

SQL/XML

Was ist SQL/XML ?

- SQL/XML ist eine *Erweiterung* von SQL, zur Ablage von XML-Daten in Tabellen und zur Ausgabe von Daten im XML-Format.
- SQL/XML ist ein ISO/ANSI Standard seit Dezember 2003.
- Verschiedene Produkte haben Implementationen, die sich an diesem Standard orientieren, meist aber noch weitergehen.
- SQL/XML ermöglicht die
 - Ablage von XML-Dokumenten oder XML-Fragmenten
 - Vermeidung zu starker Dekomposition bei der Normalisierung von Daten

Die Bezeichnung des Standards ist ISO/IEC 9075-14. Der Standard selbst ist etwas schwer verdaulich. Wesentliche Punkte sind beschrieben in einer wichtigen Vorarbeit unter: www.acm.org/sigmod/record/issues/0206/standard.pdf.

Welche Daten eignen sich für SQL/XML ?

- Datenelemente, welche in sich strukturiert sind, aber häufig als Ganzes von und zur Datenbank transferiert werden.
- Datenelemente, auf deren Substruktur gelegentlich zugegriffen werden muss, zum Beispiel in einer Abfragebedingung.
- Datenelemente, deren Substruktur gelegentlich aufgelöst, und im Rahmen einer Abfrage zusammen mit anderen Daten neu kombiniert werden müssen.

int	varchar	date	XML

Bibliotheksbeispiel

- Anhand von Sybase ASA 9.0 sollen nun die Grundfunktionen von SQL/XML vorgestellt werden.
- Folgende Beispieltabellen werden verwendet

- Variante *Classic* mit klassischen SQL-Tabellen

Book
Author
Publisher
Author_Book

- Variante *Blended* mit einer SQL/XML-Tabelle

Catalog

```
create table Publisher (
    idPub varchar(16),
    name varchar(64) ,
    url varchar(64),
    primary key ( idPub )
);
```

```
create table Book (
    idBook varchar(16),
    idPub  varchar(16),
    title  varchar(64),
    subtitle varchar(64) null,
    edition varchar(64),
    year varchar(4),
    binding varchar(32) null,
    pages numeric(4,0) null,
    isbn varchar( 16 ),
    primary key ( idBook )
);
```

```
create table Author (
    idAuthor varchar(16),
    firstname varchar(16),
    middlename varchar(16) null,
    lastname varchar(16),
    primary key ( idAuthor )
);
```

```
create table Author_Book (
    idAuthor varchar(16),
    idBook    varchar(16),
    primary key ( idAuthor, idBook )
);
```

```
create table Catalog (  
    idBook numeric(10,0) default  
                                autoincrement,  
    book XML  
);
```

Abfrageresultate im XML-Format

- Folgende Funktionen beim Abfragen von Daten ermöglichen das Konstruieren von XML-Ausgaben:

XMLELEMENT ()
XMLATTRIBUTES ()
XMLAGG ()
XMLCONCAT ()
XMLFOREST ()

XMLELEMENT ()

- Generiert ein XML Element mit SQL-Datentyp XML

```
select XMLELEMENT( name "Buch", 'Meine Kochecke' )
```

```
select XMLELEMENT( name "Buch", title )
from   Book
```

```
select XMLELEMENT( name "Buch",
                    XMLELEMENT( name "Titel", title ),
                    XMLELEMENT( name "Untertitel", subtitle ) )
from   Book
```

Kompatibel mit SQL/XML Standard

Die XML-Elementnamen werden syntaktisch als SQL-Identifizier behandelt. Falls sie keine Sonderzeichen enthalten, resp. den allgemeinen Regeln für Identifizier genügen, können die Anführungszeichen weggelassen werden. Andernfalls, insbesondere auch wenn die Indentifizier gleich heissen wie allgemeine SQL-Schlüsselwörter, müssen Anführungszeichen verwendet werden.

Falls im Inhalt eines Elementes die Zeichen ", ' oder & vorkommen, werden Sie durch die XMLELEMENT()-Funktion in entsprechende Entity-Darstellungen umgewandelt, das heisst " ' oder &

Resultat 1

```
<Buch>Meine Kochecke</Buch>
```

Resultat 2

```
<Buch>Learning XML</Buch>
```

```
<Buch>Inside XML</Buch>
```

```
<Buch>Step by Step XML</Buch>
```

...

