Chapter 9

SQL/XML

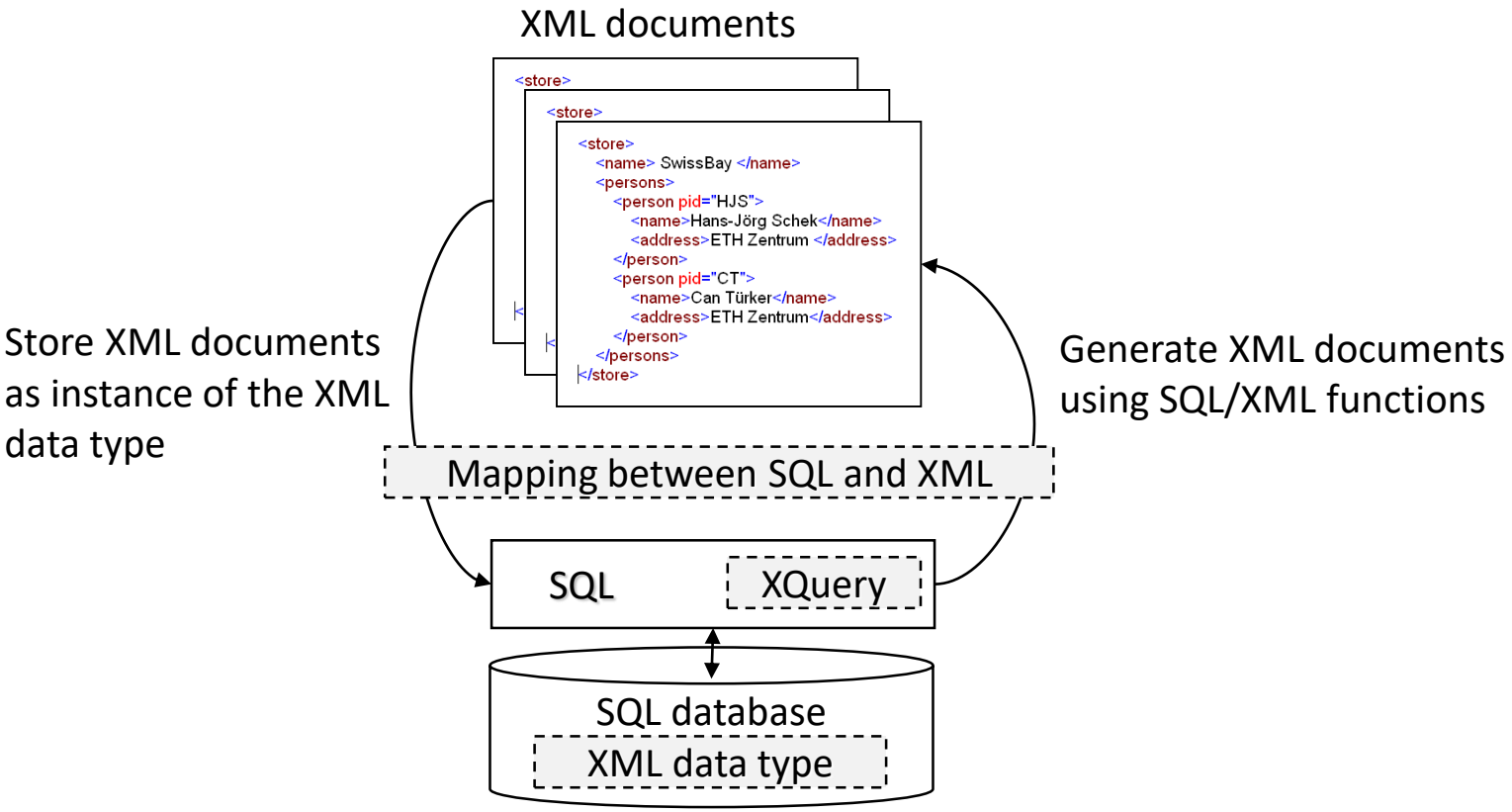
Data Type XML

Mapping SQL and XML

SQL/XML Functions

Overview

- **Data type XML in SQL together with corresponding SQL/XML functions**
- **Mapping between SQL and XML**
- **Embedding XQuery in SQL**



