Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

Chapter 9 – SQL/XML

Outline

Overview

I. Object-Relational Database Concepts

1. User-defined Data Types and Typed Tables
2. Object-relational Views and Collection Types
3. User-defined Routines and Object Behavior
4. Application Programs and Object-relational Capabilities

II. Online Analytic Processing

5. Data Analysis in SQL
6. Windowed Tables and Window Functions in SQL

III. XML

7. XML Data Modeling
8. Xquery
9. **SQL/XML**

IV. More Developments (if there is time left)

temporal data models, data streams, databases and uncertainty, …

SQL and XML?!

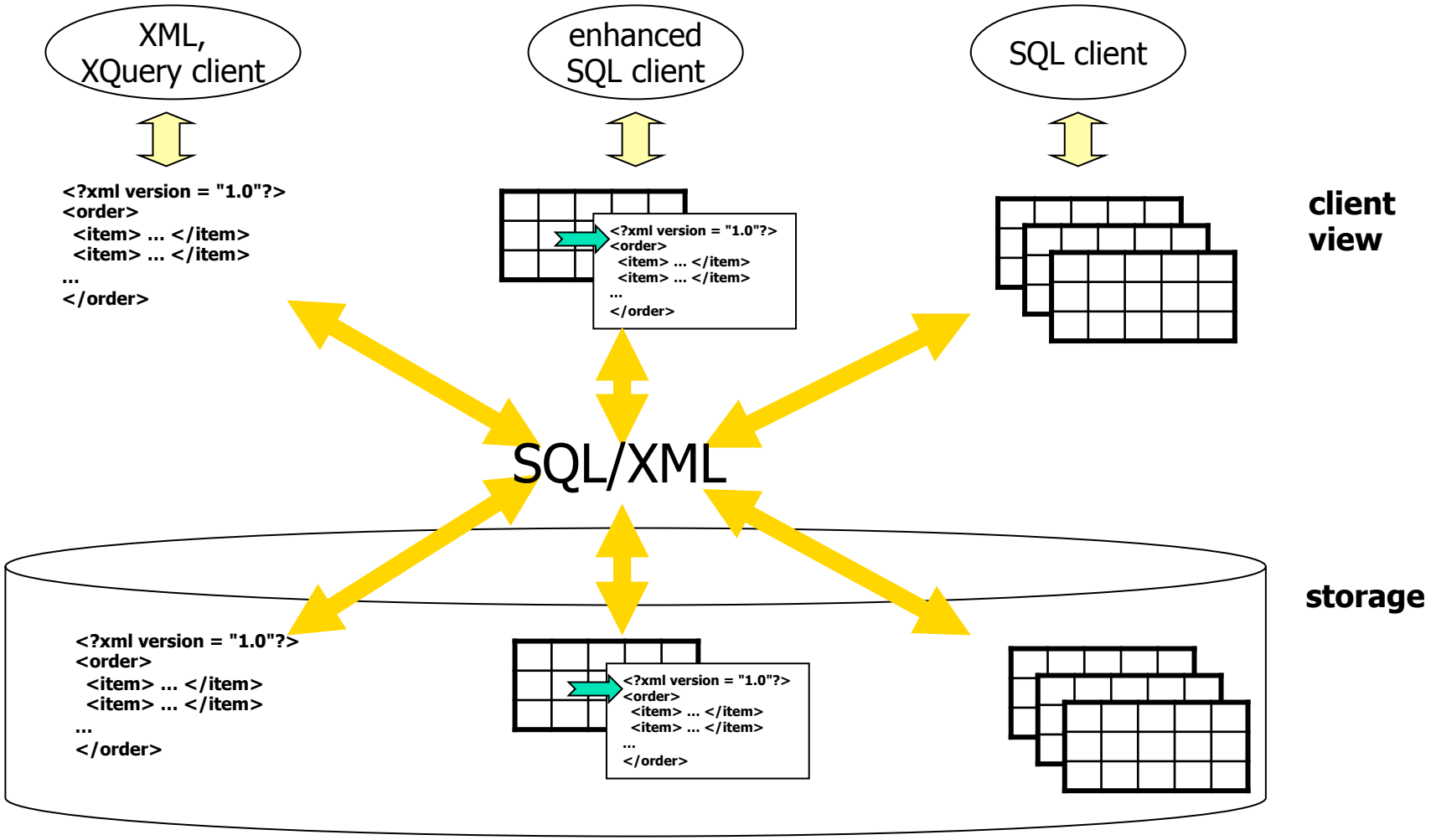
- Two major perspectives
  - Flexible exchange of relational data using XML
    - publish relational as XML
    - decompose or "shred" XML into relational
  - Reliable XML data management
    - manage, search, maintain, update, …
    - integrate with relational data
- Native-XML databases? No significant customer interest!
  - reluctance to introduce new DBMS environment
  - limited integration with relational DBMS products
  - lack of maturity (scalable, reliable, highly available, …)
  - skill revolution (not evolution) required

