

Inhaltsverzeichnis

1 Problem	1
1.1 Wozu XML in DB?	1
1.2 Relational vs XML	2
2 XML-Techniken	3
2.1 XPath	3
2.2 XQuery	5
2.3 Updates	7
3 Oracle XML-DB	7
3.1 Architektur	7
3.2 Repository	10
3.3 Views	11
3.4 Beispiele	12
4 Quellen	14

1 Problemstellung

1.1 Wozu XML in DB?

Es gibt immer mehr XML-Datenquellen

- Web-Dokumente in (X)HTML
- diverse Anwendungsformate (RSS, SVG, ODT, DOCX)
- Nachrichten in SOA-Umfeld
- Dokumente aus CMS-Systemen
- verpackte EDI-Daten

Was macht man mit XML in der DB?

- Man kann XML als CLOBs speichern ...
- ... bei Inserts serialisieren ...
- ... und bei Selects wieder DOMs bauen.
- Alle XML-Techniken finden dann außerhalb statt.
- **Das ist leider teuer!**
- **Und was wird aus Verlinkungen?**

Die ständige Serialisierung und das Parsing von XML-Daten ist nicht in allen Anwendungsfeldern akzeptabel. Hinzu kommt, dass XML-Datenbestände oft hochgradig verlinkt sind und somit eine Infrastruktur benötigt wird, die das Ziel eines Links bei Bedarf in der DB findet und bereitstellt.

1.2 Relational vs XML

Relational vs XML

RDBMS	XML
Flach	Hierarchisch
ungeordnet	geordnet
flexibel	starr
komplex/abstrakt	natürlich
Datenmodell	Schema
Verknüpfungen	Links (XInclude, XPointer)
SQL	XPath, XQuery, XSLT ...
Große Datenbestände	Nachrichten, Formulare, Dokumente
Mapping mit ORM	JAXB, DOM4J

Hierarchie: Mit XML erleben hierarchische Datenbanken, wie sie vor dem Siegeszug der RDBMS üblich waren, gewissermaßen eine Renaissance. In XML sind hierarchische Beziehungen in den Daten direkt an Enthaltensein-Beziehungen zwischen Elementen ablesbar, in einem RDBMS werden diese erst dynamisch durch verknüpfende Abfragen (wenn auch statisch assistiert durch Datenorganisation mit Primar- und Fremdschlüssel) hergestellt.

Flexibilität: In einem RDBMS wird die Flexibilität mit der Komplexität der Verknüpfungen bezahlt, während in XML die natürliche Sicht auf die Daten abgebildet wird. Dafür sind bei XML zur Transformation dieser starren Hierarchien teure Techniken wie XSLT erforderlich. Für kleinere Transformationen genügt auch das schlankere XQuery.

Datenmodell und Schema: Hat man in einem RDBMS erst einmal ein Datenmodell erstellt, sind spätere Änderungen aufwändig. In XML ist alles eher ‚fliegend‘ verdrahtet, Änderungen der validierbaren Schemata kommen immer wieder vor. Die Daten sind mit Web-Techniken wie einfachen URLs oder XPointer adressierbar, aber die Existenz des Ziels ist nicht sichergestellt. Ein RDBMS hingegen wacht über die referenzielle Integrität.¹

SQL vs XPath: In der Abfrage von Daten kann XML mit XPath 2 und dem darauf aufbauenden XQuery bereits einigermaßen mit SQL-Selects mithalten. Auch die Validierung von Daten mit Schemata steht den Constraints eines RDBMS nicht nach. Im Bereich der modifizierenden DML (Update, Insert, Delete) hat sich in der XML-Welt aber noch kein echter Standard etabliert.

Einsatzbereich: In einem RDBMS wird häufig alles Wissen eines Unternehmens universell verknüpfbar abgelegt. Die XML-Datenbestände sind oft eher kleine Inseln mit bestenfalls losen Kontakten. Im Bereich der Abbildung von Dokumenten wie Büchern, die von Natur aus hierarchisch sind, ist XML jedoch stärker.

¹Ein Vorgriff: Die Oracle XML-DB kann die referenzielle Integrität auch in XML-Daten sicherstellen

Gemeinsamkeiten von RDBMS und XML

	RDBMS	XML
Datentypen	ja	ja (mit Schema)
Strukturierung	Datenmodell	Schema (früher DTD)
Verknüpfungen	PK/FK	HREF, XPointer, XLink, XInclude
Binärdaten	BLOB	Base64-Encoding
Zugriff	SQL	XQuery, XUpdate

Die Datenablage in einem RDBMS und in XML haben viele Eigenschaften gemeinsam, wenn sie auch mit verschiedenen Techniken realisiert sind. Das vereinfacht den Brückenbau zwischen den Welten erheblich.

