alternative 2: Omit corresponding element

  <emp><name>Robledo</name>
  
  <salary>250000</salary>

  </emp>
Mapping SQL to XML
Issues Not Covered Here

- Automatic inference of XML Schema from SQL Schema
- XML Schema types: named or anonymous?
  - SQL’s built-in types
  - Types corresponding to SQL tables
- Impact of SQL privileges on mappings
- XML Namespace implications
Part III — SQL + XML

- Part I: Getting Started
- Part II: Mapping Concepts
- Part III: SQL + XML
- Part IV: Finishing Up
SQL & XPath, Storing XML, and XMLType

Still being defined, so…

All of this material is speculative
Using SQL and XML Together

- SQL plus XPath
- Storing XML in SQL databases
- XMLType
- Viewing SQL Data through an XML Lens
SQL Plus XPath

- XPath: A language for addressing parts of a document
- Compact, non-XML syntax
- Used in URIs, XLST, and other places
- Path notation to navigate hierarchical XML
- Defined on abstract document (Infoset), not on surface syntax (serialized document)
XPath fundamentals

- A *path* is a sequence of one or more *steps*

- `/` ➔ separator between steps of a path

- Each step evaluates to a *set of nodes* (no duplicates, in *document order*)

- Steps may traverse any one of several *axes*, including *child*, *parent*, *previous-sibling*, *following-sibling*, etc.
XPath fundamentals

- Several “shortcuts” to explicit axis use:
  - // → ancestor-or-self::
  - @ → attribute::
  - Element-name → child::

- Predicates
  - [5]
  - [para]
  - [@id="manager"]
XPath fundamentals

- Expanded version of shortcuts
  - `a//b` ➞ `child::a/ancestor-or-self::b`
  - `a@c` ➞ `child::a/attribute::c`
  - `a/d` ➞ `child::a/child::d`

- Predicates
  - `a[b]` ➞ all `a` elements that have a `b` element child
  - `a[@c]` ➞ all `a` elements that have a `c` attribute
  - `a[@c="xxx"]/b` ➞ all `b` elements of all `a` elements that have an attribute `c` whose value is `xxx`
XPath fundamentals

Examples:

- \//header ➔ All header elements in the document
- /book/author[1] ➔ The first author element that is a child of every book element
- /emp[@salary > "100000"] ➔ Every emp element that has a salary attribute whose value is greater than 100,000
SQL Plus XPath

- XML document in a column of a table
  CREATE TABLE my_docs ( doc_no INTEGER, doc_owner VARCHAR(50), document CLOB(2G) )
- Assume document in “raw XML” (text)
- How can we find relevant documents?
SQL Plus XPath

- Combine SQL and XPath:
  ```
  SELECT doc_no, xpath(document, '/publ/name')
  FROM my_docs
  WHERE xpath(document, '/author["Jim Melton"]')
  ```

- Needs new function, such as `xpath`, to evaluate an XPath expression on an XML “document”

- Efficient? Depends on implementation!
Storing XML in SQL Databases

- Character string form (e.g., CLOB)
- Fully or partially “shredded”
- “Native XML”
Storing XML in SQL Databases

Character String Form

- Easiest to store (no work to do)
- Best for retrieval in original form
- Easy to add full-text indexes
- Access to XML characteristics (e.g., search based on structure) requires late parsing
- Probably best for true text documents
- Suboptimal for data use or access to XML characteristics
Storing XML in SQL Databases

Shredded Form

- *Shredding*: storing some elements & contents in one place, other elements & contents in other places
- Most appropriate for “data-oriented” XML
- Element/attribute indexing: excellent
- Full-text indexing: good, if limited to selected parts
- Use of XML characteristics: depends on shredding details (*e.g.*, how thoroughly shredded)
- Often used in relational systems; few guiding principles
Storing XML in SQL Databases

“Native XML”

- Requires full parsing (e.g., into Infoset) — conceptually a tree of data structures
- Easiest access to XML characteristics of documents
- Full-text indexing awkward, but not impossible
- Data (value) indexing “interesting”, but very feasible
- Best for access to XML structure & characteristics
XMLType
Built-in or user-defined type?