Remember OO-DBMS?

SQL and XML

- Use existing (object-)relational technology?
  - Large Objects: granularity understood by DBMS may be too coarse!
    - search/retrieval of subsets, update of documents
  - Decompose into tables: often complex, inefficient
    - mapping complexity, especially for highly "denormalized" documents
  - Useful, but not sufficient
    - should be standardized as part of SQL
    - but needs further enhancement to support "native" XML support in SQL
- Enable "hybrid" XML/relational data management
  - supports both relational and XML data
    - storage, access
    - query language
    - programming interfaces
  - ability to view/access relational as XML, and XML as relational
  - all major relational DBMS vendors are moving into this direction

SQL/XML Big Picture



XML Data Type

- New SQL type “XML”
  - for storing XML data "natively" in the database
  - for capturing the data type of results and input values of SQL/XML functions that work with XML data
  - can have optimized internal representation (different from character string)
- "Shape" of an XML value
  - not just a well-formed XML document
  - but also the content of an XML element
    - element, sequence of elements, text, mixed content, …
  - based on XQuery
    - value of type XML is an instance of the XQuery data model

XML Data Type - Example

```
CREATE TABLE employees
  ( id              CHAR(6),
    lastname        VARCHAR (30),
    ...,
    resume          XML
  )
```

ID	LASTNAME	...	RESUME
940401	Long	...	<?xml version="1.0"?> <resume xmlns="http://www.res.com/resume"> <name> … </name> <address> … </address> ... </resume>
862233	Nicks	...	null
766500	Banner	...	<resume ref="http://www.banner.com/resume.html"/>

XML Data Type – Modifiers

- Permitted values can (optionally) be restricted (e.g., in column definition)
  - XML(SEQUENCE)
    - XQuery DM instance (i.e., a sequence)
  - XML(CONTENT)
    - XQuery document node
      - more flexible than well-formed documents
      - permits document nodes that have several element children (i.e., no single root)
  - XML(DOCUMENT)
    - document node with a single root element (i.e., a well-formed XML document)
- Further modifiers for CONTENT, DOCUMENT
  - UNTYPED
    - element and attribute nodes don't have type annotations (i.e., have not undergone a schema validation)
  - XMLSCHEMA
    - requires nodes contained in the XML values to be **valid** according to a registered schema or a global element in a schema
    - XML(DOCUMENT(XMLSCHEMA <XML valid according to what>))

XML Data Type *(continued)*

- Conversion to/from character strings and BLOBs
  - XMLPARSE and XMLSERIALIZE functions
  - implicit conversion during host language interaction
- Examples:

INSERT INTO employees VALUES ('123456', 'Smith', …, **XMLPARSE** (DOCUMENT '<?xml version="1.0"?> <resume xmlns="http://www.res.com/resume"><name> … </name><address> … </address>...</resume>' PRESERVE WHITESPACE));