2 XML-Techniken

XML-Techniken

Die folgenden Techniken können als XML-Fassade eines RDBMS dienen:

XPath bietet einen Zugriff auf Teile eines XML-Dokument

XQuery erweitert XPath um Merkmale wie Sortierung, Joins und einfache Transformationen

XSLT ermöglicht komplexe Transformation

XMLSchema liefert Typisierung und das ‚Datenmodell‘

Wenn XQuery das Äquivalent zu einem Select ist, so bleibt die Frage, was mit Insert, Delete und Update ist?

Auf Insert und Delete wird noch bei Speichertechniken eingegangen.

Adressierung ist der halbe Weg zum Update: Wenn der zu ändernde Teil eines XML-Dokuments adressiert werden kann, ist der Update auf Löschen und Insert dieses Teils zurückführbar.

2.1 XPath

Wir sehen in Abbildung 1 folgende 7 Knotentypen:

Drei davon, die Elemente (blaue Kreise), ihre Attribute (rote Ovale) und die Textknoten (schraffierte Rechtecke) tragen die Dateninhalte.

Die weißen Quadrate sind nur spezielle Textknoten: ignorable Whitespace, der meist nur aus Formatierungsgründen eingefügt wird. Viele Parser können so eingestellt werden, dass sie derartige Knoten von vornherein ignorieren.

Der Dokumentknoten (die Wurzel) dient als ‚Aufhänger‘, damit das Dokument eine definiert Wurzel hat. Neben dem eigentlichen Wurzelement (hier ‚Rezept‘) können ja noch beispielsweise Kommentare stehen.

Die anderen Knotentypen sind eher Metadaten: Die Namespace-Deklaration, ebenfalls über eine eigene Achse an Elemente gebunden, bindet nur einen Präfix an eine URI. Kommentare und Processing-Instructions sind normale Kindelemente und können an beliebigen Stellen vorkommen.

Das Baummodell von XPath unterscheidet sich von dem von DOM. Ein Attribut oder eine Namespace-Deklaration sind hier ein Blätter des Baumes, wie ein Text- oder Kommentarknoten.

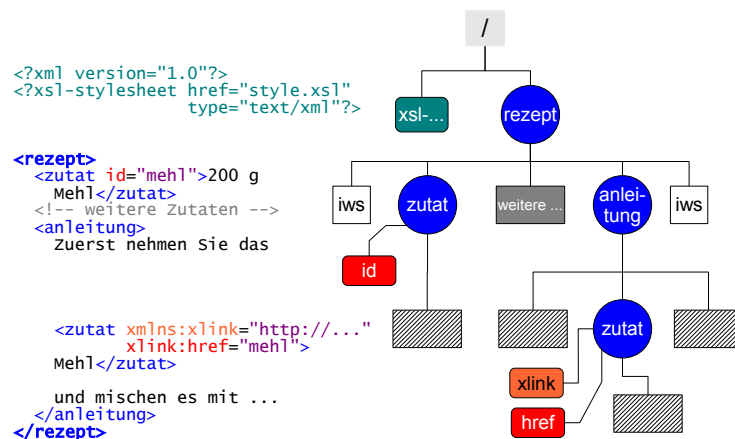


Abbildung 1: XPath-Baummodell

XPath-Ausdrücke

- XPath-Ausdrücke bestehen aus Location-Steps:
 - Step1 / Step2 / Step3 ... oder / Step1 / Step2 / Step3
- Jeder Location-Step besteht aus:
 - Achse :: Knotentest [Prädikat1] [Prädikat2] ...
- Achsen:
 - self (.), attribute (@), namespace, child, parent (..), following-sibling, preceding-sibling, ancestor, descendant-or-self (//) ...
- Knotentests:
 - name, *, text(), node(), comment()
- Prädikate:
 - boolesche Ausdrücke zur Filterung der Mengen

Ein mit / beginnender Ausdruck startet ab der Wurzel. Ansonsten beginnt die Suche ab dem Bezugsknoten, wie z.B. dem Knoten, auf den ein XSLT-Template angewendet wird.

