Vorstellung der Diplomarbeit

Implementierung der XPath-Anfragesprache für XML-Daten in RDBMS unter Ausnutzung des Nummerierungsschemas DLN

Oberseminar Datenbanken WS 05/06

Diplomand: Oliver Schmidt

Betreuer: Timo Böhme

Gliederung

1. Zielstellung der Diplomarbeit
2. Grundlagen
3. Das Relationenschema
4. Die Umsetzung
5. Probleme
6. Erste Benchmarkergebnisse
7. Ausblick

Zielstellung der Diplomarbeit

- XPath-Anfrage-Komponente des XMLRDB-Projektes
- generischer Ansatz als Aufsatz auf einem RDBMS
- Versuch der Konvertierung einer XPath-Anfrage in eine einzelne SQL-Anfrage
- praktische Ergebnisse des Nummerierungsschemas DLN
- Unterstützung des kommenden Standards XPath 2.0

- Anforderungen an das RDBMS:
- Unterstützung von Subqueries in SQL
- Unterstützung von User Defined Functions in SQL
- Nutzung von Views, falls vorhanden

Grundlagen

XPath 2.0 – Änderungen zu Version 1.0

- parallele Entwicklung zu XQuery 1.0 als Teilmenge davon

- Übernahme des Typsystems von XQuery
- Mengen als Sequenzen von Typen
- Definition von Variablen
- Schleifen und Verzweigungen
- neue Operationen auf Mengen
- Anfragen auf Kollektionen von Dokumenten

Einschränkungen

- Umsetzung Typsystem nur mit XML Schema möglich
- Verzicht auf XML Schema in XMLRDB, um beliebige XML-Daten speichern zu können
- Beschränkung auf in XPath 1.0 bekannte Typen
- Einschränkung der im Standard vorgesehenen Funktionen
- nur rudimentäre Unterstützung von Namespaces

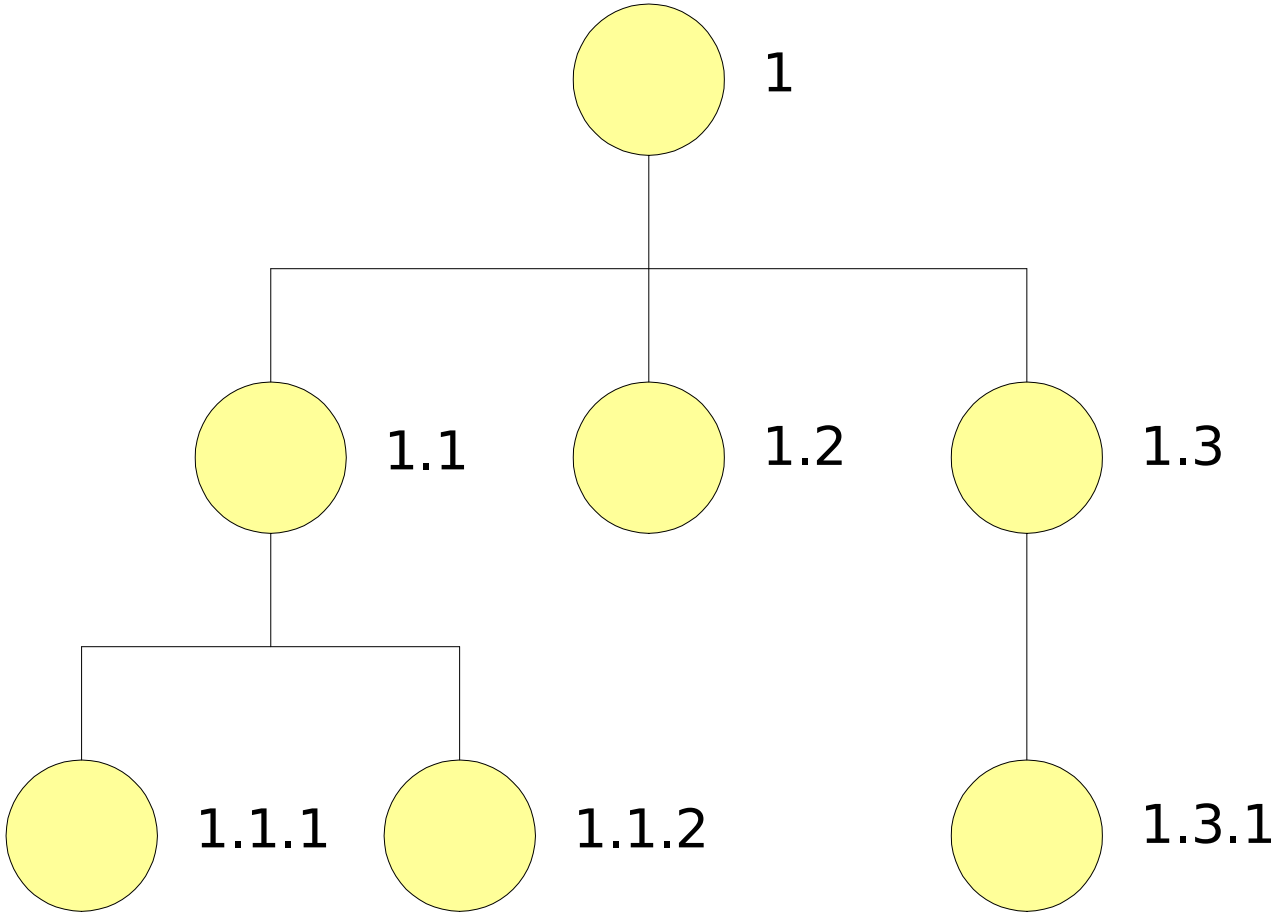
- Verzicht auf Schleifen:
- nicht in einer SQL-Anfrage umsetzbar
- für komplexere Ausdrücke XQuery sinnvoller