- Built-in type
  - Easier for SQL engine to optimize
  - No issues about routines with optional parameters or variable number of parameters

- User-Defined Type (SQL:1999 UDT)
  - May be easier to specify
  - Superficially seems easier to extend

- Conclusion: Built-in type is more attractive
XMLType
Fundamental Concepts

- May be “based on” CLOB, but not required
- Stores all sorts of “XML”
  - Documents (tree with a prolog)
  - Nodes (tree without a prolog)
  - Fragments (perhaps not well-formed)
  - Forest (sequence of documents or nodes)
  - Element content (text, including mixed content)
- Most (or all) access through functional interface
XMLType

Fundamental Concepts

- Closed under most operations
  - Concatenating 2 elements yields a forest
  - Concatenating 2 text nodes yields a text node
  - Cannot concatenate a document (with prolog) with anything else
- Cannot directly store an attribute — but can store an attribute’s value (merely a text node)
XMLType

Typical Operations

- **XMLEXTRACT**
  (XMLType, XPath [, , Namespace])
- Uses the XPath argument to locate a node in the XMLType instance, possibly using Namespace information.
- What if no “hit”? Possible design choices are: return null value, or return empty XML document.
XMLType

Typical Operations

- **XMLCONCAT**
  - Syntax: `(XMLval, XMLval [, XMLval]...)`
  - Returns XML value that is the concatenation of the argument XML values
  - If any XML value is a document (with prolog), error results
XMLType

Typical Operations

- **XMLAGG**
  
  (XMLval ORDER BY sortexpr)

  Concatenates XML values retrieved from a grouped (SQL) table after sorting by the `sortexpr`.

- None of the `XMLval` values can be document (with prolog) unless all other values are null.
XMLType

Typical Operations

- Predicates
  - XMLEXISTS
    (XMLval, XPath [, , Namespace])
  - XMLval HAS XML PROLOG
  - XMLval IS XML ELEMENT
  - XMLval IS XML TEXT
  - XMLval IS XML SCHEMA
  - XMLval IS VALID
  - XMLval IS VALID INSTANCE OF
### Publishing SQL Data in XML

Here is an example of publishing SQL data in XML:

```xml
<?xml encoding
<emp name="Go
 <sal>100000
 <dept>Softw
 </emp>
<emp name="Ma
 <sal>130000
```

This XML structure can be used to represent SQL query results in a readable and structured format.
XML Publishing Functions
“View SQL through an XML lens”

- Functions invoked in ordinary SQL statements that generate XML text
- Used to produce XML document, elements, fragments, or forests from SQL data
XML Publishing Functions

- Important characteristics — unlike most SQL functions
  - Some arguments are SQL identifiers
  - Some args allow AS (alias for arg that is expression)
  - Some functions take a variable number of arguments
  - Some args to these funcs can be of any SQL data type
  - Non-reserved words are used to identify certain arguments (e.g., XMLATTRIBUTES)

- Also, some args can be other function invocations
XML Publishing Functions

XML: XMLELEMENT

- Arguments
  - Element name
  - Zero or more attributes
  - Zero or more element content expressions

```
XMLELEMENT ("emp", e.name || ' ' || e.sal) AS "emp_result"
```
XML Publishing Functions

XMLATTRIBUTES

- Only used in second arg to XMLELEMENT
- Arguments:
  - value-expression [ AS alias ]
  - Repeated once per attribute of containing element

XMLELEMENT ( "name",
XMLATTRIBUTES (id AS "empno",
depth AS "Dept"),
first_name || ' ' last_name )
XML Publishing Functions

XMLCOLELEM

- Produces forest of XML elements using a shared pattern
- Arguments
  - value-expression [ AS alias ]
  - Repeat once per column to be used for element content