Die Achse **self** adressiert den Bezugsknoten selbst. Mit **attribute** gelangt man zu seinen Attributen und mit **namespace** zu den Namespace-Deklarationen.

Die Achsen **child**, **parent** und **following/preceding-sibling** markieren die vier möglichen Nachbarn eines Elements. Die Achsen **ancestor** und **descendant** erweitern die Reichweite von **parent** und **child**. Weiterhin stehen Kombinationen wie **descendant-or-self** und **ancestor-or-self** sowie die Achsen **following** (zuvor besuchte Knoten) und **preceding** zur Verfügung. Für einige Achsen existieren Abkürzungen. Diese sind in Klammern angegeben.

Es gibt auch noch Mengenvereinigungen mit `|` und die Möglichkeit, Steps mit Klammern zusammenzufassen. Weitere Knotentests sind `root()` und `processing-instruction()`.

XPath-Prädikate

- Ein Prädikat filtert die links davon stehende Knotenmenge
- komplexe boolesche Ausdrücke unter Einbeziehung der gesamten Dokuments möglich
- Funktionen in Prädikaten:

`position()` Position des Knotens in selektierter Menge

`last()` Anzahl der Knoten in der Selektion

`count(expr)` Anzahl Knoten in `expr`

Stringfunktionen wie `substring(s,start,end)`, `concat(a,b)` etc.

Mathematische Operatoren und Funktionen, Vergleichsoperatoren ...

Beispiele: Siehe XPath-Visualizer

Zur Vereinfachung sind diverse Abkürzungen definiert. So darf der Ausdruck

```
[position() = n
```

einfach mit `[n]` abgekürzt werden.

Ein Ausdruck, der eine Knotenmenge liefert, gilt als `false`, wenn die Menge leer ist und ansonsten als `true`. Somit kann man `attribute::*` bzw. `@*` verwenden, um das Vorhandensein von Attributen zu prüfen, statt `count(@*) = 0` zu schreiben.

In den booleschen Ausdrücken stehen natürlich auch die Operatoren `and`, `or` sowie `not(expr)` zur Verfügung.

2.2 XQuery

XQuery

- Übergang von XPath v1 auf v2 ermöglichte neue Techniken
- XPath v2 kennt Datentypen (schema aware)
- XQuery erweitert XPath wie folgt:
 - komplexere Filterungen
 - einfache Transformationen (für komplexe gibt es XSLT)
 - Verknüpfung mehrerer Datenquellen („Joins“)
 - Sortierungen
- FLWOR-Aufbau: For, Let, Where, Order-By, Return

XQuery kann Datentypen aus einem Schema ableiten und damit, wie XPath 2, eine typsichere Verarbeitung der Daten gewährleisten. XQuery ist ein W3C-Standard und hat sich mittlerweile zum Standard für Abfragen von XML-Datenquellen gemauert.

Vergleich XPath / XQuery

```
//Warehouse[country='USA']/Area

for $i in //Warehouse[country='USA']
where $i/Area > 50000
return <Details>
  <Docks num="{ $i/Docks }"/>
  <Rail>{ if ($i/RailAccess="Y") then "true"
        else "false" }
  </Rail></Details>
```

Weitere Möglichkeiten:

- Let-Klausel: Hilfsvariablen für mehrfach verwendete Ausdrücke
- Order-By-Klausel: Sortierung
- mehr als eine Datenquelle (Joins)

Der XPath-Ausdruck sucht zunächst Knoten namens ‚warehouse‘ auf beliebiger Tiefe des XML-Dokuments, filtert dann jene heraus die ein Attribute ‚country‘ mit Inhalt ‚USA‘ aufweisen und gibt dann die Inhalte der Kindelemente ‚Area‘ zurück. In diesem Beispiel sind dies Texte.

Im XQuery-Ausdruck iteriert die Variable \$i ebenfalls über die Elemente namens ‚warehouse‘ der USA, filtert in der Where-Klausel jene mit Kindelement ‚Area‘ größer 50.000 (das wäre auch noch mit einem XPath-Prädikat wie [Area>50000] möglich gewesen), und konstruiert dann einen Ergebnisbaum im return-Statement.

Die so konstruierte Rückgabe besteht vor allem aus XML-Daten, ist jedoch durchsetzt mit XPath2-Ausdrücken in geschweiften Klammern {}, mit denen Werte aus der Umgebung der Fundstelle \$i eingebaut werden. Man beachte, dass die Struktur des Ergebnisbaums von der des Ursprungdokument abweichen kann. XQuery kann also auch transformieren und nicht, wie XPath 1, nur extrahieren. Ebenso bemerkenswert ist, dass mehrere Laufvariablen verwendbar sind und somit auch Joins realisierbar sind.