Das Nummerierungsschema DLN

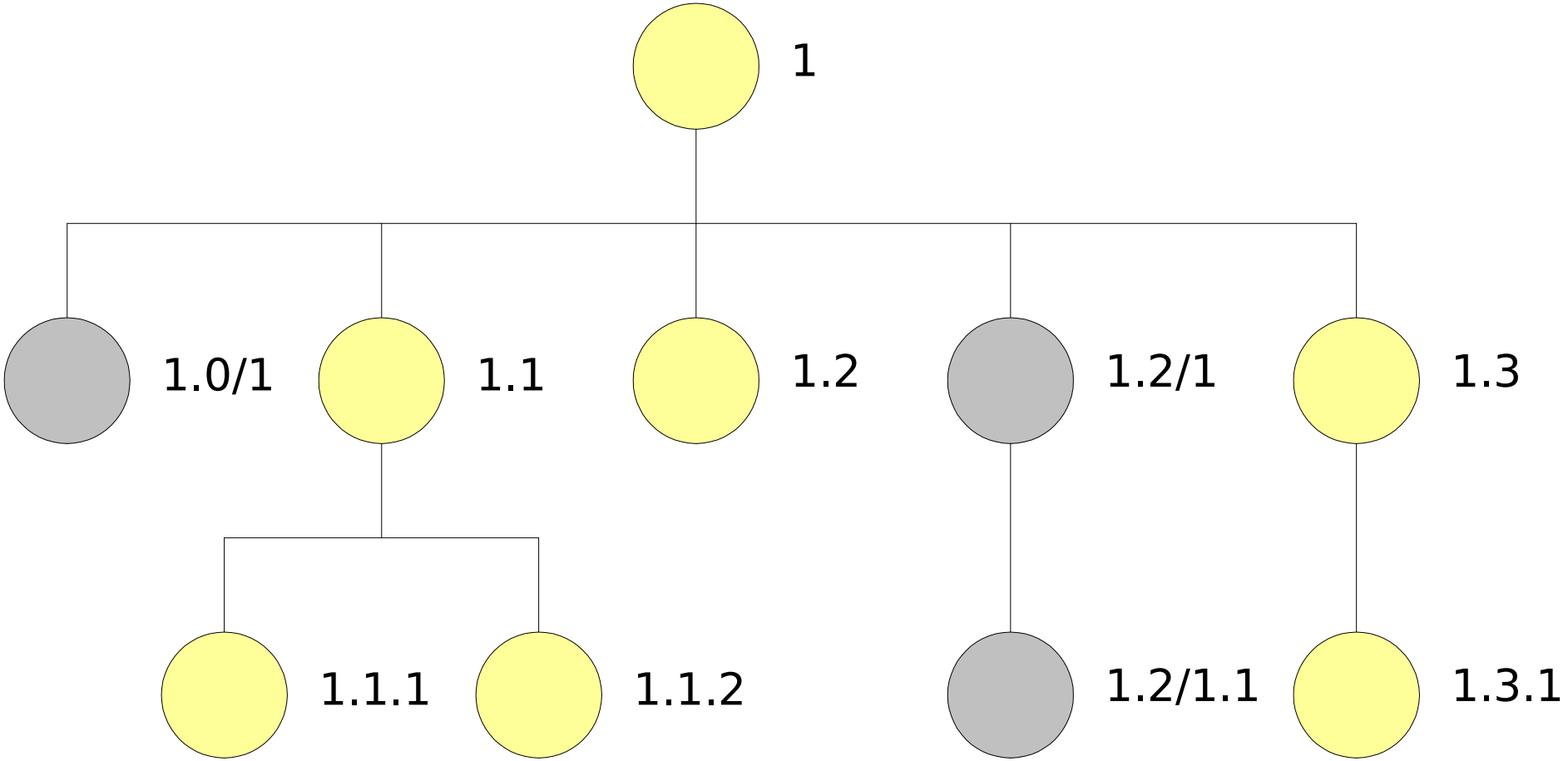
Dynamic Level Numbering

- vergibt eindeutige ID für jeden Knoten im XML-Baum
- kodiert die Position des Knotens innerhalb des Baumes
- erhält eine Preorder-Ordnung der Knoten
- Änderungen benötigen trotzdem keine Neunummerierung

