Kapitel 12
SQL und XML

12.1 XML: “Extensible Markup Language”

... mehr als nur “Language”: Datenmodell, viele Sprachen

- Instantiierung von SGML (vgl. HTML)
- Semantische Tags

⇒ wer HTML kennt, weiß, wie XML “aussieht”.

- Baumstruktur
- Elemente (Name, Attribute und Inhalt)
- rekursiver Aufbau

⇒ abstrakter Datentyp mit Konstruktoren und Operationen.

- Navigation im Baum
- vgl. Pfadausdrücke in Java, OQL, SQL
  (z.B. x.coordinates.longitude)
- Pfadausdrücke in Unix:
  (z.B. /home/may/teaching/dbp/folien.tex)

⇒ Adressierungssprache “XPath”

XML: BEISPIEL

```
<country id="D" capital="cty-Germany-Stuttgart">
    <name>Germany</name>
    <total_area>356910</total_area>
    <population>83536115</population>
    <encompassed continent="europe">100</encompassed>
    <ethnicgroup name="German">95.1</ethnicgroup>
    <ethnicgroup name="Italians">0.7</ethnicgroup>
    <religion name="Roman Catholic">37</religion>
    <religion name="Protestant">45</religion>
    <language name="German">100</language>
    <border country="F">451</border>
    <border country="A">784</border>
    <border country="CZ">646</border>
    :
```

```
  <province id="prov-Germany-Baden-Wuerttemberg">
    <name>Baden Wuerttemberg</name>
    <area>35742</area>
    <population>10272069</population>
    <city is_state_cap="yes" id="cty-Germany-Stuttgart">
      <name>Stuttgart</name>
      <longitude>9.1</longitude>
      <latitude>48.7</latitude>
      <population year="95">588482</population>
    </city>
    <city id="cty-Germany-Mannheim">
      <name>Mannheim</name>
      :
    </city>
    :
  </province>
  <province id="prov-Germany-Berlin">
    <name>Berlin</name>
    <area>889</area>
    <population>3472009</population>
    <city is_country_cap="yes" is_state_cap="yes"
          id="cty-Germany-Berlin">
      <name>Berlin</name>
      <longitude>13.3</longitude>
      <latitude>52.45</latitude>
      <population year="95">3472009</population>
    </city>
  </province>
  :
</country>
```

XML

(Siehe Vorlesung “Semistrukturierte Daten und XML”)

- Verwendung:
  - Dokumente
  - Datenaustausch
  - Datenspeicherung
- sehr flexibles “Datenmodell”: DOM-API
  rekursiv definierte Baumstruktur aus
  - Elementen,
  - Attributen und
  - Textknoten.
- Schema: DTD (Document Type Description), XML Schema
- Erweiterungen: XPath, XPointer, XLink
- Anfragesprache: XQuery
- Transformationssprache: XSL/XSLT
- Basis des Semantic Web: RDF, RDF Schema, OWL ...

12.2 Der SQL/XML bzw. SQLX Standard – Kombination relationaler Daten und XML

- Abbildung von relationalen Daten nach XML
- Speicherung von XML-Daten in RDBMS
- Entwurf eines ISO-Standards seit 2003: `www.sqlx.org`
- SQL-Objektdatentyp “XMLType”
  - mit entsprechenden Konstruktoren für XML-Strukturen,
  - und Zugriffsmethoden (basierend auf den Standards der XML-Welt),
  - benutzbar von SQL und innerhalb von PL/SQL.
- zum Teil noch unvollständig und überraschend ...

SQLX und Oracle: Literatur

- Oracle TechNet (oft eine wertvolle Informationsquelle): `http://www.oracle.com/technology/tech/xml/index.html`
- Oracle XML DB Developer's Guide: `http://ap34.ifi.informatik.uni-goettingen.de/oracle-doc/appdev.102/b14259/toc.htm`
- (mit kleinen Unterschieden zum SQL/XML Standard)

12.2.1 “XML” als SQL-Objekttyp

XML/XMLType: ein vordefinierter *SQL*-Objekttyp, der XML-Daten speichert.