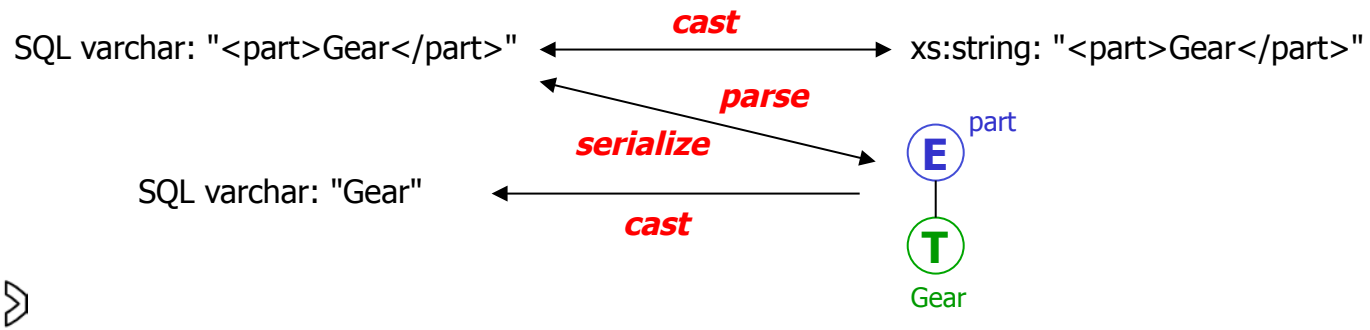
SELECT e.id, **XMLSERIALIZE** (DOCUMENT e.resume AS VARCHAR (2000)) AS resume
FROM employees AS e
WHERE e.id = '123456';

➔

ID	RESUME
123456	<?xml version="1.0" encoding ="UTF-8"> <resume xmlns="http://www.res.com/resume"> <name> … </name> <address> … </address> ... </resume>

XMLCAST - Converting SQL to XML, XML to SQL

- Convert an SQL value into an XML value
  - Values of SQL predefined types are cast to XQuery atomic types using
    - The defined mapping of SQL types/values to XML Schema types/values
    - The semantics of XQuery's cast expression
  - XMLCAST(NULL AS XML) returns the SQL null value typed as XML
- Convert an XML value into an SQL value
  - XML values are converted to values of SQL predefined types using a combination of
    - The defined mapping of SQL types to XML Schema types and SQL's CAST specification
    - XQuery's fn:data() function and cast expression
  - An XML value that is the empty sequence is converted to a NULL value of the specified SQL data type
- Note: XMLCAST to/from character strings is different from XMLSERIALIZE and XMLPARSE



XML Schema and Validating XML

- XML Schema has to be registered with the DBMS before it can be referenced
  - how this is done is left to the DBMS (i.e., implementation-defined)
    - need to supply at least a location URI, namespace information, and an SQL identifier (three-part name)
  - registered schema can be reference using the SQL identifier (ID) or the location URI (URI)
- XMLVALIDATE function
  - ensure that XML values are valid according to a certain registered XML schema
  - XMLVALIDATE() validates and annotates XML values
  - Multiple options to identify the XML schema to use
    - ID, URI
    - *xsi:schemaLocation* information provided in the input document
- Example:

```
INSERT INTO POrders
VALUES ('WO20051234', CURRENT TIMESTAMP, 'R', 'W',
  XMLVALIDATE
    (XMLPARSE       (DOCUMENT '<purchaseOrder>…</purchaseOrder>'
                     PRESERVE WHITESPACE)
  ACCORDING TO XMLSCHEMA ID PORDER) );
```

SQL/XML “constructor functions”

- Functions/operators for generating XML constructs (elements, attributes, ...) within an SQL query
- Function syntax for generating XML nodes of various types
  - XMLELEMENT, XMLATTRIBUTE, XMLCOMMENT, XMLPI, XMLTEXT
  - XMLDOCUMENT wraps an XQuery document node around an XML value
- Producing sequences of values/nodes
  - XMLFOREST generates multiple element nodes
  - XMLCONCAT concatenates XML values
- Concatenation over sets of tuples
  - XMLAGG aggregates XML across multiple tuples

XMLELEMENT

- Produces an XML value that corresponds to an XML element, given:
  - An SQL identifier that acts as its **name**
  - An optional list of **namespace** declarations
  - An optional list of named expressions that provides names and values of its **attributes**, and
  - An optional list of expressions that provides its **content**
- Options for NULL content
  - empty element
  - NULL
  - empty element with attribute nil='true'
  - empty sequence or XQuery document node with no children

XMLELEMENT (continued)

- A simple example:

```
SELECT   e.id,
         XMLELEMENT (NAME "Emp", e.lname) AS "result"
FROM  employees e
WHERE ... ;
```

➔

ID	result
1001	<Emp>Smith</Emp>
1006	<Emp>Martin</Emp>

XMLELEMENT (continued)