DLN-Beispiel



DLN-Beispiel



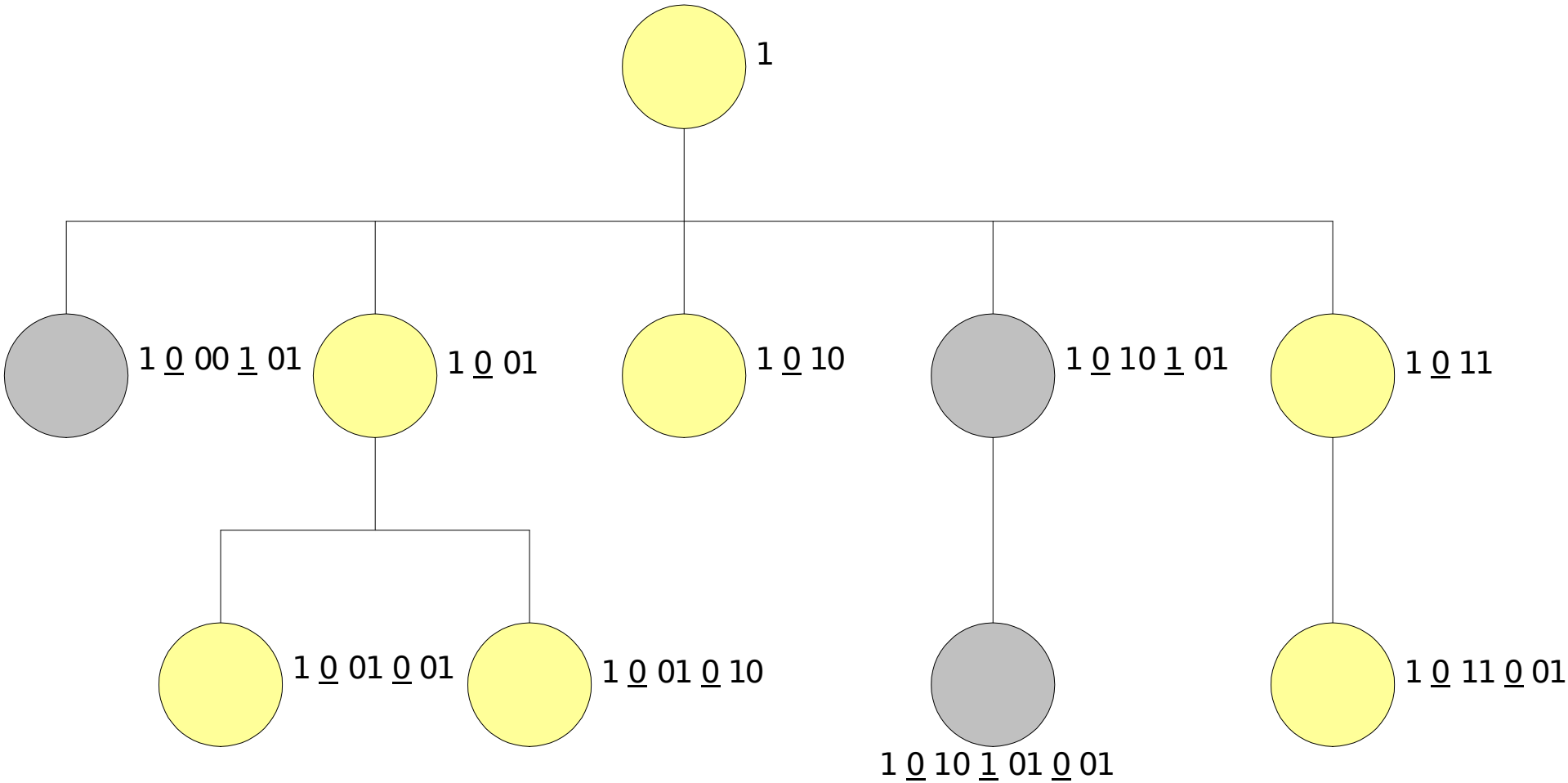
Vorteile von DLN

besteht aus Nummer des Vaterelementes als Prefix und einem Levelwert in Reihenfolge der Nachbarknoten (Dewey Order)

- Dank Preorder-Nummerierung bleibt Dokumentenordnung durch Ordnen nach DLN-ID gewahrt
- Subvalues erlauben das beliebige Einfügen von weiteren Knoten in bereits gespeicherte XML-Daten

- binäre Darstellung für schnelle Vergleiche
- Auffüllen der DLN-ID nach rechts mit Nullen
- bei Bedarf Aufspaltung der DLN-ID auf mehrere Integers
- Speicherung als große Integertypen (BIGINT bzw INT8)