XML Type

- **Create, update, and delete XML values in SQL**
- **Declaration XML type:**

```
XML [({DOCUMENT | CONTENT | SEQUENCE}
    [({ANY | UNTYPED | XMLSCHEMA <schema> [[NAMESPACE <space>] ELEMENT <name>]})])]
```

```
<schema> ::= URI <namespace> [LOCATION <schema-location>]
           | NO NAMESPACE  [LOCATION <schema-location>]
           | ID <registered-schema-name>
```

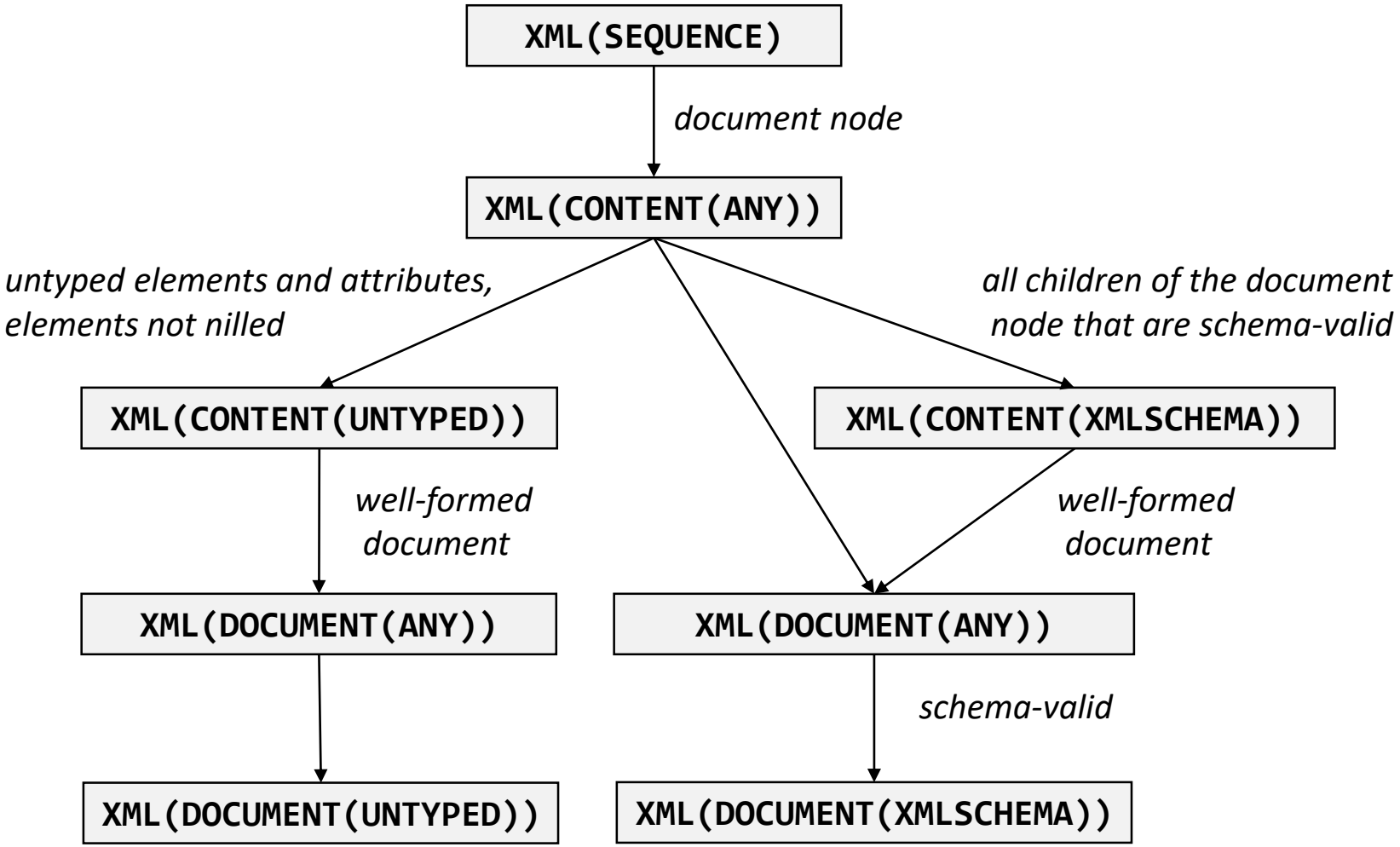
  - XML value is NULL or an XQuery sequence
  - XML value is associated with one or more types
  - XML values are not comparable (create user-defined ordering if comparisons are needed)
  - XML functions create XML values from SQL data and other XML values