Resultat 3

```
<Buch><Titel>Learning XML</Titel><Untertitel></Untertitel></Buch>
```

```
<Buch><Titel>Inside XML</Titel><Untertitel></Untertitel></Buch>
```

```
<Buch><Titel>Step by Step XML</Titel><Untertitel></Untertitel></Buch>
```

...

XMLATTRIBUTES ()

- Fügt Attribute zu einem XML-Element hinzu

```
select XMLElement( name Buch,  
                  XMLATTRIBUTES( 'Werners Kochecke' as Titel,  
                                'BFH Publishings' as Verlag ))  
  
select XMLElement( name "Buch",  
                  XMLATTRIBUTES( title as Haupttitel,  
                                subtitle as Untertitel ),  
                  'Dies ist ein Bestseller' )  
from Book
```

Kompatibel mit SQL/XML Standard

Die XML-Attributnamen werden syntaktisch als SQL-Identifizier behandelt. Falls sie keine Sonderzeichen enthalten, resp. den allgemeinen Regeln für Identifizier genügen, können die Anführungszeichen weggelassen werden. Andernfalls, insbesondere auch wenn die Indentifizier gleich heissen wie allgemeine SQL-Schlüsselwörter, müssen Anführungszeichen verwendet werden.

Falls im Wert eines Attributes die Zeichen ", ' oder & vorkommen, werden Sie durch die XMLATTRIBUTES()-Funktion in entsprechende Entity-Darstellungen umgewandelt, das heisst " ' oder &

Resultat 1

```
<Buch Titel="Werners Kochecke" Verlag="BFH Publishings"/>
```

Resultat 2

```
<Buch Haupttitel="Learning XML">Dies ist ein Bestseller</Buch>
```

```
<Buch Haupttitel="Inside XML">Dies ist ein Bestseller</Buch>
```

```
<Buch Haupttitel="Step by Step XML">Dies ist ein Bestseller</Buch>
```

```
<Buch Haupttitel="The XML Companion">Dies ist ein Bestseller</Buch>
```

...

XMLAGG ()

- Ist eine Aggregatfunktion ähnlich `count()` `sum()` etc.
- Kann mit `group by` verwendet werden.
- Erstellt einen Wald von gleichen Elementen aus allen Datensätzen einer Tabelle resp. einer SQL-Gruppierung.
- Sind die einzelnen Elemente nicht vom Datentyp XML, sondern Strings, werden sie einfach konkateniert.

```
select xmlagg(xml element( name Buch, title ))
from Book
group by idPub
```

```
select idPub, XMLAGG( title + ', ' )
from Book
group by idPub
```

Kompatibel mit SQL/XML Standard

Innerhalb der aggregierten Gruppe können die Elemente sortiert werden. Beispiel:

```
select XMLAGG( XMLELEMENT( name Buch, title ) ORDER BY title )
from Book
```

`XMLAGG()` liefert einen *einzig*en String zurück. Dieser enthält entweder die konkatenierten Einzelstrings oder die konkatenierten XML-Elemente der jeweiligen Gruppe oder ganzen Tabelle. In jedem Fall ist es technisch ein einziger String. `XMLAGG()` bietet somit die Möglichkeit, aus einzelnen Zellen einer Datenbanktabelle wieder ein einziges, vollständiges XML-Dokument zu erstellen.

Resultat 1

```
<Buch>Learning XML</Buch><Buch>Inside XML</Buch>
<Buch>Step by Step XML</Buch>
...
```

Resultat 2

```
P-ORA      Learning XML, XML in a Nutshell,
P-NRP      Inside XML,
P-MS       P Step by Step XML ,
P-AW       T he XML Companion,
...
```

XMLCONCAT ()

- Erstellt einen Wald von XML-Elementen.
- Die Parameter für XMLCONCAT sind *XML-Elemente*.
- Ausgabe mehrerer XML-Elemente in der gleichen Zeile:

```
SELECT XMLCONCAT( XMLELEMENT ( name Titel, title ),
                  XMLELEMENT( name Seiten, pages ),
                  XMLELEMENT( name Isbn, isbn )
                )
FROM Book
```