- XMLCOLELEM ( name, sal AS "Salary" )
XML Publishing Functions

XMLCOLATTVAL

- Produces forest of XML elements using a shared pattern
- Each element named `column`
- Name of column appears in attribute `name`
- Arguments same as for XMLCOLELEM
- `XMLCOLATTVAL ( name, sal AS "Salary" )`
XML Publishing Functions

XMLGEN

- Generates XML element using a pattern
- Arguments:
  - Pattern with substitution variables
  - One or more SQL expressions with names

SQLGEN ('<Emp name="{"ename}""> 
<Dept>{dept}</Dept> </Emp>', empname AS "ename", dept)
XML Publishing Functions

- Mentioned earlier — all useful for publishing SQL data in XML
  - XMLCONCAT
  - XMLAGG
  - Several predicates
XML Publishing Functions

Longer Examples

```sql
SELECT e.id,
  XML_ELEMENT ( "Emp",
    XML_ATTRIBUTES ( e.id ),
    'Employee ',
    XML_ELEMENT ( "name",
      e.fname || ' ' || e.lname ),
    ' was hired on = ',
    XML_ELEMENT ( "hiredate",
      e.hiredate ) ) AS "result"
FROM employees AS e
```
## XML Publishing Functions
### Longer Examples

<table>
<thead>
<tr>
<th>ID</th>
<th>Result</th>
</tr>
</thead>
</table>
| 1234| `<emp id="1234">Employee<br/>  
<name>E. Marcos</name> was hired on <hiredate>1998-04-25</hiredate></emp>` |
| 5432| `<emp id="1234">Employee<br/>  
<name>J. Melton</name> was hired on <hiredate>1993-11-01</hiredate></emp>` |
XML Publishing Functions

Longer Examples

```sql
SELECT e.id,
  XMLELEMENT ( "Emp",
    XMLCOLELEM ( e.name,
      e.hiredate,
      e.dept ) )
AS "result"
FROM employees e
WHERE ...
```
**XML Publishing Functions**

**Longer Examples**

<table>
<thead>
<tr>
<th>ID</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>&lt;emp&gt;&lt;name&gt;E. Marcos&lt;/name&gt; &lt;hiredate&gt;1998-04-25&lt;/hiredate&gt; &lt;dept&gt;Engineering&lt;/dept&gt;&lt;/emp&gt;</td>
</tr>
<tr>
<td>5432</td>
<td>&lt;emp&gt;&lt;name&gt;J. Melton&lt;/name&gt; &lt;hiredate&gt;1993-11-01&lt;/hiredate&gt; &lt;dept&gt;Travel&lt;/dept&gt;&lt;/emp&gt;</td>
</tr>
</tbody>
</table>
XML Publishing Functions
Longer Examples

```sql
SELECT e.id,
XMLELEMENT ( "Emp",
XMLCOLATTVAL ( e.name,
   e.hiredate,
   e.dept ) )
AS "result"
FROM employees e
WHERE ...
### XML Publishing Functions

#### Longer Examples

<table>
<thead>
<tr>
<th>ID</th>
<th><code>&lt;emp&gt;</code></th>
<th><code>&lt;column name=&quot;NAME&quot;&gt;</code></th>
<th>E. Marcos</th>
<th><code>&lt;column name=&quot;HIREDATE&quot;&gt;</code></th>
<th>1998-04-25</th>
<th><code>&lt;column name=&quot;DEPT&quot;&gt;</code></th>
<th>Engineering</th>
<th><code>&lt;/emp&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td><code>&lt;emp&gt;</code></td>
<td><code>&lt;column name=&quot;NAME&quot;&gt;</code></td>
<td>J. Melton</td>
<td><code>&lt;column name=&quot;HIREDATE&quot;&gt;</code></td>
<td>1993-11-01</td>
<td><code>&lt;column name=&quot;DEPT&quot;&gt;</code></td>
<td>Travel</td>
<td><code>&lt;/emp&gt;</code></td>
</tr>
<tr>
<td>5432</td>
<td><code>&lt;emp&gt;</code></td>
<td><code>&lt;column name=&quot;NAME&quot;&gt;</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
XML Publishing Functions

Longer Examples