- **Example: Defining a table column of type XML**

```
CREATE TABLE Employee (
  Name   VARCHAR(20),
  Salary NUMERIC(12,2),
  CV     XML
)
```

Employee	Name	Salary	CV
	Joe	2000	<cv>Joe…</cv>
	Jim	3500	<cv>Jim…</cv>

XML Types and Values



Schema Registration

- **Register XML schema before you can use it**
- **Schema registration is implementation-dependent**
- **Registered XML schema must have a unique name**
- **Pre-defined (implicitly registered) schemata**

```
xs     http://www.w3.org/2001/XMLSchema
xsi    http://www.w3.org/2001/XMLSchema-instance
sqlxml http://standards.iso.org/iso/9075/2003/sqlxml
```

XML versus SQL

- **hierarchical vs. flat**
- **loose vs. fixed schema**
- **case-sensitive names vs. case-insensitive names**
- **different type concepts**
- **strict Unicode vs. heterogeneous encodings**
- **ordered vs. unordered**

Mapping between SQL and XML

- **Mapping SQL to XML**
  - SQL character sets to XML Unicode (implementation-dependent)
  - SQL identifiers to XML names
  - SQL data types to XML schema data types
  - SQL values to XML values
  - SQL tables to XML documents and XML schema documents
  - SQL schemas to XML documents and XML schema documents
  - SQL catalogue to XML documents and XML schema documents
- **Mapping XML to SQL**
  - XML Unicode to SQL character sets (implementation-dependent)
  - XML names to SQL identifiers

Mapping between SQL Identifiers and XML Names

- **Character sets**
  - XML based on Unicode, SQL not
  - Mapping between SQL character set and Unicode is implementation-dependent
- **Names**
  - Some SQL names are not allowed as XML name
  - Masking needed, e.g., for colons or leading xml

```
Ω              → _x03A9_  (the Unicode value)
Salary:FY2000  → Salary_x003A_FY2000
Work@home      → Work_x0040_home
Home_Town      → Home_x005F_Town
```

  - SQL name mapped to XML name using upper letters

```
Employee       → <EMPLOYEE>…</EMPLOYEE>
```

Mapping Data Type Names

- **Data type parameters appended to XML names**
  - In case of TIME, TIMESTAMP, INTERVAL precision and time zone details appended

```
DECIMAL_9_2
VARCHAR_10
BLOB_4000
…
```

- **DOMAIN *d* in schema *s* of catalogue *c*:** `d.s.c`
- **DISTINCT TYPE *t* analogous to Domain:** `t.s.c`
- **ARRAY of type *t* with maximum cardinality m:** `ARRAY_m.t`
- **MULTISET analogous to ARRAY** `MULTISET_m.t`
- **ROW types are implementation-dependent; must begin with prefix ROW**
- **Interval types, structured types, reference types are not mapped**

Mapping Data Types and Values

- **Map SQL types to corresponding XML schema types**
  - Select “closest” XML schema type
  - Use facets to restrict the domain of this XML schema type as far as needed

```
length          maxLength          characterSet collation
precision       scale              minExponent  maxExponent
userPrecision   leadingPrecision   maxElements  final
catalogName     schemaName         domainName   typeName
mappedType      mappedElementType
```

  - Other SQL characteristics such as collation or character sets are added as annotations

- **Value mapping determined by mapping of the data types**
  - Exception numeric values: fraction is eliminated if it is zero

```
170407.0 → 170407
```

- **Mapping special (reserved) characters such as < or &**

```
< → &lt;
```

The sqlxml Namespace (1)

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLschema"
  targetNamespace="http://www.iso-standards.net/9075/2001/12/XMLschema"
  xmlns:sqlxml="http://www.iso-standards.net/9075/2001/12/XMLschema">
  <xsd:annotation>
    <xsd: documentation>
      This document contains the definitions and annotations
      as defined in ISO/IEC 9075-14:2003 (SQL/XML).
    </xsd: documentation>
  </xsd:annotation>
  <xsd:simpleType name= "kindKeyword">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="PREDEFINED"/>
      <xsd:enumeration value="DOMAIN"/>
      <xsd:enumeration value="ROW"/>
      <xsd:enumeration value="ARRAY"/>
      <xsd:enumeration value="MULTISET"/>
      <xsd:enumeration value="DISTINCT"/>
    </xsd:restriction>
  </xsd:simpleType>
```

The sqlxml Namespace (2)