- XMLELEMENT can produce nested elements (with mixed content):

```
SELECT   e.id,
         XMLELEMENT (NAME "Emp",
                  XMLELEMENT (NAME "name", e.lname ),
                  ' was hired on '
                  XMLELEMENT (NAME "hiredate", e.hire )
         ) AS "result"
FROM employees e
WHERE … ;
```

➔

ID	result
1001	<Emp> <name>Smith</name> was hired on <hiredate>2000-05-24</hiredate> </Emp>
1006	<Emp> <name>Martin</name> was hired on <hiredate>1996-02-01</hiredate> </Emp>

XMLELEMENT (continued)

- XMLELEMENT can take subqueries as arguments:

```
SELECT   e.id,
        XMLELEMENT (NAME "Emp",
                XMLELEMENT (NAME "name", e.lname ),
                XMLELEMENT (NAME "dependants",
                        (SELECT COUNT (*)
                         FROM dependants d
                         WHERE d.parent = e.id))
        ) AS "result"
FROM employees e
WHERE … ;
```

➔

ID	result
1001	<Emp> <name>Smith</name> <dependants>3</dependants> </Emp>

XMLATTRIBUTES (within XMLELEMENT)

- Attribute specifications must appear directly after element name and optional namespace declaration.
- Each attribute can be named implicitly or explicitly.

```
SELECT   e.id,
         XMLELEMENT (NAME "Emp",
                     XMLATTRIBUTES (e.id, e.lname AS "name")
         ) AS "result"
FROM employees e
WHERE … ;
```

➔

ID	result
1001	<Emp ID="1001" name="Smith"/>
1006	<Emp ID="1206" name="Martin"/>

XMLNAMESPACES (within XMLELEMENT)

- Namespace declarations must appear directly after element name.

```
SELECT empno,
        XMLELEMENT(NAME "admi:employee",
                XMLNAMESPACES('http://www.admi.com' AS "admi"),
                XMLATTRIBUTES(e.workdept AS "admi:department"),
                e.lastname
        ) AS "result"
FROM employees e
WHERE e. job = 'ANALYST';
```

➔

ID	result
1130	<admi:employee xmlns:admi="http://www.admi.com" admi:department="C01">QUINTANA</admi:employee>
1140	<admi:employee xmlns:admi="http://www.admi.com" admi:department="C01">NICHOLLS</admi:employee>

XMLCONCAT

- Produces an XML value given two or more expressions of XML type.
- If any of the arguments evaluate to the null value, it is ignored.

```
SELECT e.id,
        XMLCONCAT (XMLELEMENT ( NAME "first", e.fname),
                   XMLELEMENT ( NAME "last", e.lname)
        ) AS "result"
FROM employees e ;
```

➔

ID	result
1001	<first>John</first> <last>Smith</last>
1006	<last>Martin</last>

XMLFOREST

- Produces a sequence of XML elements given named expressions as arguments. Arguments can also contain a list of namespace declarations.
- Element can have an explicit name or an implicit name, if the expression is a column reference
- Same options for handling NULL values as in XMLELEMENT

```
SELECT   e.id,
         XMLELEMENT (NAME "employee", XMLFOREST (e.hire, e.dept AS "department")
                  AS "result"
FROM employees e
WHERE ... ;
```
➔

ID	result
1001	<employee> <HIRE>2000-05-24</HIRE> <department>Accounting</department> </employee>
1006	<employee> <HIRE>1996-02-01</HIRE> <department>Shipping</department> </employee>

XMLAGG

- An aggregate function, similar to SUM, AVG, etc.
  - The argument for XMLAGG must be an expression of XML type.
- Semantics
  - For each row in a group G, the expression is evaluated and the resulting XML values are concatenated to produce a single XML value as the result for G.
  - An ORDER BY clause can be specified to order the results of the argument expression before concatenating.
  - All null values are dropped before concatenating.
  - If all inputs to concatenation are null or if the group is empty, the result is the null value.

