Effiziente XPath-Ausführung

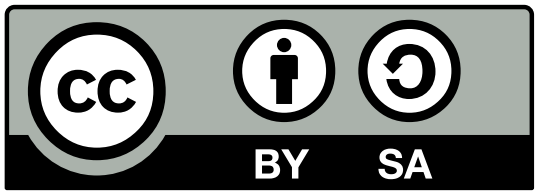
Volker Grabsch Christine Janischek

10. Dezember 2007

Allgemeines

- veröffentlicht unter
  `http://www.profv.de/uni/`

- lizensiert unter



Creative Commons BY-SA 3.0

Quelle

Dieser Vortrag basiert auf dem Paper

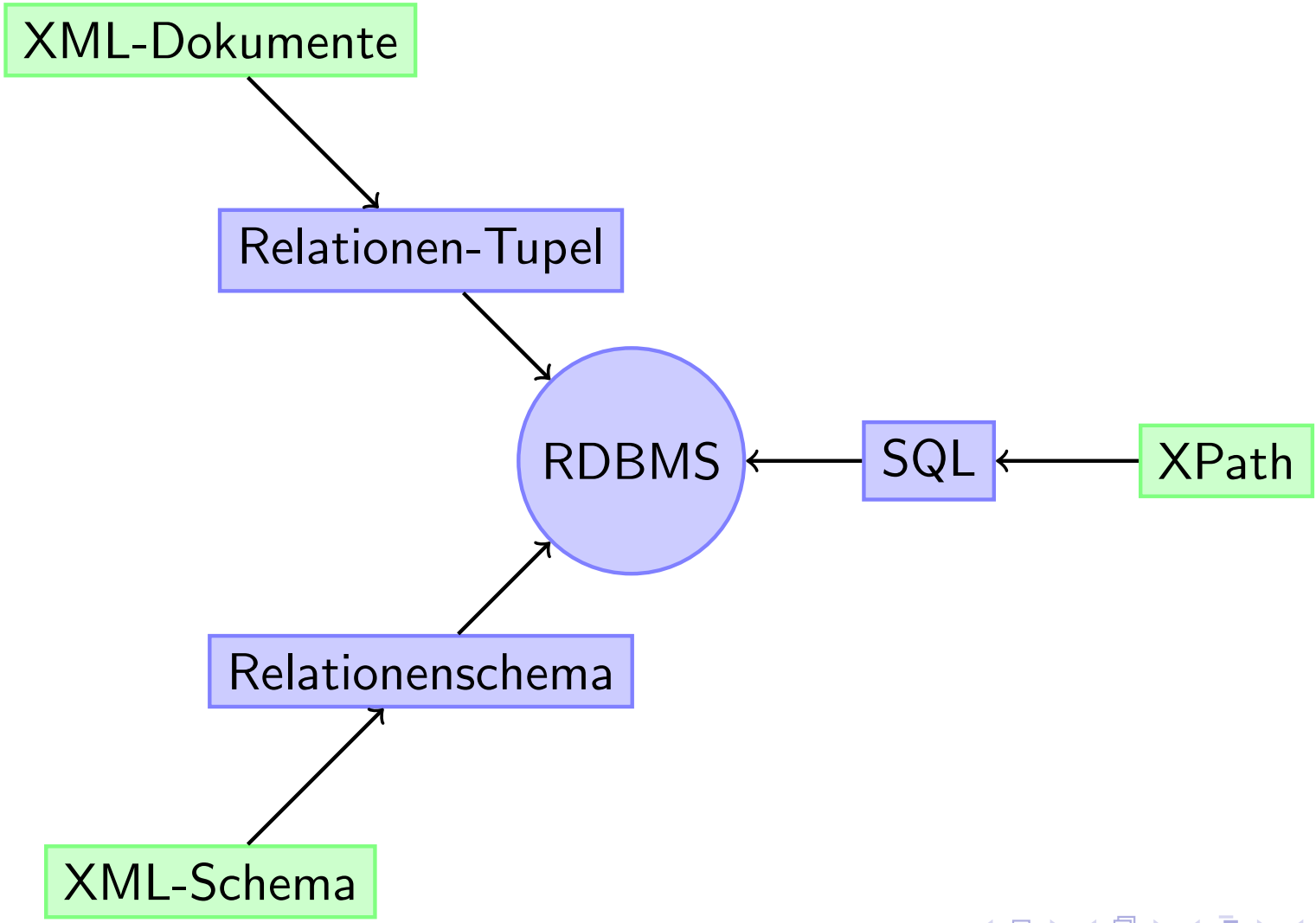
> *Improving the Efficiency of XPath Execution on Relational Systems*

von Haris Georgiadis und Vasilis Vassalos.

