Chapter 9 – SQL/XML

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

Inhalt

SQL/XML

- Subproject of SQL standard
  - Part 14 “XML-related Specifications” of upcoming SQL 2003
- Goal: standardization of interaction/integration of SQL and XML
  - how to represent SQL data (tables, results, ...) in XML (and vice versa)
  - how to map SQL metadata (information schema) to XML schema (and vice versa)
  - ...
- Potential areas of use
  - "present" SQL data as XML
  - integration of XML data into SQL data bases
  - use XML for SQL data interchange
  - XML views over relational
    - possible foundation for XQuery

SQL/XML Features

- SQL/XML includes the following:
  - XML data type
    - Enables storage and retrieval of XML documents as typed values
  - Host language bindings for values of XML type
  - XML “publishing functions”
  - Mapping SQL Tables to XML Documents
  - Mapping SQL identifiers to XML Names and vice versa
  - Mapping SQL data types to XML Schema data types
  - Mapping SQL data values to XML

XML Data Type

- New SQL type "XML"
  - based on Infoset model
  - permits forests of infoset items at the top level
  - can have optimized internal representation (different from character string)
  - no comparison operators defined
- Conversion to/from character strings
  - XMLPARSE and XMLSERIALIZE functions
  - no validation performed
  - implicit conversion during host language interaction
- XML values can contain:
  - an XML Document
  - the content of an XML element
    - element
    - sequence of elements
    - text
    - mixed content – mixture of elements and text

XML Data Type *(continued)*

```
CREATE TABLE employee
   ( 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"/>`

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…"

Rules for mapping regular identifiers

- Each character in SQL names is mapped to its upper case equivalent.
  - employee from SQL is mapped to EMPLOYEE in XML.
- What if SQL names start with XML?
  - Option 1: "Partially escaped" mode - do nothing special
    - XMLTEXT from SQL mapped to XMLTEXT in XML.
  - Option 2: "Fully escaped mode" - map X to _x0058_
    - XMLTEXT from SQL mapped to _x0058F_MLTEXT in XML.

Rules for mapping delimited identifiers

- Each character in SQL names retains its case
  - "Employee" from SQL mapped to Employee in XML.
  - "Work_home" from SQL mapped to Work_home in XML.
- What if SQL names contain characters that are illegal in XML Names?
  - Map illegal characters to _xNNNN_ or _xNNNNNN_ , where N is a hex digit and NNNN or NNNNNNN is Unicode representation of the character.
    - "work@home" from SQL mapped to work_x0040_home in XML.
    - "last.name" from SQL mapped to last_x002E_name in XML.
- What if SQL names contain _x?
  - Escape the _ in _x:
    - "Emp_xid" from SQL mapped to Emp_005F_xid in XML.

Rules for mapping delimited identifiers *(continued)*

- What if SQL names started with xml in any case combinations?
  - Option 1: "Partially escaped" mode - do nothing special.
  - Option 2: "Fully escaped" mode - map x to _x0078_ or X to _x0058_:
    - "xmlText" from SQL mapped to _x0078F_mlText in XML.
- What if SQL names included a : ?
  - Option 1: "Partially escaped" mode - map only the leading colon to _x003A_
    - ":ab:cd" from SQL mapped to _x003A_ab:cd in XML.
  - Option 2: "Fully escaped mode" - map every colon to _x003A_
    - ":ab:cd" from SQL mapped to _x003A_ab_x003A_cd in XML.

Examples

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

Mapping XML Names to SQL identifiers

- Rules
  - Map all sequences of _xNNNN_ and _xNNNNNN_ to the corresponding Unicode character; if there is no corresponding Unicode character, map to a sequence of implementation-defined characters.
  - Put double quotes around the result to make an SQL delimited identifier; double each contained double quote.
    - employee from XML is mapped to "employee" in SQL.
    - EMPLOYEE from XML is mapped to "EMPLOYEE" in SQL.
    - work_x0040_home from XML mapped to "work@home" in SQL.
  - Map the resulting string to SQL_TEXT character set using implementation-defined mapping rules - raise an exception if the mapping is not possible.

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
  - Interval type
  - Datalink 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>`

SQL/XML "publishing functions"

- SQL functions/operators for generating XML constructs (elements, attributes, ...) within a query
  - XMLCONCAT concatenates XML values
  - XMLELEMENT generates an XML element
  - XMLFOREST generates multiple elements
  - XMLAGG aggregates XML across multiple tuples
  - XMLROOT creates XML element by modifying a root information item
- Example:

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

XMLELEMENT

- Produces an XML value that corresponds to an XML element, given:
  - An SQL identifier that acts as its name
  - An optional list of named expressions that provides names and values of its attributes, and
  - An optional list of expressions that provides its content.Attributes in XMLELEMENT

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

==>
```