DLN-Beispiel



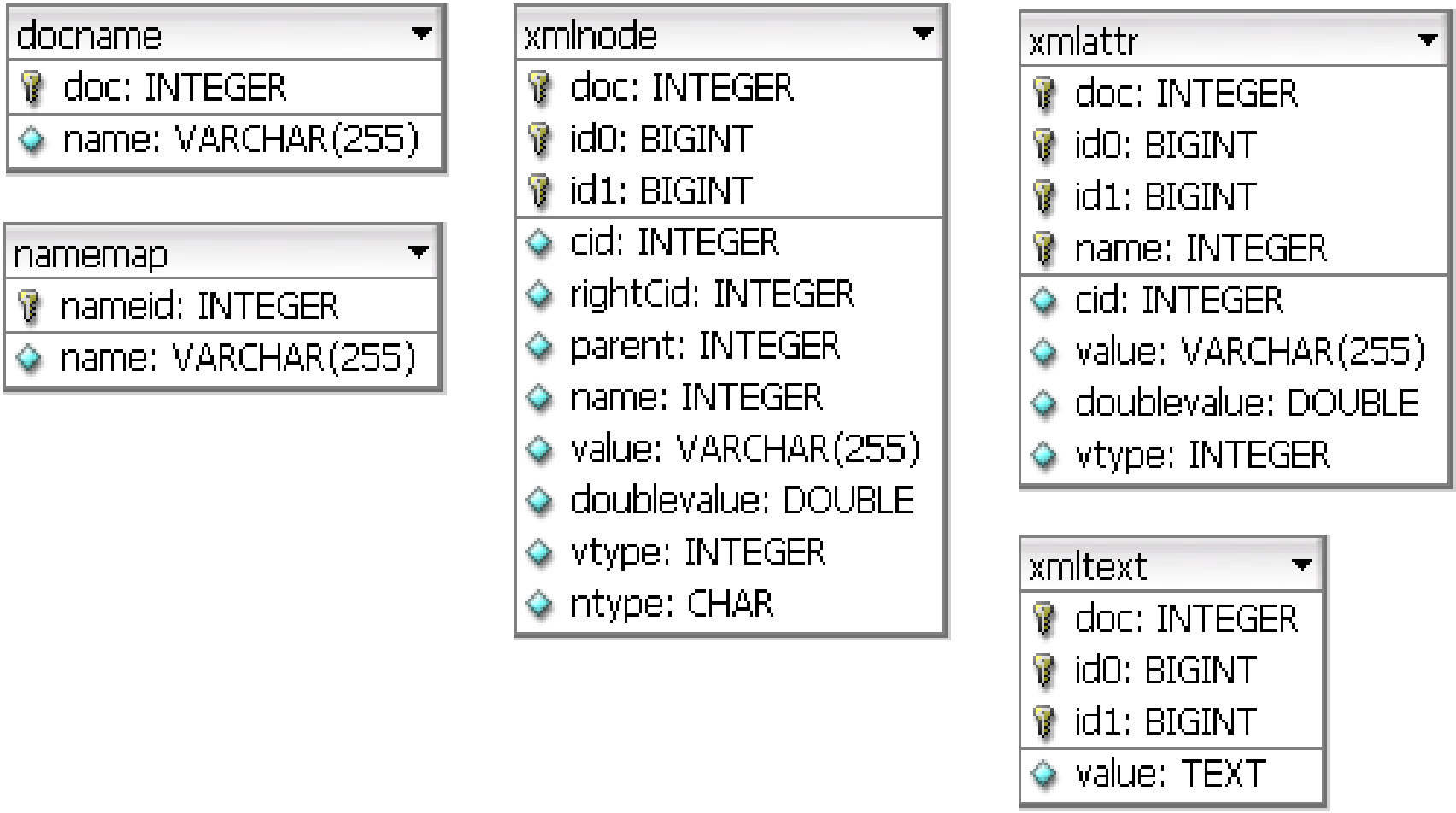
Das Relationenschema

Tabellen

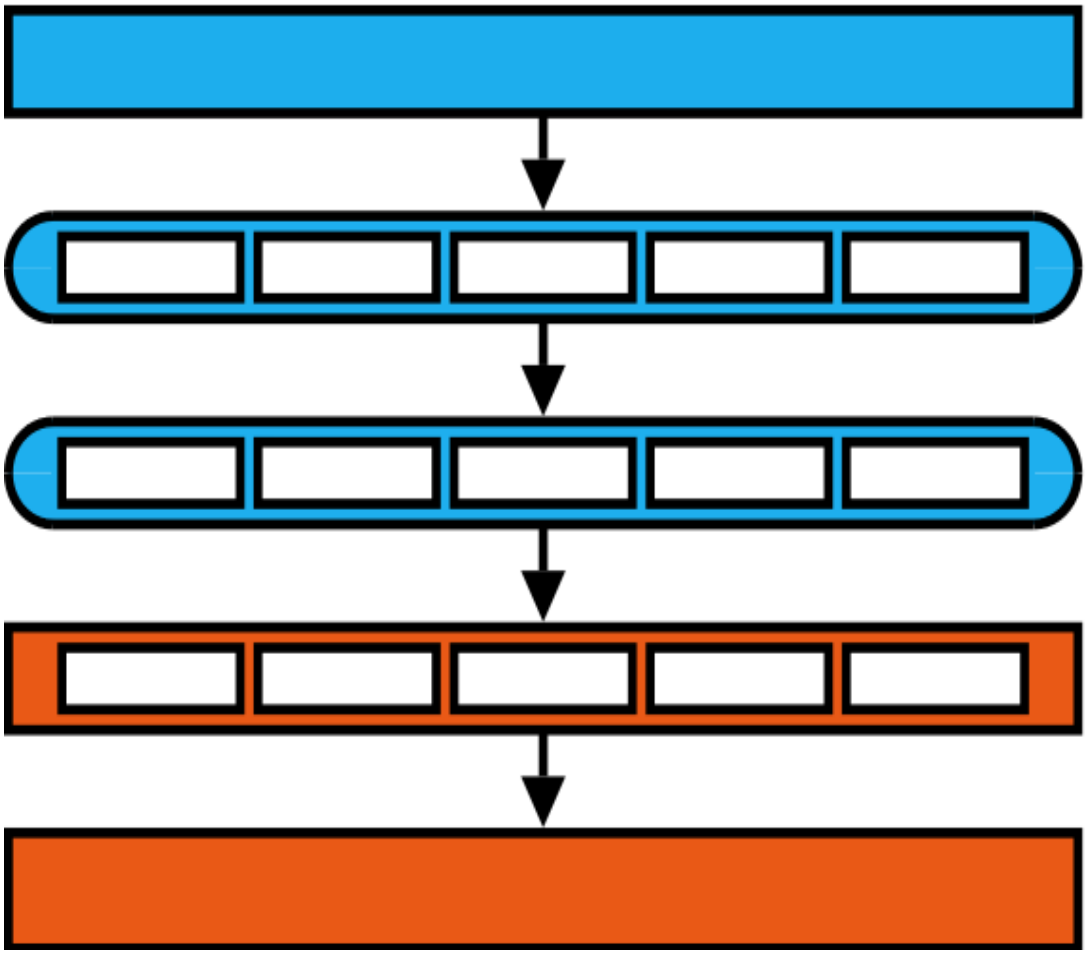
- zentrale Relation für fast alle XML-Knotentypen
- Attribute werden in einer eigenen Relation gespeichert
- Textknoten werden aufgrund unterschiedlicher Längen des Inhalts zusätzlich in einer separaten Tabelle abgelegt