Man beachte auch die Behandlung von NULL-Werten (nächstes Beispiel):

Wenn ein XPath-Ausdruck eine leere Sequence liefert, so wird das Ergebnis NULL. Bei Teilergebnissen wie dem Wert eines Attributs verschwindet das Attribut, wie im Beispiel /Details/Docks/@num.

XQuery in einer DB

- ```
SELECT warehouse_id "Id",
EXTRACTVALUE(warehouse_spec, xpath1) "Area",
XMLQuery(xquery PASSING warehouse_spec
 RETURNING CONTENT) "Big_warehouses"
FROM warehouses;
```

| ID    | Area   | Big_warehouses                                                   |
|-------|--------|------------------------------------------------------------------|
| 1     | 25000  |                                                                  |
| 2     | 50000  |                                                                  |
| 3     | 85700  | <Details><Docks></Docks><br><Rail>>false<Rail></Details>         |
| 4     | 103000 | <Details><Docks num="3"></Docks><br><Rail>>true</Rail></Details> |
| . . . |        |                                                                  |

Wo man früher mit XPath 1 und vielen Funktionen wie EXISTSNODE, EXTRACTVALUE, EXTRACT hantieren musste, kann man nun eine Funktion für alles verwenden:

`XMLQuery(string [ passing_clause ] RETURNING CONTENT)`

Mit *xquery* oder *xpath1* sind Zeichenketten gemeint, wie wir

Der Select liefert in der Spalte `warehouse_spec` ein in der DB gespeichertes XML-Dokument. Mit der XPath v1 Funktion `EXTRACTVALUE` kann man daraus einen einfachen Wert extrahieren, mit der Funktion `XMLQuery` erhalten wir hingegen einen Auszug des Baumes, der auch noch passend transformiert und/oder sortiert werden kann.

Die `PASSING`-Klausel definiert den Input für XQuery. In diesem Beispiel wird nur eine Quelle verwendet, aber mehrere sind möglich. Zusätzliche Quellen müssen aber Alias-Namen erhalten, um sie von der unbenannten Standardquelle zu unterscheiden. Es gibt auch die Möglichkeit, mit Funktionen wie `ora:view` eine Quelle innerhalb der XQuery-Abfrage zu definieren.

Die Klausel `RETURNING CONTENT` ist im Standard SQL/XML (siehe [www.sqlx.org](http://www.sqlx.org)) definiert und besagt, dass hier ein Fragment eines XML-Dokumentes zurückgegeben wird. Es ist der einzige Rückgabewert, den Oracle unterstützt. In SQL/XML sind noch der allgemeinere Super-Typ `SEQUENCE` und der einschränkendere Subtyp `DOCUMENT` definiert.

## 2.3 Updates

### Updates

- Insert und Delete mit SQL-Mitteln möglich
- Ersetzen eines Dokuments mittels Delete und neuem Insert
- Mächtiger: `updateXML(document, node, value)`
  - Ersetzen eines Teilbaums in einem XML-Dokument (`document`):
  - Dazu Teilbaum (`node`) mit XPath adressieren ...
  - ... und diesen durch neues Teildokument (`value`) ersetzen

Unter der Ersetzung eines Teildokuments kann man sich einfachste Operationen wie das Ersetzen eines Textknotens oder komplexe Vorgänge wie das Einfügen eines weiteren Elements mit tiefem Inhalt vorstellen.

## 3 Oracle XML-DB

### 3.1 Architektur

#### Architektur der XML-DB

XML-Daten können als Tabellenzeilen oder in Tabellenspalten abgelegt werden. Die Ablage kann als unstrukturiertes CLOB oder, bei Bindung an ein registriertes Schema, strukturiert (relationales Backend) erfolgen.

Vorhandene relationale Daten können mit SQL-Mitteln in XML konvertiert und somit auch als XMLType-Views bereitgestellt werden. Es existieren umgekehrt auch Funktionen, um XML-Daten via XPath oder XQuery aus SQL heraus anzusprechen.

Die XML-Daten können in einem Repository via URLs adressiert und mit HTTP, WebDAV und FTP angesprochen werden.