Kompatibel mit SQL/XML Standard

Resultat 1

```
<Titel>Learning XML</Titel><Seiten>350</Seiten><Isbn>0596000464</Isbn>
<Titel>Inside XML</Titel><Seiten>1102</Seiten><Isbn>0735710201</Isbn>
...
```

XMLFOREST ()

- Erstellt einen Wald von XML-Elementen.
- Die Parameter für XMLFOREST sind *SQL-Attribute*, Konstanten oder Ausdrücke.

```
SELECT XMLFOREST( title, isbn, pages )  
from Book
```

```
SELECT XMLFOREST( p.name as Verlag, b.title, b.isbn  
 )  
from Book b natural join Publisher p
```

Kompatibel mit SQL/XML Standard

XMLFOREST entspricht einer Zusammensetzung von XMLCONCAT und XMLELEMENT.

Resultat 1

```
<title>Learning XML</title> <isbn>0596000464</isbn> <pages>350</pages>  
<title>Inside XML</title> <isbn>0735710201</isbn> <pages>1102</pages>  
...
```

Resultat 2

```
<Verlag>O'Reilly & Associates</Verlag>  
<title>Learning XML</title>  
<isbn>0596000464</isbn>  
  
<Verlag>New Riders Publishing</Verlag>  
<title>Inside XML</title>  
<isbn>0735710201</isbn>  
  
...
```

Datentyp XML

- Der Datentyp XML ist eine vordefinierte Domäne, implementationsmässig meist auf Basis eines CLOB-Typs.
- Nach Standard gilt, dass XML-Werte wohlgeformt sein müssen, mit der Ausnahme, dass auf der obersten Ebene mehrere Elemente stehen dürfen.
- XML-Werte sind gemäss Standard *nicht vergleichbar*.
- Verschiedene proprietäre, aber mächtige Funktionen beschäftigen sich mit der Extraktion und Erzeugung von XML-Daten aus Tabelleninhalten:

XMLGEN()	Zusammenbau von neuen XML-Dokumenten aus Datenbankinhalten.
OPENXML()	XML-Dokumente als virtuelle Tabellen darstellen
extractValue()	XML-Fragmente aus XML-Feldern extrahieren
xmlUpdate()	Fragmente in XML-Feldern gezielt ersetzen

Siehe Standard, Kapitel 4.2.1

XMLGEN() findet man in SQL Server, Sybase, Oracle

OPENXML() findet man in SQL Server, Sybase

extractValue() und xmlUpdate() sind bei Oracle zu finden

Einfügen von XML-Werten

- XML-Werte können wie varchar-Felder eingefügt werden:

```
insert catalog( book )
values ( '<book>
        <title>Learning XML</title>
        <author>Erik T. Ray</author>
        <publisher>O''Reilly & Associates</publisher>
        <edition>1st edition</edition>
        <year>2001</year>
        <binding>Paperback</binding>
        <pages>350</pages>
        <isbn>0596000464</isbn>
        </book>' );
```

Die Prüfung auf syntaktische Korrektheit ist zwar vom Standard her vorgeschrieben, aber nicht überall vorhanden.

XMLGEN ()

- ein XML-Skelett Dokument kann an vordefinierten Stellen mit Text oder XML-Inhalt ergänzt werden.
- Für die Definition der Platzhalter wird auf der Syntax von XQuery aufgebaut.

```
select XMLGEN( '<catalog>{$items}</catalog>',  
              XMLAGG( book ) as items )  
from Catalog
```

Nicht mehr kompatibel mit SQL/XML Standard

XQuery ist vom W3 Konsortium als universelle Sprache für den Zugriff auf XML-Daten vorgesehen, analog zu SQL für relationale Datenbanken. Siehe <http://www.w3.org/XML/Query> . XQuery ist noch kein verabschiedeter Standard. Die meisten Hersteller von Datenbanken arbeiten aber intensiv an Implementationen.