```sql
SELECT e.id,
XMLGEN ( '<Emp name="{name}">
<hiredate>{hire}</hiredate>
<department>{dept}</department>
</Emp>',
e.name AS name,
e.hire,
e.dept ) AS "result"
FROM employees e
WHERE ...
```
### XML Publishing Functions

**Longer Examples**

<table>
<thead>
<tr>
<th>ID</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td><code>&lt;emp&gt;&lt;name&gt;E. Marcos&lt;/name&gt;&lt;hiredate&gt;1998-04-25&lt;/hiredate&gt;&lt;dept&gt;Engineering&lt;/dept&gt;&lt;/emp&gt;</code></td>
</tr>
<tr>
<td>5432</td>
<td><code>&lt;emp&gt;&lt;name&gt;J. Melton&lt;/name&gt;&lt;hiredate&gt;1993-11-01&lt;/hiredate&gt;&lt;dept&gt;Travel&lt;/dept&gt;&lt;/emp&gt;</code></td>
</tr>
</tbody>
</table>
XML Publishing Functions
Longer Examples

- `<first>Jim</first><last>Melton</last>`

```sql
XMLCONCAT(
  XMLELEMENT("first", e.fname),
  XMLELEMENT("last", e.lname)
)
```
XML Publishing Functions
Longer Examples

- SELECT XMLELEMENT ( "Department", XMLATTRIBUTES ( e.dept AS "name" ), XMLAGG ( XMLELEMENT ("emp", e.lname) ORDER BY e.lname ) ) AS "dept_list"

  FROM employees e

  GROUP BY dept

- Produces one emp element per employee of each dept, sorted by last name, one “outer” element per dept
### XML Publishing Functions

**Longer Examples**

<table>
<thead>
<tr>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;department name=&quot;Engineering&quot;&gt; &lt;emp&gt;Marcos&lt;/emp&gt;&lt;emp&gt;Piattini&lt;/emp&gt; &lt;/department&gt;</code></td>
</tr>
<tr>
<td><code>&lt;department name=&quot;Travel&quot;&gt;&lt;emp&gt;Edelberg&lt;/emp&gt;&lt;emp&gt;Melton&lt;/emp&gt; &lt;/department&gt;</code></td>
</tr>
</tbody>
</table>
Part IV — Finishing Up

Part I
Getting Started

Part II
Mapping Concepts

Part III
SQL + XML

Part IV
Finishing Up
SQL & XML

Unfinished Business

- Creating SQL data from XML documents
- Update XML documents
- Full-text search on XML documents
- ...

SQLX: Bringing XML and SQL Together
The SQL/XML Namespaces

- SQL/XML uses three XML namespaces:
  - xsd: http://www.w3.org/2001/XMLSchema
  - xsi: http://www.w3.org/2001/XMLSchema-instance
  - sqlxml: (see next slide)

- Prefixes are not relevant; URIs are
The **sqlxml** Namespace

- When first version of the *standard* is published:  

- Interim (working) versions:  
  [http://www.iso-standards.net/9075/YYYY/MM/sqlx](http://www.iso-standards.net/9075/YYYY/MM/sqlx)

- ISO provides persistent repository for actual XML schema files (but schema processors have no need to access them)
The `sqlxml` Namespace

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.iso-standards.net/9075/sqlx"
  xmlns:sqlxml="http://www.iso-standards.net/9075/sqlx">
  <xsd:element name="sqlxml" xmlns="http://www.iso-standards.net/9075/sqlx">
  </xsd:element>
</xsd:schema>
```
The sqlxml Namespace

- <xsd:annotation>
  <xsd:documentation xml:lang="en-US">
  ISO/IEC 9075-14:200n SQL/XML
  This document contains definitions of types and annotations as specified in ISO/IEC 9075-14:200n.
  </xsd:documentation>
</xsd:annotation>
The sqlxml Namespace