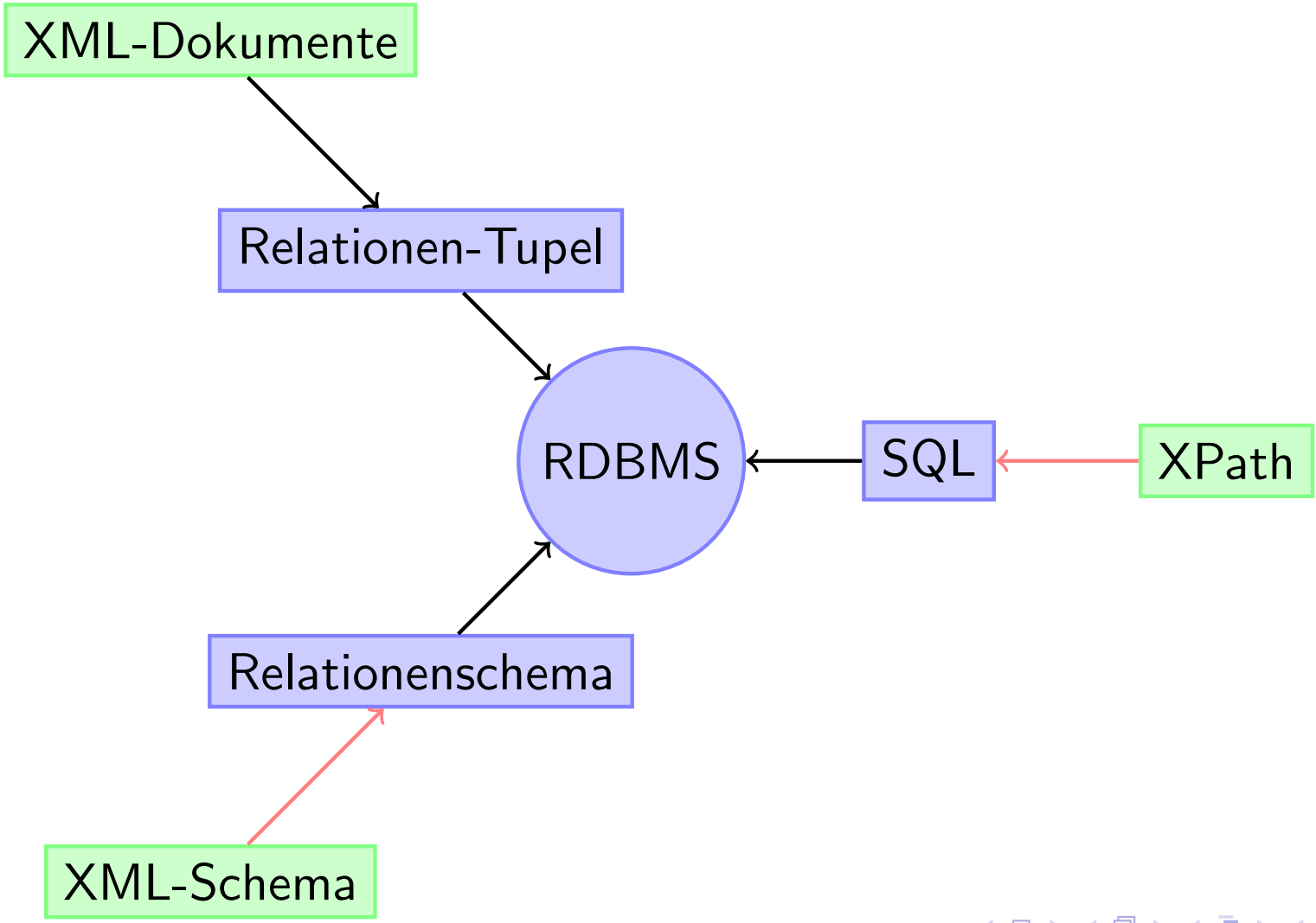
Übersicht

Übersicht

Einleitung


XML-Dokumente
Relationen-Tupel
RDBMS
SQL
XPath
Relationenschema
XML-Schema

Einleitung


XML-Dokumente
Relationen-Tupel
RDBMS
SQL
XPath
Relationenschema
XML-Schema

Grundidee

- Ziel
  - schnelle XPath-Anfragen
  - einfacherer SQL-Code
  - weniger Joins
- Strategie
  - speichere mehr Informationen über die Hierarchie
  - 1 Join = mehrere XPath-Schritte