XMLAGG - Example

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

         ) AS "dept_list",
         COUNT(*) AS "dept_count"
FROM employees e
GROUP BY dept ;
```

➔

dept_list	dept_count
<Department name="Accounting"> <emp>Smith</emp> <emp>Yates</emp> </Department>	2
<Department name="Shipping"> <emp>Martin</emp> <emp>Oppenheimer</emp> </Department>	2

SQL/XML Constructor Function Usage

- Dynamically retrieve SQL data in XML format (optionally mixed with SQL)
- Use query results to update/insert into tables with XML columns
- Use standard SQL views to create logical tables with XML columns

```
CREATE VIEW XMLDept (DeptDoc XML) AS (
    SELECT   XMLELEMENT ( NAME "Department",
                          XMLATTRIBUTES ( e.dept AS "name" ),
                          XMLATTRIBUTES ( COUNT(*) AS "count",
                          XMLAGG (XMLELEMENT (NAME "emp",
                                              XMLELEMENT(NAME "name", e.lname)
                                              XMLELEMENT(NAME "hire", e.hire))
                          )
    FROM employees e
    GROUP BY dept) ;
```

Manipulating XML Data

- Constructor functions
  - focus on publishing SQL data as XML
  - no further manipulation of XML
- More requirements
  - how do we select or extract portions of XML data (e.g., from stored XML)?
  - how can we decompose XML into relational data?
    - XMLCAST is not sufficient
  - both require a language to identify, extract and possibly combine parts of XML values

SQL/XML utilizes the XQuery standard for this!

XMLQUERY

- Evaluates an XQuery or XPath expression
  - Provided as a character string literal
- Allows for optional arguments to be passed in
  - Zero or more named arguments
  - At most one unnamed argument can be passed in as the XQuery context item
  - Arguments can be of any predefined SQL data type incl. XML
  - Non-XML arguments will be implicitly converted using XMLCAST
- Returns a sequence of XQuery nodes

```
SELECT XMLQUERY(
  'for $e in $dept[@count > 3]/emp
   where $e/hire > 2004-12-31 return $e/name'
  PASSING BY REF deptDoc AS "dept"
  RETURNING SEQUENCE) AS "Name_elements"
FROM XMLDept
```

=>

Name_elements
<name>Miller</name>
<name>Smith</name> <name>Johnson</name>
<name>Martin</name>

XMLTABLE

- Transforming XML data into table format
- Evaluates an XQuery or XPath expression – the "**row pattern**"
  - each item of result sequence is turned into a row
  - allows for optional arguments to be passed in, just like XMLQuery
- Element/attribute values are mapped to column values using path expressions (PATH) – the "**column pattern**"
- Names and SQL data types for extracted values/columns need to be specified
- Default values for "missing" columns can be provided
- ORDINALITY column can be generated
  - contains a sequential number of the corresponding XQuery item in the XQuery sequence (result of the row pattern)

XMLTABLE - Example

```
SELECT X.*
FROM    XMLDept d,
        XMLTABLE ('$dept/emp' PASSING d.deptDoc AS "dept"
                COLUMNS
                "#num"    FOR ORDINALITY,
                "name"    VARCHAR(30)        PATH 'name',
                "hire"    DATE               PATH 'hire',
                "dept"    VARCHAR(40)        PATH '../@name'
                ) AS "X"
```

=>

#num	name	hire	dept
1	Smith	2005-01-01	Accounting
2	Yates	2002-02-01	Accounting
3	Martin	2000-05-01	Shipping

SQL Predicates on XML type

- IS DOCUMENT
  - Checks whether an XML value conforms to the definition of a well-formed XML document
- IS CONTENT
  - Checks whether an XML value conforms to the definition of either a well-formed XML document or a well-formed external parsed entity
- IS VALID
  - Checks whether an XML value is valid according to a given XML Schema
  - Does not validate/modify the XML value; i.e., no default values are supplied.
- XMLEXISTS
  - Checks whether the result of an XQuery expression (an XQuery sequence) contains at least one XQuery item

SQL/XML Mapping Definitions

- Mapping SQL identifiers to XML Names and vice versa
  - rules for mapping regular and delimited identifiers
  - encoding/decoding of illegal character or character combinations
- Mapping SQL (built-in) data types to XML Schema types
  - best match, additional XML schema facets
  - schema annotations
- Mapping of values based on the type mappings
- Mapping of SQL tables, schemas, catalogs to XML documents
  - options for fine-tuning the XML schema structure
  - can be used to produce an XML-only "view" of a relational database
  - potential basis for XQuery over SQL data

see appendix

Mapping SQL Tables to XML Documents

- The following can be mapped to an XML Document:
  - Table
  - Tables of an SQL Schema
  - Tables of an SQL Catalog
- The mapping produces an XML Document and an XML Schema Document
- These XML Documents may be physical or virtual
- The mapping of SQL Tables uses the mapping of SQL identifiers, SQL data types, and SQL values
- Two choices for the mapping of null values:
  - nil: use xsi:nil="true"
  - absent: column element is omitted

Mapping Options

- Users can control whether a table is mapped to a single element or a sequence of elements.
- In a single element option:
  - The table name serves as the element name.
  - Each row is mapped to a nested element with each element named as "row".
  - Each column is mapped to a nested element with column name serving as the element name.
- In a sequence of elements option:
  - Each row is mapped to an element with the table name serving as the element name.
  - Each column is mapped to a nested element with column name serving as the element name.

Mapping Example – Single Element

- Map the EMPLOYEE table ("single element option"):