```
<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="NUMERIC"/>
    <xsd:enumeration value="DECIMAL"/>
    <xsd:enumeration value="INTEGER"/>
    <xsd:enumeration value="SMALLINT"/>
    <xsd:enumeration value="BIGINT"/>
    <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="TIMESTAMP"/>
    <xsd:enumeration value="INTERVAL YEAR"/>
    <xsd:enumeration value="INTERVAL MONTH"/>
    <xsd:enumeration value="INTERVAL DAY"/>
    <xsd:enumeration value="INTERVAL HOUR"/>
    <xsd:enumeration value="INTERVAL MINUTE"/>
    <xsd:enumeration value="INTERVAL SECOND"/>
  </xsd:restriction>
</xsd:simpleType>
```

The sqlxml Namespace (3)

```
<xsd:element name="sqlType">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="field" type="fieldType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="kind" type="sqlxml:kindKeyword"/>
    <xsd:attribute name="name" type="sqlxml:typeKeyword" use="optional"/>
    <xsd:attribute name="length" type="xsd:integer" use="optional"/>
    <xsd:attribute name="maxLength" type="xsd:integer" use="optional"/>
    <xsd:attribute name="characterSet" 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"/>
    <xsd:attribute name="minExponent" type="xsd:integer" use="optional"/>
    <xsd:attribute name="maxExponent" type="xsd:integer" use="optional"/>
    <xsd:attribute name="userPrecision" type="xsd:integer" use="optional"/>
    <xsd:attribute name="leadingPrecision" type="xsd:integer" use="optional"/>
    <xsd:attribute name="maxElements" type="xsd:integer" use="optional"/>
    <xsd:attribute name="catalogName" type="xsd:string" use="optional"/>
    <xsd:attribute name="schemaName" type="xsd:string" use="optional"/>
    <xsd:attribute name="domainName" type="xsd:string" use="optional"/>
    <xsd:attribute name="typeName" type="xsd:string" use="optional"/>
    <xsd:attribute name="mappedType" type="xsd:string" use="optional"/>
    <xsd:attribute name="mappedElementType" type="xsd:string" use="optional"/>
    <xsd:attribute name="final" type="xsd:boolean" use="optional"/>
  </xsd:complexType>
</xsd:element>
```

The sqlxml Namespace (4)

```
  <xsd:simpleType name="objectType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="CATALOG"/>
      <xsd:enumeration value="SCHEMA"/>
      <xsd:enumeration value="BASE TABLE"/>
      <xsd:enumeration value="VIEWED TABLE"/>
      <xsd:enumeration value="CHARACTER SET"/>
      <xsd:enumeration value="COLLATION"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="fieldType">
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="mappedType" type="xsd:string"/>
  </xsd:complexType>

  <xsd:element name="sqlname">
    <xsd:complexType>
      <xsd:attribute name="type" type="sqlxml:objectType" use="required"/>
      <xsd:attribute name="catalogName" type="xsd:string"/>
      <xsd:attribute name="schemaName" type="xsd:string"/>
      <xsd:attribute name="localName" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Mapping Basic SQL Data Types (1)

```
CHARACTER(20) CHARACTER SET LATIN1 COLLATION DEUTSCH
```

```
<xsd:simpleType name="CHAR_20">
 <xsd:annotation>
  <xsd:appinfo>
   <sqlxml:sqltype
    kind="PREDEFINED"
    name="CHAR"
    length="20"
    characterSetName="LATIN1"
    collation="DEUTSCH"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:restriction base="xsd:string">
  <xsd:length value="20"/>
 </xsd:restriction>
</xsd:simpleType>
```

Remember original SQL type using SQLXML namespace

Map to XML schema type and use facets to define it as close as possible

Mapping Basic SQL Data Types (2)

```
NUMERIC(12,2)
```

```
<xsd:simpleType name="NUMERIC_12_2">
 <xsd:annotation>
  <xsd:appinfo>
   <sqlxml:sqltype
    kind="PREDEFINED"
    name="NUMERIC"
    precision="12"
    scale="2"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:restriction base="xsd:numeric">
  <xsd:totalDigits value="12"/>
  <xsd:fractionDigits value="2"/>
 </xsd:restriction>
</xsd:simpleType>
```

Different attributes to exactly describe the SQL data types

Mapping Basic SQL Data Types (3)