- `<xsd:simpleType name="typeKeyword">
  `<xsd:restriction base="xsd:string">
    `<xsd:enumeration value="CHAR"/>
    `<xsd:enumeration value="VARCHAR"/>
    `<xsd:enumeration value="CLOB"/>
    `<xsd:enumeration value="BLOB"/>
    `<xsd:enumeration value="BIT"/>
    `<xsd:enumeration value="BIT VARYING"/>
    `<xsd:enumeration value="NUMERIC"/>
    `<xsd:enumeration value="DECIMAL"/>
  </xsd:restriction>
</xsd:simpleType>`
The sqlxml Namespace

- <xsd:enumeration value="INTEGER"/>
- <xsd:enumeration value="SMALLINT"/>
- <xsd:enumeration value="FLOAT"/>
- <xsd:enumeration value="REAL"/>
- <xsd:enumeration value="DOUBLE PRECISION"/>
- <xsd:enumeration value="BOOLEAN"/>
- <xsd:enumeration value="DATE"/>
- <xsd:enumeration value="TIME"/>
- <xsd:enumeration value="TIME WITH TIME ZONE"/>
- <xsd:enumeration value="TIMESTAMP"/>
The sqlxml Namespace

- `<xsd:enumeration value="TIMESTAMP WITH TIME ZONE"/>
- `<xsd:enumeration value="INTERVAL YEAR"/>
- `<xsd:enumeration value="INTERVAL YEAR TO MONTH"/>
- `<xsd:enumeration value="INTERVAL MONTH"/>
- `<xsd:enumeration value="INTERVAL DAY"/>
- `<xsd:enumeration value="INTERVAL DAY TO HOUR"/>
- `<xsd:enumeration value="INTERVAL DAY TO MINUTE"/>
The sqlxml Namespace

- `<xsd:enumeration value="INTERVAL DAY TO SECOND"/>
- `<xsd:enumeration value="INTERVAL HOUR"/>
- `<xsd:enumeration value="INTERVAL HOUR TO MINUTE"/>
- `<xsd:enumeration value="INTERVAL HOUR TO SECOND"/>
- `<xsd:enumeration value="INTERVAL MINUTE"/>
- `<xsd:enumeration value="INTERVAL MINUTE TO SECOND"/>

Copyright © 2001
Oracle Corp.

SQLX: Bringing XML and SQL Together
Jim Melton
134
The sqlxml Namespace

- <xsd:enumeration
  value="INTERVAL SECOND"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:element name="sqltype">
  <xsd:complexType>
    <xsd:attribute name="name" type="sqlxml:typeKeyword"/>
    <xsd:attribute name="length" type="xsd:integer" use="optional"/>
  </xsd:complexType>
</xsd:element>
The sqlxml Namespace