- weitere Tabellen für das Ablegen der Knotennamen und die Speicherung eindeutiger Bezeichner für Dokumente

Das Relationenschema



Umsetzung



XPath-Ausdruck

XPath-Darstellung mit einzelnen Schritten

optimierte XPath-Darstellung mit einzelnen Schritten

abstrakte SQL-Darstellung mit einzelnen Schritten

SQL-Ausdruck

Abstrakte SQL-Darstellung

- unabhängig von dem genauen Datenbank-Schema
- agiert auf abstrakter Tabelle, die alle XML-Knoten enthält
- gemeinsame Basis: Preorder-Nummerierung der Knoten

- XPath-Schritte werden einzeln umgesetzt
- schrittweises Umsetzen der Schritt-Bestandteile Achsangabe, Knotentest und Prädikat
- nummerierungsabhängige Bestandteile wie die Umsetzung der XPath-Achsen werden durchgereicht
- Prädikate werden meist über EXISTS-Subqueries realisiert
- Ersetzen von Variablen für optimierte Teilpfade in Views

Finaler SQL-Ausdruck

- Umsetzung der Nummerierungs- und Schema-abhängigen XPath-Bestandteile in SQL
- Anpassung der datenbankabhängigen Teile der Anfragen (z.B. LIMIT, UDFs)

- Ausgabe eines Strings, der als Anfrage an die Datenbank gereicht werden kann

Umsetzung der XPath-Achsen I

`child, parent, attribute` nutzen das relationale Schema

`descendant`:

- bestimmt alle Nachfahren eines Kontextknotens
- in Dokumentenordnung: alle Knoten zwischen dem Kontextknoten und dessen rechten Nachbar
- Bestimmung des rechten Nachbars nicht trivial
- mit Hilfe von DLN ist eine einfache Alternative möglich: Bestimmung des maximalen Kindes des Kontextknoten in einer UDF

Umsetzung der XPath-Achsen II

`ancestor`:

- bestimmt alle Vorgänger eines Kontextknotens
- sind mit DLN in der ID des Kontextknotens kodiert
- levelweises Parsen der DLN-ID in einer UDF

`preceding-sibling, following sibling`:

- alle Knoten mit einer kleineren (größeren) DLN-ID und dem selben Vaterknoten

Umsetzung der XPath-Achsen III

`preceding`:

- bestimmt alle Knoten mit einer kleineren ID als die des Kontextknotens, bis auf dessen direkte Vorgänger
- Vorgänger werden mit der `getAncestors`-Funktion der UDF bestimmt

`following`:

- alle Knoten mit einer größeren DLN-ID als alle Nachfolger des Kontextknotens
- alle Knoten mit einer größeren DLN-ID als das maximale Kind des Kontextknotens (per UDF bestimmt)