Grundidee

- Ziel
  - schnelle XPath-Anfragen
  - einfacherer SQL-Code
  - weniger Joins
- Strategie
  - speichere mehr Informationen über die Hierarchie
  - 1 Join = mehrere XPath-Schritte

Übersicht

Übersicht

Relationenschema

- je „Complex Type“ eine Relation
- Attribute:
  - ID
  - Eltern-IDs pro Eltern-Typ
  - Dewey-Position
  - Pfad
  - „Simple Type“-Elemente, Attribute, Text

Relationenschema

- je „Complex Type“ eine Relation
- Attribute:
  - ID
  - Eltern-IDs pro Eltern-Typ
  - Dewey-Position
  - Pfad
  - „Simple Type“-Elemente, Attribute, Text

Relationenschema

- je „Complex Type“ eine Relation
- Attribute:
  - ID
  - Eltern-IDs pro Eltern-Typ
  - Dewey-Position
  - Pfad
  - „Simple Type“-Elemente, Attribute, Text

Relationenschema

- je „Complex Type“ eine Relation
- Attribute:
  - ID
  - Eltern-IDs pro Eltern-Typ
  - Dewey-Position
  - Pfad
  - „Simple Type“-Elemente, Attribute, Text

Relationenschema

- je „Complex Type“ eine Relation
- Attribute:
  - ID
  - Eltern-IDs pro Eltern-Typ
  - Dewey-Position
  - Pfad
  - „Simple Type“-Elemente, Attribute, Text

