# Kapitel 6: SQL und XML

- Wie können die Inhalte einer Datenbank als XML-Dokumente exportiert werden (*Publizieren* von XML)?
- Wie können die Inhalte eines XML-Dokumentes in eine Datenbank importiert werden (Speichern von XML in Datenbanken)?
- Wie können Anfragen über XML-Dokumenten in SQL-Anfragen abgebildet werden?

# 6.1 SQL/XML

## SQL/XML ist Teil des SQL:2003-Standards

- XML kann direkt als Zeichenkette vom Typ `VARCHAR` oder `CLOB` (`CHARACTER LARGE OBJECT`) gespeichert werden.
- SQL/XML erweitert die zulässigen Datentypen in SQL um den vordefinierten Datentyp `XML`, so dass ganze XML-Dokumente, Elemente oder auch Mengen von Elementen als Werte vom Typ XML
  - gespeichert,
  - mittels spezieller Operatoren innerhalb SQL erzeugt werden können.
- Die Integration von XPath und XQuery in SQL wird prinzipiell möglich, ist aber noch nicht standardisiert.

## Datentyp XML

- `XMLPARSE()` wandelt ein als Zeichenkette gegebenes XML-Dokument in einen Wert vom Datentyp `XML`.
- `XMLSERIALIZE()` wandelt einen Wert vom Typ XML zu einer Zeichenkette.
- SQL/XML definiert eine Reihe von weiteren Funktionen, die, in einen SFW-Ausdruck von SQL integriert, das Publizieren von XML aus einer Datenbank erlauben.

## Relationen

**Land**

| LCode | LName | HStadt | Fläche |
|---|---|---|---|
| D | Germany | Berlin | 357 |

**Provinz**

| PName | LCode | Fläche |
|---|---|---|
| Baden | D | 15 |
| Berlin | D | 0,9 |

**Stadt**

| SName | PName | LCode | Einwohner | LGrad | BGrad |
|---|---|---|---|---|---|
| Berlin | Berlin | D | 3472 | 13,2 | 52,45 |
| Freiburg | Baden | D | 198 | 7,51 | 47,59 |
| Karlsruhe | Baden | D | 277 | 8,24 | 49,03 |

**Lage**

| LCode | Kontinent | Prozent |
|---|---|---|
| D | Europe | 100 |

**Mitglied**

| LCode | Organisation | Art |
|---|---|---|
| D | EU | member |

## SFW-Ausdruck mit SQL/XML-Funktionen (Teil 1)

```
SELECT
XMLELEMENT (
    NAME Land,
    XMLATTRIBUTES (L.LCode AS LCode),
    XMLELEMENT (NAME LName, L.LName),
    ( SELECT
       XMLELEMENT
       ( NAME Provinz,
       XMLELEMENT (NAME PName, P.PName),
       XMLELEMENT (NAME Fläche, P.Fläche),
       XMLAGG
       ( XMLELEMENT
          ( NAME Stadt,
          XMLELEMENT (NAME SName, S.SName),
          XMLELEMENT (NAME Einwohner, S.Einwohner)
          )
          ORDER BY S.Einwohner DESC
       )
       )
    FROM Provinz P, Stadt S
    WHERE P.LCode = L.LCode and P.LCode = S.LCode and
    P.PName = S.PName
    GROUP BY P.PName, P.Fläche
    ),
```

## SFW-Ausdruck mit SQL/XML-Funktionen (Teil 2)

```
    ( SELECT
       XMLELEMENT
       ( NAME Lage,
       XMLELEMENT (NAME Kontinent, La.Kontinent),
       XMLELEMENT (NAME Prozent, La.Prozent)
       )
       FROM Lage La
       WHERE La.LCode = L.LCode
    ),
    ( SELECT
       XMLELEMENT
       ( NAME Mitglied,
       XMLATTRIBUTES
       ( M.Organisation AS Organisation,
       M.Art AS Art
       )
       )
    FROM Mitglied M
    WHERE M.LCode = L.LCode
    )
) AS Mondial
FROM Land L
```

# XMLELEMENT()

- `XMLELEMENT()` erzeugt Elemente.
  - Das erste Argument von `XMLELEMENT` definiert den Namen des Elementes.
  - Das zweite Argument ist optional und dient zur Definition der zugehörigen Attribute.
  - Das dann folgende Argument definiert den Elementinhalt.
- Ein leeres Element:

```
XMLELEMENT
(   NAME Mitglied,
    XMLATTRIBUTES (
       M.Organisation AS Organisation,
       M.Art AS Art
    )
)
```