Für all diese Mechanismen existieren APIs für Java und PL/SQL sowie C/C++. Für .NET stellt ODP.NET (Oracle Data Provider) einen Datentyp `OracleXmlType` zur Verfügung.

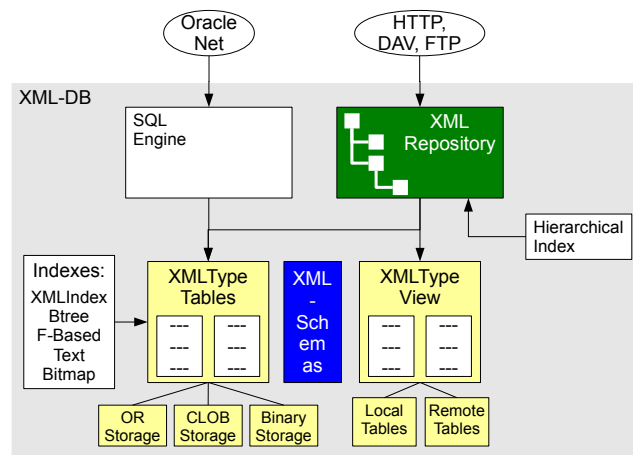


Abbildung 2: Architektur Oracle-XML-DB

## Speicherstrategien

- XML wird immer in XMLType verpackt.
- Für XMLType sind drei Speicher-Backends möglich

**Relational:** Voreinstellung, strukturiert

- Fokus auf effizienten Zugriff ins Dokument
- Wiederholende Elemente werden zu speziellen Tabellen (OCT)

**CLOB:** Unstrukturiert

- Schnell bei Insert/Select/Delete des gesamten Dokuments
- Zugriff auf Teilinhalte des Dokuments sind teuer

**Binary:** Neue Alternative zu CLOB

- als einzige Speicherform in Voreinstellung voll validierend

- Mischformen sind möglich

**Strukturierte Ablage** Bei der relationalen Speicherung wird jeder ComplexType eines Schemas mit maxOccurs größer 1 als SQL-VArray in einer OCT (Object Collection Table) gespeichert. Alle Zugriffe via XQuery oder XPath basieren auf der Anwendung von XPath-Ausdrücken auf die XML-Daten. Diese XPath-Ausdrücke werden in einem XPath-Rewrite genannten Prozess in SQL-Ausdrücke übersetzt.

**Unstrukturierte Ablage** Bei der Speicherung als CLOB müssen die Daten zur Anwendung von XPath erst von einem Parser in einen Baum transformiert werden. Dieser Prozess ist aufwändig. Bei der Variante Binary XML sind manche Zugriffe etwas effizienter als bei CLOB. Bei komplexen Abfragen muss jedoch auch diese Form der unstrukturierten Ablage passen.

**Mischformen** Es ist möglich, für verschiedene Teile eines XML-Dokuments verschiedene Speicherstrategien zu wählen. Dies lässt sich durch direkte Angaben im DDL-Statement oder durch Annotation des Schemas erreichen. Man kann also Teilbäume, bei denen keine Einblicke erforderlich sind, unstrukturiert ablegen.

**Validierung** Oracle XML-DB führt in der Voreinstellung nur bei Binary-XML eine volle Validierung gegen das Schema durch. Ansonsten wird nur geprüft, dass das Dokument zu dem relationalen Unterbau passt. Die Strukturprüfung ist so zwar vollständig, aber Verstöße gegen Restriktionen wie reguläre Ausdrücke auf Stringwerte würden nicht erkannt. Eine volle Validierung lässt sich als Table-Constraint aktivieren, ist aber natürlich teuer.

## Schema Evolution

Was ist zu tun, wenn das Schema geändert werden muss?

- Ähnlich aufwändig wie Änderung des Datenmodells
- Wahrscheinlicher als Datenmodelländerungen
- Unterstützt durch PL/SQL-Package DBMS\_XMLSCHEMA
- Zwei Möglichkeiten:

**Copy-based:** Aufwändig, erfordert meist XSLT

**In-Place:** Schnell, geht aber nur, wenn bestehende Dokumente nicht invalidiert werden

Die Funktion `dbms_schema.copyEvolve` hat einige Beschränkungen:

- Indexe, Trigger und Constraints, die mit dem alten Schema verbunden waren, müssen neu erzeugt werden.
- Änderung von Top-Level Elementen erfordert zusätzliche Aktionen.
- Ein von einem Schema abhängender Objekttyp blockiert die Änderung. Tabellen mit diesem Objekttyp müssen dann ebenfalls migriert werden.