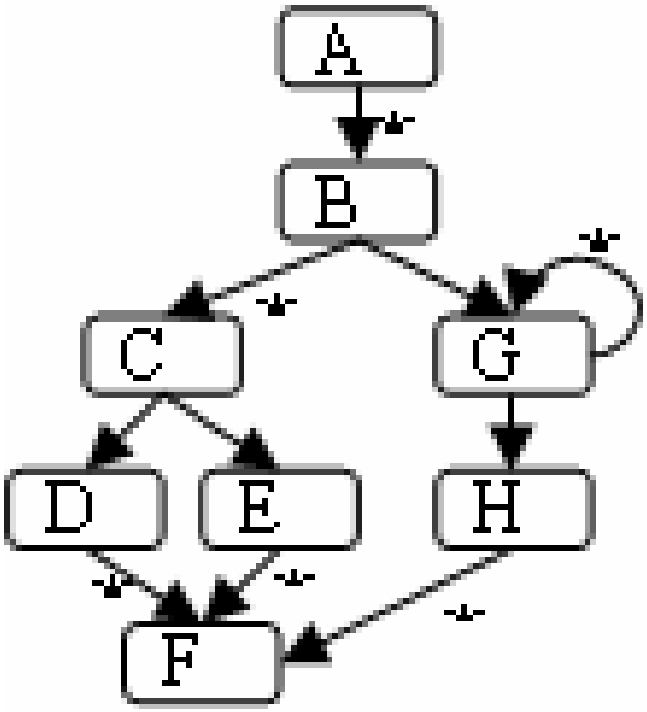
Relationenschema

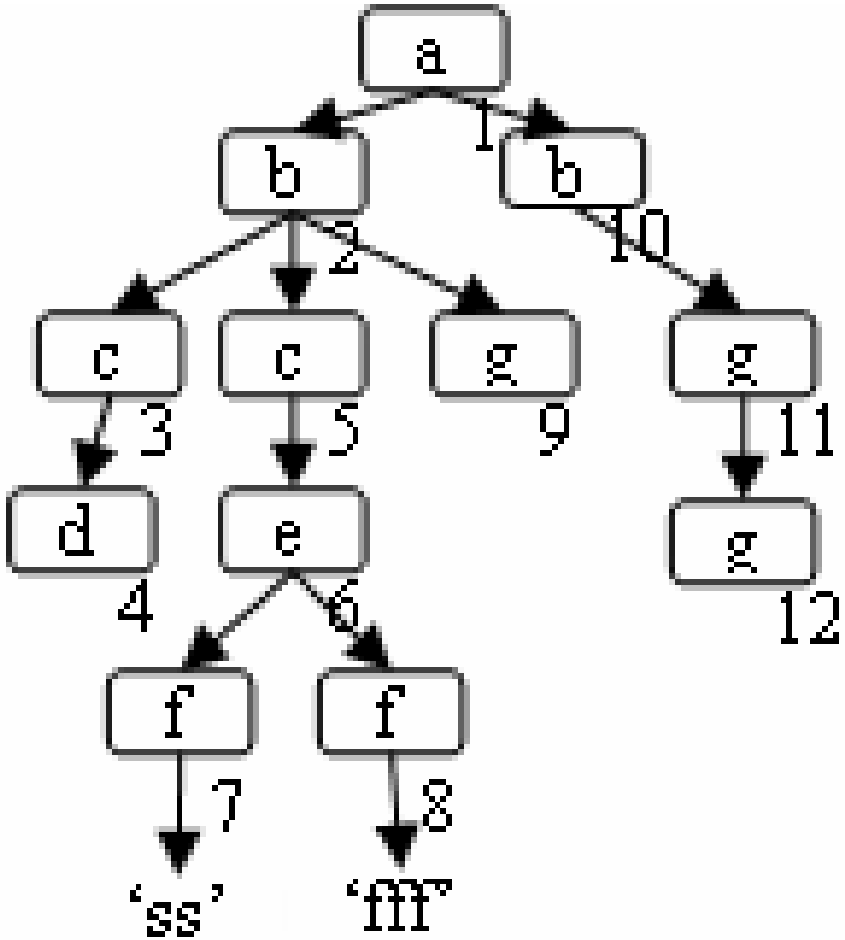
- je „Complex Type“ eine Relation
- Attribute:
  - ID
  - Eltern-IDs pro Eltern-Typ
  - Dewey-Position
  - Pfad
  - „Simple Type“-Elemente, Attribute, Text

Übersicht

Beispiel: XML-Schema und XML-Dokument

(entnommen aus dem Paper)