In der Funktion XMLGEN ist der XQuery-Konstruktor so eingeschränkt, dass nur einzelne Variablen der Form {\$x} verwendet werden dürfen. Allgemeine XQuery-Konstrukte haben wesentlich mehr Möglichkeiten.

Resultat 1

```
<catalog>  
  <book>  
    <title>Learning XML</title>  
    <author>Erik T. Ray</author>  
    <publisher>O'Reilly & Associates</publisher>  
    <edition>1st edition</edition>  
    <year>2001</year>  
    <binding>Paperback</binding>  
    <pages>350</pages>  
    <isbn>0596000464</isbn>  
  </book>  
  <book>  
    ...  
  </book>  
  ...  
</catalog>
```

OPENXML ()

- Beispiel:

```
create variable x XML;
set x = (select XMLELEMENT( name catalog,
                           XMLAGG( book ) ) from Catalog )

select *
from OPENXML( x, '//book' )
      WITH ( title varchar(25) './title',
            pages integer './pages' )
```

- OPENXML ist die einzige Möglichkeit, eine *xml-bezogene* Bedingung über den *Inhalt* von SQL/XML-Attributen zu definieren.

Nicht kompatibel mit SQL/XML Standard

Der erste Schritt erstellt eine Variable `x`, die ein XML-Dokument mit einem einzigen Wurzelement `catalog` enthält. Die einzelnen `book`-Elemente in den Datensätzen der `Catalog`-Tabelle werden zu Kindern des `catalog`-Elementes.
Zu beachten: Die `select`-Anweisung für die Variable `x` *muss* einen *einzigsten* Datensatz mit einem *einzigsten* Wert zurückliefern, der das gesamte XML-Dokument enthält.

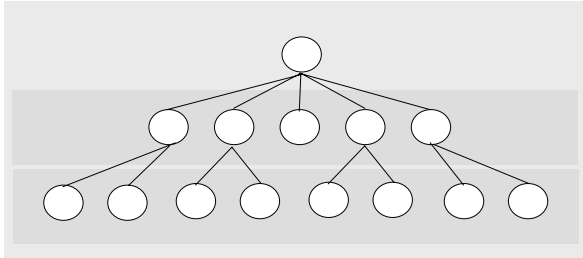
In einem zweiten Schritt wird mit `OPENXML ( )` aus der XML-Struktur eine *virtuelle* Tabelle konstruiert. Man kann das als eine Art "relationale View" auf dem XML-Dokument ansehen.

Resultat

Learning XML	350
Inside XML	1102
Step by Step XML	375
The XML Companion	566
XML	550
XML Bible	1300
XML in a Nutshell	496

Falls die zweite XPath-Abfrage innerhalb der `WITH()`-Klausel mehrere Elemente zurückliefert, wird die Ausgabe auf das erste beschränkt.

OPENXML ()



create variable: XML erstellen

openxml: Knoten = Datensatz

with: Knoten = Attribut

Die von openxml() erzeugte, virtuelle Tabelle kann überall eingesetzt werden, wo ein Tabelle vorkommen darf. Beispiel:

```
select * from tabelle, openxml(), tabelle, openxml()  
where ....
```

Nicht kompatibel mit SQL/XML Standard

OPENXML ()

1. Mit einem ersten SQL-Query wird ein gültiges (single-rooted!) XML-Dokument erstellt.
2. Mit einem XPath-Query wird darauf eine Knotenmenge generiert.
3. Jeder Knoten entspricht *einem* Datensatz einer virtuellen Tabelle.
4. Weiter unten liegende Knoten werden zu Attributen der Tabelle.
5. Die virtuelle Tabelle wird im `from`-Teil einer weiteren SQL-Abfrage verwendet, ev. zusammen mit anderen Tabellen.

```
create variable x XML;  
set x = (select xmlGeneratingExpression from ... )  
  
select *  
from OPENXML( x, XPathQuery )  
WITH ( virtualTableDefinition )
```

Nicht kompatibel mit SQL/XML Standard

xmlGeneratingExpression ist irgendein Ausdruck, der ein gültiges XML-Dokument erzeugt.

XPathQuery ist eine XPath-Abfrage, welche auf das XML-Dokument wirkt und eine Menge von Teilbäumen (Wald) extrahiert.