SMALLINT

```
<xsd:simpleType name="SMALLINT">
 <xsd:annotation>
  <xsd:appinfo>
   <sqlxml:sqltype
    kind="PREDEFINED"
    name="SMALLINT"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:restriction base="xsd:integer">
  <xsd:maxInclusive value="32767"/>
  <xsd:minInclusive value="-32768"/>
 </xsd:restriction>
</xsd:simpleType>
```

DATE

```
<xsd:simpleType name="DATE">
 <xsd:annotation>
  <xsd:appinfo>
   <sqlxml:sqltype
    kind="PREDEFINED"
    name="DATE"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:restriction base="xsd:date">
  <xsd:pattern value=
   "\p{Nd}{4}-\p{Nd}{2}-\p{Nd}{2}"/>
 </xsd:restriction>
</xsd:simpleType>
```

Summary: Mapping Basic SQL Data Types

SQL	XML schema type	Restriction
BOOLEAN	xsd:boolean	
CHAR	xsd:string	xsd:length
VARCHAR, CLOB	xsd:string	xsd:maxLength
BLOB	xsd:hexbinary, xsd:base64binary	xsd:maxLength
SMALLINT, INT, BIGINT	xsd:integer	xsd:minInclusive, xsd:maxInclusive
NUMERIC, DECIMAL	xsd:decimal	xs:totalDigits, xs:fractionDigits
REAL, FLOAT	xsd:float	
DOUBLE PRECISION	xsd:double	
DATE	xsd:date	xsd:pattern
TIME	xsd:time	xsd:pattern
TIMESTAMP	xsd:dateTime	xsd:pattern
INTERVAL	xsd:duration	xsd:pattern

Mapping SQL Domains

```
CREATE DOMAIN UZH.DBS.Jobs CHAR(4)
CHECK(VALUE IN ('Prof', 'Assi'));
```

```
<xsd:simpleType name="DOMAIN.UZH.DBS.Jobs">
 <xsd:annotation>
  <xsd:appinfo>
   <sqlxml:sqltype
    kind="DOMAIN"
    catalogName="UZH"
    schemaName="DBS"
    typeName="Jobs"
    mappedType="CHAR_4"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:restriction base="CHAR_4">
  <xsd:enumeration value="Prof"/>
  <xsd:enumeration value="Assi"/>
 </xsd:restriction>
</xsd:simpleType>
```

Mapping SQL Tuple Types

```
ROW(Firstname  VARCHAR(25),
    Lastname VARCHAR(35))
```

```
<xsd:complexType name="ROW.Name">
 <xsd:annotation>
  <xsd:appinfo>
   <sqlxml:sqltype kind="ROW">
    <sqlxml:field name="FIRSTNAME" mappedType="VARCHAR_25"/>
    <sqlxml:field name="LASTNAME"  mappedType="VARCHAR_35"/>
   </sqlxml:sqltype>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:sequence>
  <xsd:element name="FIRSTNAME" type="VARCHAR_25"/>
  <xsd:element name="LASTNAME"  type="VARCHAR_35"/>
 </xsd:sequence>
</xsd:complexType>
```

Mapping SQL Array Types

```
DECIMAL(12,2) ARRAY[10]
```

```
<xsd:complexType name="ARRAY_10.DECIMAL_12_2">
 <xsd:annotation>
  <xsd:appinfo>
   <sqlxml:sqltype
    kind="ARRAY"
    maxElements="10"
    mappedElementType="NUMERIC_12_2"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:sequence>
  <xsd:element name="Element"
   minOccurs="0"
   maxOccurs="10"
   type="NUMERIC_12_2">
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

Mapping SQL Multiset Types

```
CHARACTER(20) MULTISET
```

```
<xsd:complexType name="MULTISET.CHAR_20">
 <xsd:annotation>
  <xsd:appinfo>
   <sqlxml:sqltype
    kind="MULTISET"
    mappedElementType="CHAR_20"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:sequence>
  <xsd:element
   name="Element"
   minOccurs="0"
   maxOccurs="unbounded"
   type="CHAR_20">
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

Set cardinality not restricted

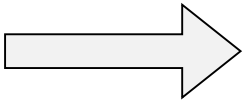
Mapping SQL Distinct Types

```
CREATE TYPE ETH.DBS.CHF AS NUMERIC(12,2) FINAL
```

```
<xsd:simpleType name="UDT.ETH.DBS.CHF">
 <xsd:annotation>
  <xsd:appinfo>
   <sqlxml:sqltype
    kind="DISTINCT"
    catalogName="ETH"
    schemaName="DBS"
    typeName="CHF"
    mappedType="NUMERIC_12_2"
    final="true"/>
   </xsd:appinfo>
  </xsd:annotation>
 <xsd:restriction base="NUMERIC_12_2"/>
</simpleType>
```

