# KeyX: ein selektiver schlüsselorientierter Index für das Index Selection Problem in XDBMS

Beda Christoph Hammerschmidt

Institut für Informationssysteme
Universität zu Lübeck
Ratzeburger Allee 160, Gebäude 64, D-23538 Lübeck
email: bchammer@ifis.uni-luebeck.de

**Zusammenfassung**

In relationalen Datenbank-Management-Systemem (RDBMS) werden Indizes verwendet, um spezifische und häufig wiederkehrende Anfragen zu beschleunigen. Die Auswahl von passenden Indizes ist ein wichtiger Prozess beim Anlegen und Optimieren der Datenbank, der meist von einem Administrator oder einem Index-Auswahl-Tool durchgeführt wird, welches eine Menge von passenden Indizes vorschlägt. Für die Indizierung von XML-Datenbanken/Dokumenten existieren zur Zeit noch keine Standardverfahren sondern verschiedene Ansätze in der wissenschaftlichen Literatur, die oft rein Pfad-basiert sind und keine oder wenig Auswahl bezüglich der indizierten Elemente zulassen. Ansätze, die das gesamte Dokument indizieren - sogenannte Structural Summaries- haben zwangsläufig einen hohen Speicherplatzbedarf und garantieren keine Leistungssteigerung, wenn häufig Änderungen am XML-Dokument vorgenommen werden, da Änderungen der Datenbank immer auch an der Indexstruktur vorgenommen werden müssen.
In dieser Arbeit wird das Konzept von spezifischen Indizes auf native XML-Datenbank-Management-Systeme (XDBMS) übertragen. Es wird eine Implementierung präsentiert, die frühere Anfragen nutzt, um die Datenbank zu optimieren, indem sie automatisch passende Indizes erstellt. Mit KeyX stellen wir einen Indizierungsansatz vor, der XML-Element- und Attributwerte mit spezifischem Pfad als Schlüssel interpretiert und die dazugehörenden oder benachbarten Knoten im Original-Dokument als Rückgabewert referenziert. Da sich Schlüssel und Wert unterscheiden können, entfällt der Aufwand, der für navigierende Pfadverfolgung zwischen beiden benötigt wird.
Wir transferieren das in der relationalen Welt wohlbekannte Index Selection Problem (ISP) auf XDBMS. Das ISP wird verwendet, um eine Menge von Indizes zu bestimmen, für die die Ausführungszeit eines Workloads von Datenbankoperationen minimal ist. Da der Workload periodisch analysiert wird und durch das ISP günstige Indizes automatisch angelegt und ggf. aufgelöst werden, garantiert die KeyX-Implementierung eine hohe Leistung über die gesamte Lebenszeit der Datenbank. Experimentell bestimmte Ergebnisse der auf einem nativen XDBMS basierenden prototypischen Implementierung zeigen, dass unser Ansatz die Ausführungszeit von Anfragen signifikant beschleunigt.

## 1 Motivation

Die Extensible-Markup-Language (XML) ist das Standardformat für den Datenaustausch zwischen Anwendungen im Internet. Der verstärkte Einsatz von XML in Web-Applikationen und E-Commerce erfordert die Anbindung von XML-Technologien an Datenbank-Management-Systeme, da letztere einen schnellen, robusten und anwendungsunabhängigen Zugriff auf Daten ermöglichen. In den letzten Jahren wurden viele (objekt-)relationale Datenbank-Management-Systeme um XML-Funktionalitäten erweitert [8, 12, 11]. Im Gegensatz zu diesen sogenannten *XML enabled database systems* wurde das Paradigma der *nativen* XML-Datenbank (XDBMS)[6, 13] eingeführt. XDBMS speichern XML-Daten persistent in einer nativen Baumstruktur und vermeiden so teure Transformationen in Tabellen. Die bekanntesten XML-Anfragesprachen wie XQuery [17] und XPath [16] nutzen Pfadausdrücke, um innerhalb der XML-Daten zu navigieren. Aus diesem Grund ist die effiziente Ausführung von Pfadausdrücken eine wichtige