- Als Zeilenobjekte:

```
CREATE TABLE Mondial OF XMLType;
CREATE TABLE CountryXML OF XMLType;
```

- Als Spaltenobjekte:

```
CREATE TABLE CityXML
  (name VARCHAR2(35),
   province VARCHAR2(32),
   country VARCHAR2(3),
   population XMLType,
   coordinates XMLType);
```

[Filename: SQLX/cityxmltable.sql]

EINFACHER KONSTRUKTOR: XMLTYPE(...)

- Syntaktische Einbindung genauso wie für selbstdefinierte Objekttypen,
- XML-Inhalt als ASCII gegeben:

```
INSERT INTO tablename
  VALUES (... , XMLType('XML in ASCII-Notation') ...)
```

```
INSERT INTO cityXML
VALUES('Northeim','Niedersachsen','D',
 XMLType('<population year="2004">10000</population>'),
 XMLType('<coordinates><longitude>10</longitude>
                       <latitude>51.7</latitude>
          </coordinates>'));
```

[Filename: SQLX/cityxmltuple.sql]

IMPORT VON KOMPLETTEN XML-FILES: ALLGEMEIN

(Standardmethode, wenn es nur eine Directory gibt, aus der XML-Dokumente geladen werden sollen)

- XML-Werte können als Dateien importiert werden.
- Anlegen einer Directory, aus der die Dateien geladen werden (Admin):

```
CREATE OR REPLACE DIRECTORY XMLDIR AS '/db' ;
CREATE OR REPLACE DIRECTORY XMLDIR AS '/home/bla/...' ;
```

  Es gibt dann ein SQL Directory Object “XMLDIR”.
  (Kann wie üblich mit `DROP directory XMLDIR` gelöscht werden).
- XML-File (z.B. m.xml) dorthin kopieren
  - darf keine Referenz auf eine DTD enthalten!
  - muss für alle lesbar sein: `chmod` *filename* `644`
- Benutzung des xdb_utilities Package (muss separat installiert werden), um Files in die Datenbank zu laden:

```
INSERT INTO mondial
VALUES(xdb_utilities.getXMLfromfile('m.xml','XMLDIR'));
```

```
set long 10000 ;
SELECT * FROM mondial;
```

IMPORT VON KOMPLETTEN XML-FILES: LOKAL

- Um beliebige XML-Files zu importieren steht *lokal* im Praktikum eine Methode `system.getxml('`*http-url*`')` zur Verfügung:

```
SELECT system.getxml('
  http://www.dbis.informatik.uni-goettingen.de/
         Teaching/DBP/XML/mondial.xml') FROM dual;
```

bzw. zum Einfügen in eine Tabelle:

```
INSERT INTO mondial VALUES(
 system.getxml(
  'http://www.dbis.informatik.uni-goettingen.de' ||
  '/Teaching/DBP/XML/mondial.xml'));
```

[Filename: SQLX/insertmondial.sql]

- XML-Instanz darf keine Referenz auf eine DTD enthalten!
- z.B. aus dem eigenen Homedirectory ...
- muss für alle lesbar sein: `chmod` *filename* `644`

Ausgabezeilenlänge anpassen

```
SET LONG 10000;
SELECT * FROM mondial;
```

12.2.2 Strukturelle XML-Konstruktoren

Der SQL/XML-Standard definiert “XML publishing functions”

- Konstruktoren des rekursiv definierten abstrakten Datentyps “XMLType”.
- Verwendung wie andere vordefinierte oder selbst definierte Funktionen (u.a. in der SELECT-Klausel).
- Erzeugen Fragmente oder Instanzen von XMLType
- Ausgabe (z.B. in SELECT) als ASCII

XMLElement

- XMLElement: Name $\times$ Element-Body $\rightarrow$ Element:
  - Element-Body: Text oder rekursiv erzeugt (Attribute und Elemente)

```
SELECT XMLElement("Leer") FROM DUAL;
```

(Ergebnis ist eigentlich nicht korrekt: `<x/>` ist ein leeres Element, während `<x></x>` ein Element mit leerem Inhalt ist!)

```
SELECT XMLElement("Country",'bla') FROM DUAL;
SELECT XMLElement(Country,'bla') FROM DUAL;
```

- Hinweis: mit “...” zur Kontrolle über Groß-/Kleinschreibung (sonst alles groß).
  (man darf hierbei auch einfache und doppelte “...” nicht anders verwenden als im ersten Beispiel).