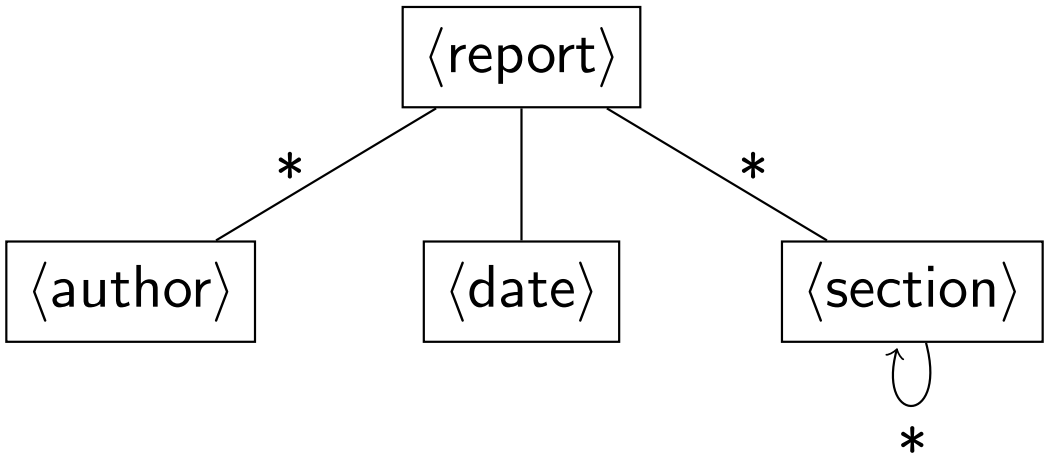




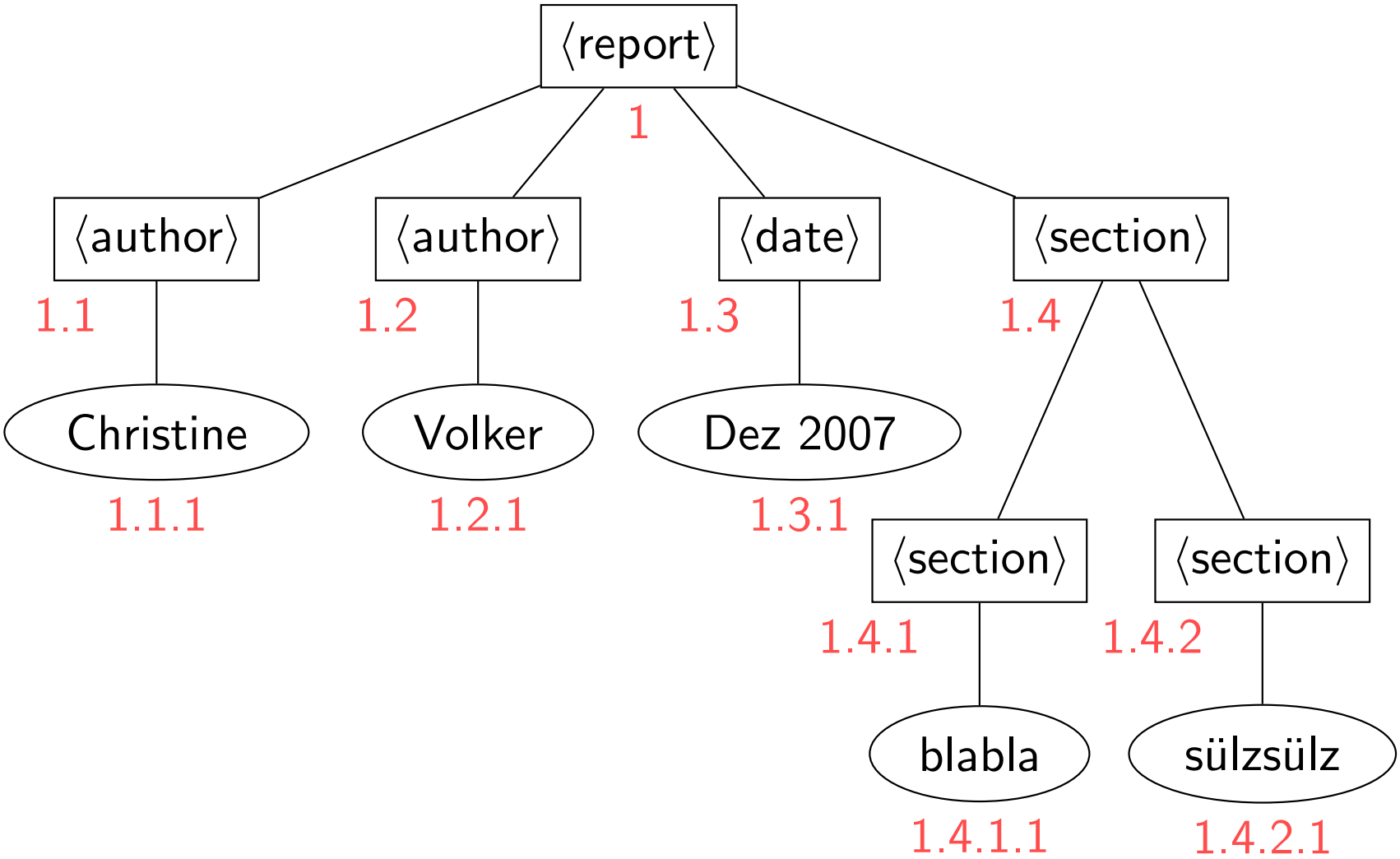
Beispiel: XML-Schema und XML-Dokument

Nur ein Scherz!

Beispiel: XML-Schema



Beispiel: XML-Dokument (mit Dewey-Positionen)



Beispiel: Relation ⟨report⟩

ID	Dewey-Pos	Pfad	date
1	1	/report	Dez 2007

Beispiel: Relation ⟨author⟩

ID	⟨report⟩	Dewey-Pos	Pfad	text
1	1	1.1	/report/author	Christine
2	1	1.2	/report/author	Volker

Beispiel: Relation ⟨section⟩

ID	⟨report⟩	⟨section⟩	D-Pos	Pfad	text
1	1		1.4	/report/section	
2		1	1.4.1	/report/section/section	blabla
3		1	1.4.2	/report/section/section	sülzsülz

Übersicht

Reservierte Dewey-Positionen

- höchster Index (hier: 9) darf nicht verwendet werden
  - 1.4.9 ist nicht erlaubt
  - maximal 8 (statt 9) direkte Kindknoten
  - $x < 1.5 \Leftrightarrow x < 1.4.9$

Reservierte Dewey-Positionen

- höchster Index (hier: 9) darf nicht verwendet werden
  - `1.4.9` ist nicht erlaubt
  - maximal 8 (statt 9) direkte Kindknoten
  - $x < 1.5 \Leftrightarrow x < 1.4.9$

Reservierte Dewey-Positionen

- höchster Index (hier: 9) darf nicht verwendet werden
  - `1.4.9` ist nicht erlaubt
  - maximal 8 (statt 9) direkte Kindknoten
  - $x < 1.5 \Leftrightarrow x < 1.4.9$

Reservierte Dewey-Positionen

- höchster Index (hier: 9) darf nicht verwendet werden
  - `1.4.9` ist nicht erlaubt
  - maximal 8 (statt 9) direkte Kindknoten
  - $x < 1.5 \Leftrightarrow x < 1.4.9$

Übersicht

Übersicht

Einfache Pfadfragmente (PPFs)

Welche Teile eines XPath können wir mit einem Schlag abarbeiten?