## Indexierung

- Bei rein strukturierter Speicherung genügen normale Indizes auf den Basistabellen
- Für Volltextsuche von Textknoten genügt Typ CONTEXT
- Manche Indexe werden automatisch von Oracle umgeschrieben
- Bei (teilweise) unstrukturierter Speicherung hilft XMLIndex
  - Anwendbar mit und ohne Schema
  - hat strukturierte und unstrukturierte Komponente
  - strukturierte Komponente adressiert unstrukturierte Inseln
  - unstrukturierte Komponente verwendet XPath-Table

Oracle empfiehlt, die früher verwendeten Indextypen CTXXPath oder Function-Based-Indexe durch XMLIndex zu ersetzen.

Oracle erkennt mitunter selbst, wenn ein function based Index angelegt wird, dass ein Ausdruck der Form `extractValue` oder `XMLCAST(XMLQUERY(...))` durch einen B-Tree-Index ersetzt werden kann und tut dies stillschweigend. Das funktioniert aber nicht bei Collection-Elementen.

### 3.2 XML-DB-Repository

#### Was ist das XML-DB-Repository?

- Dateisystemartige Ablage von Ressourcen
- Ressource ist **Folder** oder **File**
- Ressource wird identifiziert über **Path** und **Name**
- Ressource besteht aus **Metadaten** und **Content**
- Content kann XML sein, muss aber nicht
- Zugriffskontrolle via ACLs
- Eingebaute Versionierung
- hard und weak Links

Für viele Anwendungszwecke ist die Ablage auch und insbesondere von XML-Dateien in einem Dateisystem bequemer als in einer Tabellenzeile oder Spalte. Für diese Fälle stellt die XML-DB das XML-DB-Repository bereit.

Die Metadaten sind unterteilt in einen systemdefinierten Teil, den Oracle zur Verwaltung verwendet und einen optionalen benutzerdefinierten, in dem der Benutzer freies Spiel hat.

Änderungen der Daten erzeugen neue Versionen gemäß dem IETF WebDAV Standard. So bleiben frühere Datenbestände zugreifbar.

XML-Content wird speziell behandelt, indem er z.B. in XMLType verpackt wird, so dass die XML-Zugriffstechniken effizient angewendet werden können. Für ein Schema kann eine Standardtabelle eingerichtet werden, in der dann neue Dokumente abgelegt werden. Somit wird der Upload einer Datei auf die URL eines Folders wie

```
http://oe:oe@10.0.0.153:18082/home/OE/PurchaseOrders/2002/Sep/
```

effektiv einen Insert in einer Tabelle `OE.PurchaseOrder` auslösen. Zusätzliche werden Metadaten wie Dateiname, Pfad, Datumstempel etc. im Repository abgelegt. Jede Resource trägt diese Metadaten auf den oberen Ebenen des XML-Baumes. Der eigentliche Inhalt befindet sich dann unterhalb eines Elementes namens ‚Content‘.

Außer Ressourcen befinden sich im Repository auch noch Links auf andere Ressourcen. Harte Links verhindern, dass das Ziel gelöscht wird.

#### Wie greift man auf das Repository zu?

- Mit SQL über die Views `RESOURCE_VIEW` und `PATH_VIEW`
- Mit PL/SQL über das Package `DBMS_XDB`
- Mit Java über das Oracle XML DB resource API for Java
- Mit Protokollen wie FTP, HTTP(s) oder WebDAV ...
- ... also beispielsweise mit Browser oder Windows Explorer

Die Lokalisierung von Ressourcen via Pfad wird durch einen speziellen hierarchischen Index beschleunigt.

## Verwaltung von Links

- HTTP-Zugriff auf Repository bietet Ziele für Hyperlinks via URL
- Mit Folder-Links kann eine Resource mehrere Zugriffspfade haben
- Dokument-Links werden mit XInclude oder XLink genutzt
  - XInclude definiert Verbunddokumente
  - XLink stellt komplexe anwendungsspezifische Verbindungen her
  - Traversierung mittels XPath-Funktion `fn:doc`.
  - XML-DB unterstützt einfache unidirektionale XLinks
- XML-DB erkennt und verwaltet XInclude-Daten und XLink-Daten

Außer Folder-Links, die verschiedene Pfade zu einer Resource darstellen, gibt es auch Links in einem Dokument auf ein anderes. Wenn beide im XML-DB-Repository liegen, werden diese Dokumentenlinks von XML-DB erkannt und verwaltet. Je nach Konfiguration werden daraus hard oder soft Links auf die referenzierte Resource.