Aus jedem durch *XPathQuery* definierten Teilbaum wird ein Datensatz in Tabelle erstellt.

OPENXML ist auf den ersten Blick etwas umständlich, jedoch potentiell ein sehr mächtiger Befehl, weil mit XPath auf ein zugrundeliegendes XML-Dokument zugegriffen werden kann. Die virtuelle Tabelle kann Teil einer Verknüpfung mit weiteren SQL-Tabellen oder OPENXML-Definitionen sein.

OPENXML () lässt in der gegenwärtigen Implementation keine Prädikate innerhalb von XPathQuery zu. Es können also lediglich Teilbäume extrahiert werden. Allfällige Bedingungen müssen über die where-Klausel auf die virtuelle Tabelle gesetzt werden, oder vorgängig beim Zuweisen der Variable `x` für OPENXML.

OPENXML () ist noch kein Teil des vorgeschlagenen Standards.

Ausgabe mit `select ... for xml`

- Viele Datenbanksysteme haben eine einfache XML-Ausgabeformatierung mit folgenden oder ähnlichen Konstrukten eingeführt:

```
select ... for xml raw
select ... for xml auto
select ... for xml auto, elements
select ... for xml explicit
```

- Der Vorteil, respektive auch Nachteil, liegt darin, dass die XML-Fähigkeiten auf die reine *Ausgabe* beschränkt sind.
- Die Datenbank muss XML als Datentyp nicht kennen.

Nicht kompatibel mit SQL/XML Standard

Variante 1 gibt jeden Resultatdatensatz in der Form aus

```
<row att1="wert1" att2="wert2" ... />
```

Variante 2 gibt jeden Resultatdatensatz in der Form aus

```
<tablename att1="wert1" att2="wert2" ... />
```

Variante 3 gibt jeden Resultatdatensatz in der Form aus

```
<tablename>
  <att1>wert von att1</att1>
  <att2>wert von att2</att2>
</tablename>
```

Variante 4 ermöglicht eine individuelle Darstellung von jedem Tabellenattribut als XML-Element oder als XML-Attribut. Ausserdem ist ein Zusammenzug von Datensätzen zu einer verschachtelten XML-Struktur möglich. Der Gebrauch dieser Variante ist jedoch umständlich und schwerfällig. Die Verwendung der Funktionen `XMLELEMENT()`, `XMLAGG()`, `XMLGEN()`, `XMLCONCAT()` usw. scheint besser geeignet.

Beispiel für select ... for xml raw

```
select firstname, lastname  
from Author  
for xml raw
```

```
<row firstname="Elliotte" lastname="Harold"/>  
<row firstname="Erik" lastname="Ray"/>  
<row firstname="Michael" lastname="Young"/>  
<row firstname="Lisa" lastname="Rein"/>  
<row firstname="Neil" lastname="Bradley"/>  
<row firstname="Steven" lastname="Holzner"/>  
<row firstname="W." lastname="Means"/>
```

Nicht kompatibel mit SQL/XML Standard

Beispiel für select ... for xml auto

```
select firstname fn, lastname ln from Author  
for xml auto
```

```
<Author fn="Elliotte" ln="Harold"/>  
<Author fn="Erik" ln="Ray"/>  
...
```

```
select firstname fn, lastname ln from Author  
for xml auto, elements
```

```
<Author><fn>Elliotte</fn><ln>Harold</ln></Author>  
<Author><fn>Erik</fn><ln>Ray</ln></Author>  
...
```

Beispiel für select ... for xml explicit

```
select
1      as Tag,
null   as Parent,
''     as [Buch!1],
idBook as [!1!id!],
title  as [!1!Titel!element],
name   as [!1!Verlag!element]
from Book natural join Publisher
for xml explicit

<Buch id="B1">
  <Titel>Learning XML</Titel>
  <Verlag>O'Reilly & Associates</Verlag>
</Buch>
...
```



Für Beziehungen von Datensätzen untereinander.

Die SQL-Spaltenalias werden für Formatierungszwecke missbraucht. Die Spalten `Tag` und `Parent` müssen zwingend vorkommen, wenn mit `for xml explicit` gearbeitet wird. Sie können verwendet werden, um anhand einer Menge von Datensätzen eine hierarchische Struktur zu erstellen.