- einfache Abwärts-Pfade
- einfache Aufwärts-Pfade
- Einzelschritte entlang der Seitenachsen
- ... jeweils mit abschließendem Prädikat
  - kann komplexe XPath-Ausdrücke enthalten (→ Rekursion!)

```
/report/date/..//section[.='blabla']/following::*
```

Einfache Pfadfragmente (PPFs)

Welche Teile eines XPath können wir mit einem Schlag abarbeiten?

- einfache Abwärts-Pfade
- einfache Aufwärts-Pfade
- Einzelschritte entlang der Seitenachsen
- ... jeweils mit abschließendem Prädikat
  - kann komplexe XPath-Ausdrücke enthalten (→ Rekursion!)

```
/report/date/..//section[.='blabla']/following::*
```

Einfache Pfadfragmente (PPFs)

Welche Teile eines XPath können wir mit einem Schlag abarbeiten?

- einfache Abwärts-Pfade
- einfache Aufwärts-Pfade
- Einzelschritte entlang der Seitenachsen
- ... jeweils mit abschließendem Prädikat
  - kann komplexe XPath-Ausdrücke enthalten (→ Rekursion!)

```
/report/date/..//section[.='blabla']/following::*
```

Einfache Pfadfragmente (PPFs)

Welche Teile eines XPath können wir mit einem Schlag abarbeiten?

- einfache Abwärts-Pfade
- einfache Aufwärts-Pfade
- Einzelschritte entlang der Seitenachsen
- ... jeweils mit abschließendem Prädikat
  - kann komplexe XPath-Ausdrücke enthalten (→ Rekursion!)

```
/report/date/..//section[.='blabla']/following::*
```

Einfache Pfadfragmente (PPFs)

Welche Teile eines XPath können wir mit einem Schlag abarbeiten?

- einfache Abwärts-Pfade
- einfache Aufwärts-Pfade
- Einzelschritte entlang der Seitenachsen
- ... jeweils mit abschließendem Prädikat
  - kann komplexe XPath-Ausdrücke enthalten (→ Rekursion!)

```
/report/date/..//section[.='blabla']/following::*
```

Einfache Pfadfragmente (PPFs)

Welche Teile eines XPath können wir mit einem Schlag abarbeiten?

- einfache Abwärts-Pfade
- einfache Aufwärts-Pfade
- Einzelschritte entlang der Seitenachsen
- ... jeweils mit abschließendem Prädikat
  - kann komplexe XPath-Ausdrücke enthalten (→ Rekursion!)

```
/report/date/..//section[.='blabla']/following::*
```