Elemente mit Nichttrivialem Inhalt

- XMLElement: zweites Argument enthält Subknoten des Elements (Attribute, Text und Subelemente),
- XMLAttributes: Liste von Wert-Name-Paaren, aus denen Attribute erzeugt werden.

```
SELECT XMLElement("Country",
  XMLAttributes(code AS "car_code", capital AS "capital"),
  name,
  XMLElement("Population",population),
  XMLElement("Area",area))
FROM country
WHERE area > 1000000;
```

[Filename: SQLX/xmlelement.sql]

Ein Ergebnis-Element:

```
<Country car_code="R" capital="Moscow">
  Russia
  <Population>148178487</Population>
  <Area>17075200</Area>
</Country>
```

Optionale Substrukturen

- XML als abstrakter Datentyp, funktionale Konstruktoren
- semistrukturierte Daten: Flexible und optionale Substrukturen

```
SELECT XMLElement("City",
  XMLAttributes(country AS country),
  XMLElement("Name",name),
  CASE WHEN longitude IS NULL THEN NULL
       ELSE XMLElement("Longitude",longitude) END,
  CASE WHEN latitude IS NULL THEN NULL
       ELSE XMLElement("Latitude",latitude) END)
FROM city
WHERE longitude IS NOT null;
```

[Filename: SQLX/xmlelement2.sql]

- Hinweis: `CASE WHEN` $cond$ `THEN` a `ELSE` b `END` ist ein *funktionales* Konstrukt.

Einfache Elementinhalte

- XMLForest: erzeugt eine Liste einfacher XML-Elemente aus gegebenen Namen und Werten:

```
SELECT XMLElement("Country",
          XMLForest(name AS Name,
                    code AS car_code,
                    population AS "Population",
                    area AS "Area"))
FROM country
WHERE area > 1000000;
```

[Filename: SQLX/xmlforest.sql]

```
<Country>
  <NAME>Brazil</NAME>
  <CAR_CODE>BR</CAR_CODE>
  <Population>162661214</Population>
  <Area>8511965</Area>
</Country>
```

⇒ kanonische Abbildung von Tupeln auf einfache XML-Elemente.

Subqueries

Textinhalte können auch durch (korrelierte) Subqueries bestimmt werden:

```
SELECT XMLElement("Country",
         XMLAttributes(code AS car_code),
         XMLElement("Name",name),
         XMLElement("NoOfCities",
                    (SELECT count(*)
                     FROM City
                     WHERE country=country.code)))
FROM country
WHERE area > 1000000;

SELECT XMLElement("Country",
         XMLAttributes(code AS car_code),
         XMLElement("Name",name),
         (SELECT XMLElement("NoOfCities",count(*))
                       FROM City
                       WHERE country=country.code))
FROM country
WHERE area > 1000000;
```

[Filename: SQLX/xmlsubquery.sql]

Gruppierung: XMLAgg

- XMLAgg: Bildung einer Collection aus den Zeilen der Gruppierung nach GROUP BY:
  In XML kann man auch die *Liste* der Items innerhalb der Gruppe verwenden:

```
SELECT XMLElement("Country",
        XMLAttributes(country AS car_code),
        XMLElement("NoOfCities", count(*)),
        XMLAgg(XMLElement("city",name)
                          ORDER by population))
FROM city
GROUP BY country;
```

[Filename: SQLX/xmlagg.sql]

Element des Ergebnisses:

```
<Country CAR_CODE="D">
  <NoOfCities>85</NoOfCities>
  <city>Erlangen</city>
  <city>Kaiserslautern</city>
  :
  <city>Berlin</city>
</Country>
```

Gruppierung: XMLAgg

- ähnlich über eine (korrelierte) (Sub)query:

```
SELECT XMLElement("Country",
       XMLAttributes(name AS name),
        (SELECT XMLAgg(XMLElement("city",name))
         FROM City
         WHERE country=code))
FROM country;
```

... wobei jetzt `XMLElement(‘‘NoOfCities’’, count(*))` (ebenfalls aus der Subquery zu berechnen) fehlt.