- Sind Attributnamen und Spaltennamen identisch, dann kann eine abgekürzte Schreibweise gewählt werden, hier `XMLATTRIBUTES(M.Organisation, M.Art)`

## Beispiel

```
SELECT
        XMLELEMENT (
                NAME Lage,
                XMLELEMENT (NAME Kontinent, La.Kontinent),
                XMLELEMENT (NAME Prozent, La.Prozent)
        )
FROM Lage La
WHERE La.LCode = L.LCode
```

# XMLAGG()

- `XMLAGG()` erzeugt zu jeder mittels `GROUP BY` definierten Gruppe von Zeilen einer Tabelle eine Folge von Elementen.

## Beispiel

```
SELECT
    XMLELEMENT (
       NAME Provinz,
       XMLELEMENT (NAME PName, P.PName),
       XMLELEMENT (NAME Fläche, P.Fläche),
       XMLAGG (
          XMLELEMENT (
             NAME Stadt,
             XMLELEMENT (NAME SName, S.SName),
             XMLELEMENT (NAME Einwohner, S.Einwohner)
          )
       )
    )
FROM Provinz P, Stadt S
WHERE P.LCode = L.LCode and P.LCode = S.LCode and
       P.PName = S.PName
GROUP BY P.PName, P.Fläche
```

# XMLCONCAT()

- `XMLAGG` erlaubt lediglich ein einziges Argument, das, jedoch mehrere Kindelemente haben kann.
- Mittels `XMLCONCAT()` kann eine Folge von Elementen mittels Konkatenation ihrer Inhalte erzeugt werden.

## Beispiel

```
XMLAGG (
    XMLCONCAT (
        XMLELEMENT (NAME SName, S.SName),
        XMLELEMENT (NAME Einwohner, S.Einwohner)
    )
)
```

# XMLFOREST()

- Alternativ zu `XMLCONCAT()`:

```
XMLAGG (
    XMLFOREST (
        S.SName AS SName,
        S.Einwohner AS Einwohner
    )
)
```

- Die Elementnamen ergeben sich implizit aus den betreffenden Spaltenbezeichnern, oder explizit über die `AS`-Klausel.
- Abgekürzte Schreibweise: `XMLFOREST(S.SName, S.Einwohner)`
- Die einzelnen Elemente haben keine Struktur und keine Attribute.

## Abbildungsregeln

SQL/XML enthält Abbildungsregeln,

- um Tabellen generisch in XML-Dokumente zu transformieren,
- um SQL-Datentypen XML-Schema Datentypen zuzuordnen, deren Werte die in SQL zulässigen Werte möglichst exakt darstellen.

# Abbildungsregeln für Tabellen

```
<Land>
    <row>
       <LName>France</LName>
       <LCode>F</LCode>
       <HStadt>Paris</HStadt>
       <Fläche>547</Fläche>
    </row>
    <row>
       <LName>Germany</LName>
       <LCode>D</LCode>
       <HStadt>Berlin</HStadt>
       <Fläche>357</Fläche>
    </row>
</Land>
```

(a)

```
<Land>
       <LName>France</LName>
       <LCode>F</LCode>
       <HStadt>Paris</HStadt>
       <Fläche>547</Fläche>
</Land>
<Land>
       <LName>Germany</LName>
       <LCode>D</LCode>
       <HStadt>Berlin</HStadt>
       <Fläche>357</Fläche>
</Land>
```

(b)

Es kann festgelegt werden, ob bei Vorliegen von Nullwerten in Spalten einer Tabelle im zugehörigen XML-Dokument die entsprechenden Elemente fehlen, oder durch das Attribut `nil = "true"` repräsentiert werden sollen.