Aufgabe von XDBMS.
In RDBMS werden Indizes zur Beschleunigung von häufig wiederkehrenden Anfragen genutzt. Ein Index ist hierbei eine Datenstruktur, die spezifische Anfragen optimal unterstützt und Zugriffe auf die physikalische Datenbankschicht reduziert. Ein unvermeidbarer Seiteneffekt von Indizes ist der Speicherverbrauch und die notwendige Aktualisierung bei Änderungen der Originaldaten. Aus diesem Grund wird nicht jede Spalte einer relationalen Datenbank indiziert.
In RDBMS sind Indizes gut verstanden und werden tagtäglich eingesetzt. Für XDBMS und semistrukturierte Daten sind Indizes ein aktives Forschungsgebiet der letzten Jahre. Hierbei existieren verschiedene Ansätze, die z.T. nur navigierende Anfragen ohne Wertvergleiche unterstützen (z.B. Lore Dataguides[10], APEX[3]) oder ausschliesslich Wertanfragen ohne Pfadberücksichtigung unterstützen (z.B: Lore Value Index [10]). Kombinierte Ansätze (z.B. Index Fabrics[5]) unterstützen beide Anfragearten. Ansätze wie der Dataguide oder Index Fabric Raw Paths sind sogenannte *Structural-Summaries*, die alle Knoten der XML-Daten in einer zweiten Datenstruktur ablegen. Solche Indizes sind nicht selektiv, benötigen viel Speicher und verbessern die Leistung eines Datenbanksystems nicht, wenn häufig Änderungen vorgenommen werden, da jedesmal die Indexstruktur aktualisiert werden muss.

## 2 Der schlüsselorientierte XML-Index KeyX

Im Folgenden führen wir den KeyX-Indizierungsansatz ein, der sowohl Wertanfragen, rein navigierende Anfragen und Kombinationen spezifisch unterstützt. Mit spezifisch meinen wir, dass ein Index nur definierte Teile der XML-Daten und nicht das gesamte Dokument wie ein Structural-Summary indiziert. Abbildung 1 zeigt ein XML-Beispieldokument; eine semistrukturierte Repräsentation von Publikationen, bestehend aus Büchern und Artikeln. Die linke Seite zeigt die textuelle Darstellung, die rechte Seite den dazugehörenden *Document Object Model*(DOM)-Baum.

```
<publications>
  <book title="XPath Essentials" isbn="0471205486">
    <author>Andrew Watt</author>
  </book>
  <book title="Data on the Web" isbn="155860622X">
    <author>Serge Abiteboul</author>
    <author>Peter Bunemann</author>
    <author>Dan Suciu</author>
  </book>
  <article title="Containment  for XPath" year="2002">
    <author>Gerome Miklau</author>
    <author>Dan Suciu</author>
  </article>
</publications>
```

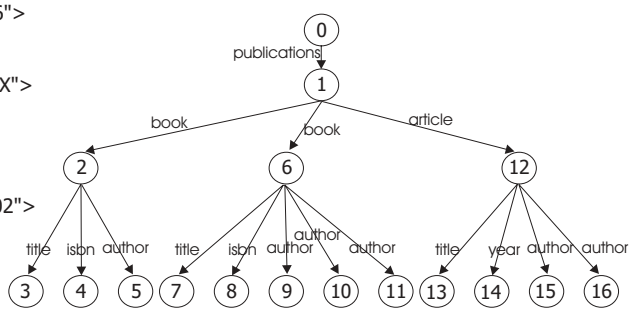


Abbildung 1: XML-Beispieldaten

Für dieses Dokument können zum Beispiel die folgenden Pfadausdrücke von Interesse sein. Aus Gründen der Lesbarkeit wird hier die Kurzform von XPath verwendet.

- $p_1 = /publications/article[year > X]$
  Zurückgegeben werden alle Publication-Knoten, die nach $X$ erschienen sind.
- $p_2 = /publications/book[author = "X"]$
  Pfadausdruck $p_2$ selektiert alle Book-Knoten eines Autors $X$.
- $p_3 = /publications/article[author = "X"\ and\ year > Y]$
  Pfadausdruck $p_3$ ist ein Beispiel für eine mehrwertige Anfrage.
- $p_4 = /publications/article$
  Pfadaudruck $p_4$ selektiert alle Artikel ohne weiteres Prädikat. Dies ist eine rein navigierende Anfrage.