Beispiel Umsetzung XPath – SQL I

```
... /child::kindname[child::*/@id = 72]/ ...
```

```
SELECT
FROM xmlnode k, xmlnode x1
WHERE k.cid = x1.parent AND x1.name = '13' AND EXISTS
(
  SELECT x2.cid
  FROM xmlnode x2, xmlattr a1
  WHERE x1.cid = x2.parent AND x2.cid = a1.cid AND
  a1.doublevalue = '72'
)
```

Beispiel Umsetzung XPath – SQL II

```
... /descendant::*/ ...
```

```
SELECT
FROM xmlnode k, xmlnode x1
WHERE k.doc = x1.doc AND (
( x1.id0 > k.id0 AND x1.id0 < xrdb_getMaxChild0(k.id0,
  k.id1) ) OR ( x1.id0 = k.id0 AND x1.id1 > k.id1 AND
  x1.id1 < xrdb_getMaxChild1(k.id0, k.id1) ))
```

Beispiel Umsetzung XPath – SQL III

```
... /child::kindname[position() = 2]/ ...
```

```
SELECT
FROM xmlnode k, xmlnode x2
WHERE x2.cid = (
  SELECT x1.cid
  FROM xmlnode x1
  WHERE k.cid = x1.parent AND x1.name = '13'
  ORDER BY x1.id0, x1.id1
  LIMIT 1 OFFSET 1
)
```

Probleme

- Datenbank-spezifisch:
- Query Planner bei PostgreSQL
- Erweiterungsmöglichkeiten bei MySQL

- Darstellung von Wurzel- und Dokumentenknoten in XPath
- Verarbeitung von rückwärts gerichteten Achsen

PostgreSQL

- entwickelt einen katastrophalen Queryplan bei der Benutzung der UDF `getAncestors(id0, id1)`
- PostgreSQL wertet Subqueries nicht vor der umgebenden Query aus und prüft deshalb sequenziell die gesamte Tabelle
- vollständiges Umschreiben der gesamten Anfrage notwendig

MySQL

- Erweiterbarkeit sehr eingeschränkt
- UDFs können nur einen Datentyp aus STRING | INTEGER | REAL zurückgeben
- UDFs können keine Mengen zurückgeben
- Problem bei `getAncestors(id0, id1)`
- Testen der gesamten XMLNODE-Tabelle notwendig

Dokumentenknoten

- ersetzt in XQuery und XPath 2.0 den Wurzelknoten
- DLN bildet diesen Knoten nicht explizit nach
- kein Problem bei Anfragen, die am Wurzelknoten beginnen
- aber bei aufwärts gerichteten Achsen kann der Dokumentenknoten in der Sequenz enthalten sein

- Aufnahme des Knotens in DLN hätte den Nachteil, dass jede Kontextmenge zusätzlich danach geprüft werden müsste
- Komplexität der Anfragen würde steigen

Optimierungen

- Einsparung von nutzlosen Tabellenjoins und Bedingungen

- Vorhalten der Namenstabellen bei der Konvertierung
- Nutzung der `doublevalue`-Spalte
- bei `/self` Beibehaltung der Kontexttabelle
- Attributnamen sind eindeutig – benötigen keine Subquery
- nur Elemente haben Namen

Erste Benchmarks I

Q1: `//closed_auction[./price > 50]/seller`

Q2: `//profile[@income > 90000]`

Q3: `//item[contains(shipping, 'internationally')]`

Q4: `//address[city = 'Panama']/zipcode`

Query	PSQL	PSQL mit FK	MySQL	MySQL mit FK
1	3203	3218	3922	3601
2	5234	5110	2173	2300
3	23125	23328	16405	16141
4	4422	4454	6386	6404

Zeit für die Beantwortung der Queries in ms

Erste Benchmarks II

Q5: `//people//name[contains(.,'Lindenberg')]`

Q6: `//people/person/name[contains(.,'Lindenberg')]`

Q7: `//open_auctions/open_auction[25]/seller/@person`

Query	PSQL mit FK	MySQL mit FK
5	29937	22338
6	6593	6639
7	8797	5662

Zeit für die Beantwortung der Queries in ms

Ausblick

- Datenbankspezifische Optimierungen
- Einbinden von Views für oft verwendete Schrittfolgen
- Implementierung von Mengenoperationen (`some satisfies`)
- vollständige Integration in XMLRDB