## Abbildungsregeln für Datentypen

- Abbildungsregeln, um gegebenen Basis-SQL-Datentypen, wie `CHARACTER`, `NUMERIC`, `DATE` und konstruierten Datentypen, wie `ROW`, `ARRAY`, `MULTISET`, jeweils XML-Schema Datentypen zuzuordnen.
- Jedem Basis-SQL-Datentyp wird ein globaler XML-Schema Datentyp zugeordnet, wobei sich der Zusammenhang zwischen den einzelnen Typen über eine normierte Namensgebung ergibt.

```
<simpleType name = "INTEGER">
  <restriction base = "int"/>
</simpleType>
```

```
<simpleType name = "CHAR_50">
  <restriction base = "string">
    <length value = "50"/>
  </restriction>
</simpleType>
```

## Annotationen

Einige Unterscheidungen in SQL, wie beispielsweise `CHARACTER VARYING` versus `CHARACTER LARGE OBJECT`, haben keine Entsprechung in XML-Schema.

```
<simpleType name = "CHAR_50">
  <annotation>
    <appinfo>
      <sqltype kind = "PREDEFINED" name = "CHAR" length = "50"
        characterSetName = "LATIN1" collation = "DEUTSCH"/>
    </appinfo>
  </annotation>
  <restriction base = "string">
    <length value = "50"/>
  </restriction>
</simpleType>
```

## konstruierter Datentyp ROW

```
<sequence>
  <element name = "LName" nillable = "true"
           minoccurs = "1" maxoccurs = "1" type = "VARCHAR\_50">
  <element name = "LCode"
           minoccurs = "1" maxoccurs = "1" type = "CHAR\_5">
</sequence>
```

# XML-Datentyp zur Tabelle Land

```
<complexType name = "RowType.Mondial.Land">
  <sequence>
     <element name = "LName" nillable = "true"
              minoccurs = "1" maxoccurs = "1" type = "VARCHAR_50">
    <element name = "LCode"
             minoccurs = "1" maxoccurs = "1" type = "CHAR_5">
  </sequence>
</complexType>

<complexType name = "TableType.Mondial.Land">
  <sequence>
     <element name = "row" type = "RowType.Mondial.Land
              minoccurs = "0" maxoccurs = "unbounded"/>
  </sequence>
</complexType>
```

# 6.2 Speichern von XML

- *strukturorientierte* Speicherung: Jedem XML-Dokument wird ein (erweiterter) XML-Baum zugeordnet; die resultierende Tabelle nennen wir *XML-Kantentabelle*.
- *inhaltsorientierte* Speicherung: In Abhängigkeit vom Typ des XML-Dokumentes ist ein möglichst gut geeignetes relationales Schema zu finden. Hierarchische Beziehungen im XML-Dokument sind geeignet durch Fremdschlüsselbeziehungen zu repräsentieren.

## strukturorientierte Speicherung

- XML-Bäume sind geordnete Bäume.
  - Ordnung durch Nummerierung der Knoten in Dokument-Ordnung,
  - Alternativ Ordnung der Kindknoten eines Knotens.
- Die Speicherung eines XML-Dokumentes als XML-Baum ist geeignet wenn zu den Dokumenten keine Typ-Information bekannt ist, oder die Struktur der zu speichernden Dokumente stark variiert.
- Nachteile:
  - Alle Typ-Informationen gehen verloren.
  - Außerdem macht das Halten sämtlicher Informationen in einer Tabelle die Formulierung von Anfragen und ihre effiziente Auswertung problematisch: Auswertung des '//'-Operators eines XPath-Ausdrucks mittels SQL verlangt Rekursion im Allgemeinen.

## Beispiel

XML-Kantentabelle

| Knoten | Vorgänger | Ordnung | Name | Wert |
|---|---|---|---|---|
| 0 | null | 1 | Mondial | |
| 1 | 0 | 1 | Land | |
| 2 | 1 | | @LCode | D |
| 3 | 1 | 1 | LName | |
| 4 | 2 | 1 | | Germany |
| 5 | 1 | 2 | Provinz | |
| 6 | 5 | 1 | PName | |
| 7 | 6 | 1 | | Baden |
| 8 | 5 | 2 | Fläche | |
| 9 | 8 | 1 | | 15 |
| 10 | 5 | 3 | Stadt | |
| 11 | 10 | 1 | SName | |
| 12 | 11 | 1 | | Freiburg |
| 13 | 10 | 2 | Einwohner | |
| 14 | 13 | 1 | | 198 |
| 15 | 5 | 4 | Stadt | |
| 16 | 15 | 1 | SName | |
| 17 | 16 | 1 | | Karlsruhe |
| 18 | 15 | 2 | Einwohner | |
| 19 | 18 | 1 | | 277 |
| | | ... | | |

## inhaltsorientierte Speicherung

- Inverses Problem zum Publizieren von XML aus einer Datenbank.
- Jedem Elementtyp, zu dem Kindelemente oder Attribute existieren, wird eine Tabelle zugeordnet.
- Allen skalaren Attributen und allen Kindelementen einfachen Typs, die nicht mehrfach auftreten können, wird jeweils eine Spalte zugeordnet.