XMLConcat

- XMLConcat: Aneinanderhängen von mehreren “Spalten” einer Anfrage zu einem XML-Fragment:

```
SELECT XMLElement("Country",
  XMLAttributes(code AS code),
  XMLELEMENT(name, name),
  (SELECT XMLConcat(
     XMLElement("NoOfCities", count(*)),
     XMLAgg(XMLElement("city",name)))
   FROM City
   WHERE country=code))
FROM country;
```

[Filename: SQLX/xmlconcat.sql]

Auf diese Weise erzeugtes XML wird verwendet, um Instanzen von XMLType in Tabellen zu erzeugen:

XML-ZEILENOBJEKTE

```
CREATE TABLE CountryXML OF XMLType;

INSERT INTO CountryXML
(SELECT XMLElement("Country",
  XMLAttributes(code AS "Code",
                population AS "Population"),
  XMLElement("Name",name),
  (SELECT XMLElement("Capital",
            XMLForest(name AS "Name",
                      population AS "Population"))
   FROM city
   WHERE country=country.code
     AND city.name=capital))
FROM country);
```

[Filename: SQLX/fillcountry.sql]

- Ergebnis des SELECT sind Objekte vom Typ XMLType.

XML-SPALTENOBJEKTE

```
CREATE TABLE CityXML
  ( name VARCHAR2(35),
    province VARCHAR2(32),
    country VARCHAR2(3),
    population XMLType,
    coordinates XMLType);

INSERT INTO CityXML
(SELECT name, province, country,
  XMLElement("Population",
             XMLAttributes(95 as year),
             population),
  CASE WHEN longitude IS NULL THEN NULL
       ELSE XMLElement("Coordinates",
              XMLElement("Longitude", longitude),
              XMLElement("Latitude", latitude))
   END
 FROM city);
```

[Filename: SQLX/fillcity.sql]

... so weit zum Erzeugen und Abspeichern von XML-Daten.

12.2.3 Anfragen an XML-Daten innerhalb SQL

- XMLType als Abstrakter Datentyp: Selektoren, Modifikatoren.
- Diese bieten Schnittstellen für Standard-XML-Sprachen.
- Signatur:

  extract: XMLType $\times$ XPath_Expression $\rightarrow$
  XMLType/text/number
  extractValue: XMLType $\times$ XPath_Expression $\rightarrow$ text/number
  existsNode: XMLType $\times$ XPath_Expression $\rightarrow$ Boolean
- als "freie" Methoden (XMLType-Objekt mit `value` selektieren):

```
SELECT extract(value(m), '//city[name="Berlin"]')
FROM mondial m;
```

- oder als "member methods" wie bei benutzerdefinierten Objekttypen:

```
SELECT m.extract('//city[name="Berlin"]')
FROM mondial m;
```

SELECT: "EXTRACT"-FUNKTION

extract(XMLType_instance, XPath_string)
XMLType_instance.extract(XPath_string)

- Erstes Argument: SQL – selektiert ein (SQL-)Attribut des gegenwärtigen Tupels (muss ein Objekt des Typs XMLType ergeben),
- Zweites Argument: wendet einen XPath-Ausdruck darauf an,
- Ergebnis: vom Typ XMLType oder anderer SQL-Typ (mehrwertige Ergebnisse werden aneinandergehängt).

XML-ZEILENOBJEKTE

Zeilenwert ist vom Typ XMLType:

```
SELECT extract(value(c), '/Country/@CODE'),
       extract(value(c), '/Country/Capital/Name')
FROM CountryXML c;

SELECT c.extract('/Country/@CODE'),
       c.extract('/Country/Capital/Name')
FROM CountryXML c;
```

EINSCHUB: KURZÜBERSICHT ÜBER XPATH

- Navigation wie in Unix: /*step*/*step*/*step*
  /mondial/country/name
- Groß-Kleinschreibung beachten!
- Ergebnis: Menge/Liste von XML-Knoten (können nicht nur Werte, sondern ganze Teilbäume sein):
  /mondial/country
- Schritte überspringen:
  /mondial//city/name
  //city/name
  (ergibt /mondial/country/city und
  /mondial/country/province/city)
- Attribute: .../@*attributname*:
  /mondial/country/@area
- Zugriff auf Textinhalt:
  /mondial/country/name/text()
- Tests während der Navigation:
  /mondial/country[@code='D']/@area
  /mondial/country[name/text()='Germany']/@area
- Bei Vergleichen wird automatisch der Textinhalt verwendet:
  /mondial/country[name='Germany']/@area

(Weitere Details und Systematik siehe XML-Vorlesung)

SELECT: "EXTRACT"-FUNKTION (FORTS.)