Die Struktur der Formatierungsangabe ist wie folgt zu verstehen

[ElementName!TagNr!AttributeName!Direktive]

ElementName: Der Name des Elementes, das aus diesem Datensatzes gebildet werden soll. Wenn mehrere Elementnamen für dieselbe TagNr angegeben werden, gilt nur der erste Elementname.

TagNr: Alle SQL-Attribute mit derselben TagNr steuern Informationen zum XML-Element mit dieser TagNr bei, sei es als Attribut für diesen Tag, sei es als Textinhalt für diesen Tag oder sei es als weitere Unterelemente.

AttributeName: Wenn diese Angabe vorkommt, wird das SQL-Attribut als XML-Attribut des angegebenen Tags eingefügt.

Direktive: Heisst die Direktive `element`, wird das SQL-Attribut nicht als XML-Attribut, sondern als XML-Element eingefügt.

Zu beachten: Die TagNr ist *statisch*. Es kann keine Variable oder ein SQL-Attribut für die TagNr angegeben werden. Das Konstrukt `for xml explicit` ist damit nicht geeignet, *dynamisch* eine verschachtelte XML-Struktur zu generieren (siehe Folien über die generische XML-Datenstruktur).

Der Oracle Ansatz

- Datentyp `XMLTYPE` analog XML im Standard.
- Wesentliche Erweiterungen für das Extrahieren von XML-Daten mit `extractValue()` Funktion.
- Die `extractValue()` -Funktion nimmt einen allgemeinen XPath-Ausdruck entgegen.
- Die Indexierung über einen XPath-Ausdruck ist möglich.
- Teilweiser Update in einem `XMLTYPE`-Attribut ist möglich mit `xmlupdate()` -Funktion.

Beim Einfügen oder Ändern von XML-Daten wird die wohlgeformtheit überprüft.

Die `extractValue()` Funktion nimmt einen XPath-Ausdruck entgegen und liefert ein Resultat vom Typ `XMLTYPE`.

siehe Beispiele in `Oracle-SQL-XML-Beispiele.pdf`

Oracle9i SQL-Beispiele

Diplomarbeit Bruno Hofer, November 03

Gekürzt, Arno Schmidhauser, Februar 04

Beispiele mit dem XML-Table Ansatz von Oracle

create table

create table myTable of xmltype;

insert

```
insert into myTable values(
XMLTYPE('
    <user>
      <id>123</id>
      <lastname>hofer</lastname>
      <firstname>bruno</firstname>
      <identification>hob</identification>
    </user>
'));
```

insert mit einem nicht wohlgeformten XML
-> Endtag lastname nicht korrekt

```
insert into myTable values(
XMLTYPE('
    <user>
      <id>124</id>
      <lastname>mueller<lastname>          <-- hier fehler
      <firstname>josef</firstname>
      <identification>muj</identification>
    </user>
'));
```

*

FEHLER in Zeile 2:

ORA-31011: XML-Parsing nicht erfolgreich

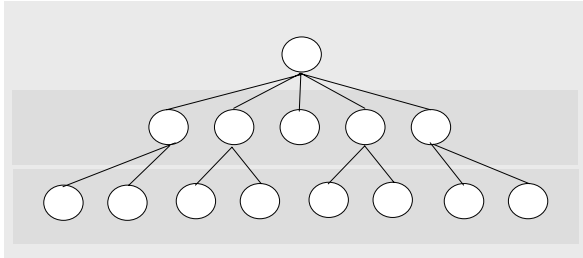
ORA-19202: Fehler bei XML-Verarbeitung

LPX-00225: end-element tag "user" does not match

start-element tag "lastname"

Error at line 29

OPENXML ()



create variable: XML erstellen

openxml: Knoten = Datensatz

with: Knoten = Attribut

Die von openxml() erzeugte, virtuelle Tabelle kann überall eingesetzt werden, wo ein Tabelle vorkommen darf. Beispiel:

```
select * from tabelle, openxml(), tabelle, openxml()  
where ....
```

Nicht kompatibel mit SQL/XML Standard