## 3.3 XML-Views

### XML-Views

- Verkleidung relationaler Daten als XMLType
- ermöglicht Verwendung von XML-Techniken
- alternative Speicherstrategie für XMLType
- Views mit und ohne Schema möglich
- Erzeugung von XML mit SQL/XML-Funktionen
  - XMLElement, XMLAttribute, XMLForest etc.

Auch bei dieser selbstgestrickten strukturierten Speicherung von XMLType kann Oracle Zugriffe via XPath umschreiben. Dies funktioniert auch mit nicht schemabasierten Views.

### Beispiel für XML-View

```
CREATE OR REPLACE VIEW emp_view OF XMLType
 WITH OBJECT ID (XMLCast(xmlquery AS BINARY_DOUBLE))
AS
SELECT XMLElement("Emp",
 XMLAttributes(employee_id AS "empno"),
 XMLForest(e.first_name || ' ' || e.last_name AS "name",
 e.hire_date AS "hiredate"))
 AS "result"
FROM employees e WHERE salary > 15000;
```

Ein `SELECT * FROM emp_view` liefert:

```
<Emp empno="100"><name>Steven King</name>
 <hiredate>2003-06-17</hiredate></Emp>
```

Für Object-Views muss eine OBJECT ID definiert werden. Diese gewinnt man durch Extraktion geeigneter Wert aus den Daten mittels der Funktion XMLQuery.

Mit der Funktion XMLElement wird ein Element gebaut, das dann mit der Funktion XMLAttributes mit einem Attribut und mit XMLForest mit gemischtem Inhalt versehen wird. Auch wenn Namespace-Deklarationen in XML-Sinne keine Attribute sind, so werden sie dennoch mit XMLAttributes erzeugt.

Aus einer Objekttablelle lassen sich ebenfalls XML-Views ableiten. Man setzt dann die Funktionen sys\_xmlgen ggf. mit einer Formatspezifikation (generiert mit der Funktion XMLFormat) ein. Mit DBMS\_XMLSCHEMA.generateSchema kann man aus einem Objekttyp auch automatisch ein Schema generieren.

### DML auf XMLType Views

- Im Allgemeinen ist ein INSTEAD-OF-Trigger erforderlich
- Insert, Delete und Update führen zu Trigger-Aufrufen
- Setzt der XML-View auf einem inhärent DML-fähigen View auf, so ist auch der XML-View DML-fähig
- Typisches Beispiel: XML-View auf Object-View oder Konstruktor

Beispiel:

```
CREATE OR REPLACE VIEW dept_xml of XMLType
XMLSchema "URI" element "Department"
WITH OBJECT ID (XMLCast(XMLQuery(...) AS BINARY_DOUBLE))
AS SELECT dept_t(d.dept_id, d.dept_name, d.location_id)
FROM departments d;
```

In diesem Beispiel sieht man auch, wie ein View an ein Schema gebunden wird. Das verwendete Schema und das Dokument-Element werden mit einer XMLSchema-Klausel benannt.

## 3.4 Beispiele

### Relationale Sicht auf XML

```
CREATE OR REPLACE VIEW purchaseorder_master_view AS
SELECT po.*
FROM purchaseorder pur,
XMLTable('$p/PurchaseOrder' PASSING pur.OBJECT_VALUE as "p"
COLUMNS
reference VARCHAR2(30) PATH 'Reference',
requestor VARCHAR2(128) PATH 'Requestor',
userid VARCHAR2(10) PATH 'User',
costcenter VARCHAR2(4) PATH 'CostCenter',
ship_to_name VARCHAR2(20)
PATH 'ShippingInstructions/name') po;
```

Zum Bau eines relationalen Views auf eine XML-Tabelle ist XMLTable besonders geeignet. Wie die Funktion XMLQuery basiert sie auf XQuery, wird aber in der FROM-Klausel eingesetzt und bricht das XML-Resultat in Spalten auf.

Bei tiefen Strukturen kann man zusätzliche XMLType-Aufruf in die FROM-Klausel packen, die dann weitere Ebenen aufbrechen.

In diesem Beispiel sieht man auch, dass XQuery auch einfach nur aus einem XPath 2 Ausdruck bestehen kann.

## Schema registrieren und Tabelle anlegen

```
CREATE DIRECTORY "XMLSCHEMA" AS '/u01/schema'
;
BEGIN
 DBMS_XMLSCHEMA.registerSchema(
 SCHEMAURL => 'http://server/path/purchaseOrder.xsd',
 SCHEMADOC => bfilename('XMLSCHEMA','purchaseOrder.xsd'),
 CSID => nls_charset_id('AL32UTF8'));
END;
/
CREATE TABLE PURCHASEORDER2 OF XMLTYPE
 XMLSCHEMA "http://server/path/purchaseOrder.xsd"
 ELEMENT "PurchaseOrder"
;
INSERT INTO purchaseorder2
 SELECT XMLQUERY('for $i in /PurchaseOrder return $i'
 PASSING object_value RETURNING CONTENT)
 FROM oe.purchaseorder;
```

## Daten anschauen

XML formatieren mit: XMLSerialize(DOCUMENT|CONTENT ...)

```
SELECT XMLSerialize(DOCUMENT
 XMLType('<root><a>1234</root>')
 AS varchar2(100) INDEXT)
FROM dual;
```

Resultat:

```
"<root>
 <a>12
 34
</root>
"
```

Hier sieht man zum Einen den Einsatz des XMLType-Konstruktors, der aus einem Text ein untypisiertes XMLType-Objekt macht und zum anderen die Funktion XMLSerialize, die genau das umgekehrte bewerkstelligt. Die Serialisierung kann durch Verwendung des Schlüsselworts DOCUMENT statt CONTENT vor dem eigentlichen Argument zu einer Prüfung veranlasst werden, ob das XML auch wohlgeformt ist. Der Konstruktor XMLType akzeptiert nur wohlgeformte Daten.

## Updates

```
UPDATE oe.purchaseorder po
 SET po.OBJECT_VALUE = updateXML(po.OBJECT_VALUE,
 '/PurchaseOrder/Actions/Action[1]/User/text()',
 'SKING')
WHERE XMExists(
 '$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
 PASSING po.OBJECT_VALUE AS "p");

SELECT XMLQuery('/PurchaseOrder/Actions/Action[1]/User/text()')
```

```
PASSING po.object_value RETURNING CONTENT)
FROM oe.purchaseorder po
WHERE ...
```

In updateXML können mehrere Ersetzungspaare angegeben werden. Man ist auch nicht auf die Änderung von Textknoten beschränkt, sondern kann auch als Wert einen XMLType-Konstruktor verwenden und so ganze Teilbäume ändern.

## DBMS\_XMLGEN

```
DECLARE
 qryCtx DBMS_XMLGEN.ctxHandle;
 result CLOB;
BEGIN
 qryCtx := DBMS_XMLGEN.newContext(
 'SELECT d.*, CURSOR(SELECT e.* FROM emp e
 WHERE e.deptno=d.deptno) emps
 FROM dept d ');
 DBMS_XMLGEN.setRowTag(qryCtx, 'DEPT');
 result := DBMS_XMLGEN.getXML(qryCtx);
 dbms_output.put_line(result);
 DBMS_XMLGEN.closeContext(qryCtx);
END;
/
```

Das Besondere an diesem Beispiel ist, dass Sie aus einer SQL-Query, die doch ‚flache‘ liefern soll, hier hierarchische Daten erhalten. Das Package DBMS\_XMLGEN führt mit dem Cursor-Ausdruck eine rekursive Abfrage durch und liefert so einen Teilbaum.

Auf der obersten Ebene erhält man also eine Liste von ‚depts‘-Elementen, die wiederum in ‚emps‘ Collections von Elementen ‚emps\_row‘ enthält.

Diese Funktion kann man auch mit dem Java-API des Oracle XSU (XML-SQL-Utility) verwenden. Via Kommandozeile kann man dieses mit

```
java OracleXML getXML -user scott/tiger 'SQL-Query'
aufrufen.
```

## 4 Quellen

### Quellen

- Oracle 11g Dokumentation (<http://www.oracle.com/pls/db112/portal.all.books>)
  - Oracle XML DB Developer’s Guide
  - Oracle Database SQL Reference
  - Oracle XML Developer’s Kit Programmer’s Guide
- Spezifikationen des W3C
  - XQuery (<http://www.w3.org/TR/xquery>)
  - XPath 2 (<http://www.w3.org/TR/xpath20>)
- Spezifikationen der SQL/XML-Gruppe ([www.sqlx.org](http://www.sqlx.org))