Analog zu relationalen Datenbanken arbeitet unser Index mit Schlüsseln. Die Schlüssel sind die Werte von XML-Elementen oder Attributen, die über einen spezifischen Pfadausruck erreichbar sind. Für die Anfrage $p_1$ sind dies die Werte aller *year*-Elemente. Die Schlüssel werden aus den XML-Daten

extrahiert, indem der Schlüsselpfad einer Anfrage ausgewertet und in einer Baumstruktur abgelegt wird, um logarithmische Komplexität beim Suchen eines Schlüssels zu garantieren. Jeder Schlüssel verweist auf einen oder mehrere Knoten (Element oder Attribut) in den XML-Daten. Zum Beispiel verweist der Schlüssel *'2004'* der Indexstruktur für Anfrage $p_1$ auf alle Artikel, die $2004$ oder später geschrieben wurden. Der Schlüssel und der Wert einer Anfrage dürfen unterschiedlich sein. In Anfrage $p_2$ sind die Werte der *author*-Elemente die Schlüssel, während die *book*-Elemente den (Rückgabe)-Wert darstellen. Dieses Konzept vermeidet die teure Navigation zwischen Schlüssel und Wert in den XML-Daten bei Anfragen, die nur einmal beim Anlegen des Index und beim Einfügen neuer Knoten ausgeführt werden muß.
Die für Anfrage $i_{p_2}$ spezifische Indexstruktur $i_{p_2}$ ist in Abbildung 2 dargestellt. Da dieser Index ausschliesslich Buchautoren indiziert, verweist der Schlüssel *'Dan Suciu'* nur auf sein Buch und nicht auf seinen Artikel.

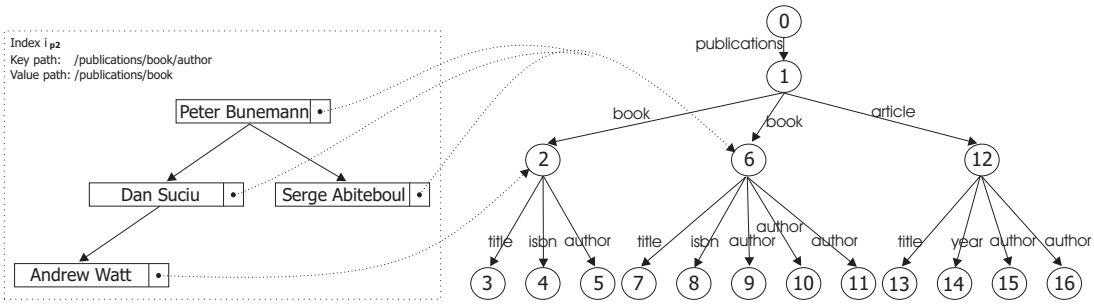


Abbildung 2: Index für Anfrage $i_{p_2}$

Wenn eine Anfrage mehr als einen Schlüssel besitzt (z.B. $p_3$), muss ein mehrwertiger Index angelegt werden. Dies geschieht analog zu RDBMS durch eine geschachtelte Baumstruktur. Rein navigierende Anfragen ohne Prädikat(z.B. $p_4$) können leicht und ebenfalls spezifisch durch KeyX unterstützt werden. Der gesamte Pfadausdruck wird als Schlüssel betrachtet, in einem Suchbaum abgelegt und mit allen über diesen Pfadausdruck erreichbaren Knoten verknüpft. Hierdurch entfallen aufwendige Pfadverfolgungen in den XML-Daten.
Beim Ausführen einer Anfrage sucht der Query-Optimizer nach einem angelegten Index, der zur Anfrage passt. Dies kann durch vergleichen der Schlüssel- und Wertpfade von Anfrage und Index entschieden werden. Wenn ein geeigneter Index gefunden wurde, kann die Anfrage ohne Navigation in den XML-Daten ausgeführt werden. Ohne geeigneten Index muss die Anfrage konventionell durch die XPath-Engine des XDBMS ausgeführt werden. Abgesehen von der beschleunigten Ausführung, macht es keinen Unterschied, ob eine Anfrage mit oder ohne Index ausgewertet wird.

Formal kann ein *schlüsselorientierter XML-Index* $i$ als Paar $i = (k, v)$ definiert werden, wobei $k$ eine Liste von absoluten Pfadausdrücken zu den Schlüsseln und $v$ ein relativer Pfad zum Wert ist.

# 3 Das Index Selection Problem angewandt auf KeyX

Aus Platzgründen kann das Index Selection Problem (ISP) an dieser Stelle nur motiviert und überblicksartig vorgestellt werden. Beim ISP geht es darum, zu einer gegebenen Menge von abfragenden und modifizierenden Datenbankoperationen eine optimale Menge von Indizes zu bestimmen, für die die Ausführungszeit aller Datenbankoperationen minimal ist. Da das ISP NP-vollständig ist[4], kann in realistischen Szenarien nur mit einer Heuristik eine Approximation der exakten Lösung gefunden werden. Ausgehend der in einem *Workload* gesammelten Datenbankoperationen werden *Indexkandidaten* bestimmt. Indexkandidaten definieren einen Index, der einzelne Operationen des Workloads beschleunigt. Da es modifizierende Operationen im Workload geben kann, die diesen Index beeinflussen, ist es nicht sinnvoll, jeden Indexkandidaten zu realisieren. Ziel des ISP ist es, die Teilmenge (*Konfiguration*) der Indexkandidatenmenge