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

XMLATTRIBUTES (within XMLELEMENT)

- Attribute specifications must be bracketed by XMLATTRIBUTES keyword and must appear as the second argument of XMLELEMENT.
- 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"/>

XMLELEMENT (continued)

- XMLELEMENT can produce nested element structures:

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

==>

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

XMLELEMENT (continued)

- XMLELEMENT can produce elements with mixed content:

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

==>

ID	result
1001	`<Emp>` `Employee <name>Smith</name>` `was hired on <hiredate>2000-05-24</hiredate>` `</Emp>`
1006	`<Emp>` `Employee <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>`

XMLFOREST

- Produces a sequence of XML elements given named expressions as arguments.
- Element can have an explicit name:
  - **`e.salary AS "empSalary"`**
- Element can have an implicit name, if the expression is a column reference:
  - **`e.salary`**

```
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>`

XMLCONCAT

- produces an XML value given 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	`<first>Mary</first>` `<last>Martin</last>`

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

==>
```

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

XMLAGG and ORDER BY

```
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

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>
```

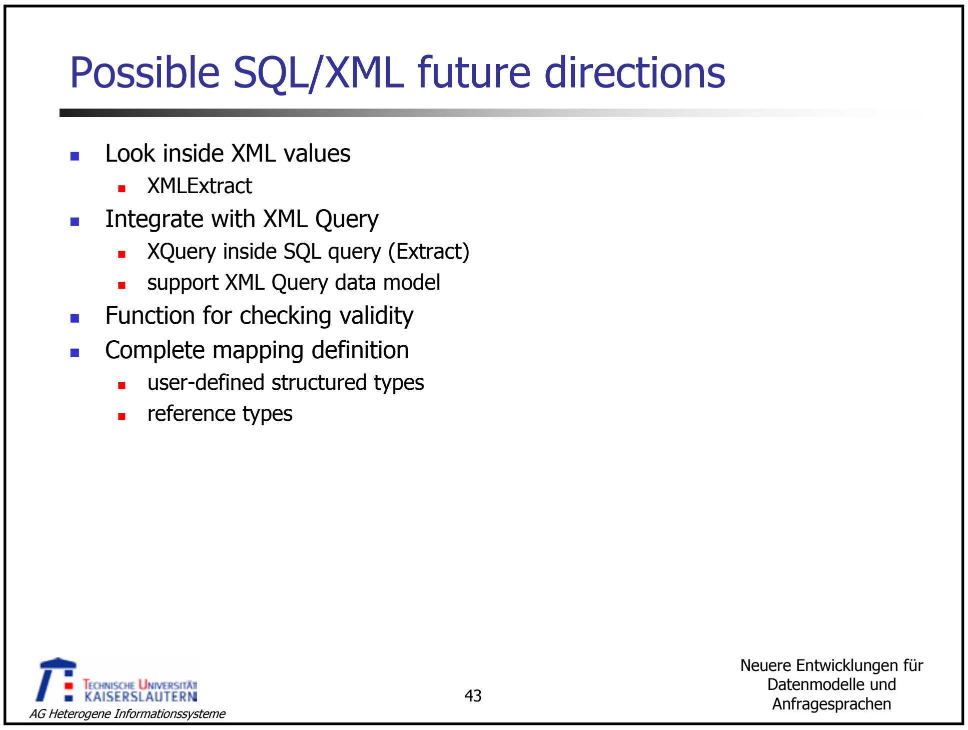
Corresponding XML-Schema document

```
<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>
```



Possible SQL/XML future directions

- Look inside XML values
  - XMLExtract
- Integrate with XML Query
  - XQuery inside SQL query (Extract)
  - support XML Query data model
- Function for checking validity
- Complete mapping definition
  - user-defined structured types
  - reference types