  Lässt sich mit diesen Spalten kein geeigneter Primärschlüssel definieren, so wird eine zusätzliche Primärschlüsselspalte aufgenommen.
- Mehrwertigen Attributen und mehrfach auftretenden Kindelementen einfachen Typs, bzw. Kindelementen komplexen Typs, wird ebenfalls eine eigene Tabelle zugeordnet.

  Der Bezug zu dem jeweiligen Elter-Element wird durch Hinzunahme seines Primärschlüssels als Fremdschlüssel realisiert.

Ein so erzeugtes Datenbankschema kann nur als ein Ausgangspunkt für einen umfassenderen Datenbankentwurf gelten.

## Beispiel

```
<Mondial>
  <Land LCode = "D">
    <LName>Germany</LName>
    <Provinz>
      <PName>Baden</PName>
      <Flaeche>15</Fläche>
      <Stadt>
        <SName>Freiburg</SName>
        <Einwohner>198</Einwohner>
      </Stadt>
      <Stadt>
        <SName>Karlsruhe</SName>
        <Einwohner>277</Einwohner>
      </Stadt>
    </Provinz>
    <Provinz>
      <PName>Berlin</PName>
      <Flaeche>0,9</Fläche>
      <Stadt>
        <SName>Berlin</SName>
        <Einwohner>3472</Einwohner>
      </Stadt>
    </Provinz>
    <Lage>
      <Kontinent>Europe</Kontinent>
      <Prozent>100</Prozent>
    </Lage>
    <Mitglied Organisation = "EU" Art = "member"/>
  </Land>
</Mondial>
```

## Beispiel fortgesetzt

Land

| LCode | LName |
|---|---|
| D | Germany |

Provinz

| PName | LCode | Fläche |
|---|---|---|
| Baden | D | 15 |
| Berlin | D | 0,9 |

Stadt

| SName | PName | LCode | Einwohner |
|---|---|---|---|
| Berlin | Berlin | D | 3472 |
| Freiburg | Baden | D | 198 |
| Karlsruhe | Baden | D | 277 |

Lage

| LCode | Kontinent | Prozent |
|---|---|---|
| D | Europe | 100 |

Mitglied

| LCode | Organisation | Art |
|---|---|---|
| D | EU | member |

# 6.3 XPath und SQL

Anfragen nach den Nachfolgern oder Vorgängern eines Knotens eines Baumes können in einem geschlossenen SQL-Ausdruck im Allgemeinen nur mittels Rekursion ausgedrückt werden.

- Anfragen dieser Art können mit einer *festen* Anzahl Verbundoperationen ausgewertet werden,
- wenn wir etwas mehr Informationen über die einzelnen Knoten vorsehen.

## Vermeiden von Rekursion

- Den einzelnen Knoten eines XML-Baumes werden Intervalle so zugeordnet, dass direkte und indirekte Vorgänger und Nachfolgerbeziehungen durch Analyse der Intervalle ohne Rekursion entschieden werden können.
- Des Weiteren nutzen wir die Pattern-Matching-Möglichkeiten von SQL geschickt aus, so dass Lokationspfade, die keine Prädikate enthalten, ohne Verbundoperationen ausgewertet werden können.

## benötigte Relationen

Zur Repräsentation eines XML-Baumes führen wir unterschiedliche Tabellen ein.

- Für Element-, Attribut- und Textknoten werden jeweils eigene Tabellen verwendet.
- Zusätzlich werden alle über dem XML-Baum ausdrückbaren *einfachen* XPath-Ausdrücke, die ausgehend von der Wurzel ein Blatt lokalisieren, in einer eigenen weiteren Tabelle gespeichert.

  Ein *einfacher* XPath-Ausdruck ist eine Folge von Ausdrücken der Form `#/EName` und `#/@Attr`, wobei '#' ein für das spätere Pattern-Matching benötigtes Trennzeichen ist.