XML-SPALTENOBJEKTE

Die Werte von CityXML.population sind XMLType-Zeilenobjekte:

```
SELECT extract(population,'/') FROM CityXML;
SELECT c.population.extract('/') FROM CityXML c;
```

```
SELECT name,
   extractValue(population,'/Population/@YEAR'),
   extractValue(population,'/Population')
FROM CityXML;
```

```
SELECT name,
 c.population.extract('/Population/@YEAR').getNumberVal(),
 c.population.extract('/Population/text()').getNumberVal()
FROM CityXML c
ORDER BY 3;
```

- exakte Groß-/Kleinschreibung im XPath-Ausdruck,
- extractValue momentan nicht als member method implementiert (Fehler)
- Benutzung von getNumberVal() und getStringVal()-Funktionen zum Casting:
  XML kennt soweit keinen Unterschied zwischen Strings und numerischen Werten.

SUBQUERIES AN XMLTYPE IN DER WHERE-KLAUSEL

... zum Heraussuchen und Vergleichen ebenfalls mit extract():

```
SELECT name
FROM CityXML c
WHERE c.population.extract('/Population/text()')
        .getNumberVal() > 1000000;

SELECT c.extract('/Country/Name/text()')
FROM CountryXML c
WHERE c.extract('/Country/@Population')
        .getNumberVal() > 1000000;
```

- Hierbei findet der Vergleich auf der SQL-Ebene statt (Vorteil: man kann Joins bilden).
- Hinweis: Wenn der XPath-Ausdruck mehrere Ergebnisse liefert, werden diese (bereits bei der Auswertung der extract()-Funktion) aneinandergehängt ...
- ... dann muss man es anders machen.

"EXISTSNODE"-FUNKTION

existsNode(XMLType_instance, XPath_string)

- Erstes Argument: SQL – selektiert ein (SQL-)Attribut des gegenwärtigen Tupels (muss ein Objekt des Typs XMLType ergeben),
- Zweites Argument: testet ob der angegebene XPath-Ausdruck für das Objekt ein nichtleeres Ergebnis hat
- Vergleich findet für jeden betroffenen Knoten einzeln auf XML-Ebene statt.
  Der Vergleichwert muss im XPath-String angegeben werden, man kann also damit *keine* Joins bilden.
- Ergebnis: 1 falls es einen Knoten gibt, 0 sonst.

```
SELECT name, extractValue(Population,'/')
FROM CityXML
WHERE existsNode(population,
                 '/Population[text()>1000000]') = 1;

SELECT name, extractValue(Population,'/')
FROM CityXML c
WHERE c.population.
      existsNode('/Population[text()>1000000]') = 1;
```

12.2.4 Ändern von XML-Daten

- das XMLType-Objekt wird komplett *ersetzt*,
- updateXML(...) als (Transformations-)Funktion:

updateXML(XMLType_instance, XPath_string, new_value)

- Erstes Argument: SQL – selektiert ein (SQL-)Attribut des gegenwärtigen Tupels (muss ein XMLType-Objekt sein),
- $2n$-tes Argument: selektiert den/die zu ändernden Knoten durch einen XPath-Ausdruck,
- $2n+1$-tes Argument: neuer Wert,
- Ergebnis: geänderte Instanz vom Type XMLType.
- Der Ausdruck "SELECT updateXML(...) FROM ..." ändert nicht die Datenbank, sondern gibt nur den Wert aus, der aus dem Update resultieren würde.

```
SELECT updateXML(c.population,
                 'Population/text()','1000000',
                 'Population/@YEAR','2004')
FROM CityXML c WHERE name='Gottingen';

SELECT updateXML(value(c),
                 '/Country/Name/text()','Fidschi')
FROM CountryXML c
WHERE extractValue(value(c),'Country/Name')='Fiji';
```

[Filename: SQLX/updatexml.sql]