Mapping SQL Tables

```
CREATE TABLE Employee
(
 Name    CHAR(20),
 Salary NUMERIC(12, 2)
);
```



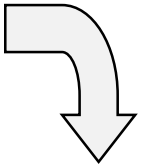
Map table schema to XML schema document

```
<xsd:complexType name="ROW.EMPLOYEE">
 <xsd:sequence>
  <xsd:element name="NAME" type="CHAR_20"/>
  <xsd:element name="SALARY type="NUMERIC_12_2"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TABLE.EMPLOYEE">
 <xsd:annotation>
  <xsd:appinfo>
   <sqlxml:sqlname
    type="BASE TABLE"
    localName="EMPLOYEE"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:sequence>
  <xsd:element
   name="row" type="ROW.EMPLOYEE"
   minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="EMPLOYEE" type="TABLE.EMPLOYEE"/>
```

Name	Salary
Joe	2000
Jim	3500



Map table content to XML document

```
<EMPLOYEE>
 <row>
  <NAME>Joe</NAME>
  <SALARY>2000</SALARY>
 </row>
 <row>
  <NAME>Jim</NAME>
  <SALARY>3500</SALARY>
 </row>
</EMPLOYEE>
```

XML Type Operators

SQL → XML

- **XMLCOMMENT** creates XML comment node
- **XMLELEMENT** creates XML element from SQL values
- **XMLFOREST** creates XML element sequence from a table
- **XMLPI** creates processing instruction node
- **XMLTEXT** creates text node
- **XMLPARSE** maps SQL text to XML value
- **XMLAGG** aggregates XML values to groups

*** → XML**

- **XMLQUERY** evaluates an XQUERY expression

XML → XML

- **XML document** creates document node from an XML value
- **XMLCONCAT** concatenates XML values
- **XMLVALIDATE** validates XML value against a schema and delivers a copy of the value

XML → SQL

- **XMLTABLE** transforms XQuery result to SQL table
- **XMLITERATE** transforms XQuery sequence to SQL table
- **XMLSERIALIZE** transforms XML value to SQL text

XMLELEMENT

```
XMLELEMENT(NAME <element-name>
           [, XMLATTRIBUTES(<attribute-list>)]
           [, <value-expression-list>]
           [, OPTION <content-option>]
           [, <returning-clause>])

<attribute> := <value-expression> [AS <attribute-name>]

<content-option> := {NULL|EMPTY|ABSENT|NIL} ON NULL | NIL ON NO CONTENT

<returning-clause>:= RETURNING {CONTENT|SEQUENCE}
```

Element content

Renaming

Document Element

creates an XML element

Example:

Employee	Name	Salary
	Joe	2000
	Jim	3500

```
SELECT XMLELEMENT(NAME "Emp",
                  XMLATTRIBUTES(Salary),
                  Name) AS Element
FROM Employee;
```

Element
<Emp SALARY="2000">Joe</Emp>
<Emp SALARY="3500">Jim</Emp>

XMLFOREST

```
XMLFOREST(<element-list>
          [, OPTION <content-option>]
          [, <returning-clause>])

<element> := <value-expression> [AS <element-name>]
```

creates XML element sequence

Example:

Employee	Name	Salary
	Joe	2000
	Jim	3500

```
SELECT XMLFOREST(Name,
 XMLELEMENT(NAME "Euro", 12*Salary) AS Income)
 AS Salary
FROM Employee;
```

Salary
<NAME>Joe</NAME> <Income><Euro>24000</Euro></Income>
<NAME>Jim</NAME> <Income><Euro>42000</Euro></Income>

XMLCONCAT

```
XMLCONCAT(<xml-value-expression-list> [, <returning-clause>])
```

concatenates XML elements to a sequence

Example:

Employee	Name	Salary
	Joe	2000
	Jim	3500

```
SELECT XMLCONCAT(
         XMLELEMENT(NAME "Employee", Name),
         XMLELEMENT(NAME "Euro", 12*Salary))
       AS Salary
FROM Employee;
```

Salary
<Employee>Joe</Employee> <Euro>24000</Euro>
<Employee>Jim</Employee> <Euro>42000</Euro>