```
<EMPLOYEE>
   <row>
      <EMPNO>000010</EMPNO>
      <FIRSTNME>CHRISTINE</FIRSTNME>
      <LASTNAME>HAAS</LASTNAME>
      <BIRTHDATE>1933-08-24</BIRTHDATE>
      <SALARY>52750.00</SALARY>
   </row>
   <row>
      <EMPNO>000020</EMPNO>
      <FIRSTNME>MICHAEL</FIRSTNME>
      <LASTNAME>THOMPSON</LASTNAME>
      <BIRTHDATE>1948-02-02</BIRTHDATE>
      <SALARY>41250.00</SALARY>
   </row>
   …
</EMPLOYEE>
```

Mapping Example – Sequence of Elements

- Map the EMPLOYEE table ("sequence of elements option"):

```
<EMPLOYEE>
  <EMPNO>000010</EMPNO>
  <FIRSTNME>CHRISTINE</FIRSTNME>
  <LASTNAME>HAAS</LASTNAME>
  <BIRTHDATE>1933-08-24</BIRTHDATE>
  <SALARY>52750.00</SALARY>
</EMPLOYEE>

<EMPLOYEE>
  <EMPNO>000020</EMPNO>
  <FIRSTNME>MICHAEL</FIRSTNME>
  <LASTNAME>THOMPSON</LASTNAME>
  <BIRTHDATE>1948-02-02</BIRTHDATE>
  <SALARY>41250.00</SALARY>
</EMPLOYEE>

  ...
```

Mapping All Tables of a Schema

- Map the ADMINISTRATOR schema:

```
<ADMINISTRATOR>
   <DEPARTMENT>
      <row>
         <DEPTNO>A00</DEPTNO>
         <DEPTNAME>SPIFFY COMPUTER SERVICE DIV.</DEPTNAME>
         <MGRNO>000010</MGRNO>
      </row>
      …
   </DEPARTMENT>
   <ORG>
      <row>
         <DEPTNUMB>10</DEPTNUMB>
         <DEPTNAME>Head Office</DEPTNAME>
         <MANAGER>160</MANAGER>
      </row>
      …
   </ORG>
</ADMINISTRATOR>
```

Mapping All Tables of a Catalog

- Mapping the HR catalog:

```
<HR>
   <ADMINISTRATOR>
      <DEPARTMENT>
         <row>…</row>
         …
      </DEPARTMENT>
      …
   </ADMINISTRATOR>
   <SYSCAT>
      …
   </SYSCAT>
</HR>
```

Corresponding XML Schema

- XML Schema that is generated:
  - provides named type for every column, row, table, schema, and catalog
  - allows annotation to be included in each of these definitions
- SQL data types map to XML Schema type names

SQL Data Type	XML Schema type name
INTEGER	INTEGER
CHAR (12)	CHAR_12
DECIMAL (6,2)	DECIMAL_6_2
INTEGER ARRAY [20]	ARRAY_20.INTEGER

SQL/XML Mapping - Example

- SQL table "EMPLOYEE"
  - XML document:

```
<EMPLOYEE>
  <row>
    <EMPNO>000010</EMPNO>
    <FIRSTNME>CHRISTINE</FIRSTNME>
    <LASTNAME>HAAS</LASTNAME>
    <BIRTHDATE>1933-08-24</BIRTHDATE>
    <SALARY>52750.00</SALARY>
  </row>
  <row>
    <EMPNO>000020</EMPNO>
    <FIRSTNME>MICHAEL</FIRSTNME>
    <LASTNAME>THOMPSON</LASTNAME>
    <BIRTHDATE>1948-02-02</BIRTHDATE>
    <SALARY>41250.00</SALARY>
  </row>
  ...
</EMPLOYEE>
```

Corresponding XML-Schema document

```
<xsd:schema>

<xsd:simpleType name="CHAR_6">
  <xsd:restriction base="xsd:string">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
...
<xsd:simpleType name="DECIMAL_9_2">
  <xsd:restriction base="xsd:decimal">
    <xsd:totalDigits value="9"/>
    <xsd:fractionDigits value="2"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name=
   "RowType.HR.ADMINISTRATOR.EMPLOYEE">
  <xsd:sequence>
    <xsd:element name="EMPNO" type="CHAR_6"/>
    <xsd:element name="FIRSTNME"
        type="VARCHAR_12"/>
    <xsd:element name="LASTNAME"
        type="VARCHAR_15"/>
    <xsd:element name="BIRTHDATE" type="DATE"
        nillable="true"/>
    <xsd:element name="SALARY"
        type="DECIMAL_9_2" nillable="true"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name=
    "TableType.HR.ADMINISTRATOR.EMPLOYEE">
  <xsd:sequence>
    <xsd:element name="row"
      type=
        "RowType.HR.ADMINISTRATOR.EMPLOYEE"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="EMPLOYEE" type=
      "TableType.HR.ADMINISTRATOR.EMPLOYEE"/>

</xsd:schema>
```

XML Schema Annotations

- Annotations may be included:

```
<xsd:complexType name="TableType.HR.ADMINISTRATOR.EMPLOYEE">
   <xsd:annotation>
      <xsd:appinfo>
         <sqlxml:sqlname
            type="BASE TABLE"
            catalogName="HR"
            schemaName="ADMINISTRATOR"
            localName="EMPLOYEE"/>
      </xsd:appinfo>
   </xsd:annotation>
   <xsd:sequence>
      <xsd:element name="row"
         type="RowType.HR.ADMINISTRATOR.EMPLOYEE"
         minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

Product Support

- The "big three" support XML in SQL databases
  - IBM, Oracle implement (almost) complete support of SQL/XML
  - Microsoft supports similar capabilities using proprietary syntax
  - all three support XQuery inside SQL
  - differences in implementation of XML storage
- IBM DB2 V9 (SIGMOD2005, VLDB2005)
  - CLOB-based as well as native storage for XML values
    - efficient storage, indexing, processing techniques
  - allows to include SQL requests in XQuery expressions, too
- Oracle 10g (Oracle XML-DB technical whitepaper, VLDB2004)
  - storage based on CLOBs or object-relational tables
    - additional indexing capabilities, XML query rewrite
  - protocols (ftp, WebDAV, …) for supporting file-oriented XML storage/access
- Microsoft SQL Server 2005 (MSDN whitepaper, VLDB2005)
  - stored as BLOB in an internal format
    - primary (B+ tree) and secondary indexes, query processing based on mapping to RDM

Summary

- Increasing importance of XML in combination with data management
  - flexible exchange of relational data using XML
  - managing XML data and documents
  - trend towards "hybrid" approaches for relational DBMS
- SQL/XML standard attempts to support the following
  - "Publish" SQL query results as XML documents
  - Ability to store and retrieve (parts of) XML documents with SQL databases
  - Rules and functionality for mapping SQL constructs to and from corresponding XML concepts
- Relies partly on XQuery standard
  - XML data model
  - queries over XML data
- Broad support by major SQL DBMS vendors

Appendix

Mapping SQL identifiers to XML Names

- SQL identifiers and XML Names have different rules:
  - SQL regular identifiers are case insensitive
  - SQL delimited identifiers can have characters like space and "<"
  - SQL identifiers use an implementation-defined character set
- Map SQL identifiers to XML Names by:
  - Encoding characters that cannot be included in an XML Name as "_xNNNN_" or "_xNNNNNN_" (N is hex digit)
  - "_x" is represented with "_x005F_x"
  - ":" is represented with "_x003A_"
  - For <identifier>s that begin with "XML" or "xml", encode the "X" or "x"
    - "XML…"will be encoded as "_x0078_ML…"