zu bestimmen, die optimal für den gesamten Workload ist. Die Güte einer Indexkonfiguration ergibt sich dabei aus einem Kostenmaß, das der Query-Optimizer errechnet. Hierbei wird die Ausführungszeit des Workloads berechnet, wobei die in der Konfiguration enthaltenen Indizes berücksichtigt werden. Die Kosten einer Anfrage, die mit Hilfe eines Index ausgeführt wird, ergibt sich u.a. aus der Kardinalität der Schlüsselmenge, weswegen der Query-Optimizer die XML-Daten heranzieht. Die optimale Indexkonfiguration besitzt die minimalen Gesamtkosten der enthaltenen Indizes. Die in dieser Indexkonfiguration enthaltenen Indizes werden dann realisiert und bei zukünftigen Anfragen berücksichtigt. Als Nebenbedingung des ISP existiert eine Speicherbeschränkung für alle zu erzeugenden Indizes; in diesem Sinne kann das ISP mit dem ebenfalls NP-vollständigen Knapsack-Problem verglichen werden.
Durch Adaption der Problemdefinition konnten wir zeigen, dass die relationalen Lösungsansätze und Heuristiken des ISP[1][4] auf XDBMS anwendbar sind. Die Anzahl der Indexkandidaten ist jedoch größer, da nicht nur die Schlüssel eines Index, sondern auch die Werte berücksichtigt werden müssen. Auch gestaltet es sich deutlich schwieriger zu entscheiden, ob ein Index durch eine modifizierende Operation betroffen ist. In einigen Fällen ist dieses sogenante *Containment*-Problem[15] sogar unentscheidbar.
In vielen kommerziellen RDBMS werden Index-Tuning-Tools bereitgestellt, die das ISP auf verschiedene Weise aufgreifen und eine Lösung approximieren. Als Beispiele seien der *Microsoft Index Tuning Wizard* (a.k.a *Autoindex selection tool)* [2], IBMs *DB2 Advisor* [14] und der *Index Tuning Wizard* für Oracle Datenbanken genannt. Alle diese Systeme arbeiten jedoch in der Design-Phase der Datenbank und benötigen einen Administrator. Unsere Implementierung von KeyX bestimmt und aktualisiert die optimale Indexkonfiguration periodisch ohne Eingriff eines Administrators anhand der gesammelten Datenbankoperationen. Hierdurch passt sich das Datenbanksystem fortlaufend den Anfragen an.

# 4 Implementierung

Die Architektur unserer Index-Umgebung ist in Abbildung 3 dargestellt. Wie die Abbildung zeigt, besteht

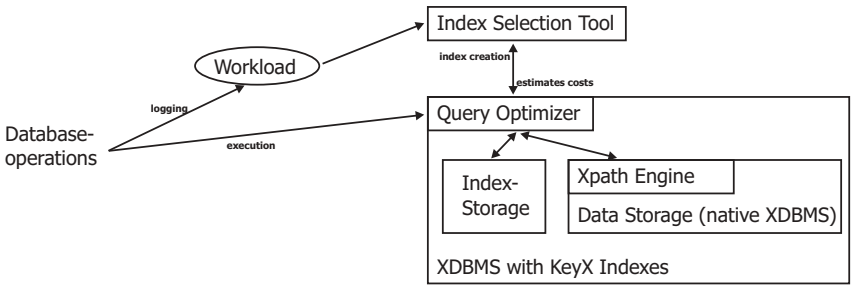


Abbildung 3: Architektur der prototypischen Implementierung

unser System aus zwei Hauptkomponenten: dem Index-Selection-Tool und dem um KeyX-Indizes erweiterten XDBMS[7], das sich in drei Subkomponenten *Data-Storage*, *Index-Storage* und *Query-Optimizer* gliedert. Die Persistenzschicht *Data-Storage* ist durch das native XDBMS *Infonyte DB*[9] mit integrierter XPath-Engine realisiert worden. Der Query-Optimizer analysiert die Pfadausdrücke von Anfragen und prüft, ob geeignete Indizes im Indexspeicher *Index-Storage* vorliegen. Unser Index-Selection-Tool versucht, zu einem Workload eine gute Indexkonfiguration zu bestimmen. Aus diesem Grund werden alle Anfragen, die an das System gestellt werden, mitgeschnitten und im Workload gespeichert. Das Tool kommuniziert mit dem Query-Optimizer, um die Auswertungskosten für eine Anfrage und eine Indexkonfiguration zu bestimmen. Nachdem das Tool eine Indexkonfiguration bestimmt hat, wird die Erzeugung der enthaltenen Indizes automatisch angeregt.