XMLQUERY (1)

```
XMLQUERY(<xquery-expression> [PASSING <xml-argument-list>]
        [<returning-clause>] {NULL|EMPTY} ON EMPTY)

<xml-argument> := <xml-value-expression> AS "<xquery-variable-name>"
```

evaluates given XQuery and returns an XML value

Example:

Employee	Name	Salary
	Joe	2000
	Jim	3500

```
SELECT XMLQUERY(
 '<Emp Name="{$n}"><Salary>{$s}</Salary></Emp>'
 PASSING Name AS "n", 12*Salary AS "s"
 NULL ON EMPTY) AS Element
FROM Employee;
```

Element
<Emp Name="Joe"><Salary>24000</Salary></Emp>
<Emp Name="Jim"><Salary>42000</Salary></Emp>

XMLQUERY can be used to define XML views on SQL data

XMLQUERY (2)

```
CREATE TABLE Report (Id INTEGER, Content XML);
```

Report	Id	Content
	1	<report><author>Johnny</author><date>13.03.2014</date><title>Dead Man Walking</title></report>
	2	<report><author>Jim</author><title>Beam Me Up</title><date>17.03.2014</date></report>
	3	<report><title>Great Single Malts</title><author>Jack</author><author>Jim</author><date>01.04.2014</date></report>

```
SELECT Id, XMLQUERY('for $a in $c//author return <writer>{$a/text()}</writer>'
                    PASSING content AS "c") AS Writers
FROM Report;
```

Id	Writers
1	<writer>Johnny</writer>
2	<writer>Jim</writer>
3	<writer>Jack</writer><writer>Jim</writer>

XMLTABLE

```
XMLTABLE (<xquery-expression> PASSING <xml-argument-list>
                              COLUMNS <column-path-list>)

<column-path> := <column-name> <column-type> PATH <xquery-expression>
```

creates SQL table from an XQuery expression

Example:

```
SELECT Id, t.*
FROM Report r,
XMLTABLE('for $root in $c
          where $root//author/text() = "Jim"
          return $root'
          PASSING r.Content AS "c"
          COLUMNS Report  VARCHAR(30) PATH '/report/title',
                  Written VARCHAR(10) PATH '/report/date') AS t;
```


Id	Report	Written
2	Beam Me Up	17.03.2014
3	Great Single Malts	01.04.2014

XMLTABLE can be used to define SQL views on XML data

XMLAGG

```
XMLAGG(<xml-value-expression>[ORDER BY <sort-expression-list>])
```

aggregates XML elements of a group

Example:

Employee	Name	Salary
	Joe	2000
	Jim	3500
	Jim	5000

```
SELECT Name, XMLAGG(XMLELEMENT(NAME "Euro", Salary))
             AS Income
FROM Employee
GROUP BY Name;
```

Name	Income
Joe	<Euro>2000</Euro>
Jim	<Euro>3500</Euro> <Euro>5000</Euro>

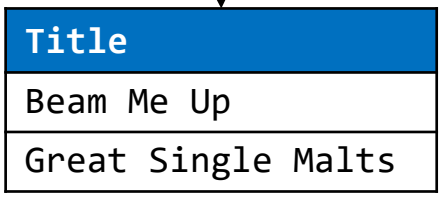
XMLEXISTS

```
XMLEXISTS (<xquery-expression> PASSING <xml-argument-list>)
```

is true if XQuery expression yields a non-empty result

Example:

```
SELECT XMLQUERY('$c//title/text()' PASSING content AS "c") AS Title
FROM Report
WHERE XMLEXISTS('$c//author[text()="Jim"]' PASSING Content AS "c");
```



Title
Beam Me Up
Great Single Malts

XMLPARSE

```
XMLPARSE ({DOCUMENT | CONTENT} <string-value-expression>
          {STRIP | PRESERVE} WHITESPACE)
```

creates XML value from a String value

Example:

```
SELECT XMLPARSE(DOCUMENT '<id>'||id||'</id>' PRESERVE WHITESPACE)
       AS XMLdocument
FROM Report;
```

XMLdocument
<id>1</id>
<id>2</id>
<id>3</id>

```
SELECT XMLPARSE(DOCUMENT 'Report <id>'||id||'</id>' PRESERVE WHITESPACE)
       AS XMLsequence
FROM Report;
```