- `<xsd:attribute name="maxLength" type="xsd:integer" use="optional"/>
- `<xsd:attribute name="characterSetName" type="xsd:string" use="optional"/>
- `<xsd:attribute name="collation" type="xsd:string" use="optional"/>
- `<xsd:attribute name="precision" type="xsd:integer" use="optional"/>
- `<xsd:attribute name="scale" type="xsd:integer" use="optional"/>"
The sqlxml Namespace

```xml
<xsd:attribute name="maxExponent" type="xsd:integer" use="optional"/>
<xsd:attribute name="minExponent" type="xsd:integer" use="optional"/>
<xsd:attribute name="userPrecision" type="xsd:integer" use="optional"/>
<xsd:attribute name="leadingPrecision" type="xsd:integer" use="optional"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```
<xml version="1.0">
<welcome>
<from>Jim Melton & Oracle Corp.</from>
<subject>SQLX: Bringing XML and SQL Together</subject>
</welcome>
</xml>
XML Query

- Very brief overview
- Still under development — very incomplete
- Many SQL-like capabilities, but doesn’t “feel” like SQL…
- Because data model requirements are different — semi-structured trees instead of regular tables with explicit metadata

Copyright © 2001 Oracle Corp.
SQLX: Bringing XML and SQL Together
Jim Melton
XML Query

- SQL: SELECT...FROM...WHERE...
- XQuery: FOR...LET...WHERE...RETURN

- FLWR ("flower") expressions
- Fully composable
- Functional language
- Uses XPath for some document navigation and addressing
XML Query

- Type system based on (an extended subset of) XML Schema
- Operates on XPath 2.0 and XQuery 1.0 Data Model, not on surface syntax
- Results in a Data Model instance
- Defined for well-formed XML, DTD-valid XML, Schema-valid XML, and collections (forests) of each/all
XML Query

- Each query is an expression
- Easy to combine and nest expressions
- Allows construction of elements and attributes
- Keywords not case-sensitive, identifiers are
- Keywords may be reserved (open issue)
- Variable names start with $ (e.g., $sal)
XML Query

- FLWRExpr ::= (ForClause | LetClause)+ WhereClause? "return" Expr
- ForClause ::= "for" Variable "in" Expr ("," Variable "in" Expr)*
- LetClause ::= "let" Variable ":=" Expr ("," Variable ":=" Expr)*
- WhereClause ::= "where" Expr
XML Query
FOR clause

- Used for iteration
- Introduces variables and binds them to an expression
  
  FOR $x$ IN /library/book binds many <book> elements to $x$, one per iteration

- Multiple FOR expressions do “joins”
XML Query

LET clause

- Binds one or more variables to expressions
- No iteration

LET $y := /library/book$ binds $y$ to a sequence of <book> nodes…not to individual <book> nodes
XML Query

FOR and LET clauses

- FLWR without FOR generates exactly one “binding-tuple”.
- Multiple independent FORs generates Cartesian product: number of binding-tuples is equal to product of tuples in each FOR
XML Query

WHERE clause

- Filters binding-tuples returned from FOR & LET clauses
- Closely analogous to SQL’s WHERE clause
- Variables bound by FOR are bound to individual nodes at the WHERE:
  WHERE $x/color="red"
- Variables bound by LET are bound to collections at WHERE:
  WHERE avg($y/price) > 100
XML Query

RETURN clause

- Generates the output of the FLWR
- Executed once for each binding-tuple resulting from FOR and LET and remaining after WHERE
- Usually contains element constructors, often uses subexpressions
XML Query Example

- FOR $b IN document("bib.xml")//book
  WHERE $b/publisher = "Morgan Kaufmann"
  AND $b/year = "2001"
  RETURN $b/title

SQLX: Current Status Redux

- Representation of XML data in SQL and *vice versa* — *well under way*
- Mapping between XML schemas and SQL schemata — *in progress*
- Representation of SQL schemata in XML — *in progress*
SQLX: Current Status Redux

- Representation of SQL actions in XML — some proposals have been drafted
- SQL-friendly syntax for querying XML — some research and prototypes under way
The SQLX Group

- Interested in joining? Send email to our webmaster at: mailto:campbelc@acm.org
- Check out http://www.sqlx.org (but substantive contents are mostly protected — to get access, join!)
SQL/XML

- Current draft available at:

- Next draft expected in early December at:
Relevant W3C Documents

- http://www.w3.org/TR/xml-infoset
- http://www.w3.org/TR/xmlschema-0
- http://www.w3.org/TR/xmlschema-1
- http://www.w3.org/TR/xmlschema-2
- http://www.w3.org/TR/xpath
- http://www.w3.org/TR/REC-xml-names
Relevant W3C Documents

- http://www.w3.org/TR/xmlquery-use-cases
- http://www.w3.org/TR/query-datamodel
- http://www.w3.org/TR/query-semantics
- http://www.w3.org/TR/xquery
- http://www.w3.org/TR/xquery-operators
SELECT questions
FROM audience
WHERE topic = 'SQLX'

FOR $q IN
document("JISBD")//audience
WHERE $q/topic = "SQLX"
RETURN $q/question/text()