Einfacher Abwärts-Pfad

```
/*/section//section
```

→

```
WHERE REGEX('^/[^/]+/section/(.+/)?section$',
Rel.Pfad)
```

Einfacher Abwärts-Pfad

```
/*/section//section
```

→

```
WHERE REGEX('^/[^/]+/section/(.+/)?section$',
Rel.Pfad)
```

Einfacher Abwärts-Pfad

```
/*/section//section
```

→

```
WHERE REGEX('^/[^/]+/section/(.+/)?section$',
Rel.Pfad)
```

Einfacher Abwärts-Pfad

```
/*/section//section
```

→

```
WHERE REGEX('^/[^/]+/section/(.+/)?section$',
Rel.Pfad)
```

Einfacher Abwärts-Pfad

```
/*/section//section
```

→

```
WHERE REGEX('^/[^/]+/section/(.+/)?section$',
Rel.Pfad)
```

Einfacher Abwärts-Pfad

```
/*/section//section
```

→

```
WHERE REGEX('^/[^/]+/section/(.+/)?section$',
Rel.Pfad)
```

Einfacher Abwärts-Pfad

```
/*/section//section
```

→

```
WHERE REGEX('^/[^/]+/section/(.+/)?section$',
Rel.Pfad)
```

Einfacher Abwärts-Pfad

```
/*/section//section
```

→

```
WHERE REGEX('^/[^/]+/section/(.+/)?section$',
Rel.Pfad)
```

Einfacher Aufwärts-Pfad

```
/parent::section/ancestor::*
```

→

```
WHERE REGEX('^/.+/section/[^/]+$', VorherigeRel.Pfad)
```

Alternativ:

```
WHERE REGEX('^/.+/section$', VorherigeElternRel.Pfad)
```

Einfacher Aufwärts-Pfad

```
/parent::section/ancestor::*
```

→

```
WHERE REGEX('^/.+/section/[^/]+$', VorherigeRel.Pfad)
```

Alternativ:

```
WHERE REGEX('^/.+/section$', VorherigeElternRel.Pfad)
```

Einfacher Aufwärts-Pfad

```
/parent::section/ancestor::*
```

→

```
WHERE REGEX('^/.+/section/[^/]+$', VorherigeRel.Pfad)
```

Alternativ:

```
WHERE REGEX('^/.+/section$', VorherigeElternRel.Pfad)
```

Einfacher Aufwärts-Pfad

```
/parent::section/ancestor::*
```

→

```
WHERE REGEX('^/.+/section/[^/]+$', VorherigeRel.Pfad)
```

Alternativ:

```
WHERE REGEX('^/.+/section$', VorherigeElternRel.Pfad)
```

Einzelschritte entlang der Seitenachsen

```
/preceding-sibling::author
```

→

```
WHERE REGEX('^.*/author$', Rel.Pfad)
```

Prädikate

- einfaches Prädikat:

```
//*[date='Dez 2007']
→
WHERE ... AND date='Dez 2007'
```

- Prädikat mit komplexem Teilpfad:

```
//*[section/section='blabla']
→
... AND EXISTS (SELECT ... WHERE ... AND
text='blabla')
```

Prädikate

- einfaches Prädikat:

```
//*[date='Dez 2007']
→
WHERE ... AND date='Dez 2007'
```

- Prädikat mit komplexem Teilpfad:

```
//*[section/section='blabla']
→
... AND EXISTS (SELECT ... WHERE ... AND
text='blabla')
```

Zusammenführen der PPFs mittels Dewey-Positionen

```
//A/ancestor::B
```

→

```
WHERE A.Dewey-Pos > B.Dewey-Pos
  AND A.Dewey-Pos < B.Dewey-Pos || '.9'
```

Zusammenführen von Vorwärts-Pfaden

- Join über Dewey-Positionen
- Pfad des bisherigen Elements in die Regex des nächsten Elements einbauen!

```
//A[@x=0]//*/B
→
WHERE ... AND REGEX('^' || A.Pfad || '/.+/B$',
B.Pfad)
```

Zusammenführen von Vorwärts-Pfaden

- Join über Dewey-Positionen
- Pfad des bisherigen Elements in die Regex des nächsten Elements einbauen!

```
//A[@x=0]//*/B
```

→

```
WHERE ... AND REGEX('^' || A.Pfad || '/.+/B$',
B.Pfad)
```

Zusammenführen von Vorwärts-Pfaden

- Join über Dewey-Positionen
- Pfad des bisherigen Elements in die Regex des nächsten Elements einbauen!

```
//A[@x=0]//*/B
```

→

```
WHERE ... AND REGEX('^' || A.Pfad || '/.+/B$',
B.Pfad)
```

Übersicht

Binärkodierung der Dewey-Positionen

- Dewey-Positionen binär kodieren
- Verzicht auf Trennzeichen
- Beispiel:
  - 3 Bytes je Position
  - Bereich: 000000 – `feffff`
  - Reservierter Bereich: `ff....`
  - `dewey_pos || '\xff'`

Binärkodierung der Dewey-Positionen

- Dewey-Positionen binär kodieren
- Verzicht auf Trennzeichen
- Beispiel:
  - 3 Bytes je Position
  - Bereich: 000000 – `feffff`
  - Reservierter Bereich: `ff....`
  - `dewey_pos || '\xff'`

Binärkodierung der Dewey-Positionen

- Dewey-Positionen binär kodieren
- Verzicht auf Trennzeichen
- Beispiel:
  - 3 Bytes je Position
  - Bereich: `000000` – `feffff`
  - Reservierter Bereich: `ff....`
  - `dewey_pos || '\xff'`