XMLsequence
Report <id>1</id>
Report <id>2</id>
Report <id>3</id>

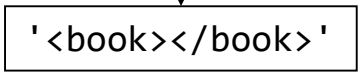
XMLSERIALIZE

```
XMLSERIALIZE({DOCUMENT | CONTENT} <xml-value-expression> AS <sql-type>
[VERSION <XML version>] [(INCLUDING | EXCLUDING) XMLDECLARATION])
```

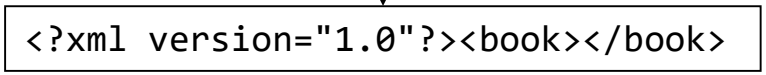
converts XML value to SQL string or BLOB

Examples:

```
XMLSERIALIZE(XMLQUERY('<book></book>') AS CHAR(20))
```




```
XMLSERIALIZE(DOCUMENT XMLQUERY('document {<book></book>}') AS CHAR(100)
VERSION '1.0' INCLUDING XMLDECLARATION)
```



XMLCOMMENT, XMLPI, XMLTEXT

```
XMLCOMMENT (<string-value-expression> [, <returning-clause>])
```

creates XML comment node (CONTENT is default)

```
XMLPI (NAME <identifier> [, <returning-clause>]))
```

creates XML processing instruction node

```
XMLTEXT (<string-value-expression> [, <returning-clause>]))
```

creates XML text node

Examples:

```
XMLCOMMENT('hello')
```
→
```
<!--hello-->
```

```
XMLPI(NAME php, 'echo "hello world" ')
```
→
```
<?PHP echo "hello world"?>
```

```
XMLTEXT('hello world')
```
→
```
hello world
```

XMLDOCUMENT

```
XMLDOCUMENT (<xml-expression> [RETURNING SEQUENCE])
```

creates XML document node from an XML value

- result is NULL if the XML value is null
- result type is **XML(SEQUENCE)** or **XML(CONTENT)**

Note:

```
XMLDOCUMENT (expr)
≡ document { $expr }
≡ XMLQUERY ('document {$E}' PASSING BY REF <xml-expression> AS "E")
```

XML Type Predicates

```
<xml-value> IS [NOT] CONTENT
```

returns false [true] if XML value is [not] a document node

```
<xml-value> IS [NOT] DOCUMENT
```

returns true [false] if XML value is [not] a (well-formed) document

```
<xml-value> IS [NOT] VALID {SEQUENCE | CONTENT | DOCUMENT}
[ACCORDING TO XMLSCHEMA <schema>]
```

returns true if XML value is [in]valid w.r.t. the given schema

```
XMLVALIDATE(<type> <xquery-expression>
ACCORDING TO XMLSCHEMA <schema> [<element>])

<schema> ::= URI <namespace> [LOCATION <schema-location>]
           | NO NAMESPACE  [LOCATION <schema-location>]
           | ID <schema-name>

<element> ::= ELEMENT <element-name>
            | NAMESPACE <namespace> ELEMENT [<element-name>]
            | NO NAMESPACE ELEMENT [<element-name>]
```

returns true if XQuery expression yields a value of type XML(<type>)

XMLCAST

- **SQL to XML**

```
XMLCAST(<sql-value-expression> AS <xml-type>)
```

  - converts SQL value to an XML value and transforms it then with XMLPARSE (CONTENT … PRESERVE WHITESPACE) to a document node

- **XML to SQL**

```
XMLCAST(<xml-value-expression> AS <sql-type>)
```

  - removes the document node
  - applies fn:data() to each node of the result
  - converts the result to the corresponding XML type
  - maps this XML to the corresponding SQK value

- **XML to XML (compatibility of type modifier and identical schema assumed)**

```
CAST(<xml-value-expression>) AS XML(<type-modifier>) BY {REF | VALUE}
```

  - **REF** preserves node identities, **VALUE** looses them

Conclusions

- **We know how SQL/XML defines common usage of SQL and XML**
- **Data Type XML in SQL**
  - Store XML data in SQL database
  - Support XML functionality by database engine
- **SQL-XML mappings for data types and values**
- **SQL/XML functions including seamless integration of XQuery**
  - XML views on SQL data
  - Querying SQL and XML data together
- **Open issues:**
  - Updates on XML values (instead of just replacing entire XML values)
  - Full-text search on XML values
  - No framework for mapping constraints
  - No specification of (XML) indexes
- **We now want to know how leadings Database Systems support SQL/XML**