Ändern von XML-Daten

Diese Funktion wird dann im SET-Statement verwendet:

```
UPDATE CityXML c
SET c.population   -- an XMLType element
    = updateXML(c.population,
                'Population/text()','1000000',
                'Population/@YEAR','2004')
WHERE name='Gottingen';

UPDATE CountryXML c
SET value(c) = updateXML(value(c),
                '/Country/Name/text()','Fidschi')
WHERE extractValue(value(c),'Country/Name')='Fiji';

UPDATE CountryXML c
SET value(c) = updateXML(value(c),
                '/Country/Name/text()','Fidschi')
WHERE existsNode(value(c),'/Country[Name="Fiji"]') = 1;
```

12.2.5 XML-Spezifische Funktionalität

Member Methods von XMLType

- Anwenden von XSLT-Stylesheets auf XMLType

Java-Funktionalität und PL/SQL Packages

- Implementierungen vieler XML-Sprachen mitgeliefert,
- dbms_xmldom: implementiert DOM (API um direkt mit der Baumstruktur zu arbeiten):
  PL/SQL: dbms_xmldom.*dosomething*(*object,args*)
- dbms_xmlparser: parst XML-Dokumente und DTDs (gegeben als CLOB oder URL) uns speichert das Ergebnis: Zugriff auf die DOM-Instanz und die DTD innnerhalb des Parsers durch “getdocument” bzw. “getdoctype”
- dbms_xslprocessor: processxsl(*verschiedene Argumente*); clob2file/file2clob liest/schreibt; selectnodes/selectsinglenode/valueof: XPath-Anfragen

... Details: Oracle Dokumentation, google ...

XSLT in Oracle: "Transform" Member Method

Member Method von XMLType:
XML-instance.transform(*Stylesheet-as-XMLValue*)
als SQL-Funktion anwendbar: SELECT
XMLTransform(*XML-instance*,*Stylesheet-als-XMLType*)

```
CREATE TABLE stylesheets
 (name VARCHAR2(100),
  stylesheet XMLTYPE);

INSERT INTO stylesheets VALUES('mondial-simple.xsl',
 system.getxml(
  'http://www.dbis.informatik.uni-goettingen.de' ||
   '/Teaching/DBP/XML/mondial-simple.xsl'));

SELECT value(m).transform(s.stylesheet)
FROM mondial m, stylesheets s
WHERE s.name = 'mondial-simple.xsl';

SELECT XMLTransform(value(m),s.stylesheet)
FROM mondial m, stylesheets s
WHERE s.name = 'mondial-simple.xsl';
```

[Filename: SQLX/applystylesheet.sql]

```
CREATE OR REPLACE FUNCTION xslexample RETURN CLOB IS
  xmldoc      CLOB;   xsldoc  CLOB;  html  CLOB;
  myParser    dbms_xmlparser.Parser;
  indomdoc    dbms_xmldom.domdocument;
  xsltdomdoc  dbms_xmldom.domdocument;
  xsl         dbms_xslprocessor.stylesheet;
  outdomdocf  dbms_xmldom.domdocumentfragment;
  outnode     dbms_xmldom.domnode;
  proc        dbms_xslprocessor.processor;
BEGIN
  -- Get the XML document as CLOB
  SELECT value(m).getClobVal() INTO xmldoc FROM mondial m;
  -- Get the XSL Stylesheet as CLOB
  SELECT s.stylesheet.getClobVal() INTO xsldoc
  FROM stylesheets s WHERE name='mondial-simple.xsl';

  -- Get the new xml parser instance
  myParser := dbms_xmlparser.newParser;
  -- Parse the XML document and get its DOM
  dbms_xmlparser.parseClob(myParser, xmldoc);
  indomdoc := dbms_xmlparser.getDocument(myParser);

  -- Parse the XSL document and get its DOM
  dbms_xmlparser.parseClob(myParser, xsldoc);
  xsltdomdoc := dbms_xmlparser.getDocument(myParser);

  xsl  := dbms_xslprocessor.newstylesheet(xsltdomdoc, '');
  -- Get the new xsl processor instance
  proc := dbms_xslprocessor.newProcessor;

  -- Apply stylesheet to DOM document
  outdomdocf := dbms_xslprocessor.processxsl(proc, xsl, indomdoc);
  outnode    := dbms_xmldom.makenode(outdomdocf);

  -- Write the transformed output to the CLOB
  dbms_xmldom.writetoCLOB(outnode, html);
  return(html);  -- Return the transformed output
END;
/
```

[Filename: SQLX/xslexample.sql]