Examples

SQL <identifier>	XML Name
employee	EMPLOYEE
"employee"	employee
"hire date"	hire_x0020_date
"comp_xplan"	comp_x005F_xplan
xmlcol	_x0078_MLCOL

Mapping SQL data types to XML Schema data types

- Each SQL data type is mapped to an XML Schema data type; with the exception of:
  - Structured type
  - Reference type
- Appropriate XML Schema facets are used to constrain the range of values of XML Schema types to match the range of values of SQL types.
- XML Schema annotations may be used to keep SQL data type information that would otherwise be lost (optional).

Mapping Character Strings - Example

CHAR (10) CHARACTER SET LATIN1 COLLATION DEUTSCH	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:annotation> <xsd:appinfo> <sqlxml:sqltype name="CHAR" length="10" characterSetName="LATIN1" collation="DEUTSCH"/> </xsd:appinfo> </xsd:annotation> <xsd:length value="10"/> </xsd:restriction> </xsd:simpleType>

Mapping Integer - Example

INTEGER	<xsd:simpleType> <xsd:restriction base="xsd:integer"> <xsd:annotation> <xsd:appinfo> <sqlxml:sqltype name="INTEGER"/> </xsd:appinfo> </xsd:annotation> <xsd:maxInclusive value="2157483647"/> <xsd:minInclusive value="-2157483648"/> </xsd:restriction> </xsd:simpleType>

Mapping Unnamed Row Types

```
ROW
  (city  VARCHAR(30),
   state CHAR(2)
  )
```

```
<xsd:complexType name='ROW.001'>
   <xsd:sequence>
      <xsd:element name='CITY'
         nillable='true'
         type='VARCHAR_30'/>
      <xsd:element name='STATE'
         nillable='true'
         type='CHAR_2'/>
   </xsd:sequence>
</xsd:complexType>
```

Mapping Array Types - Example

`CHAR(12) ARRAY[4]`	`<xsd:complexType name='ARRAY_4.CHAR_12'>`
	`  <xsd:sequence>`
	`  <xsd:element name='element'`
	`  minOccurs='0' maxOccurs='4'`
	`  nillable='true' type='CHAR_12'/>`
	`  </xsd:sequence>`
	`</xsd:complexType>`

Mapping SQL values to XML

- Data type of values is mapped to corresponding XML schema types.
- Values of predefined types are first cast to a character string and then the resulting string is mapped to the string representation of the corresponding XML value.
- Values of numeric types with no fractional part are mapped with no periods.
- NULLs are mapped to either xsi:nil="true" or to absent elements, except for values of collection types whose NULLs are always mapped to xsi:nil="true".

Mapping SQL values to XML (continued)

- For scalar types it is straightforward:

SQL data type	SQL literal	XML value
VARCHAR (10)	'Smith'	Smith
INTEGER	10	10
DECIMAL (5,2)	99.95	99.95
TIME	TIME'12:30:00'	12:30:00
TIMESTAMP	TIMESTAMP'2001-09-14 11:00:00'	2001-09-14T11:00:00
INTERVAL HOUR TO MINUTE	INTERVAL'2:15'	PT02H15M

Mapping SQL values to XML (continued)

- ROW data type:

SQL data type:	`ROW (city VARCHAR(30), state CHAR(2))`
SQL value:	`ROW ('Long Beach', 'NY')`

XML Value: (in birth column)	`<BIRTH>`
	`  <CITY>Long Beach</CITY>`
	`  <STATE>NY</STATE>`
	`</BIRTH>`

Mapping SQL values to XML (continued)

- ARRAY data type:

SQL data type:	`CHAR(12) ARRAY[4]`

SQL value:	`ARRAY ['1-333-555-1212',`
	`NULL,`
	`'1-444-555-1212'`
	`]`

XML Value: (in phone column)	`<PHONE>`
	`<element>1-333-555-1212</element>`
	`<element xsi:nil="true"/>`
	`<element>1-444-555-1212</element>`
	`</PHONE>`