# 5 Ausblick

Nachdem die prototypische Implementierung von XML abgeschlossen ist, sollen weitere in der wissenschaftlichen Literaur vorgestellte XML-Indizierungsansätze implementiert und durch Messungen mit KeyX verglichen werden. Hierbei muss insbesondere herausgearbeitet werden, welche Arten von Anfragen ein Indizierungsansatz prinzipiell unterstützen kann. Ein weiteres Gebiet ist die Integration der verschiedenen Indizierungsansätze hinter einer gemeinsamen Schnittstelle, um die Vorteile der Ansätze zu verbinden. Offen ist, wie der Query-Optimizer den besten Index findet und wie das ISP die Kosten einer so erweiterten Indexkonfiguration bestimmen kann. Die Unterstützung von Volltextanfragen mit dem *LIKE*-Operator durch KeyX ist angedacht und erfordert eine Erweiterung der Index-Datenstruktur.

# Literatur


[1] Alberto Caprara, Matteo Fischetti, and Dario Maio. Exact and approximate algorithms for the index selection problem in physical database design. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):955–967, December 1995.

[2] Surajit Chaudhuri and Vivek R. Narasayya. An efficient cost-driven index selection tool for microsoft sql server. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*. Morgan Kaufmann, 1997.

[3] Chin-Wan Chung, Jun-Ki Min, and Kyuseok Shim. Apex: an adaptive path index for xml data. In *SIGMOD 2002, Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA*, pages 121–132. ACM Press, 2002.

[4] Douglas Comer. The difficulty of optimum index selection. *ACM Transactions on Database Systems*, 3(4):440–445, December 1978.

[5] Brian F. Cooper, Neal Sample, Michael J. Franklin, Gísli R. Hjaltason, and Moshe Shadmon. A fast index for semistructured data. In *Proceedings of 27th International Conference on Very Large Data Bases*, Roma, Italy, September 11-14 2001. Morgan Kaufmann.

[6] Thorsten Fiebig, Carl-Christian Kanne, and Guido Moerkotte. Natix - ein natives xml-dbms. *Datenbank-Spektrum*, 1(1), September 2001. Themenschwerpunkt: XML und Datenbanken.

[7] Tim Germann and Alexander Pfalzgraf. Indizierung von schemalosen xml-dokumenten anhand von xpath-anfragen. Bachelor thesis, Institute for Information Systems, University of Lübeck, 2004.

[8] IBM Corporation. IBM DB2 XML Extender.
URL: `http://www-3.ibm.com/software/data/db2/extenders/xmlext/`.

[9] Infonyte GmbH. Infonyte DB. URL: `http://www.infonyte.com`, 2003.

[10] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database managment system for semistructured data. *SIGMOD Record*, 26(3), 1997.

[11] Microsoft Corporation. SQL Server 2000 Web Services Toolkit.
URL: `http://www.microsoft.com/sql/techinfo/xml/default.asp`, 2002.

[12] Oracle Corporation. Oracle XML DB.
URL: `http://otn.oracle.com/tech/xml/xmldb/index.html`, 2003.

[13] Harald Schöning. Tamino - a dbms designed for xml. In *Proceedings of the 17th International Conference on Data Engineering*, pages 149–154, Heidelberg, Germany, April 2-6 2001. IEEE Computer Society.

[14] Gary Valentin, Michael Zuliani, and Daniel C. Zilio. Db2 advisor: An optimizer smart enough to recommend its own indexes. In *Proceedings of the 16th International Conference on Data Engineering, 28 February - 3 March, 2000, San Diego, California, USA*. IEEE Computer Society, 2000.

[15] Jean-Yves Vion-Dury and Nabil Layaïda. Containment of xpath expressions: an inference and rewriting based approach. In *Proceedings of Extreme Markup Languages*, Montreal, Quebec, Canada, 2003.

[16] World Wide Web Consortium (W3C). XML Path Language (XPath).
URL: `http://www.w3.org/TR/xpath`.

[17] World Wide Web Consortium (W3C). XQuery 1.0: An XML Query Language.
URL: `http://www.w3.org/TR/xquery`.