Binärkodierung der Dewey-Positionen

- Dewey-Positionen binär kodieren
- Verzicht auf Trennzeichen
- Beispiel:
  - 3 Bytes je Position
  - Bereich: 000000 – `feffff`
  - Reservierter Bereich: `ff....`
  - `dewey_pos || '\xff'`

Binärkodierung der Dewey-Positionen

- Dewey-Positionen binär kodieren
- Verzicht auf Trennzeichen
- Beispiel:
  - 3 Bytes je Position
  - Bereich: 000000 – `feffff`
  - Reservierter Bereich: `ff....`
  - `dewey_pos || '\xff'`

Binärkodierung der Dewey-Positionen

- Dewey-Positionen binär kodieren
- Verzicht auf Trennzeichen
- Beispiel:
  - 3 Bytes je Position
  - Bereich: `000000` – `feffff`
  - Reservierter Bereich: `ff....`
  - `dewey_pos || '\xff'`

LIKE statt Regex

- Einfache reguläre Ausdrücke mit LIKE formulieren

```
WHERE REGEX('^/report/.+/section$', Rel.Pfad)
→
Rel.Pfad LIKE '/report/%/section'
```

LIKE statt Regex

- Einfache reguläre Ausdrücke mit LIKE formulieren

```
WHERE REGEX('^/report/.+/section$', Rel.Pfad)
```

→

```
Rel.Pfad LIKE '/report/%/section'
```

Zusätzliches Attribut für Elementnamen

- Neues Attribut: Elementname

```
WHERE REGEX('^.*/section$', Rel.Pfad)
→
WHERE Rel.Elementname = 'section'
```

Zusätzliches Attribut für Elementnamen

- Neues Attribut: Elementname

```
WHERE REGEX('^.*/section$', Rel.Pfad)
```

→

```
WHERE Rel.Elementname = 'section'
```

Pfad-ID

- Pfade wiederholen sich oft
- Neue Relation: (Pfad-ID, Pfad)

Pfad-ID

- Pfade wiederholen sich oft
- Neue Relation: (Pfad-ID, Pfad)

Unnötige Pfadfilterungen

- Zu jeder Relation („Complex Type“):
  - Welche Pfade sind laut Schema möglich?
- Zu jeder Regex:
  - Erlaubt die Regex all diese Pfade?
  - Erlaubt die Regex keinen der Pfade?
- Falls ja:
  - Regex-Vergleich weglassen
  - Join mit Pfad-ID-Relation weglassen

Unnötige Pfadfilterungen

- Zu jeder Relation („Complex Type“):
  - Welche Pfade sind laut Schema möglich?
- Zu jeder Regex:
  - Erlaubt die Regex all diese Pfade?
  - Erlaubt die Regex keinen der Pfade?
- Falls ja:
  - Regex-Vergleich weglassen
  - Join mit Pfad-ID-Relation weglassen

Unnötige Pfadfilterungen

- Zu jeder Relation („Complex Type“):
  - Welche Pfade sind laut Schema möglich?
- Zu jeder Regex:
  - Erlaubt die Regex all diese Pfade?
  - Erlaubt die Regex keinen der Pfade?
- Falls ja:
  - Regex-Vergleich weglassen
  - Join mit Pfad-ID-Relation weglassen

Unnötige Pfadfilterungen

- Zu jeder Relation („Complex Type“):
  - Welche Pfade sind laut Schema möglich?
- Zu jeder Regex:
  - Erlaubt die Regex all diese Pfade?
  - Erlaubt die Regex keinen der Pfade?
- Falls ja:
  - Regex-Vergleich weglassen
  - Join mit Pfad-ID-Relation weglassen

Unnötige Pfadfilterungen

- Zu jeder Relation („Complex Type“):
  - Welche Pfade sind laut Schema möglich?
- Zu jeder Regex:
  - Erlaubt die Regex all diese Pfade?
  - Erlaubt die Regex keinen der Pfade?
- Falls ja:
  - Regex-Vergleich weglassen
  - Join mit Pfad-ID-Relation weglassen

Übersicht

Kritik

- Mehrere Rückwärtsschritte dürfen nicht einfach zusammengefasst werden!

```
//F/parent::B/parent::*/ancestor::B
⟶
WHERE REGEX('^.*/B/.+/B/F$', F.Pfad)
```

Kritik

- Mehrere Rückwärtsschritte dürfen nicht einfach zusammengefasst werden!

```
//F/parent::B/parent::*/ancestor::B
```

→

```
WHERE REGEX('^.*/B/.+/B/F$', F.Pfad)
```