## Beispiel

Element

| Start | End | PId |
|---|---|---|
| 0 | 480 | 1 |
| 9 | 470 | 2 |
| 25 | 46 | 4 |
| 47 | 233 | 5 |
| 56 | 75 | 6 |
| 76 | 94 | 7 |
| 95 | 158 | 8 |
| 102 | 124 | 9 |
| 125 | 150 | 10 |
| 159 | 223 | 8 |
| 166 | 189 | 9 |
| 190 | 215 | 10 |
| 234 | 356 | 5 |
| 243 | 263 | 6 |
| 264 | 283 | 7 |
| 284 | 346 | 8 |
| 291 | 311 | 9 |
| 312 | 338 | 10 |
| 357 | 420 | 11 |
| 363 | 391 | 12 |
| 392 | 413 | 13 |
| 421 | 462 | 14 |

Attribut

| PId | Start | End | Wert |
|---|---|---|---|
| 3 | 10 | 10 | D |
| 15 | 422 | 422 | Europe |
| 16 | 422 | 422 | member |

Text

| Start | End | Wert | PId |
|---|---|---|---|
| 32 | 38 | Germany | 4 |
| 63 | 67 | Baden | 6 |
| 84 | 85 | 15 | 7 |
| 109 | 116 | Freiburg | 9 |
| 136 | 138 | 198 | 10 |
| 173 | 181 | Karlsruhe | 9 |
| 201 | 203 | 277 | 10 |
| 250 | 255 | Berlin | 6 |
| 272 | 274 | 0,9 | 7 |
| 298 | 303 | Berlin | 9 |
| 323 | 326 | 3472 | 10 |
| 374 | 379 | Europe | 12 |
| 401 | 403 | 100 | 13 |

## Beispiel fortgesetzt

Pfad

| PId | PathExpr |
|---|---|
| 1 | #/Mondial |
| 2 | #/Mondial#/Land |
| 3 | #/Mondial#/Land#/@LCode |
| 4 | #/Mondial#/Land#/LName |
| 5 | #/Mondial#/Land#/Provinz |
| 6 | #/Mondial#/Land#/Provinz#/PName |
| 7 | #/Mondial#/Land#/Provinz#/Fläche |
| 8 | #/Mondial#/Land#/Provinz#/Stadt |
| 9 | #/Mondial#/Land#/Provinz#/Stadt#/SName |
| 10 | #/Mondial#/Land#/Provinz#/Stadt#/Einwohner |
| 11 | #/Mondial#/Land#/Lage |
| 12 | #/Mondial#/Land#/Lage#/Kontinent |
| 13 | #/Mondial#/Land#/Lage#/Prozent |
| 14 | #/Mondial#/Land#/Mitglied |
| 15 | #/Mondial#/Land#/Mitglied#/@Organisation |
| 16 | #/Mondial#/Land#/Mitglied#/@Art |

## Regionen

- Die *Region* eines Text- und Elementknotens ist ein Intervall der Form $[a, b]$, wobei $a$ die Start- und $b$ die Endposition der zugehörigen Tags im Dokument ist.
- Die Region eines Attributknotens ist gegeben durch zwei identische Zahlen, die gleich der Startposition seines Elterelementes erhöht um 1 sind. Hat ein Element mehrere Attribute, so ist in dieser Weise keine Ordnung auf den Attributen impliziert.
- Um die Zuordnung der Element-, Attribut- und Textknoten in den Tabellen zu ihren Positionen im Dokument zu rekonstruieren, sind lediglich die *relativen* Verhältnisse der Werte zu `Start` von Bedeutung.

## Beispiel

- /Mondial/Land/Provinz/Stadt:

```
SELECT E.Start, E.End
FROM Element E, Pfad P
WHERE P.PathExp LIKE '#/Mondial#/Land#/Provinz#/Stadt' AND
  E.PId = P.PId
```

- /Mondial//Stadt:

## Beispiel

```
//Land[Lage/Kontinent = "Europe"]//Stadt/SName:
```

```
SELECT E2.Start, E2.End
FROM Pfad P1, Pfad P2, Pfad P3,
  Element E1, Element E2, Text T
WHERE P1.PathExp LIKE '#%/Land'
AND P2.PathExp LIKE '#%/Land#/Lage#/Kontinent'
AND P3.PathExp LIKE '#%/Land#%/Stadt#/SName'
AND E1.PID = P1.PID
AND E2.PID = P3.PID
AND T.PID = P2.PID
AND E1.Start < T.Start AND E1.End > T.End
AND E1.Start < E2.Start AND E1.End > E2.End
AND T.Wert = 'Europe'
```

**Beispiel**

`/Mondial//Provinz//SName`: