Effizienzbewertung für XML-Anwendungen

Studienarbeit

Universität Rostock
Institut für Informatik
Lehrstuhl DBIS

angefertigt von: Anke Diderich
geboren am: 25. August 1982 in Bergen
Betreuer: Dr.-Ing. Meike Klettke
Abgabedatum: 10. April 2006

Zusammenfassung

Im Bereich Datenbanken gibt es sowohl verschiedene Methoden zum konzeptuellen Entwurf als auch bestimmte Kriterien, um die Eigenschaften von Datenbanken zu sichern. Im Bereich XML fehlen solche Methoden und Kriterien teilweise noch. Es sind somit keine Aussagen möglich, welche XML-Konstrukte aufwendig oder effizient in der Verarbeitung sind.

Um beurteilen zu können, wie „gut“ eine XML-Anwendung ist, muss man bewerten können, wie effizient diese ausgewertet werden kann. Man benötigt also Aussagen über die Effizienz beim Einfügen und Anfragen der Dokumente. Ziel dieser Studienarbeit ist mittels solcher Einfüge- und Anfrageoperationen die Effizienz von verschiedenen XML-Anwendungen zu testen. Auf Grundlage dieser Untersuchungen können dann später Methoden zur Erstellung der XML-Konstrukte entwickelt werden, die den gesamten Entwurf unterstützen, sodass das Kriterium Effizienz schon auf konzeptueller Ebene von Bedeutung ist.

Aufbauend auf dem Wissen über Benchmarks wird im Rahmen dieser Studienarbeit die Effizienz von verschieden strukturierten XML-Konstrukten auf drei unterschiedlichen Datenbanksystemen untersucht. Bei den drei Systemen handelt es sich einerseits um IBM DB2 UDB Version 8 und andererseits um IBM DB2 Viper und Tamino von der Software AG. Es wurden drei unterschiedliche Datenbanksysteme gewählt, um herauszufinden, ob sich die Ergebnisse der einzelnen Tests verallgemeinern lassen.

Die Unterscuhungen der Konstrukte erfolgen sowohl hinsichtlich der Speicherung der Dokumente mit und ohne XML-Schemavalidierung als auch hinsichtlich der Bearbeitung unterschiedlicher Anfragen (XPath-Anfrage, COUNT- und FLWR-Query). Beispiele für verschiedene Testfälle sind: Verwendung von Attributen oder Elementen, Einfluss von Kommentaren, Einfluss der Schachtelungstiefe oder auch Verwendung von ID’s oder KEY’s. Nach der Auswertung der einzelnen Test wird am Ende der Arbeit eine allgemeine Bewertung bezüglich der Beeinflussung der Effizienz durch verschiedene XML-Konstrukte gegeben.

Abstract

For working with XML documents it's useful to know, how efficiently these are processed. In order to be able to evaluate this, knowledge about the efficiency for inserting and querying the documents is essentially. So, the principal aim is to find out, how efficiently variously structured XML documents are processed. Based on this conclusions it is theoretically possible to decide which structures are most suitable for XML applications.

Based on the knowledge about benchmarks the evaluation of the efficiency of variously structured XML documents are performed on three different systems. These three systems are IBM DB2 Version 8, IBM DB2 VIPER (Version 9.1) and Tamino from the Software AG. The reason for choosing three different systems was to determine if it's possible to generalize the results about the document structures.

For the efficiency evaluation a measurement of the time to insert the documents and to process selected queries is taken. The insert test includes inserting the documents without schema definition and validation as well as with schema validation. The query test contains an XPath query, a COUNT and a FLWR XQuery. The evaluation of the efficiency bases on the times for these operations. Examples for the different test cases within this student research project are: Using attributes instead of elements, the influence of comments and of the levels of nesting as well as the usage of IDs or KEYs. After the evaluation of every test a summary of the influence on the efficiency for variously structured XML documents is given.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation der Arbeit

Die *e**X**tensible **M**arkup **L**anguage* (XML) erfreut sich auch heute noch, fast ein Jahrzehnt nach der Verabschiedung durch das World Wide Web Consortium (W3C), in vielen unterschiedlichen Bereichen wachsender Beliebtheit.

Bei XML handelt es sich um ein einfaches Textformat, das auf dem SGML[1] - Standard basiert. Damit lässt sich die gute Lesbarkeit für den Nutzer begründen. Das XML-Format ist nicht nur für den Nutzer, sondern auch von Maschinen gut les- und verarbeitbar, da neben dem Inhalt auch Strukturinformationen in XML-Dokumenten gegeben werden. Eine Sprache, bei der sowohl Daten als auch Informationen über die Bedeutung der Daten in einem Dokument auftreten, wird als Markup Language bezeichnet. Diese Eigenschaft wird auch *selbstbeschreibend* genannt. XML ist ein einfaches und sehr flexibles Datenformat. Das bedeutet, dass die logische Struktur der Dokumente frei wählbar und das Format somit vielseitig einsetzbar ist. Aus der frei wählbaren Struktur resultieren sehr viele unterschiedliche Darstellungsmöglichkeiten und so ist XML neben der Darstellung strukturierter Daten auch für die Darstellung semistrukturierter Daten geeignet. XML ist eine Sprache zur Beschreibung von Sprachen.

Das breite Anwendungsgebiet für XML lässt sich direkt aus den oben genannten Eigenschaften von XML ableiten. Mittlerweile ist es so, dass viele Unternehmen ihre Informationen im XML-Format ablegen und weniger auf proprietäre Datenformate zurückgreifen. Das Anwendungsgebiet reicht vom Dokumentformat für Text, Formeln und Grafik über Basis für Web-Dokumente bis hin zum Zwischenformat zur Datenrepräsentation oder zum Datenaustausch. Auf Grund des großen Anwendungsgebietes hat sich auch eine Vielzahl an Technologien für die Verarbeitung von XML-Dokumenten etabliert. Zum Durchsuchen und Parsen von Dokumenten können beispielsweise XPath, die Anfragesprache XQuery oder Parser wie DOM und SAX verwendet werden. Das Transformieren von Dokumenten wird durch XSLT ermöglicht. Um einen großen Dokumentbestand auch effizient verwalten zu können, werden spezielle Datenbanken zur Speicherung der Dokumente benötigt. Hierfür gibt es auf der einen Seite XML-Datenbanksysteme (native Systeme) und auf der anderen Seite um XML-Funktionalität erweiterte objektrelationale Datenbanksysteme.

Für all diese Technologien werden Schemabeschreibungen entweder gebraucht oder sind für die Verarbeitung vorteilhaft. Die Beschreibungen geben die allgemeine Struktur für ein XML-Dokument vor. Die am häufigsten verwendeten Schemadefinitionssprachen sind DTD und XML Schema. Mit Hilfe dieser Sprachen kann die Gültigkeit eines XML-Dokuments bezüglich eines Schemas geprüft werden.
Im Rahmen dieser Arbeit wird unter einer XML-Anwendung die Modellierung eines

[1] SGML: Standard Generalized Markup Language; textbasierte Metasprache zur Definition verschiedener Auszeichnungssprachen (engl. markup languages)

bestimmten Szenarios durch solche Schemabeschreibungen verstanden.

Im Datenbankbereich gibt es sowohl verschiedene Methoden zum konzeptuellen Entwurf als auch bestimmte Kriterien, um die Eigenschaften von Datenbanken zu sichern. Solche Methoden und Kriterien fehlen im Bereich XML teilweise noch. Es gibt in der Phase des konzeptuellen Entwurfs keine Prinzipien, nach denen verfahren werden kann und es sind auch keine Aussagen möglich, welche XML-Konstrukte aufwendig oder effizient in der Verarbeitung sind.

Mit dieser Problematik beschäftigt sich der Themenkomplex „Entwurfsmethoden für semistrukturierte Anwendungen - Bewertung und Optimierung von Dokumentkollektionen“ am Lehrstuhl für Datenbanken und Informationssysteme an der Universität Rostock. Basierend auf dieser Thematik sollen Methoden entwickelt werden, die den gesamten Entwurf unterstützen, so dass das Kriterium Effizienz schon auf konzeptueller Ebene von Bedeutung ist.

Im Rahmen dieser Arbeit wird eine Teilaufgabe dieses Themenkomplexes untersucht. Ziel ist die Effizienzbewertung verschieden strukturierter XML-Dokumente. Dazu werden unterschiedliche Dokument-Strukturen auf mehreren Datenbanksystemen bezüglich der Zeit für das Einfügen und der Zeit fürs Bearbeiten ausgewählter Anfragen miteinander verglichen.

1.2 Aufbau der Arbeit

Die Arbeit ist in zwei Teile unterteilt. Beim ersten Teil handelt es sich um den eher theoretischen Teil, der sich mit den Grundlagen für die Bearbeitung des Themas befasst. Der zweite Teil stellt die praktische Arbeit dar.

Das folgende Kapitel 2 befasst sich zunächst mit Benchmarks. Benchmarks werden im Bereich XML hauptsächlich zum Testen und Bewerten der Leistungsfähigkeit (Performanz) unterschiedlicher XML-Datenbanksysteme verwendet. Durch das Testen der unterschiedlichen XML-Datenbanksysteme mit Benchmarks ist auch ein Vergleich der Systeme untereinander möglich. Um von den Benchmarks Rückschlüsse für das Testen der Dokumentstrukturen ziehen zu können, werden diese in Kapitel 2 vorgestellt. Dabei werden zunächst allgemeine Anforderungen und Eigenschaften von Benchmarks aufgezeigt und danach die XML-Datenbankbenchmarks XMach-1, XMark und XOO7 genauer erläutert. Abschließend wird in diesem Kapitel eine Bewertung der XML-Datenbankbenchmarks hinsichtlich ihrer Verwendbarkeit für das Herleiten der Testanfragen gegeben.

In Kapitel 3 werden die für das Testen der Effizienz von XML-Anwendungen verwendeten Datenbanksysteme erklärt. Es werden im Zusammenhang mit dieser Arbeit drei verschiedene Datenbanksysteme herangezogen. Auf der einen Seite handelt es sich um das native XML-Datenbankensystem Tamino von der Software AG und auf der anderen Seite um das objektrelationale Datenbanksystem DB2 Version 8 mit dem XML-Extender von IBM sowie die Weiterentwicklung dieser Version zu DB2

Viper (V9.1). DB2 Viper ermöglicht im Rahmen eines relationalen Datenbanksytems die native Speicherung von XML-Dokumenten. Bei der Vorstellung der Systeme wird vor allem auf die Architektur, die Speicherung von XML-Dokumenten und die Anfrage- und Updateverarbeitung eingegangen.

Auf der Basis der in Kapitel 2 und 3 getätigten Vorüberlegungen, werden in Kapitel 4 die einzelnen Dokumentstrukturen und Anfrageoperationen hergeleitet sowie die entsprechende Implementierung gezeigt. Mit Hilfe der hier entwickelten Dokumentstrukturen kann die Effizienz getestet und bewertet werden.

Die Ergebnisse der Tests für die Effizienz von XML-Anwendungen werden im folgenden Kapitel 5 präsentiert. Hierbei wird zunächst eine Unterteilung entsprechend der einzelnen Dokumenttypen unternommen und schließlich allgemeinere Aussagen für die Effizienz von unterschiedlich strukturierten XML-Dokumenten abgeleitet.

Die wichtigsten Aspekte der Arbeit werden abschließend in der Zusammenfassung noch einmal aufgezeigt. Weiterhin wird im letzten Kapitel 6 ein Ausblick auf Erweiterungen der Effizienzbewertung von XML-Anwendungen gegeben. Dieses erfolgt vor allem hinsichtlich der Vielzahl verschiedener Dokumente, aber auch hinsichtlich unterschiedlicher Datenbanksysteme.

Teil I
Grundlagen

2 Benchmarks

Benchmarks, vor allem Formen der TPC[2]-Benchmarks, haben sich auf dem Gebiet der relationalen Datenbanksysteme als objektives Kriterium zum Testen und Vergleichen der Funktionalität und der Leistungsfähigkeit unterschiedlicher Systeme etabliert.

Mit der zunehmenden Verarbeitung von XML z.B. als Datenaustauschformat, als natives Speicherformat oder als Basis für Web-Dokumente und die daraus resultierende wachsende Zahl von XML-Dokumenten wird die Notwendigkeit von XML-Datenbanksystemen zur Speicherung deutlich. Durch das breite Anwendungsspektrum von XML werden unterschiedliche XML-Datenbanksysteme benötigt. Herkömmliche Datenbankbenchmarks arbeiten vor allem auf strukturierten Daten und sind somit für semistruktukturierte Daten, wie sie typischerweise bei XML-Dokumenten vorliegen, nicht verwendbar. Mit Hilfe spezieller XML-Benchmarks können aber auch XML-Datenbanksysteme hinsichtlich ihrer Performanz und Leistungsfähigkeit verglichen werden.

Benchmarks, sowohl Datenbankbenchmarks als auch XML-Datenbankbenchmarks, verfolgen somit ein ähnliches Ziel wie diese Arbeit. Ziel von Benchmarks ist das Testen der verschiedenen Datenbanksysteme bezüglich ihrer Leistungsfähigkeit abhängig von bestimmten Anfragen und Updates. Die vorliegende Arbeit setzt sich mit der Effizienzbewertung von XML-Anwendungen basierend auf verschiedenen Dokumentstrukturen auseinander. Getestet wird dabei, wie bei Benchmarks, die Effizienz (Zeit) der Verarbeitung der Dokumente abhängig von bestimmten Anfragen und Updates. Beide Verfahren haben somit die gleiche Gesamtkonstellation unterscheiden sich aber in den konstanten und variablen Parametern für den Test. Bei Benchmarks ist das Testsystem die variable Größe und bei der Effizienzbewertung von XML-Anwendungen die einzelnen Dokumente (vgl. Abbildung 1).

Um Rückschlüsse von den Benchmarkverfahren auf das Testen der Effizienz von XML-Anwendungen ziehen zu können, werden im Folgenden allgemeine Kriterien für Benchmarks und XML-Benchmarks vorgestellt und darauf basierend ausgewählte XML-Benchmarks untersucht und hinsichtlich ihrer Verwendbarkeit für diese Arbeit bewertet.

[2]Transaction Processing Performance Counsil; Institution für das Entwickeln von Benchmarks [Gra93]

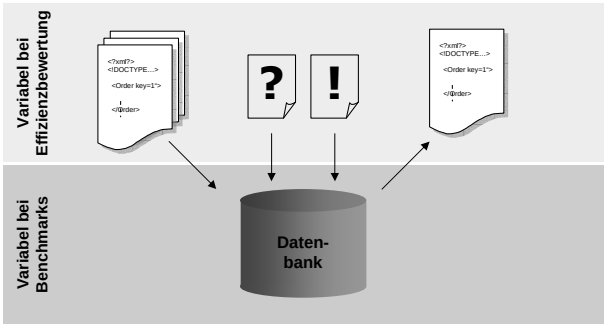

Abbildung 1: Vergleich der variablen Größen bei Benchmarks/Effizienzbewertung

2.1 Anforderungen an Benchmarks

Ein Benchmark sollte ein objektives Kriterium zum Testen der Leistungsfähigkeit und Funktionalität eines Systems darstellen. Um dieser Aufgabe gerecht zu werden, muss ein Benchmark eine Reihe von Anforderungen, die in [Gra93] nauchzulesen sind, erfüllen. Damit von einem *aussagekräftigen* Benchmark die Rede sein kann, sollte er mindestens die folgenden vier Anforderungen erfüllen.

- **Relevanz:** Um aussagekräftige Ergebnisse zu erzielen, sollte der Benchmark die für den Anwendungsbereich wesentlichen Merkmale und Operationen abdecken. Das bedeutet, dass der Benchmark beispielsweise die Leistung des Systems und das Preis-Leistungs-Verhältnis ausgewählter Operationen bestimmen können muss.

- **Portabilität:** Der Benchmark sollte einfach auf allen relevanten Rechner-Plattformen implementiert werden können.

- **Skalierbarkeit:** Der Benchmark sollte für unterschiedlich große Konfigurationen eine Leistungsbewertung ermöglichen. Die verschiedenen Konfigurationen reichen dabei von einem einzelnen PC bis hin zu Großrechner-Konfigurationen, für zentralisierte und verteilte Architekturen sowie für beliebig große Datenmengen.

- **Einfachheit:** Die Architektur und Funktionsweise eines Benchmarks muss so einfach sein, dass er leicht verständlich, nachvollziehbar und implementierbar ist. Diese Eigenschaften sind Grundlage für die Akzeptanz eines Benchmarks und sichern gleichzeitig die Überprüfbarkeit der Ergebnisse.

2.2 XML-Benchmarks

Für herkömmliche Datenbankbenchmarks gibt es die vier oben genannten Kriterien: Relevanz, Portabilität, Skalierbarkeit und Einfachheit. Diese gelten natürlich auch für XML-Datenbankbenchmarks. Neben diesen Kriterien müssen für XML Benchmarks weitere ergänzt bzw. angepasst werden. Die Notwendigkeit dafür ergibt sich

aus dem breiten Anwendungsgebiet von XML und den abgeänderten Anforderungen beim Testen eines XML-Datenbanksystems im Gegensatz zu einem herkömmlichen Datenbanksystem.
Im Folgenden soll auf die Kriterien Domäne, Bewertungsziel, Anzahl der Nutzer und Operationen-Mix eingegangen werden.

Domäne Im XML-Benchmarkbereich gibt es keinen universellen Benchmark, der alle Anwendungsbereiche abdecken kann. Dies resultiert vor allem aus dem breiten Einsatzspektrum von XML, welches es nicht erlaubt ohne eine Verletzung der Einfachheit eine Datenbankstruktur und Operationen so zu definieren, dass alle Bereiche bedient werden. Ein solch universeller Benchmark ist aber auch nicht das Ziel der Benchmarkentwicklung, vielmeht sollte jeder Benchmark nur einem bestimmten Anwendungsschwerpunkt (Domäne) genügen. Dieser wird charakterisiert durch die unterschiedlichen Arten von XML-Daten und den entsprechenden Operationen. Es werden drei unterschiedliche Arten von XML-Daten unterschieden: dokumentorientierte, datenorientierte und gemischte Daten.

Dokumentkollektionen Diese Domäne beschreibt die Verwaltung von dokumentorientierten Daten im Rahmen von Dokumentkollektionen. Die spezifischen Operationen reichen von Volltextsuche auf Elementen über das Abrufen bis zum Einfügen und Löschen ganzer Dokumente.

Strukturierte Daten Bei dieser Domäne werden datenorientierte oder strukturierte XML-Daten verwaltet. Operationen sind beispielsweise das Suchen und Ändern von Attributwerten oder auch Anfragen mit Sortierung, Aggregation und Verbundanweisungen.

Gemischte Daten In vielen Anwendungsbereichen wird nicht ausschließlich mit datenorientierten oder dokumentorientierten Daten gearbeitet, sondern vielmehr mit einer Mischung beider. Neben strukturierten Daten müssen auch Informationen zu den Daten in textueller Form verwaltet werden. Für die Operationen dieser Domäne ergibt sich ebenfalls eine Mischung aus den beiden Vorangegangenen.

Bewertungsziel Hierbei wird zwischen der Bewertung eines gesamten Systems oder einzelner Komponenten eines Systems unterschieden. In den meisten Fällen ist das Leistungsverhalten des gesamten Systems, welches sich aus dem Zusammenwirken der einzelnen Komponenten ergibt, interessant. Aber teilweise ist auch die Bewertung von Einzelkomponenten, wie z.B. dem Anfrageprozessor, aussagekräftig und hilfreich.

Anzahl der Nutzer Im Allgemeinen sind Datenbanksysteme, auch XML-Datenbanksysteme, für die gleichzeitige Nutzung durch mehrere Personen konzipiert. In diesem Fall kann mit einem Benchmark eine Messung der Durchsatzleistung erfolgen

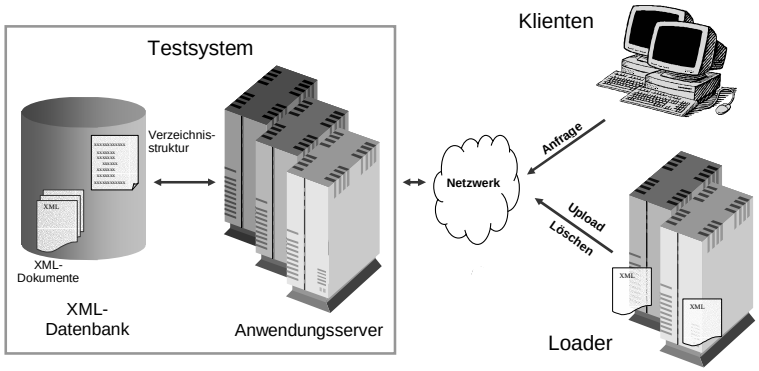

Abbildung 2: Systemarchitektur XMach-1 nach [BR01]

und der Mehrbenutzerbetrieb bewertet werden. Der Einzelnutzerbetrieb hingegen kann hilfreich beim Testen einzelner Komponenten sein, wobei dann die Antwortzeiten gemessen werden.

Operationen-Mix Im Rahmen eines Benchmarks muss eine Arbeitslast zum Testen bestimmt werden, die sich aus verschiedenen Operationen zusammensetzt. Die Anzahl der unterschiedlichen Operationen sollte dabei möglichst gering gehalten werden, was meist den Einsatz komplexer Operationen erfordert. Hierbei sollte beachtet werden, dass komplexe Operationen die Bewertung eines Systems erschweren können.

Mehr Informationen zu den einzelnen Domänen, den weiteren Kriterien für XML-Datenbankbenchmarks und auch zu Problemen sind in [RV03] nachzulesen. Im Folgenden werden die drei bekanntesten XML-Benchmarks XMach-1, XMark und XOO7 vorgestellt und bewertet.

2.2.1 XMach-1

Der XMach-1 Benchmark (XML Data Management Benchmark) wurde im Jahr 2000 an der Universität Leipzig entwickelt und Anfang 2001 in [BR01] veröffentlicht. Beim XMach-1 Benchmark handelt es sich um den ersten, speziell für XML-Systeme entwickelten Benchmark. Er wurde für den Mehrbenutzerbetrieb konzipiert und simuliert eine Web-Anwendung zur Verwaltung von Textdokumenten.

Systemaufbau Die Systemarchitektur des XMach-1 setzt sich aus dem zu testenden System und der Anwenderseite zusammen. Zu dem Testsystem gehören dabei die XML-Datenbank und die Anwendungsserver und zur Anwenderseite der Loader und die Klienten (Browser). Die Anzahl der Datenbanken und Anwendungsserver ist beim Testsystem frei wählbar. Der Anwendungsserver dient zur Abbildung generischer Anfragen der Klienten in Anfragen für die Datenbank und zum Lösen von

Mapping- und Transformationsaufgaben. Die Interaktion mit den Anwendungsservern erfolgt über eine HTTP-Schnittstelle.

Domäne Der XMach-1 simuliert eine Webanwendung zur Verwaltung unterschiedlicher textorientierter Dokumente. Die Anwendung besteht aus einer Verzeichnisstruktur und einzelnen XML-Dokumenten. Auf Grund der Vielzahl verschiedener Textdokumente steht bei XMach-1 die Domäne „Dokumentkollektion“ im Vordergrund. Im Zusammenhang mit der Verzeichnisstruktur werden aber auch strukturierte Daten verwendet. Die Verzeichnisstruktur dient zur Verwaltung der Metadaten der einzelnen Dokumente. Es handelt sich bei dem Benchmark somit um eine gemischte Domäne mit Schwerpunkt auf den Dokumentkollektionen.

XML-Anforderungen Bei dem XMach-1 Benchmark werden nicht alle möglichen Spezifikationen von XML berücksichtigt. So wird beispielsweise XML-Schema nicht unterstützt. Diese fehlende Unterstützung ist allerdings auf die spätere Entwicklung von XML-Schema zurückzuführen. Weiterhin finden auch Namespace, Abschnitte, Entities und Character Data keine Beachtung.

Daten Wie oben bereits angedeutet gibt es beim XMach-1 Benchmark zwei verschiedene Dokumenttypen. Auf der einen Seite strukturierte Daten und auf der anderen Seite Textdokumente, die aus Kapiteln, Paragraphen und Abschnitten bestehen.

Die strukturierten Daten werden durch die Verzeichnisstruktur (Directory Document), welche Metadaten über alle gespeicherten Textdokumente enthält, vertreten.

Die verschiedenen Textdokumente repräsentieren die dokumentorientierten Daten. Sie werden bei XMach-1 synthetisch erzeugt. Das bedeutet, aus einer Wortliste[3] wird ein natürlich-sprachlicher Text generiert. Die Größe der einzelnen Dokumente liegt zwischen 2 und 100 KByte und es werden sowohl flache als auch tief verschachtelte Dokumente erzeugt.

An dieser Stelle ist noch zu erwähnen, dass die Initalisierung der Datenbank mit einer bestimmten Anzahl an Dokumenten erfolgt. Um die Skalierbarkeit des Benchmarks zu gewährleiten, gibt es vier verschiedene Datenbankgrößen mit entweder 10.000, 100.000, 1.000.000 oder 10.000.000 Dokumenten. Die Anzahl der Dokumente verändert sich aber im Laufe des Benchmarks.

Bezogen auf die Speicherung der Dokumente gibt es zwei verschiedene Varianten des XMach-1 Benchmarks: auf der einen Seite mit und auf der anderen Seite ohne Schemavalidierung.

[3] Wortliste: Verwendet wird eine Liste der 10000 gebräuchlichsten englischen Wörter

Operationen Der XMach-1 Benchmark beinhaltet acht Anfragetypen und drei verschiedene Änderungsoperationen. Die Operationen sind in natürlicher Sprache vorgegeben und können somit zum Testen den entsprechenden Anfragesprachen angepasst werden. Die verschiedenen Anfragetypen reichen vom Abfragen vollständiger Dokumente über Textsuche in Dokumenten bis hin zu Anfragen mit Sortier- und Verbundoperationen. Somit wird der wertorientierte, der textorientierte und auch der strukturierte Bereich mit den Anfragen abgedeckt. Bei den drei Änderungsoperationen handelt es sich auf der einen Seite um das Löschen und Hinzufügen eines Dokuments und auf der anderen Seite um die Änderung von Attributwerten (Metadaten) in der Verzeichnisstruktur. Für das Gesamtergebnis des Benchmarks kommt den einzelnen Operationen eine unterschiedliche Wichtung zu. Bei den Anfrageoperationen ist die Wichtung zwischen 30% und 4%. Die Änderungsoperationen bilden zusammen nur einen Anteil von 2% am Gesamtergebnis. Die Wichtung ist unter anderem von der Umsetzbarkeit der einzelnen Operationen in den Systemen abhängig. Für eine detailiertere Beschreibung der einzelnen Operationen sei auf [BR01] und [RB02] verwiesen.

Metriken An dieser Stelle wird die Unterscheidung des XMach-1 in eine schemalose und schemaunterstützte Variante noch einmal wichtig. In beiden Fällen liegt das Hauptaugenmerk auf der Messung des Durchsatzes, wie es für einen Mehrbenutzer-Benchmark typisch ist. Der Durchsatz wird im schemaunterstützten Fall als XML-Anfragen pro Sekunde (Xqps) angegeben und entspricht den ausgeführten Q1-Anfragen (Zurückliefern eines Dokuments zu einer gegebenen URL). Im schemalosen Fall wird die Einheit um „sl“ zu Xqpssl ergänzt. Neben dem Durchsatz werden auch die Antwortzeiten der einzelnen Operationen gemessen. Hierbei wurde festgelegt, dass 90% der acht Anfragetypen (Q1-Q8) und der dritten Änderungsoperation (Änderung eines Attributwertes) unter drei Sekunden ausgeführt werden müssen. Dieses gewährleistet, dass der Durchsatz nicht zu Lasten der Anwortzeit verbessert werden kann. Bei den übrigen beiden Änderungsoperationen liegt die Zeitspanne bei 20 Sekunden.

2.2.2 XMark

Der XMark Benchmark wurde von einem unabhängigen Forscherteam am National Research Institute for Mathematics and Computer Science (CWI) in den Niederlanden entwickelt. Veröffentlicht wurde er nach dem XMach-1 Benchmark Mitte 2001 in [SWK+01]. Es handelt sich beim XMark um einen Einzelnutzer-Benchmark, der auf einem lokalen Computersystem durchgeführt wird. Das vordergründige Ziel des Benchmarks ist die Abdeckung eines breiten Anfragespektrums, wie es bei realen Anwendungsszenarien üblich ist.

Systemaufbau Wie oben bereits angedeutet besteht das Testsystem nur aus einem einzigen Rechner und zwar dem zu testenden Server. Die Belastung des Systems

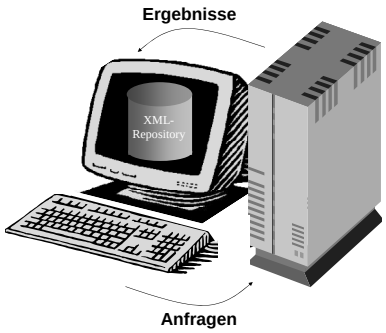

Abbildung 3: Systemarchitektur XMark

durch Client-Anfragen wird lokal auf diesem Rechner simuliert. Die schematische Systemarchitektur des XMark ist in Abbildung 3 dargestellt.

Domäne Das XMark Anwendungsszenario simuliert die Datenbank eines Internet-Auktionshauses. Auch bei diesem Benchmark gibt es sowohl daten- als auch dokumentorientierte Ansätze. Die Domäne ist somit ebenfalls von gemischtem Charakter. Der datenorientierte Aspekt steht dabei jedoch im Vordergrund. Er äußert sich in Form des XML-Dokument, welches die gesamten XML-Daten enthält. Die Vielzahl an Fakten in dem XML-Dokument unterliegen nämlich einer festen Struktur. Da neben diesen Fakten aber auch Beschreibungstexte berücksichtigt werden, ist der dokumentorientierte Aspekt ebenfalls vertreten.

XML-Anforderungen Bei XMark wird auf einige Features von XML verzichtet. Dies bedeutet, dass keine Dokumente mit Entities oder Notations generiert werden können. Weiterhin wird keine Unterscheidung zwischen Parseable Character Data und Character Data unternommen und Namespace wird vollständig vernachlässigt. Unterstützt werden bei XMark sowohl DTD's als auch XML-Schemainformationen, um den Testsystemen effizientere Transformationen zu ermöglichen. Auf die Leistungsbewertung hat die Nutzung dieser Informationen jedoch keinen Einfluss.

Daten Die verschiedenen Daten werden wie oben bereits erwähnt in einem einzigen Dokument verwaltet. In diesem Dokument sind u.a. alle Gegenstände, Kategorien, Personen und Auktionen eines Auktionshauses gespeichert. Die einzelnen Elemente werden über entsprechende Verweise miteinander verbunden. Auf Grund dieser Tatsache kann das Dokument sehr groß werden. Allerdings ist die Dokumentgröße (10KByte bis 10GByte) unabhängig von den einzelnen Verweisen über einen Skalierungsfaktor regelbar. Für die Erzeugung der Daten und somit des XML-Dokumentes wird bei XMark das Tool xmlgen[4] verwendet.

[4]xmlgen: Tool zur Generierung des XML-Dokumentes; für nähere Informationen zu xmlgen sei auf [SWK+02] verwiesen

Operationen Der XMark Benchmark beinhaltet 20 verschiedene Anfragen, die in XQuery formuliert sind. Auf die Definition von Änderungsoperationen wurde auf Grund eines fehlenden Standards verzichtet. Die große Anzahl an Anfragen lässt sich mit dem Ziel der Behandlung eines möglichst breiten Anfragespektrums begründen. Die einzelnen Anfragen können in bestimmte Kategorien unterteilt werden. Diese reichen von exakter Suche über Volltextsuche bis hin zu Sortierung, Aggregation und Rekonstruktion von Teildokumenten.
Eine ausführlichere Beschreibung der einzelnen Operationen ist in [SWK+02] zu finden.

Metriken Im Gegensatz zum Mehrbenutzer-Benchmark XMach-1 wird bei XMark nicht das gesamte System bewertet, sondern nur die Ausführungszeiten der Operationen. Dafür wird jede Operation einzeln betrachtet und der Benchmark liefert als Ergebnis die entsprechende Zeit. Als Einheit für das Ergebnis wird Millisekunde pro Anfrage verwendet. Bei XMark besteht die Möglichkeit Operationen mit sehr kurzen Ausführungszeiten mehrmals zu wiederholen, wobei aber das Problem des Caching die Zeit beeinflussen kann. Solche Wiederholungen sind für sehr schnelle Operationen jedoch notwendig, um die Zeit für die Bearbeitung messen zu können. Letzlich wird die ermittelte Zeit durch die Anzahl der Wiederholungen geteilt.

2.2.3 XOO7

Der XOO7 Benchmark ist ebenfalls ein Einzelnutzer-Benchmark und wurde Mitte 2001 in [BDL+01] veröffentlicht. Er wurde von der National University of Singapore und der Arizona State University entwickelt. Bei dem XOO7 handelt es sich um eine Erweiterung bzw. Anpassung des damals schon existierenden OO7 Benchmarks [CDN93]. Dieser ist ein Benchmark für objekt-orientierte Datenbanksysteme und wurde auf Grund der vorhandenen Ähnlichkeiten zwischen objekt-orientierten Systemen und XML gewählt. Die Anpassung des Benchmarks an XML wird in [BDL+01] beschrieben.

Es existiert für den XOO7 Benchmark noch eine Weiterentwicklung bzw. Erweiterung, die sich genauer mit dem dokumentorientierten Aspekt bei Operationen befasst. Hierbei wurden dem XOO7 neben den bereits vorhandenen dokumentorientierte Anfragen noch weitere, speziellere hinzugefügt.
Im Zusammenhang mit der vorliegenden Arbeit soll auf diese Erweiterung jedoch nicht weiter eingegangen werden. Für weitere Einzelheiten bezüglich des Themas ist [NLB+01] zu empfehlen.

Systemaufbau Wie oben bereits erwähnt ist der XOO7 Benchmark ein Einzelnutzer-Benchmark und benötigt genauso wie der XMark Benchmark nur einen einzigen Rechner, der gleichzeitig das zu testende System darstellt. Die Systemarchtiktur entspricht somit der des XMark Benchmarks (vgl. Abbildung 3).

Domäne Dem XOO7 Benchmark liegt im Gegensatz zu den beiden vorhergehenden Benchmarks kein konkretes Testszenario zu Grunde. Es handelt sich aber auch bei diesem Benchmark um eine Domäne von gemischtem Charakter. Der Benchmark beschreibt komplex aufgebaute Objekte mit „Teil-von“ Beziehungen. Das bedeutet, es gibt Entwurfsobjekte, die aus vielen Einzelobjekten bestehen. Die Einzelobjekte können wiederum aus mehreren atomaren Objekten bestehen. Die Modellierung der Objekte erfolgt mit einer festen und regelmäßigen Struktur. Die Daten werden alle in den Attributen gespeichert. Durch diese Modellierung wird der datenorientierte Aspekt vertreten. Der dokumentorientierte Aspekt zeigt sich bei den textuellen Beschreibungen der Objekte und den Operationen auf diesen.

XML-Anforderungen Auch bei diesem Benchmark werden nicht alle XML-Features unterstützt. So ist beispielsweise die Referenzierung mittels ID/IDREF nicht möglich. Das führt dazu, dass Objekte redundant vorliegen können. Eine Besonderheit des XOO7 Benchmarks ist die Verwirklichung eines Vererbungskonzeptes. Hierfür müssen für die Elemente die Attribute dupliziert werden.

Daten Der XOO7 Benchmark basiert auf einem einzigen Dokument, indem alle relevanten Daten enthalten sind. Das den Daten zugrunde liegende Schema ist fest und regelmäßig. Das bedeutet, dass keine optionalen Attribute oder Elemente möglich sind und weiterhin auch kein freier Elementinhalt erlaubt ist. Durch dieses festgelegte Schema wird der datenorientierte Aspekt des Benchmarks nochmals verstärkt.
Die Daten werden mit Hilfe eines Generators erzeugt und es besteht die Möglichkeit, durch Angabe bestimmter Parameter Dokumente in verschiedenen Größen zu generieren. Die Größe eines Dokuments kann dabei von 4 MByte bis 1 GByte reichen. Die Erzeugung der Zeichenketten und Texte erfolgt über Zähler oder durch die Wiederholung von kurzen Zeichenketten.

Operationen Der XOO7 Benchmark beinhaltet ausschließlich Anfrageoperationen, auf die Definition von Änderungsoperationen wurde verzichtet. Die Anfragen sind zunächst verbal beschrieben und später um eine XQuery Notation ergänzt worden. Die verbale Formulierung gewährleistet die Anpassung der einzelnen Anfragen auf die entsprechende Anfragesprache des zu testenden Systems.
Die Arbeitslast des XOO7 umfasst 18 verschiedenen Anfragen an die Datenbank. Diese Anzahl setzt sich aus den acht Anfragen des OO7 Benchmarks und zehn zum Testen der XML Funktionalität ergänzte Anfragen zusammen. Wie zu Beginn des XOO7 Benchmarks bereits erläutert, erhöht sich die Anzahl der Anfragen, wenn die Erweiterung um spezielle dokumentorientierte Anfragen betrachtet wird.

Metriken Beim XOO7 Benchmark handelt es sich um einen Einzelnutzer- Benchmark. Dies bedeutet, dass nur ein einziger Rechner (Testsystem) benötigt wird, auf

dem alle Daten lokal gespeichert werden. Die Anfragen werden unabhängig voneinander betrachtet und das Ergebnis des Benchmarks sind die entsprechenden Antwortzeiten. Als Einheit des Ergebnisses wird die Zeit pro Anfrage verwendet.

2.2.4 Bewertung

An dieser Stelle wird keine Bewertung der einzelnen Benchmarks hinsichtlich ihrer Eignung zur anwendungsspezifischen Leistungsbewertung verschiedener Datenbanksysteme vorgenommen. Die Entscheidung, welcher Benchmark zum Testen eines bestimmten Systems verwendet werden soll, sollte abhängig von der Anwendungssituation anhand der Eigenschaften der vorgestellten Benchmarks gefällt werden. Vordergründig ist dabei die Frage, ob es sich um die Bewertung eines gesamten Systems oder eines einzelnen Anfrageprozessors handelt. Ein weiteres Entscheidungskriterium ist beispielsweise die Anzahl der Nutzer oder auch das vorliegende Testszenario.

Für die vorliegende Arbeit ist ein Vergleich bestimmter Eigenschaften und Besonderheiten der Benchmarks wichtiger. Hierbei geht es vor allem um das Bewertungsziel der einzelnen Benchmarks. Wie gesehen beschränken sich XMark und XOO7 auf das Bewerten einer einzelnen Komponenten des Systems, den Anfrageprozessor. Da es sich auch beim Testen der Effizienz von XML-Anwendungen lediglich um das Testen einer einzelnen Komponente handelt, können die damit verbundenen Besonderheiten wie der Systemaufbau übernommen werden. Weiterhin ist auch die Beschränkung auf den Einbenutzer-Betrieb (vgl. XMark und XOO7) für die Effizienzbewertung ausreichend. Eine entscheidene Rolle für diese Arbeit spielen außerdem die unterschiedlichen Anfrage- und Updateoperationen der Benchmarks. Diesbezüglich werden im Konzept (siehe Kapitel 4) Anfragen, die von allen Benchmarks unterstützt werden, ausgewählt und zum Testen der Effizienz verwendet.

3 Datenbanksysteme

Für die Speicherung und Verarbeitung von XML-Dokumenten in Datenbanksystemen existieren unterschiedliche Entwicklungslinien. Nachfolgend sollen zum einem Weiterentwicklungen vorhandener (objekt-)relationaler DBS um XML-Funktionalität und zum anderen die Entwicklung neuer, spezieller XML-Datenbanksysteme (native XML-Datenbanksysteme), die den XML-Anforderungen angepasst wurden, vorgestellt werden.

Bei den objektrelationalen Datenbanken handelt es sich im Kern um relationale Datenbanksysteme, die um bestimmte objektorientierte Konzepte (u.a. Objektidentität, Typhierarchien, Typkonstruktoren) erweitert wurden. Auf diese Weise wird das Konstruieren spezieller, benutzerdefinierter Datentypen ermöglicht. Die erzeugten Typen können als Attributtyp einer Relation oder als Tabellentyp verwendet werden. Weiterhin ist auch die Definition spezieller Methoden für diese Typen möglich. Eine Normierung für das Vorgehen beinhaltet der Standard SQL/99. Dieser Standard stellt auch für die Speicherung großer Datenmengen neue Datentypen zur Verfügung: sogenannte LOB (Large Objects). Hierbei wird in BLOB und CLOB unterschieden. BLOB's sind Binärdaten, z.B. Bilder, Audio- oder Videodaten und CLOB's sind lesbare Zeichen. Mit dem Datentyp CLOB existiert also eine einfache Methode, um XML-Dokumente in objektrelationalen Datenbanksystemen zu speichern. In diesem Zusammenhang werden sowohl die Tags als auch der Inhalt als Fließtext gespeichert. Die Erschließung des Inhalts erfolgt z.B. durch Volltextsuche oder durch Zerlegung der XML-Dokumente in eine relationale Speicherstruktur (Tabellen), wie sie relationalen Datenbanken zu Grunde liegt (Mapping).

Alternativ zu den (objekt-)relationalen Datenbanksystemen, gibt es auch Datenbanksysteme, die direkt für die Speicherung und Manipulation von XML-Daten entworfen wurden, sog. „native XML-Datenbanken“ . Der Zugriff und die Bearbeitung der Daten erfolgt bei diesen Systemen über XML-spezifische Technologien wie z.B. XPath bzw. XSLT, DOM oder SAX. Es existieren verschiedene Anforderungen an ein natives XML-Datenbanksystem, welche bei der XML:DB Initiative [xml:db] zusammengetragen wurden. Es sollte die Definition eines logischen Modells erfolgen und weiterhin sollte die grundlegende Einheit ein XML-Dokument sein (vgl. relationale DB: Tabelle). Die Art der physischen Speicherung ist nicht vorgeschrieben.
Mehr Informationen zu diesem Thema sind in dem Artikel „XML und Datenbanken“ von R. Bourret [Bou05] nachzulesen.

In den folgenden Abschnitten wird jeweils ein Vertreter beider Entwicklungslinien vorgestellt. Auf der einen Seite handelt es sich um DB2 Universal Database von IBM und auf der anderen Seite um Tamino von der Software AG. Die vorgestellten Systeme entsprechen den für den Effizienztest verwendeten Systemen (vgl. Kapitel 5).

3.1 IBM DB2 Universal Database (DB2 V8.2)

Die DB2 Universal Database von IBM ist ein objektrelationales Datenbanksystem nach dem Standard SQL/99 (vgl. oben). Es bietet die Möglichkeit der funktionellen Erweiterung des Systems durch verschiedene Softwarepakete, sog. Extender. Es gibt zwei solche Extender, die für die Verarbeitung von XML-Dokumenten geeignet sind. Das sind einerseits der XML-Extender und andererseits der Text-Extender. Der XML-Extender wird im nächsten Kapitel bezüglich seiner Funktionalität genauer untersucht. Der Text-Extender hingegen soll nur als Ergänzung zur Volltextsuche betrachtet werden.

3.1.1 DB2 UDB mit XML-Extender

Der XML-Extender stellt neue Datentypen zur Speicherung und neue Funktionen zum Arbeiten mit XML-Dokumenten zur Verfügung. Er bietet zwei verschiedene Möglichkeiten zum Speichern der XML-Dokumente in einer Datenbank. Auf der einen Seite XML Column und auf der anderen Seite XML Collection. Bei XML Column erfolgt die Speicherung des Dokumentes als Ganzes, bei XML Collection hingegen wird der Inhalt des XML-Dokumentes auf objektrelationale Strukturen (Tabellen) abgebildet. Die Dokumente können sowohl intern in der Datenbank als auch extern abgelegt sein. Neben der Möglichkeit zur Speicherung der Dokumente existieren auch Funktionen zum Anfragen, Ändern und Löschen.

3.1.1.1 XML Column

XML Column [IBM04] dient zur Speicherung eines gesamten XML-Dokumentes oder -Fragmentes in der ursprünglichen Form. XML Column ist ein *abstrakter Datentyp*, der als *User Defined Type* (UDT) definiert wird. Das Verhalten des Datentyps wird durch *User Defined Functions* (UDFs) gesteuert.

Speicherung Es werden drei verschiedene Datentypen zur Speicherung als XML Column vom XML-Extender angeboten. Dabei handelt es sich um XMLCLOB für interne, große Dokumente, XMLVARCHAR für interne, kurze Dokumente und XMLFILE zur Speicherung einer Referenz auf eine externe Datei, die durch das Datenbanksystem verwaltet wird. Nachdem für eine Spalte der Datenbank der entsprechende User Defined Type definiert wurde, wird von einer XML-Spalte gesprochen, in der die Dokumente gespeichert werden können. Zum Abspeichern von Dokumenten, bei denen der Ursprungstyp verschieden vom Zieltyp ist, werden zu jedem UDT vom XML-Extender spezielle UDF's zur Verfügung gestellt. Hierauf soll in dieser Arbeit jedoch nicht näher eingegangen, sondern auf [IBM04] verwiesen werden.

Indexierung Ein Nachteil von XML Column ist das Fehlen einer direkten Indexierungsmethode für XML-Typen. Um dennoch eine effiziente Suche nach Attribut-

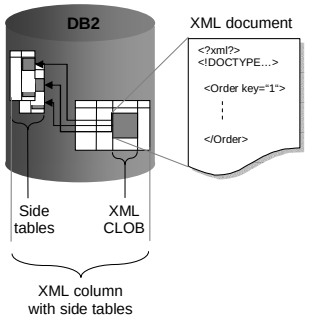

Abbildung 4: Speicherung als XML Column mit Side tables nach [IBM04]

oder Elementwerten zu ermöglichen, können Seitentabellen (*side tables*) genutzt werden. Hierbei handelt es sich um relationale Tabellen, in denen die Element- und Attributwerte gespeichert werden. Für diese Seitentabellen können dann mit herkömmlichen DB2-Methoden Zugriffspfade definiert werden. Nachdem erstmalig die Einrichtung der Seitentabellen erfolgt ist, wird die Verwaltung und Aktualisierung dieser von DB2 übernommen. Zum Definieren des Mappings von XML-Inhalten auf Seitentabellen muss auf DAD's[5] zurückgegriffen werden.

Anfragen Es gibt unterschiedliche Möglichkeiten Anfragen an eine XML Column zu stellen. Auf der einen Seite kann ein gesamtes Dokument abgefragt werden oder auf der anderen Seite einzelne Element- und Attributwerte. Elemente können jedoch nur abgefragt werden, wenn es sich um Blattelemente handelt. Das Anfragen erfolgt mit Hilfe eines XPath-Ausdrucks, der in SQL eingebettet ist. Da es sich bei DB2 UDB um ein objektrelationales Datenbanksystem handelt, müssen die Anfragen in SQL formuliert werden.
Um ein gesamtes XML-Dokument abzufragen, muss eine SQL-Anweisung konstruiert werden, die die entsprechende Tabelle und Spalte, in der das Dokument gespeichert ist, beinhaltet.

Syntax:
```
SELECT Spalte
FROM Tabelle
WHERE Selektionsbedingung
```

Zum Abfragen von Element- und Attributwerten wird vom XML-Extender die *extract*-Funktion zur Verfügung gestellt.

Syntax:
```
SELECT extractDatentyp(Spalte, Pfad)
FROM Tabelle
WHERE Selektionsbedingung
```

[5]Document Access Definition; spezielle XML-Dateien zur Beschreibung der Abbildung von XML-Inhalten auf Datenbanktabellen (siehe auch: „Beschreibung des Mapping“ bei XML Collection, Kapitel 3.1.1.2)

Datentyp:	Datentyp der Rückgabe der *extract*-Funktion
Spalte:	Name der XML-Spalte, in der sich das gesuchte Element bzw. Attribut befindet
Pfad:	XPath-Ausdruck zur Beschreibung des Elements bzw. Attributs im XML-Dokument

Ergänzend ist zu sagen, wenn es sich bei dem von einer Anfragen betroffenen Wert um einen eindeutigen Wert handelt, kann eine skalare Funktion (z.B. `extractInteger()`) verwendet werden, ansonsten muss eine Tabellenfunktion genutzt werden (`extractIntegers()`).

Updates Dieser Bereich umfasst das Ändern und Löschen eines Dokumentes. Für das Ändern eines Dokumentes existieren zwei Varianten. Zum einen das Ersetzen eines gesamten Dokuments durch ein Neues

Syntax:
```
UPDATE Tabelle
SET Spalte = neuesDokument
WHERE Selektionsbedingung
```

und zum anderen das Ändern bzw. Ersetzen von Attribut- oder Elementwerten in einem XML-Dokument. Hierfür muss der Pfad des zu ändernden Wertes angegeben werden.

Syntax:
```
UPDATE Tabelle
SET Spalte = Update(Spalte, Pfad, neuerWert)
WHERE Selektionsbedingung
```

Bei mehrfachem Vorkommen eines Elements oder Attributs wird zwingend jedes Vorkommen mit dem neuen Wert ersetzt. Daran ist schon zu erkennen, dass das Ändern von Element- und Attributwerten mit vielen Einschränkungen verbunden ist. Weiterhin sind beispielsweise das Einfügen oder Löschen eines Elements und das relative Ändern eines Wertes nicht möglich. Für mehr Informationen zu diesen und anderen Einschränkungen ist [Sch03] zu empfehlen. Zum Löschen eines oder mehrerer ganzer Dokumente wird die SQL-Anweisung `DELETE` verwendet. Die `WHERE`-Klausel dient dabei zur genaueren Spezifizierung der zu löschenden Dokumente.

Syntax:
```
DELETE FROM Tabelle
WHERE Selektionsbedingung
```

3.1.1.2 XML Collection

Mit XML Collection ist sowohl das Zerlegen als auch das Erzeugen von XML-Dokumenten möglich. Das Konzept beinhaltet, dass XML-Dokumente nicht als Ganzes, sondern intern in verschiedenen Tabellen gespeichert werden. In diesem Zusammenhang wird auch oft von einer inhaltsorientierten Abbildung gesprochen. Die-

se Technik ist somit nur für datenorientierte Dokumenten mit einer regelmäßigen Struktur geeignet. Gemischter Inhalt kann nicht abgebildet werden. Die Zerlegung vorhandener und das Erzeugen neuer Dokumente erfolgt über ein benutzerdefiniertes Mapping. Eine XML Collection ist somit eine Menge relationaler Tabellen, die Daten aus XML-Dokumenten beinhalten. Durch die Speicherung in Tabellen stehen für die Verwaltung der Daten alle Datenbank-Funktionen zur Verfügung.

Beschreibung des Mapping Eine XML Collection wird in einer speziellen Datei definiert, der sogenannten Document Access Definition (DAD). Die DAD ist selbst in Form eines XML-Dokuments geschrieben und definiert die Abbildung von XML-Inhalten wie Attribute und Elemente auf Tabellen, indem in der DAD genau angegeben wird, welche Inhalte wo abgespeichert werden.

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM „C:\dxx\dtd\dad.dtd">
<DAD>
  <DTDID>C:\dtds\Abt.dtd</DTDID>
  <validation>YES</validation>
  <Xcollection>
    <prolog>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE Person SYSTEM „C:\dtds\Abt.dtd"</doctype>
    <root_node>
      <element_node name=„Abteilung">
        <RDB_node>
          <table name=„Abteilungen" key=„Abtnr"/>
          <table name=„Angestellte" key=„Personalnr"/>
          <condition>
            Abteilungen.Abtnr=Angestellte.Abtnr
          </condition>
        </RDB_node>
        <attribute_node name=Abteilungen"/>
          <RDB_node>
            <table name=„Abteilungen"/>
            <column name=„Abtnr" type=„integer"/>
          </RDB_node>
        </attribute_node>
      ...

      </element_node>
    </root_node>
  </Xcollection>
</DAD>
```

Das Beispiel wurde aus [Sch03] entnommen.

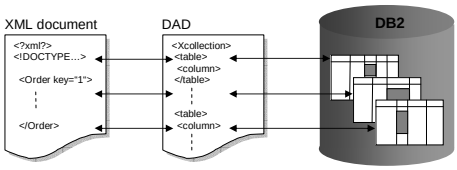

Abbildung 5: Speicherung als XML Collection nach [IBM04]

Wie bereits erwähnt handelt es sich um ein benutzerdefiniertes Mapping. Es stehen zwei unterschiedliche Arten zur Definition des Mappings zur Verfügung, auf der einen Seite das SQL Mapping und auf der anderen Seite die RDB-Knotenzuordnung[6]. Beim SQL Mapping werden mittels einer SQL-`SELECT`-Anweisung die Daten, die später das XML-Dokument bilden, aus den Tabellen ausgelesen. Mit dieser Variante ist somit nur die Komposition, also die Abbildung der relationalen Speicherung in ein XML-Dokument möglich. Das RDB-Node Mapping hingegen ermöglicht sowohl die Komposition als auch die Dekomposition von XML-Daten. Hierbei wird keine komplexe SQL-Anweisung benötigt, sondern es wird zu jedem Element oder Attribut die Quelle explizit definiert. Die genaue Definition einer DAD ist in [IBM04] zu finden. Nachteil dieser Speicherungsvariante ist jedoch, dass Kommentare und Verarbeitungsanweisungen verloren gehen.

Zerlegung von XML-Dokumenten (Speicherung) Die Inhalte eines XML-Dokumentes werden im Gegensatz zur XML Column-Technik nicht als Ganzes gespeichert, sondern zerlegt und auf eine oder mehrere Datenbanktabellen abgebildet. Dies bedeutet, dass die einzelnen Elementinhalte und Attributwerte als Datenbankattribute gespeichert werden. Der XML-Extender bietet zwei verschiedene *Stored Procedures* zum Zerlegen von XML-Dokumenten. Voraussetzung für das Anwenden einer solchen Stored Procedure ist die Definition einer DAD-Datei, die die Abbildung zwischen XML-Dokumenten und DB2 UDB Tabellenstrukturen beschreibt. Die Stored Procedure nutzt die DAD-Datei für die Zerlegung der Dokumente. Die Datei muss das RDB-Node Mapping nutzen, da mit dem SQL Mapping das Zerlegen nicht möglich ist. Die beiden verschiedenen Stored Procedures sind einerseits `dxxShredXML()` und andererseits `dxxInsertXML()`. Ein Unterschied der beiden Prozeduren liegt bei der beabsichtigten Häufigkeit der Änderungen der gespeicherten Daten. `dxxShredXML()` sollte bei seltenen Änderungen der Daten verwendet werden und `dxxInsertXML()` bei häufigen Änderungen. Vorteil der Speicherung von XML-Dokumenten mit Hilfe der `dxxShredXML()` ist, dass zur Nutzung dieser Prozedur keine Collection definiert und aktiviert werden muss. Die Nutzung von `dxxShredXML()` ist somit einfacher und nicht so aufwendig. Entsprechend der Anforderungen sollte dann die geeignete *Stored Procedure* zum Zerlegen und Abspeichern der Daten gewählt werden.

[6]RDB-Knotenzuordnung: rational databse node mapping

Erzeugung von XML-Dokumenten Hiermit ist die Generierung einer Menge von XML-Dokumenten aus relationalen Daten gemeint. Das neue XML-Dokument wird nach der Erzeugung in einer Tabelle abgelegt. Auch hier gibt es wie schon bei der Zerlegung verschiedene *Stored Procedures*. Zur Nutzung der Prozeduren wird ebenfalls eine DAD als Grundlage für die Abbildung benötigt. Diese DAD kann sowohl als SQL Mapping als auch als RDB-Node Mapping definiert sein. Die am häufigsten verwendete Stored Procedure ist `dxxGenXML()`, sie ist das Äquivalent zur `dxxShredXML()`-Prozedur. Neben `dxxGenXML()` gibt es noch die Prozedur `dxxRetrieveXML()` zur Erzeugung von XML-Dokumenten. Die Auswahl der entsprechenden Prozedur erfolgt ebenfalls nach der Häufigkeit der Datenänderungen. Bei der Generierung von Dokumenten ist neben den oben genannten Möglichkeiten auch die Speicherung in einen Wert vom Typ CLOB möglich. Dies erfolgt mit den Stored Procedures `dxxGenXMLCLOB()` und `dxxRetrieveXMLCLOB()`.

Anfragen und Updates Der XML-Extender bietet die Möglichkeit Daten der XML Collection zu ändern, zu löschen und auch nach ihnen zu suchen. Die Daten, die nach der Zerlegung in den Datenbanktabellen gespeichert sind, haben nichts mehr mit dem ursprünglichen XML-Dokument, aus dem sie stammen, zu tun. Für das Ändern, Löschen und Suchen der Daten stehen somit die für relationale Datenbanksysteme bekannten SQL-Anweisungen zur Verfügung.
Um die Effizienz der Anfragen zu unterstützen, können für die Daten der XML Collection Indexe erstellt werden. Diese Indexierung erfolgt mit herkömmlichen Datenbanktechniken auf den entsprechenden Spalten der einzelnen Tabellen, die die Daten enthalten.

Nähere Informationen und Erläuterungen zu einer XML Collection und den Möglichkeiten zur Anfrage und Manipulation der Daten dieser sind in [IBM04] nachzulesen.

3.1.2 DB2 UDB mit Text-Extender

Der Text-Extender [EDO+00] [IBM03] ist zuständig für das Hinzufügen von Information Retrieval-Funktionalität in das Datenbanksystem DB2 UDB von IBM. Information Retrieval bezeichnet dabei eine unscharfe Suche auf den Daten. Das bedeutet, es wird nicht nur genau das Gesuchte, sondern auch Variationen und Synonyme gefunden. Typisch für Information Retrieval sind vage Anfragen auf einer unsicheren Wissenbasis.

Die Funktionen des Text-Extenders reichen von einer Suche nach Dokumenten, die ein bestimmtes Wort enthalten, über eine positionsabhängige Suche, z.B. Wörter im gleichen Satz oder im gleichen Absatz, bis zur Suche nach ähnlich geschriebenen oder gesprochenen Worten.

Für die Unterstützung des Retrievals kann ein Text-Index angelegt werden. Beim Einfügen eines neuen Dokumentes werden die einzelnen Terme des Dokumentes für

den Index extrahiert. Welche Terme indexiert werden, hängt vom Typ des gewählten Indexes ab. Der Text-Extender unterstützt die folgenden Indexe:

- *Linguistischer Index*: Dieser Index basiert auf einem Wörterbuch und erlaubt verschiedene Varianten eines Wortes zu finden (Stammwortreduktion). Durch diese Methode wird die Anzahl der Worte und somit auch die Größe des Indexes minimiert.
- *Exakter Index*: Bei diesem Index wird genau das gefundene Wort gespeichert (sogar Unterscheidung zwischen Groß- und Kleinschreibung möglich). Somit wird das Indexing und Retrieval beschleunigt, aber der Index wird sehr groß.
- *N-Grams*: Hierbei handelt es sich nicht um einen wortbasierten, sondern um einen zeichenbasierten Index. Das bedeutet, es werden jeweils die ersten n Zeichen eines Wortes indexiert. Dieser Index ermöglicht somit die Suche nach Worten anhand der ersten n Zeichen.

Die Auswahl des Indexes ist wie gesehen von der jeweiligen Anwendung abhängig.

Der Text-Extender ermöglicht nicht nur das Durchsuchen von intern in der Datenbank gespeicherten Texten, sondern auch externer Dateien. Neben reinen Textdateien können auch strukturierte Dokumente wie HTML- oder XML-Dateien verarbeitet werden. In diesem Fall wird von strukturierter Suche gesprochen. Hierfür ist es erforderlich, die relevanten Markup-Tags bzw. Abschnitte des Dokuments, welche indexiert werden sollen, zu definieren. Ein Beispiel dafür ist in [EDO+00] zu finden.

Die am häufigsten verwendeten Funktionen des Text-Extender sind:

- *CONTAINS*: Anfrage, ob ein Dokument ein bestimmtes Suchargument enthält

  ```
  SELECT id
  FROM XMLFILE
  WHERE db2tx CONTAINS (hfile, 'model letter sections
  (/letter/subject) „introduction")
  ```

  Beispiel für die Suche nach dem Wort „introduction“ nur im Abschnitt „subject“ des Dokumentes
- *NO_OF_MATCHES*: Liefert die Anzahl der Treffer bei der Suche nach einem bestimmten Argument im Dokument
- *RANK*: Liefert eine Bewertung (Ranking) des Dokumentes für ein Suchargument (zwischen 0 und 1)

Mit dem Text-Extender ist nicht nur die Suche nach einem Argument möglich, sondern auch nach Verknüpfungen durch UND (&) und ODER (|) oder Negierungen (NOT).

3.1.3 Bewertung

In diesem Kapitel wurden zwei unterschiedliche Möglichkeiten vorgestellt, um XML-Dokumente in dem Datenbanksystem DB2 UDB abzuspeichern. Diese beiden Varianten stellt der XML-Extender als XML Column oder XML Collection zur Verfügung. Genaugenommen gibt es bei DB2 auch noch eine dritte Möglichkeit um XML-Dokumente abzuspeichern. Der Text-Extender bietet die Speicherung eines XML-Dokumentes als Volltext an. Auf diese Möglichkeit soll jedoch nicht weiter eingegangen werden. Im Rahmen dieser Arbeit soll der Text-Extender ergänzend für die Volltextsuche in XML-Dokumenten genutzt werden.

XML Column dient zur Speicherung eines gesamten XML-Dokumentes in seiner ursprünglichen Form. Diese Funktionalität ist vor allem zur Archivierung ganzer Dokumente, die nur selten geändert werden sollen, geeignet. Der Vorteil ist nämlich, dass die Dokumente intakt bleiben und beliebig oft verwendet werden können. Es gibt jedoch auch Nachteile bei diesem Verfahren, wie z.B. die Notwendigkeit von Seitentabellen für die Indexierung und die Einschränkungen beim Manipulieren der Daten. Weiterhin ist diese Variante nicht für das Zusammensetzen von Dokumenten aus gespeicherten Daten geeignet.
Bei XML Collection erfolgt eine Zerlegung des Dokumentes in die einzelnen Bestandteile und eine Speicherung dieser in relationalen Datenbankstrukturen. Diese Methode eignet sich zum Kreieren neuer Dokumente, wenn die Struktur der Dokumente nicht im Vordergrund steht. Außerdem werden bei dieser Speicherungsvariante häufige Manipulationen der Daten unterstützt und Indextechniken sind gut nutzbar. Nachteile sind jedoch die obligatorische Definition einer DAD und die Schwierigkeit Dokumente zu rekonstruieren.
Die Möglichkeit Anfragen an die Dokumente zu stellen oder Retrieval-Funktionen zu nutzen bieten beide Alternativen.

XML Collection hat jedoch einen entscheidenen Nachteil, wenn es um das Testen und Bewerten der Effizienz von verschieden strukturierten XML-Dokumenten geht. Die Zerlegung der Dokumente erfolgt entsprechend der Definition in der DAD. Es handelt sich also um ein benutzerdefiniertes Mapping. Durch diese Art der Zerlegung gehen die Auswirkungen unterschiedlicher Strukturen der XML-Dokumente auf die Effizienz verloren. Aus diesem Grund wird zum Testen und Bewerten der unterschiedlichen Dokumentstrukturen auf dem System DB2 Universal Database XML Column verwendet. Die Dokumente werden somit als Ganzes gespeichert und Anfragen und Änderungen erfolgen mit XPath-Ausdrücken eingebettet in SQL-Anweisungen. Die Indexierung zur Verbesserung der Effizienz ist, wie bereits erwähnt, auch möglich. Diese basiert jedoch ebenfalls auf einem benutzerdefinierten Mapping bei dem Strukturbesonderheiten verloren gehen. Aus diesem Grund wird auf eine Indexierung der Daten bei DB2 UDB verzichtet.

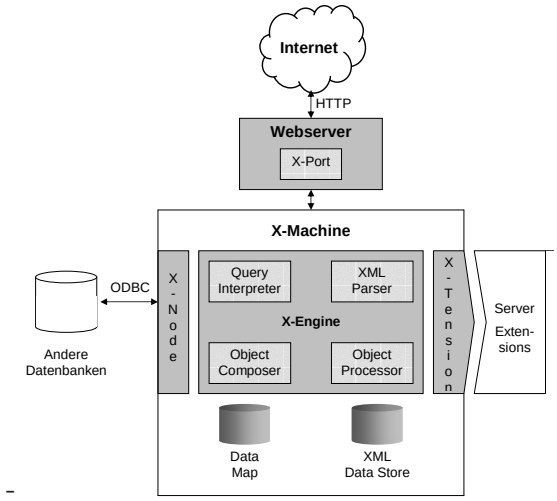

Abbildung 6: Schematischer Aufbau des Tamino XML Servers

3.2 Tamino XML Server

Der Tamino XML Server ist eine Entwicklung der Software AG Darmstadt. Der Name Tamino steht für ***T**ransaction **A**rchitecture for the **M**anagement of **IN**ternet **O**bjects*. Es handelt sich bei Tamino um eine native XML-Datenbank, also eine speziell für die Anforderungen von XML-Dokumenten entwickeltes Datenbanksystem.

3.2.1 Aufbau und Organisation

In der Abbildung 6 sind die einzelnen Komponenten des Tamino XML Servers dargestellt. Der Zugriff auf den Server erfolgt über das HTTP-Protokoll. Dazu müssen Webserver wie Apache um die Tamino-Komponente **X-Port** erweitert werden. Dieses Modul dient zur Weiterleitung der Anweisungen, die an Tamino gesendet werden, zur **X-Machine**-Komponente. Das Kernstück des XML Servers ist die **X-Engine**, ein Teil der X-Machine. Die X-Engine hat wiederum mehrere Komponenten: XML-Parser, Query Interpreter, Object Processor und Object Composer. Diese Komponenten dienen zur Verarbeitung und Speicherung von ankommenden XML-Objekten, DTD's und zur Bearbeitung von Anfragen, die als URL zur X-Machine gelangen. Gespeichert werden die XML-Dokumente und andere Objekte im **XML Data Store**. Neben dem XML Data Store gibt es noch die **Data Map**, die auch als Wissensbasis des Tamino XML Servers bezeichnet wird. Darin befinden sich unter anderem DTD's, XML-Metadaten und Stylesheets zur Abspeicherung von XML-Objekten oder zur Abbildung von Daten nach XML.

Letztlich gibt es noch die Komponenten **X-Node** und **X-Tension**, die Schnittstellen zur Anbindung an andere Systeme darstellen. Mit Hilfe dieser beiden Komponenten ist sowohl der Import von Daten aus anderen Systemen als auch der Export in andere Systeme möglich. X-Node stellt eine Verbindung zu anderen Datenbanksyste-

men auf und X-Tension sorgt für die Verbindung und Zusammenarbeit mit externen Programmen (**Server Extensions**).

3.2.2 Speicherung von XML-Dokumenten

Eine Tamino Datenbank besteht aus einer Vielzahl von **Collections**. Eine Collection ist dabei eine Sammlung von XML-Dokumenten. Bei der Abspeicherung eines XML-Dokumentes muss eine Zuordnung zu genau einer Collection erfolgen. Diese wird explizit beim Speichern angegeben. Innerhalb einer Collection werden die XML-Dokumente in **Doctypes** verwaltet. Ein Doctype ist die Menge der Dokumente einer Collection mit demselben Wurzelknotentyp. Beim Abspeichern wird ein Dokument somit entsprechend seines Root-Elementes einem Doctype zugewiesen.

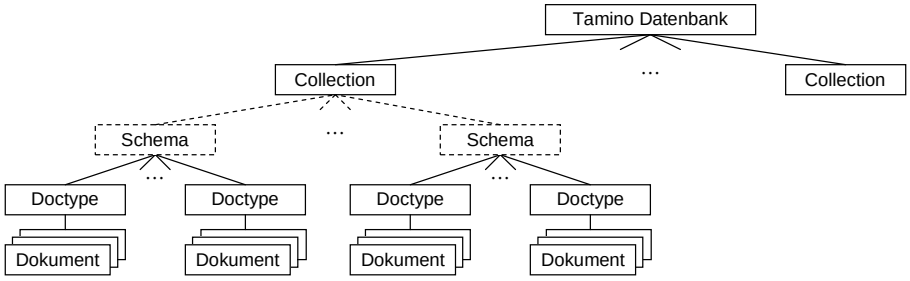

Abbildung 7: Organisation einer Tamino Datenbank nach [Sch03]

In Tamino besteht die Möglichkeit einem Doctype eine **schematische Beschreibung** zuzuordnen. XML-Dokumente müssen jedoch für die Speicherung in einer Datenbank keiner Schemabeschreibung unterliegen. Wenn ein Doctype aber durch ein Schema beschrieben wird, müssen alle zugehörigen Dokumente diesem Schema unterliegen. Für jede Collection sind mehrere XML-Schemabeschreibungen möglich und innerhalb dieser Collection kann es für mehrere Doctypes gelten. Diese Doctypes müssen dann explizit im Schema angegeben werden.

Im Zusammenhang mit der Schemadefinition unterstützt Tamino zwei unterschiedliche **Inhaltsmodelle**: das geschlossene und offene Inhaltsmodell. Bei der Verwendung des geschlossenen Inhaltsmodells muss ein Dokument bei der Validierung genau der Schemadefinition entsprechen. Beim Offenen ist es hingegen erlaubt, Dokumente mit zusätzlichen Elementen oder Attributen für die Speicherung zuzulassen. Die Schemadefinition für XML-Dokumente innerhalb einer Tamino-Datenbank ist Voraussetzung für die Indexierung der Daten.

3.2.3 Validierung

Wie bereits im vorherigen Kapitel angedeutet, gibt es bei Tamino die Möglichkeit XML-Dokumente mit und ohne Schemabeschreibung abzuspeichern. Zur Speicherung von Dokumenten ohne Schemabeschreibung existiert beim Tamino XML Server

eine vordefinierte Collection namens *ino:etc*. In diese Collection können alle möglichen Dokumente ohne Schemabeschreibung abgespeichert werden.

Zur Speicherung der XML-Dokumente mit Validierung gegen ein Schema muss eine eigene Collection angelegt werden, für die dann gleichzeitig ein Schema definiert werden kann. Somit erfolgt die Schemadefinition zeitgleich zum Anlegen der Collection. Das bedeutet, das Schema für die Collection wird direkt beim Anlegen dieser definiert. Sobald ein Dokument dann in eine Collection eingefügt werden soll, für die ein Schema definiert ist, wird es vor dem Einfügen gegen das Schema validiert und wenn keine Schemaverletzungen auftreten in die zugehörige Collection gespeichert. Wenn das Dokument gegen das Schema verstößt, werden entsprechende Fehlermeldungen generiert und das Dokument nicht in die Collection gespeichert.

3.2.4 Anfragen

Von der Software AG werden zwei verschiedene Anfragesprachen für Tamino XML Server angeboten. Dabei handelt es sich zum einen um den aktuellen Spezifikationsentwurf für XQuery vom W3C. Diese Sprache wird bei Tamino als XQuery 4 bezeichnet. Sie ist entsprechend den speziellen Anforderungen von XML-Dokumenten als Datenbankanfragesprache entwickelt worden. Daneben ist bei Tamino die eigens entwickelte Sprache X-Query [Sch03], die eine Erweiterung von XPath darstellt, als Anfragesprache verfügbar.
XQuery zeichnet sich durch die Verwendung von FLWR-Anfragen[7] aus. Mit Hilfe dieser Ausdrücke können die Anfragen auf den XML-Dokumenten genau spezifiziert werden. Zur Bedeutung der einzelnen Schlüsselwörter:

- Die `for`-Klausel bindet alle Werte, die durch den Ausdruck nach dem Schlüsselwort `in` angesprochen werden, in geordneter Reihenfolge an eine Variable.

- Der `where`-Ausdruck wird zur Einschränkung der Anfrage auf spezielle Daten verwendet. Er dient somit zum Definieren eines Filters.

- Mit Hilfe des `return`-Ausdrucks wird das Ergebnis der Anfrage konstruiert.

```
for $b in input()/bib/book
where $b/@year > 1994
return
<book>
{$b/@year}
{$b/title}
</book>
```
[8]

[7] FLWR: ausgesprochen engl. Flower; steht für die Schlüsselwörter `FOR-LET-WHERE-RETURN` vgl. bei relationalen DB: `SELECT-FROM-WHERE`-Ausdrücke

[8] Beispiele stammen größtenteils aus der Tamino XML Server Dokumentation [Tam03]

- Neben diesen drei bereits beschriebenen Ausdrücken gibt es noch `let`. `Let` bietet die Möglichkeit eines zusätzlichen Bindens von Werten an eine Variable. Im Gegensatz zum `for`-Ausdruck, der die Worte iterativ an eine Variable bindet, bindet der `let`-Ausdruck in einem Schritt.

```
for $b in input()/bib/book
let $y := $b/@year
where $y > 1994
return
<book>
{$y}
{$b/title}
</book>
```

Die Ausgabe der einzelnen Anfragen ist bei Tamino wohlgeformtes XML. Mit den FLWR-Ausdrücken sind neben den einfachen Anfragen auch Sortierungen, Verbünde oder Gruppierungen möglich, die aber für diese Arbeit nicht von Bedeutung sind.

Neben diesen genauen Anfragen gibt es beim Tamino XML Server auch die Möglichkeit mit Hilfe von Text-Retrieval-Funktionalität ungenaue Anfragen zu stellen. Für diese Anwendung werden die drei Funktionen `containsText`, `containsAdjacentText` und `containsNearText` zur Verfügung gestellt. Die `containsText`-Funktion dient zur Suche nach einer bestimmten Zeichenkette in einem Text.

```
for $a in input()/bib/book
where tf:containsText ($a/title, „XML")
return $a
```

Mit dieser Retrieval-Anfrage wird nach Büchern gesucht, die die Zeichenkette „XML“ im Tiel enthalten.

Die Funktion `containsAdjacentText` (abhängig von der Reihenfolge der Worte) und `containsNearText` (unabhängig von der Reihenfolge der Worte) berücksichtigen jeweils den Abstand zwischen zwei Worten bei der Suche.

Bei Tamino besteht außerdem die Möglichkeit Suchergebnisse im Text hervorzuheben, nach verschiedenen Formen eines Wortstammes zu suchen oder eine phonetische Suche durchzuführen [Tam03].

3.2.5 Updates

Zur Manipulation der XML-Daten wird bei Tamino ebenfalls XQuery 4 verwendet. Diese Sprache lässt Änderungsoperationen basierend auf XQuery zu. Die Manipulationsanweisungen ermöglichen das Einfügen, Ändern und Löschen von Knoten. Den Beginn einer solchen Anweisung stellt immer das Schlüsselwort `update` dar. Innerhalb einer Updateoperation können mehrere Änderungen spezifiziert werden.

Im Fall eines Widerspruches untereinander wird die gesamte Änderungsoperation abgewiesen.

Der `insert`-Ausdruck dient zum Einfügen neuer Knoten (Elemente oder Attribute) in ein bestehendes Dokument. Die Stelle, an der der Knoten eingefügt werden soll, kann näher bestimmt werden. Als Beispiel wird nachfolgend der Befehl zum Einfügen eines Buches mit dem Titel „XML“ in eine bestehende Bibliothek gezeigt:

```
update insert
<book>
  <title> XML </title>
</book>
into input()/bib
```

Der `delete`-Befehl löscht bestimmte Knoten aus einem Dokument, z.B. Löschen eines Buches mit dem Titel „XML“ aus dem Datenbestand:

```
update delete
into input()/bib/book/title[.=„XML"]
```

Der `rename`-Ausdruck ermöglicht das Umbenennen eines Knotens. Diese Funktion ist sowohl auf Elemente als auch auf Attribute anwendbar. Nachfolgend ist der Befehl zum Umbenennen des Elements „book“ in „buch“ zu sehen.

```
update rename
input()/bib/book as buch
```

Der `replace`-Ausdruck schließlich dient zum Ersetzen eines Knotens in einem XML-Dokument. Dieser Befehl ist ebenfalls für Attribute und Elemente möglich.

```
update replace
input()/bib/book/title[.="XML"]
with <title> XML - Extensible Markup Language </title>
```

Anstelle der o.a. Ausdrücke kann auch eine Variante der FLWR-Ausdrücke verwendet werden. Diese Variante sind FLWU-Ausdrücke (U-Update). So ist beispielsweise die Anweisung

```
update delete
into input()/bib/book/title[.="XML"]
```

äquivalent zu dem FLWU-Ausdruck

```
update for $a in input()/bib/book
do delete $a /title[.="XML"]
```

3.2.6 Indexierung

Um Anfragen auf den verschiedenen XML-Dokumenten effizient durchführen zu können, ist die Definition von Indexen erforderlich [Sch03]. Voraussetzung für die Indexierung bei Tamino XML Server ist eine Schemabeschreibung für die entsprechenden Dokumente. Bei Tamino gibt es drei verschiedene Indexarten, um die Suche auf XML-Dokumenten zu beschleunigen. Die Indexe müssen vom Nutzer selbst ausgewählt und angegeben werden. Die drei unterschiedlichen Indexe sind der Wertindex, Textindex und Strukturindex.

- *Wertindex:* Ein solcher wertbasierter Index ist aus dem Bereich der traditionellen Datenbanksysteme bereits bekannt. Er dient zur Beschleunigung beim Zugriff auf bestimmte Attributwerte und Elementinhalte. Es werden hierbei die vollständigen Werte der Elemente oder Attribute in den Index aufgenommen. Somit sind sowohl Punktanfragen (`//Name[.='Müller']`) als auch Bereichsanfragen (`//Preis[.<20 and .>10]`) effizienter zu bearbeiten. Die Organisationsform des Wertindexes ist bei Tamino ein B*-Baum und das Resultat der Suche ist der Dokumentknoten des entsprechenden Dokuments.

- *Textindex:* Der textbasierte Index unterstützt den Zugriff auf Texte. Der Index beinhaltet Worte eines Textes und deren Kombinationen. Dadurch wird es beispielsweise ermöglicht in diesem XML-Fragment

  ```
  <text>
    <bold>X</bold>ML ist super
  </text>
  ```

  den String 'XML' zu finden. Die Struktur des Indexes ist so angelegt, dass auch Nachbarschaftssuche und Suche nach Teilwörter effizienter wird. Das Ergebnis der Suche ist die zugehörige Dokumentreferenz.

- *Strukturindex:* Mit Hilfe des Strukturindexes werden spezielle Anfragen, die die Struktur eines Dokumentes betreffen, unterstützt. Inhalt des Strukturindexes sind die Pfade, die in den einzelnen Dokumenten eines Doctypes vorkommen. Tamino stellt zwei verschiedene Arten des Strukturindexes zur Verfügung: Auf der einen Seite der verdichtete Strukturindex (condensed), der Informationen über das Auftreten von bestimmten Pfaden in einem Doctype bereithält. Mit dieser Variante kann also nur festgestellt werden, ob ein Pfad in einem Dokument eines Doctypes auftritt oder nicht. Der volle Strukturindex (full) auf der anderen Seite liefert auch Informationen darüber, in welchen Dokumenten ein bestimmter Pfad auftritt. Auch bei diesem Index ist das Resultat der Suche die entsprechende Dokumentreferenz.

Alle drei Indexarten sind bei Tamino in einem Doctype frei kombinierbar. Nachdem die einzelnen Indexe angelegt wurden, werden sie durch das System automatisch aktualisiert.

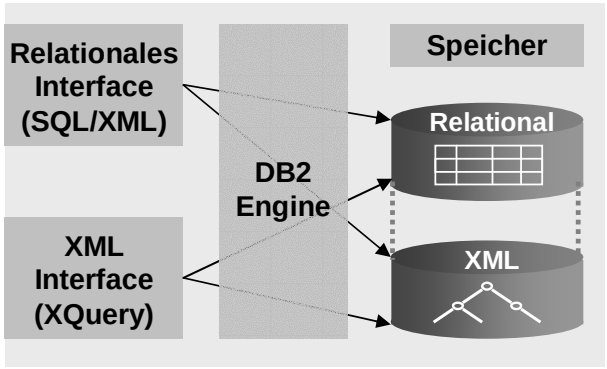

Abbildung 8: Aufbau des DB2 Viper Systems nach [NL05a]

3.2.7 Bewertung

Der Tamino XML Server ist ein natives Datenbanksystem zur Speicherung von XML-Dokumenten. Die Dokumente bleiben dabei aus Sicht des Nutzers intakt. Eine eventuelle interne Zerlegung der Dokumente bleibt dem Nutzer verborgen.

Mappingvorgänge der Dokumentinhalte, wie sie bei DB2 nötig waren, entfallen bei Tamino komplett. Es wird ein direkter Zugriff auf die Daten mittels der Anfrage- und Manipulationssprache XQuery gewährleistet. Diese speziell für XML entwickelte Sprache bietet verschiedenste Möglichkeiten beim Anfragen und Ändern der Dokumente [siehe [Tam03] und Kapitel 3.2.4 und 3.2.5]. Die Definition der Indexe für die entsprechenden Elemente und Attribute erfolgt bei Tamino in der Schemabeschreibung. Beim Einfügen und Anfragen der Dokumente werden diese vom System automatisch aktualisiert und genutzt. Tamino ermöglicht somit vergleichende Tests mit und ohne Indexierung der Daten.

Für die Erleichterung der Arbeit mit dem Tamino XML Server stehen noch zahlreiche Tools zur Verfügung. So kann beispielsweise der Schema Editor zur Definition und Änderung von Schemabeschreibungen und der Tamino Manager zur Administration genutzt werden.

3.3 IBM DB2 Viper (DB2 V9.1)

DB2 Viper ist das dritte für die Tests verwendete Datenbanksystem. Bei der IBM Datenbank handelt es sich um die Weiterentwicklung von DB2 Version 8.2. Aus diesem Grund ist DB2 Viper gleichzusetzen mit DB2 Version 9.1. „Viper“ ist der interne Codename von DB2 UDB Version 9 und wird in dieser Arbeit zur besseren Unterscheidung von Version 8 verwendet. Da es sich um eine Weiterentwicklung der UDB DB2 handelt, sind bekannte Funktionalitäten aus vorherigen Versionen, wie sie auch in Kapitel 3.1 bereits vorgestellt wurden, erhalten geblieben. Der schematische Aufbau des DB2 Viper Datenbanksystems ist in Abbildung 8 zu sehen.

An dieser Stelle wird darauf hingewiesen, dass sich das DB2 Viper Programm noch in der „closed-beta-Phase“ befindet. Das bedeutet, das Datenbanksystem ist noch nicht für die Öffentlichkeit zugänglich oder erwerbbar und viele Informationen bezüglich der DB2 Viper Version sind noch nicht veröffentlicht oder IBM vertraulich zu behandeln. Hieraus resultiert eine kürzere und nicht so detailierte Beschreibung des Datenbanksystems.

Der Unterschied zwischen vorherigen Versionen und der Version 9.1 liegt hauptsächlich bei den Möglichkeiten zur Speicherung von XML-Dokumenten. Bei DB2 Viper ist ein neuer Datentyp zur Speicherung von XML-Dokumenten hinzugekommen. Dieser neue Datentyp erlaubt, wohlgeformte XML-Dokumente in ihrer nativen, hierarchischen Struktur abzuspeichern. Der wesentliche Unterschied zu der in Abschnitt **??** beschriebenen Möglichkeit benutzerdefinierte Datentypen (UDFs) zu definieren besteht darin, dass für diesen Datentyp auch intern spezielle Datenstrukturen zur Speicherung der XML-Daten existieren (vgl. Abbildung 8).

Im den folgenden Abschnitten werden die Möglichkeiten zur Speicherung, zum Anfragen und Ändern sowie zum Indexieren von XML-Dokumenten mit DB2 Viper genauer erläutert.

3.3.1 Speicherung von XML-Dokumenten

Durch die Erweiterung des IBM DB2 Datenbanksystems um einen speziellen Datentyp zur Speicherung von XML-Dokumenten können solche Dokumente in ihrer ursprünglichen, hierarchischen (nativen) Form abgespeichert werden. Der neue Datentyp heißt `XML` und dient der Definition von Spalten einer Tabelle zur Speicherung von XML-Dokumenten. Voraussetzung für die Speicherung ist die Wohlgeformtheit der Dokumente. Mit Hilfe des XML-Datentyps ist es möglich sowohl relationale als auch XML-Daten in ihrer nativen Form in einer Tabelle zu Speichern und gemeinsam zu verarbeiten (z.B. anzufragen).

Um eine Datenbank zur Speicherung von XML-Daten nutzen zu können, muss diese zunächst als Unicode UTF-8 Datenbank angelegt werden. Datenbanken, die nicht als Unicode Datenbank angelegt wurden, unterstützen die Speicherung von XML-Daten nicht. Der Befehl zum Anlegen einer solchen Datenbank lautet:

```
CREATE DATABASE dbname USING CODESET UTF-8 TERRITORY US
```

Wie bereits erwähnt können Tabellen einer UTF-8 Datenbank sowohl relationale als auch XML-Spalten enthalten. Ein Beispiel für die Definition einer Tabelle mit einer relationalen ID-Spalte und einer Spalte für XML-Dokumente sieht folgendermaßen aus.

```
CREATE TABLE tablename (id integer, documents XML)
```

Die XML-Spalte *documents* dient in dieser Tabelle zum Speichern von XML-Dokumenten.

3.3.2 Validierung

Zum DB2 Viper Datenbanksystem gehört ein XML Schema Repository (XSR). Dieses Repository dient zur Speicherung von XML Schemadefinitionen. Gegen dort abgespeicherte XML Schemata können Dokumente beim Einfügen validiert werden. Das Speichern von XML Schemata in das Repository erfolgt mit Hilfe folgender Befehle.

```
REGISTER XMLSCHEMA 'http://localhost/myschema.xsd'
FROM 'file://C:/schema.xsd'
AS schemaname
```

Dabei wird zunächst der Pfad angegeben, in dem sich das XML Schema (xsd-Datei) befindet und dem Schema wird zur Speicherung im Repository ein Name zugeordnet. Optional können weitere zugehörige Schemabeschreibungen hinzugefügt werden. Abschließend wird mit dem `COMPLETE`-Befehl

```
COMPLETE XMLSCHEMA schemaname
```

der Speichervorgang für ein XML Schema abgeschlossen. Nachdem das Schema im XML Schema Repository von DB2 Viper abgelegt wurde, können XML-Dokumente beim Abspeichern gegen dieses validiert werden. Der Befehl zum Validieren von Dokumenten lautet `XMLVALIDATE`. Dieser Befehl wird eingebettet in den `INSERT INTO`-Befehl.

```
INSERT INTO table (spalte) VALUES (XMLVALIDATE (? ACCORDING TO
XMLSCHEMA ID schemaname))
```

Neben den üblichen Bestandteilen eines `INSERT INTO`-Kommandos enthält der Ausdruck `XMLVALIDATE` noch die Angabe, gegen welches Schema das Dokument validiert werden soll. Mit Hilfe dieses Kommandos wird ein XML-Dokument vor der Speicherung in eine Tabelle einer Datenbank zunächst gegen das angegebene Schema validiert und nur bei Schemakonformität gespeichert. Wenn ein Dokument nicht der Schemadefinition entspricht, wird eine entsprechende Fehlermeldung ausgegeben.

3.3.3 Anfragen und Updates

Für das Anfragen von XML-Daten in einer Tabelle stehen bei DB2 Viper verschiedene Möglichkeiten zur Verfügung. Die vorrangige Anfragesprache für XML-Daten ist XQuery. DB2 Viper unterstützt das Anfragen mit ausschließlich XQuery-Ausdrücken oder auschließlich SQL-Ausdrücken. Neben diesen beiden Varianten besteht aber auch die Möglichkeit XQuery-Anfragen eingebettet in SQL oder XQuery-Anfragen, die SQL-Ausdrücke beinhalten, zu verwenden. Da für diese Arbeit vor allem die XQuery-Anfragen eingebettet in SQL-Ausdrücken benötigt werden, soll auf diese genauer eingegangen werden. Mit Hilfe dieser Variante ist es möglich, jegliche SQL-Anfragen aber auch XQuery-Anfragen zu formulieren. Sie erlaubt somit das Ausführen von XQuery in einem SQL-Kontext. Der Hauptbestandteil einer eingebetteten XQuery-Anfrage ist die `XMLQUERY`-Funktion.

```
SELECT XMLQUERY ('$doc/VOC/voyage/master' PASSING DOCUMENTS AS
„doc") FROM tablename
```

Der Rückgabewert der `XMLQUERY`-Funktion, die für jede Zeile der Tabelle ausgeführt wird, ist eine XML-Sequenz. Diese Sequenz enthält das Ergebnis der Anfrage. Sie kann sowohl leer als auch ein oder mehrere Ergebnisse enthalten.

Zum Ändern und Löschen von Dokumenten stehen die von DB2 V8 bekannten Varianten (vgl. Kapitel 3.1.1.1) zur Verfügung. Das Ändern eines XML-Dokumentes in einer XML-Spalte erfolgt mit Hilfe des `UPDATE` SQL-Kommandos. Hinter der `SET`-Klausel muss sich ein wohlgeformtes XML-Dokument verbergen. Die `WHERE`-Klausel dient zum Ansprechen spezieller Zeilen. Wie auch bei DB2 V8 wird beim Ändern eines Dokumentes das ganze Dokument ersetzt. Zum Löschen eines Dokumentes aus einer XML-Spalte wird das `DELETE` SQL-Kommando verwendet.

3.3.4 Indexierung

Indexe über XML-Spalten dienen zur Verbesserung der Effizienz von Anfragen auf die indexierten Werte der in einer XML-Spalte gespeicherten Dokumente. Im Gegensatz zu relationalen Indexen, bei denen Schlüssel über eine oder mehrere Spalten angelegt werden, benötigt ein Index auf XML-Spalten spezielle `XMLPATTERN`-Ausdrücke. Diese dienen zur Beschreibung der zu indexierenden Werten in XML-Dokumenten (Pfadausdrücke). Zum Anlegen und Löschen von Indexen werden `CREATE INDEX` und `DROP INDEX` verwendet. Zu beachten bei der Indexierung in DB2 Viper ist, dass Schlüsselwörter eine andere Bedeutung haben können, als bei den aus vorherigen DB2 Versionen bekannten, relationalen `CREATE INDEX`-Anweisungen. Die Syntax für einen solchen Index über eine XML-Spalte kann folgendermaßen aussehen.

```
CREATE INDEX indexname on table(spalte)
GENERATE KEY USING XMLPATTERN 'pfadausdruck'
AS SQL DOUBLE
```

Mit diesem Befehl wird ein Index auf den angegebenen Pfad der Dokumente der entsprechenden Spalte angelegt. Die Indexwerte werden als *Double*-Wert abgespeichert.

3.3.5 Bewertung

Da es sich bei DB2 Viper um eine Weiterentwicklung von DB2 V8.2 handelt, stimmen beide Systeme in ihrer Funktionalität weitgehend überein. Der Vorteil von Viper ist Möglichkeit der nativen Speicherung von XML-Dokumenten in speziell definierten XML-Spalten (Datentyp `XML`). Mit dieser Funktionalität ist natürlich auch die Unterstützung von XQuery verbunden. Aber auch das Be- und Verarbeiten von herkömmlichen relationalen Daten ist mit DB2 Viper weiterhin möglich. Somit ist das Anfragen, Manipulieren und Indexieren sowohl von relationalen als auch XML-Daten möglich. Es ist hervorzuheben, dass das Verwenden eines speziellen

Extenders, wie es noch bei DB2 V8.2 zur Verarbeitung von XML-Daten üblich war (XML-Extender vgl. Kapitel 3.1.1), nicht mehr notwendig ist. Auch sollte DB2 Viper durch die Integration der nativen Speicherung von XML-Dokumenten im Gegensatz zu DB2 V8.2 eine Performanzverbesserung bei der Verarbeitung dieser aufweisen. Mit dieser Erweiterung verschmelzen bei DB2 Viper relationale Ansätze mit XML-Ansätzen und Viper ist mit nativen XML-Datenbanksystemen wie Tamino von der Software AG vergleichbar.

Teil II
Konzept und Test

4 Entwicklung und Implementierung der Tests

In diesem Kapitel wird zunächst das zum Testen verwendete XML-Dokument vorgestellt und näher beschrieben. Aufbauend auf diesem Dokument werden anschließend die unterschiedlichen Testszenarien erläutert und die dafür notwendigen Änderungen im Dokument gezeigt. Weiterhin werden auch die systemabhängigen Anfrage- und Updateoperationen abgeleitet. Zum Abschluss des vierten Kapitels werden noch einige Informationen zur Implementierung der Tests auf den beiden verschiedenen Testsystemen gegeben.

4.1 XML-Dokument

Das ausgewählte XML-Dokument bildet die Grundlage für alle folgenden Tests der Effizienz. Bei dem zum Testen verwendeten XML-Dokumentes[9] handelt es sich um ein Fahrtenbuch für Schiffe vorwiegend aus dem 17. und 18. Jahrhundert. Der Inhalt dieses Fahrtenbuches wurde auf ein XML-Dokument übertragen.

```
<?xml version="1.0" encoding=UTF-8"?>
<VOC>
  <voyage>
    <number>4408</number>
    <trip>3</trip>
    <leftpage>
      <boatname>BRESLAU</boatname>
      <master>Jan Kornelis Roos</master>
      <tonnage>1150</tonnage>
      <hired>1774</hired>
      <chamber>Z</chamber>
      <departure>1783-02-15</departure>
      <harbour>Rammekens</harbour>
      <callatcape>
        <arrival>1783-05-27</arrival>
        <departure>1783-06-12</departure>
      </callatcape>
      <destination>
        <arrival>1783-08-06</arrival>
        <harbour>Batavia</harbour>
```

[9] Das Dokument voc.xml stammt von Monet DB (http://monetdb.cwi.nl). VOC steht für Vereenigde geoctrooieerde Oostindische Compagnie (aus dem Niederländischen: The Dutch East Indian Company). Das Dokument basiert auf Daten, die in dem Buch „Dutch-Asiatic Shipping in the 17th and 18th Centuries“ von J.R. Bruijn, F.S. Gaastra und I. Schaar veröffentlicht wurden und beinhaltet eine Auflistung der Fahrten von Schiffen in den „Osten”. Die Liste enthält detailierte Informationen von über 8000 Fahrten.

```
      </destination>
    </leftpage>
    <rightpage>
      <onboard>
        <total>
          <one>124</one>
          <two>1</two>
          <three>0</three>
        </total>
      </onboard>
      <particulars>
        <next>8101</next>
        The BRESLAU flew the Prussian flag. After having
        arrived at Batavia the ship sailed on to China,
        where it cast anchor on 11-10-1783.
      </particulars>
    </rightpage>
  </voyage>
...
</VOC>
```

Das Root-Element ist *VOC* und die Daten zu den einzelnen Fahrten sind in einer Vielzahl von *voyage*[10]-Elementen enthalten. Jedes einzelne *voyage*-Element entspricht somit einem Datensatz des Dokumentes. Weiter unterteilt und strukturiert wird das Dokument in die linke und rechte Seite des ursprünglichen Fahrtenbuches. Auf der linken Seite sind einige Informationen zum Schiff und der Fahrtroute wie beispielsweise das Element „*master*“ und „*number*“ zu finden und auf der rechten Seite Informationen zur Besatzung und zur nächsten Fahrt des Schiffes.
Insgesamt besteht das Dokument aus 8076 verschiedenen Datensätzen mit einer Gesamtgröße von 5,74 MByte (6.023.823 Byte). Jeder einzelne Datensatz vertritt dabei eine Fahrt eines Schiffes mit den oben genannten Einzelheiten. Zu beachten ist hierbei, dass nicht alle Tags zwingend in jedem Datensatz auftreten müssen. So enthält beispielsweise nicht jeder Datensatz den Namen des Kapitäns (vgl. *master*) des Schiffes. Im Allgemeinen ist die Struktur aller Datensätze jedoch gleich und es fehlen teilweise nur einzelne Tags.
Der Inhalt des gesamten Dokumentes ist sowohl daten- als auch dokumentorientiert. Dies bedeutet, dass neben strukturierten Daten (vor allem die Einzelheiten unter *leftpage*) auch Informationen in textueller Form vorliegen. Diese sind vor allem innerhalb des *particulars*-Tags zu finden.
Im nächsten Abschnitt werden die verschiedenen Testszenarien und auch die damit verbundenen Modifikationen des Dokumentes beschrieben.

[10] aus dem Englischen: Seereise, Seefahrt

4.2 Szenarien

In diesem Abschnitt werden die einzelnen Tests mit ihren zugehörigen Dokumenten und Anfragen vorgestellt. Dabei wird vor allem auf die für die speziellen Tests vorgenommenen Veränderungen des Originaldokumentes (vgl. Abschnitt 4.1) eingegangen. Bei den verschiedenen Tests handelt es sich im Einzelnen um: Elemente vs. Attribute, Länge der Tagnamen, Einfluss von Kommentaren, Schachtelungstiefe, verteilte vs. kompakte Speicherung und IDREF's vs. KEYREF's. Neben dem Testen für das Anfragen der XML-Dokumente und das Einfügen ohne Validierung wird auch jeweils ein Test für das Einfügen mit Validierung gegen ein XML-Schema zur Bewertung herangezogen.

Die einzelnen Tests werden für unterschiedliche Größen der Dokumente bzw. eine unterschiedliche Anzahl an *voyage*-Einträgen durchgeführt. Es beginnt mit einem einzigen *voyage*-Eintrag im Dokument und geht über 8, 127, 505, 1010, 2019, 3028, 4038, 5048, 6057 und 7067 bis hin zu 8076 Einträgen. Mit Hilfe dieses Verfahrens können auch Auswirkungen der unterschiedlichen Dokumentstrukturen in Abhängigkeit von der Dokumentgröße untersucht werden. Außerdem werden die einzelnen `INSERT` und `QUERY`-Operationen für jedes Dokument 12 mal ausgeführt, damit später der kleinste und größte Wert (Ausreißer) unberücksichtigt bleiben können und trotzdem noch ein repräsentativer Mittelwert von jeweils 10 Testergebnissen gebildet werden kann.

Bei den Anfragen auf die Dokumente und bei der Speicherung mit Validierung wird zwischen einerseits DB2 V8 und andererseits Tamino und DB2 V9.1 als Testsystem unterschieden. Auf DB2 V8 werden lediglich XPath-Anfragen durchgeführt, da das System nur diese Art der Anfragen unterstützt. Eine Unterstützung von XQuery ist erst mit der neuen Version (DB2 V9.1) hinzugekommen. Auf Tamino und Viper können somit im Gegensatz zu DB2 V8 auch komplexe XQuery-Anfragen ausgeführt werden. Bei den beiden durchgeführten XQuery-Anfragen handelt es sich zum einen um eine `COUNT`- und zum anderen um eine `FLWR`-Anfrage. Bei der `FLWR`-Anfrage ist zu beachten, dass die in der `WHERE`-Klausel selektierte Dokumentmenge maximal 34 Dokumente umfasst und die eigentliche Anfrage dann nur auf diese Menge angewendet werden muss. Auch die Validierung der Dokumente gegen ein Schema wird nur auf Viper und Tamino getestet. Auch das ist mit der fehlenden Unterstützung dieser Funktionalität bei DB2 V8 zu erklären. Diese Version bietet die Validierung gegen ein XML-Schema nicht an.

Tests mit Indexierung der angefragten Werte werden auf keinem der drei Systeme durchgeführt, da weder DB2 V8 noch Viper oder Tamino die benötigten Funktionen zur Verfügung stellen. Bei DB2 V8 wird zum Indexieren von Werten (Atrribute oder Elemente) in XML-Dokumenten ein benutzerdefiniertes Mapping benötigt (vgl. Kapitel 3.1.1.1). Da durch ein solches benutzerdefiniertes Mapping jegliche Strukturierung der Dokumente verloren geht, kann auf DB2 V8 keine Indexierung der Werte zum Testen vorgenommen werden. Für Viper entfällt das Testen mit Indexen, weil bei der Auswertung der `XMLQUERY`-Funktion eventuell vorhandene Indexe

nicht verwendet werden. Indexe werden vor allem bei der Funktion `XMLEXIST` verwendet, die für einen XPath-Ausdruck einen boolschen Wert mit der Aussage, ob sich das betreffende Dokument entsprechend dem Ausdruck qualifiziert, zurückliefert. Die `XMLEXIST`-Funktion kann somit innerhalb der `WHERE`-KLausel einer SQL-Anfrage verwendet werden, um nur einen Teil der Dokumente zu selektieren. Auf dieser reduzierten Dokumentmenge wird dann aber die Anfrage unabhängig von jeglichen Indexen ausgeführt. Für das Testen der unterschiedlichen Dokumentstrukturen ist die `XMLEXIST`- Funktion somit nicht nutzbar, da sie aufgrund der Selektiereigenschaften nur bei mehreren angefragten Dokumenten in einer Datenbank sinnvoll ist. Indexe sind somit nur für Anfragen mit Selektierung hilfreich und machen innerhalb eines Dokumentes beim Anfragen keinen Unterschied. Im Rahmen dierser Arbeit durchgeführte Tests deuten daraufhin, dass bei Tamino Indexe in ähnlicher Weise verwendet werden. Bei der Verwendung von Indexen auf bestimmten angefragten Attributen oder Elementen in einem großen Dokument können somit keine Performanzverbesserungen gegenüber den Anfragen ohne Indexe festgestellt werden. Das weist daraufhin, dass der Index innerhalb eines Dokumentes nicht zur Beschleunigung der Anfragen verwendet wird. Verbesserungen durch Indexe können nur erzielt werden, wenn viele Dokumente in einer Collection abgelegt sind und der Index dann dazu dient, die zutreffenden Dokumente auszuwählen. Vermutlich arbeitet Tamino intern ähnlich wie DB2 Viper und der Index wird nur für die Selektion zutreffender Dokumente genutzt, aber nicht für die Abarbeitung der Query selbst innerhalb eines Dokumentes.

Aus den oben genannten Gründen ist es im Rahmen dieser Studienarbeit nicht sinnvoll Performanztests mit Indexen durchzuführen, weil die eigentlichen Anfragen ohne die Nutzung der Indexe ausgeführt werden. Es ist aber festzuhalten, dass es für die Performanz bei vielen Dokumenten in einer Datenbank durchaus empfehlenswert ist Indexe zu verwenden, aber beim Testen der Struktureigenschaften von XML-Dokumenten sind sie nicht hilfreich.

4.2.1 Elemente vs. Attribute

Dieser Test dient dazu herauszufinden, inwiefern die Effizienz bei der Verarbeitung von Dokumenten beeinflusst wird, wenn ein Datum als Attribut oder als einzelnes Element modelliert ist. Da es in vielen Fällen möglich ist, ein Element als Attribut oder umgekehrt darzustellen, wäre ein Unterschied bei der Verarbeitung sehr interessant.

Das Originaldokument (vgl. Abschnitt 4.1), bei dem der Wert als einzelnes Element modelliert ist, dient als Grundlage für den Test. Für die Durchführung des Tests wurde das Element *master* ausgewählt. Für den zweiten Teil des Tests wird der Inhalt des Elements *master* als Attribut des übergeordneten *voyage*-Elements dargestellt (vgl. Abbildung 9).

```
<?xml version="1.0" encoding="UTF-8"?>
<VOC>
  <voyage>
    <number>4408</number>
    <trip>3</trip>
    <leftpage>
      <boatname>BRESLAU</boatname>
      <master>Jan Kornelis Roos</master>
      <tonnage>1150</tonnage>
      …
</VOC>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<VOC>
  <voyage master="Jan Kornelis Roos">
    <number>4408</number>
    <trip>3</trip>
    <leftpage>
      <boatname>BRESLAU</boatname>
      <tonnage>1150</tonnage>
      …
</VOC>
```

Abbildung 9: Vergleich der Dokumentstrukturen: Elemente (links) und Attribute (rechts)

Zum Testen der Performanz der verschiedenen Strukturen werden neben der Speicherung mit und ohne Validierung gegen ein XML-Dokument unterschiedliche Anfragen verwendet. Da DB2 V8 keine XQuery-Ausdrücke unterstützt, wird hier lediglich ein Test bezüglich des Pfades durchgeführt.
Beim Test wird auf der einen Seite die Zeit für das Anfragen des Elements *master*

```
'input()/VOC/voyage/master'
```

und auf der anderen Seite für das Anfragen des Attributs *master* gemessen.

```
'input()/VOC/voyage/@master'
```

Neben dieser Anfrage werden bei Tamino und Viper noch zwei weitere XQuery-Anfragen gestellt. Bei der einen handelt es sich um ein `COUNT`

```
'COUNT (input()/VOC/voyage/master)'
bzw.
'COUNT (input()/VOC/voyage/@master)'
```

[11]

und bei der anderen um eine Anfrage mit `FOR-LET-RETURN`-Klausel.

```
FOR $t IN input()/VOC/voyage[leftpage/boatname=„ZEELANDIA"]
LET $l:=$t/leftpage
RETURN (<result> {$l/departure/text()}
{$t/@master bzw. master/text()} </result>)
```

Die Auswertung dieses Tests auf den drei verschiedenen DBS erfolgt in Kapitel 5.2.

4.2.2 Länge der Elementnamen

Dieser Test befasst sich mit der Länge der Elementnamen. Dabei wird untersucht, wie groß der Einfluss der Länge auf die Effizienz der Anfragen ist. Um ein möglichst aussagekräftiges Ergebnis zu erhalten, werden drei verschiedene Fälle untersucht. Bei dem für diesen Test ausgewählten Element handelt es sich um das Element

[11] Beispiele für Tamino angeführt

number des Dokumentes. Wie bereits erwähnt, besteht der Test aus drei Teilen, die sich jeweils in der Länge der Elementnamen unterscheiden. Die verschiedenen Namen sind: *nr*, *number* und *number_of_the_voyage*. Für den Test mit *number* als Tagname kann erneut das Originaldokument verwendet werden. Zum Testen mit *nr* und *number_of_the_voyage* als Tagname muss das *number*-Tag entsprechend modifiziert werden. Um die aus der Verkürzung von *number* auf *nr* resultierende kleinere Dokumentgröße zu kompensieren, wird das darunterliegende *trip* in *trippppp* umgewandelt. Für die Veränderung bei *number_of_the_vogaye* wird *leftpage* auf *l* und *boatname* auf *b* verkürzt. Die Dokumentgröße wird kompensiert, damit später bei den Tests keine Unterschiede in den für das Speichern und Anfragen benötigten Zeiten aufgrund unterschiedlicher Dokumentgrößen auftreten.

```
<VOC>
  <voyage>
    <number>4408</number>
    <trip>3</trip>
    <leftpage>
      <boatname>BRESLAU</boatname>
      …
</VOC>
```

```
<VOC>
  <voyage>
    <nr>4408</nr>
    <trippppp>3</trippppp>
    <leftpage>
      <boatname>BRESLAU</boatname>
      …
</VOC>
```

```
<VOC>
  <voyage>
    <number_of_the_voyage>4408</number_of_the_voyage>
    <trip>3</trip>
    <l>
      <b>BRESLAU</b>
      …
</VOC>
```

Abbildung 10: Vergleich der Dokumentstrukturen für die Länge der Elementnamen

Wie auch bei dem vorhergehenden Test werden abgesehen von DB2 V8 mehrere Anfragen zum Testen der Effizienz verwendet. Zum einen die Anfrage nach dem Inhalt des *number*-Tags (bei DB2 V8 nur diese Anfrage)

```
'input()/VOC/voyage/number'
```

und zum anderen wieder eine `COUNT`- und eine `FOR-RETURN`-Anfrage. Die `COUNT`-Anfrage wird jeweils über die entsprechenden Elementnamen formuliert. Es folgt das Beispiel für *nr*.

```
'COUNT (input()/VOC/voyage/nr)'
```

Die `FOR-RETURN`-Anfrage sieht folgendermaßen aus:

```
FOR $t IN input()/VOC/voyage[leftpage/boatname=„ZEELANDIA"]
LET $l:=$t/leftpage
RETURN (<result> {$t/nr/text()} </result>)
```

Informationen zur Auswertung dieses Tests sind in Kapitel 5.3 zu finden.

4.2.3 Einfluss von Kommentaren

Bei diesem Test wird der Einfluss von Kommentaren in einem XML-Dokument auf die Effizienz untersucht. Dafür wird zum einen das Originaldokument, welches keine Kommentare enthält, verwendet und zum anderen das Originaldokument angereichert mit Kommentaren. Das veränderte Dokument enthält in jedem Datensatz einen Kommentar hinter dem Element *number* und einen weiteren hinter dem Element *boatname*. Aufgrund der Anreicherung des Dokuments mit Kommentaren ist das veränderte Dokument 7,5 MB groß im Gegensatz zu 5,8 MB.

```
…
<voyage>
  <number>4408</number>
  <trip>3</trip>
  <leftpage>
    <boatname>BRESLAU</boatname>
    <master>Jan Kornelis Roos</master>
…
```

```
…
<voyage>
  <number>4408</number>
  <!-- Das ist ein Kommentar… -->
  <trip>3</trip>
  <leftpage>
    <boatname>BRESLAU</boatname>
    <!-- Das ist ein Kommentar… -->
  <master>Jan Kornelis Roos</master>
…
```

Abbildung 11: Dokumentstruktur ohne (links) und mit Kommentaren (rechts)

Die Anfragen sind für die beiden Dokumente indentisch. Es wird jeweils eine Anfrage nach dem Inhalt des Elements *master* durchgeführt und bei Tamino und Viper eine `COUNT` und eine `FOR-LET-RETURN` XQuery-Anfrage über dieses Element.
Die Ergebnisse dieses Tests sind in Kapitel 5.4 nachzulesen.

4.2.4 Schachtelungstiefe der Dokumente

Dieser Test vergleicht eine flache Struktur eines XML-Dokumentes mit einer verschachtelten Struktur eine Dokumentes. Allerdings ist die Hierarchietiefe für diesen Test begrenzt, da in beiden Dokumenten der gleiche Inhalt bei vergleichbarer Dokumentgröße dargestellt werden muss und somit das Dokument nicht beliebig tief geschachtelt werden kann. Der Ausgangspunkt des Tests ist erneut das Originaldokument mit den einzelnen Schiffsrouten. Dieses Dokument hat vier Level und vertritt somit die tiefe Hierarchie. Dieses Dokument wird umgewandelt in ein Dokument mit größtenteils zwei Level. Zwei Level sind notwendig, da das Dokument mehrere *voyage*-Elemente enthält und diese bereits ein Level darstellen. Die Informationen zu den einzelnen Routen stellen dann das zweite Level dar. An dieser Stelle wird aus Gründen der Übersichtlichkeit nur das modifizierte Dokument dargestellt. Dieses kann mit dem Originaldokument aus Kapitel 4.1 verglichen werden. Um den Größenverlust des Dokumentes nach der Entschachtelung zu kompensieren, müssen noch diverse Elementnamen in ihrer Länge verändert werden. Auch dies ist in der Abbildung 12 zu sehen.

Die Anfragen werden jeweils auf das *arrival*-Element ausgeführt. Bei dem Dokument mit der tiefen Hierarchie befindet sich das Element im vierten Level und bei

```
<?xml version="1.0" encoding="UTF-8" ?>
<VOC>
  <voyage>
    <number>4408</number>
    <trip>3</trip>
    <leftpage_boatname>BRESLAU</leftpage_boatname>
    <master>Jan Kornelis Roos</master>
    <tonnage>1150</tonnage>
    <hired>1774</hired>
    <yard>A</yard>
    <chamber>Z</chamber>
    <departure>1783-02-15</departure>
    <harbour>Rammekens</harbour>
    <arrival>1783-05-27</arrival>
    <c_departure>1783-06-12</c_departure>
    <destination_arrival>1783-08-06</destination_arrival>
    <destination_harbour>Batavia</destination_harbour>
    <onboard_total>
      <one>124</one>
      <two>1</two>
      <three>0</three>
    </onboard_total>
    <particulars_informationsss>
      <next_trip>8101</next_trip>
      The BRESLAU …
    </particulars_informationsss>
  </voyage>
  <voyage>
  …
</VOC>
```

Abbildung 12: Struktur des flach geschachtelten Dokuments

dem flach strukturierten Dokument im zweiten Level. Auch bei diesem Test werden die bekannten `COUNT`- und `FOR-LET-RETURN`-Anfragen über das *arrival*-Element durchgeführt.

4.2.5 Kompakte vs. verteilte Speicherung

Mit Hilfe dieses Tests soll die verteilte Speicherung mit der kompakten Speicherung der Dokumente verglichen werden. Das bedeutet auf der einen Seite sind alle *voyage*-Elemente in einem XML-Dokument zusammengefasst und werden in einer Spalte der Tabelle abgelegt. In diesem Fall befinden sich alle Informationen in einem einzigen Dokument. Auf der anderen Seite wird jedes einzelne *voyage*-Element in einer neuen Zeile der Tabelle abgespeichert. Jedes abgespeicherte XML-Dokument enthält somit nur die Information eines *voyage*-Elements. Bei dieser Methode gibt es dann genauso viele XML-Dokumente (1, 8, 127,..., 8076) wie *voyage*-Elemente.

Zur Durchführung dieses Tests muss das Originaldokument inhaltlich nicht modifiziert werden. Es ist aber eine Zerlegung der unterschiedlich großen Dokumente in die einzelnen *voyage*-Elemente notwendig. Für drei *voyage*-Elemente sehen die beiden verschiedenen Strukturen wie in Abbildung 13 aus.

Die drei verschiedenen Anfragen dieses Tests werden jeweils auf das *voyage*-Element

```
<VOC>                          <voyage>
  <voyage>                     …
  …                            </voyage>
  </voyage>
  <voyage>                     <voyage>
  …                            …
  </voyage>                    </voyage>
  <voyage>
  …                            <voyage>
  </voyage>                    …
</VOC>                         </voyage>
```

Abbildung 13: Vergleich der Dokumentstrukturen für kompakte (links) und verteilte Speicherung (rechts)

der Dokumente durchgeführt. Bei den Anfragen handelt es sich erneut um eine XPath-Anfrage, eine `COUNT` und eine `FOR-LET-RETURN`-Anfrage.

Zur Bewertung der Speicherung der verschiedenen XML-Dokumente wird bei diesem Test nur die Speicherung ohne Schemadefinition und Schemavalidierung untersucht. Der Grund hierfür ist, dass beide Strukturen annähernd dasselbe Schema besitzen und somit bei der Validierung kein signifikanter Unterschied zu erwarten ist. Der Einfluss der Validierung bei der Speicherung der Dokumente ist in den vorhergehenden Tests auch bereits mehrmals untersucht worden und daher genügt es bei diesem Test den Einfluss der verschiedenen Speicherungsvarianten (kompakt und verteilt) zu messen.

4.2.6 IDREF vs. KEYREF

Mit Hilfe dieses Tests soll der Einfluss der unterschiedlichen Schlüsselkonzepte auf die Effizienz untersucht werden. Dieser Test wird nur bezüglich der Speicherung der XML-Dokumente mit Validierung durchgeführt. Der Grund dafür ist, dass beim Anfragen der Dokumente die *ID*- und *KEY*-Elemente bzw. Attribute genauso wie übliche Elemente oder Attribute behandelt werden, denn sie werden als solche abgespeichert. Erst bei der Validierung gegen ein XML Schema jedoch werden die ID/IDREF- bzw. KEY/KEYREF-Eigenschaften des entsprechenden Dokumentes berücksichtigt. Unter anderem wird an dieser Stelle überprüft, ob jede Referenz auch als ID oder KEY in dem Dokument enthalten ist und ob die ID's und KEY's eindeutig sind. Bei der Untersuchung der Effizienz der Schlüsselkonzepte IDREF und KEYREF ist zu beachten, dass die Konzepte eine unterschiedliche Mächtigkeit besitzen und somit nicht beliebig austauschbar sind.

Um diesen Test durchführen zu können, muss zunächst das Dokument an die Eigenschaften der beiden Schlüsselkonzepte angepasst werden. Für den ID/IDREF-Test bedeutet dies, dass zunächst für jedes *voyage*-Element ein eindeutiger Schlüssel (ID) als Attribut definiert werden muss. Dafür wird das *number*-Element entsprechend modifiziert. Somit hat jedes *voyage*-Element eine eindeutige ID. Das *next*-Element

des Dokumentes wird als IDREF umgewandelt. Das bedeutet, der Inhalt des Elementes wird gelöscht und es bekommt ein entsprechendes Attribut mit gleichem Inhalt, welches vom Typ IDREF ist. Dieses Attribut dient jeweils als Verweis auf die nächste Fahrt des Schiffes. Zu beachten ist, dass die Nummer jedes *IDREF*-Attributes auch in dem Dokument vorkommen muss, denn ansonsten ist eine ID/IDREF-Eigenschaft verletzt.

Ähnliche Veränderungen am Dokument müssen auch für den KEY/KEYREF-Test vorgenommen werden. Auch hier erhält jedes *voyage*-Element ein Attribut namens *KY* (KY steht für KEY, da aber ID nur zwei Zeichen lang ist, darf auch KEY auf Grund der resultierenden Dokumentgrößen nicht länger sein). Das *KY*-Attribut wird anschließend im Schema als Key definiert. Im zweiten Teil wird der Inhalt des *next*-Element wieder entfernt und das Element bekommt ein Attribut *KYREF* mit gleichem Inhalt (analog IDREF). Die Dokumentstruktur für diesen Test ist exemplarisch in Abbildung 14 zu sehen.

```
<VOC>
<voyage id/ky="number0001">
  <trip>3</trip>
  <leftpage>
  …
  </leftpage>
  <rightpage>
  …
    <particulars>
      <next idref/kyref="number0002"/>
      The BRESLAU …
    </particulars>
  </rightpage>
</voyage>
…
```

Abbildung 14: Dokumentstruktur für den IDREF vs. KEYREF Test

Um die Unterschiede zwischen der Verwendung von ID/IDREF und KEY/KEYREF zu verdeutlichen, sind in Abbildung 15 jeweils ein Auszug aus den beiden Schemata vergleichend gegenübergestellt. Die entscheidenen Zeilen sind mit fetter Schrift hervorgehoben. Neben dem Vergleich der Effizienz bei der Verwendung von ID/IDREF's und KEY/KEYREF's wird auch die Speicherung der selben Dokumente ohne Überpfrüfung der Schlüsseleigenschaften untersucht. Damit ist es möglich den Aufwand der jeweiligen Überprüfung abzuschätzen. Hierfür werden die selben Dokumente gegen ein Schema validiert, in dem die *ID*- und *IDREF*-Attribute ganz normale Attribute (vom Typ NCName) mit nicht eindeutigen Namen sind. Bei dieser Validierung entfällt dann die Schlüsselprüfung.

An dieser Stelle muss darauf hingewiesen werden, dass der vollständige Test leider nur auf DB2 Viper durchgeführt werden kann. Das ist darin begründet, dass der Tamino XML Server Version 4.2.1 die Überprüfung der KEY/KEYREF-Eigenschaften

```
<xs:element name="voyage">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0"
        ref="trip"/>
      <xs:element ref="leftpage"/>
      <xs:element ref="rightpage"/>
    </xs:sequence>
    <xs:attribute name="id"
      use="required" type="xs:ID"/>
  </xs:complexType>
</xs:element>
…
<xs:element name="next">
  <xs:complexType>
    <xs:attribute name="idref"
      type="xs:IDREF"/>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="VOC">
…
  <xs:key name="voyageKey">
    <xs:selector xpath="./voyage"/>
    <xs:field xpath="@ky"/>
  </xs:key>
  <xs:keyref name="voyageKeyref"
    refer="voyageKey">
    <xs:selector
      xpath="./voyage/rightpage/
      particulars/next"/>
    <xs:field xpath="@kyref"/>
  </xs:keyref>
</xs:element>
…
<xs:element name="voyage">
  <xs:complexType>
  …
    <xs:attribute name="ky"
      use="required" type="xs:NCName"/>
  </xs:complexType>
</xs:element>
…
<xs:element name="next">
  <xs:complexType>
    <xs:attribute name="kyref"
      type="xs:NCName"/>
  </xs:complexType>
</xs:element>
```

Abbildung 15: Auszug aus den Schemata von IDREF (links) und KEYREF (rechts)

nicht unterstützt.

4.3 Implementierung

Die Implementierung der Tests erfolgt in Java. Die Art der Speicherung und der Zugriff auf das DBS sind jeweils systemspezifisch und werden im Folgenden genauer erläutert.

Allgemein ist zu den drei Implementierungen zu sagen, dass bei jedem System sowohl beim Einfügen der Dokumente als auch beim Anfragen der Dokumente die Ergebnisse (Zeiten) jeweils in eine Datei geschrieben werden. Diese Datei dienen später als Grundlage für die Auswertung der einzelnen Tests. Die Zeit wird auch bei allen drei Systemen gleich bestimmt. Es wird sowohl direkt vor der Ausführung als auch direkt nach der Ausführung der `INSERT`- und `Query`-Kommandos die Systemzeit genommen und anschließend die Differenz gebildet, so dass am Ende das Ergebnis die Zeit für die Abarbeitung des Befehls darstellt.

Außerdem werden alle Testläufe mit Hilfe von Skripten ausgeführt. Die Sripte dienen zum Aufrufen der entsprechenden Java-Klassen und zum Übergeben der testspezifischen Informationen, wie beispielsweise den Pfade der einzufügenden Datei, die speziellen Anfragen für das gespeicherte Dokument oder auch den NAmen oder

Pfad der Datei, in der das Testergebnis abgespeichert werden soll.

4.3.1 IBM DB2 UDB (V8.1.8) mit XML-Extender

Speicherung der XML-Dokumente
Für jeden Testlauf wird eine neue Tabelle angelegt, die jeweils eine Spalte mit einer systemgenerierten Identität und eine Spalte für das einzufügende XML-Dokument enthält:

```
CREATE TABLE xmldocs (id integer not null primary key generated
always as identity, xmlcolumn db2xml.xmlclob not logged)
```

Das wiederholte Anlegen und darauffolgende Löschen der Tabelle für jeden Testlauf ist notwendig, um annähernd identische Randbedingungen (initiale Größe der Datenbank) zu gewährleisten.

Das Einfügen der XML-Datei in die Datenbank erfolgt mittels Java über eine JDBC Verbindung. Der SQL-Befehl (prepared statement) hierfür lautet:

```
INSERT INTO xmldocs (XMLCOLUMN) VALUES (?)
```

Der Platzhalter „?“ innerhalb des *PreparedStatement*s wird dabei durch die zuvor aus der XML-Datei eingelesene Zeichenkette (in Form eines *ByteArray*s) belegt. Außerdem sind entsprechende JDBC-Befehle zum Auf- und Abbauen der Datenbankverbindung erforderlich. Gemessen wird jedoch lediglich die Zeit für das Insert-Kommando selbst.

Eine Anmerkung zu Test Nr. 5 (Kompakte vs. verteilte Speicherung [Kapitel 4.2.5]): Beim Einfügen der vielen kleinen XML-Dokumente in die Datenbank (verteilte Speicherung) tritt nach einer bestimmten Anzahl von Dokumenten folgender Fehler auf:

```
COM.ibm.db2.jdbc.DB2Exception:
IBM CLI Driver DB2/NT
SQL0954C
Not enough storage is available in the application heap to process
the statement.
SQLSTATE=57011
```

Das Problem lässt sich auch durch eine Vergrößerung der Heapgröße nicht derart beheben, dass alle einzelnen kleinen Dokumente eingefügt werden können und somit wird die folgende Lösung implementiert: Statt der ursprünglichen Variante, bei der zunächst eine DB-Verbindung hergestellt wird, dann das Einfügen aller Dokumente (einschl. Messung der benötigten Zeit) erfolgt und letztlich die Verbindung wieder geschlossen wird, wird die DB-Verbindung nach jeweils 1000 Dokumenten einmal ab- und wieder aufgebaut. Da jedoch nur die Netto-Zeit (ohne Auf- und Abbauen der DB-Verbindung) gemessen und kumuliert wird, sind die Ergebnisse vergleichbar. Das Auf- und Abbauen der Verbindung für jedes einzelne Dokument ist nicht erforderlich und wird wegen des hohen Zeitbedarfs nicht umgesetzt.

XPath-Anfragen
Die XPath-Anfragen werden in SQL eingebettet und ebenfalls aus einer Java-Applikation über eine JDBC-Verbindung abgesetzt. Da im allgemeinen Fall die XPath-Anfrage mehr als nur einen Wert zurückliefern kann, muss die tabellenwertige UDF des Text-Extenders verwendet werden. Der Datentyp des Rückgabewertes ist eine Zeichenkette (varchar), so dass der SQL-Befehl folgendermaßen aussieht:

```
SELECT id, returnedvarchar FROM xmldocs,
table (db2xml.extractvarchars(xmlcolumn, '[query]')) AS RESULT
```

Der Platzhalter [query] wird in der Java-Applikation jeweils durch die entsprechende XPath-Anfrage ersetzt. Die einzelnen Ergebnistupel werden dann zeilenweise in der Java-Anwendung ausgelesen (Cursor-Konzept). Für die Effizienzbewertung wird die Zeit für das Absetzen des SQL-Befehls und Abfragen aller Tupel gemessen, jedoch auch hier wieder ohne das Auf- und Abbauen der JDBC-Verbindung.

Anmerkung: Die Ausgabe der XPath-Anfrageergebnisse auf den Bildschirm mittels `System.out.println()` hat einen erheblichen Einfluss auf die gemessene Zeit. Aus diesem Grund wird diese Ausgabe nur zur Kontrolle durchgeführt. Während der Tests selbst erfolgt jedoch keine Ausgabe.

Die einzelnen XPath-Anfragen werden jeweils mehrfach durchgeführt und die Zeiten der einzelnen Messungen schließlich gemittelt. Um dabei zu verhindern, dass nach der ersten Anfrage die Ergebnisse bereits in einem Zwischenspeicher des Datenbanksystems vorhanden sind, wurde das DBS vor jeder Anfrage angehalten und neu gestartet:

```
db2 terminate
db2stop force
db2start
```

4.3.2 Tamino XML Server

Beim Tamino XML Server ist keine Kommandozeile wie z.B. bei DB2 vorhanden, auf der einzelne Befehle (DB2: SQL) abgesetzt werden können. Dafür stehen beim Tamino XML Server aber verschiedene Tools zur Verfügung, mit denen auf der Datenbank gearbeitet werden kann. Beispielsweise gibt es den Schema Editor, mit dessen Hilfe XML-Collections mit einer XML-Schemadefinition angelegt werden können. Weiterhin gibt es auch ein XQuery Tool zur Definition und Durchführung von XQueries auf den Datenbankinhalten und eine Webschnittstelle zur Systemadministration. Neben den genannten Tools sind noch eine Reihe weiterer vorhanden, die in [Sch03] ausführlich vorgestellt werden.

Der Zugriff auf die Datenbank erfolgt bei Tamino über spezielle Bibliotheken. Im

Wesentlichen findet hierbei die **Tamino API for Java** [SAGa] Verwendung. Die Tamino API for Java-Bibliothek wird von der Software AG für den Zugriff auf Tamino-Datenbanken empfohlen. Der Vorteil dieser proprietären Schnittstelle ist, dass alle Möglichkeiten des System genutzt und angesprochen werden können. Mit Hilfe dieser Bibliothek ist die Unterstützung verschiedener Datenzugriffsmodelle (DOM, JDOM, SAX) und Standards (JAXP, DOM2) gegeben. Weiterhin wird auch die Bibliothek **XML:DB API** [SAGb], die eine einheitliche und systemunabhängige Schnittstelle für native XML-Datenbanken definiert, unterstützt. Der Vorteil dieser Bibliothek liegt in der Vereinfachung der Entwicklung von Anwendungen, die verschiedene XML-DBS unterstützen. Dies ist mit der Anfragesprache SQL bei relationalen DBS zu vergleichen. Die Implementierung dieser allgemeinen, systemunabhängigen Beschreibung der Schnittstelle wird durch systemspezifische Bibliotheken realisiert.

Speichern der XML-Dokumente
Die XML-Dokumente werden bei Tamino in *Collections* gespeichert (siehe Kapitel 3.2.2, vgl. Tabelle in einer Datenbank). Zu unterscheiden sind hierbei *Collections* zur Speicherung von Dokumenten mit und ohne Schemadefinition. XML-Dokumente ohne Schemadefinition werden standardmäßig in der Collection `ino:etc` gespeichert. Für die Tests mit Dokumenten ohne Indexe ist keine Schemadefinition erforderlich und die Speicherung dieser erfolgt in der *Collection* `ino:etc`. Ansonsten ist das Anlegen einer speziellen *Collection* erforderlich. *Collections* werden (analog den Tabellen bei DB2) für jeden Test stets neu angelegt und anschließend wieder gelöscht, um identische Randbedingungen zu gewährleisten.

Ausnahme: Da es sich bei der *Collection* `ino:etc` um eine spezielle *Collection* des XML-Servers handelt, wird diese nicht gelöscht, sondern lediglich alle XML-Dokumente der *Collection*. Die Testergebnisse zeigen aber keine signifikanten Unterschiede oder Abweichungen zwischen den einzelnen Tests, die hierin begründet sein könnten.

Für das Einfügen eines XML-Dokumentes wird zunächst eine Verbindung zur Datenbank aufgebaut und ein Objekt für den Zugriff auf die jeweilige *Collection* angefordert. Anschließend wird das XML-Dokument durch einen entsprechenden XML-Parser eingelesen. Gemessen wird jedoch (vgl. DB2) auch hier nur die Zeit für das Einfügen des Dokumentes in die Datenbank, welches sich aus dem Anlegen einer neuen XML-Ressource und dem Zuweisen des zugehörigen Inhalts (Content, also das XML-Dokument) zusammensetzt.

XQuery-Anfragen
Im Gegensatz zu DB2 müssen die XPath- oder XQuery-Anfragen nicht umständlich in eine SQL-Anfrage eingebettet werden, sondern können direkt gestellt werden. Nach dem Aufbauen der Verbindung zur Datenbank wird zunächst ein Objektes für den Zugriff auf die XML-Objekte der jeweiligen *Collection* (*TXMLObjectAccessor*) erzeugt. Anschließend wird ein Objekt, welches die XQuery-Anfrage beinhaltet

(*TXQuery*), erzeugt. Für die Effizienzbewertung wird die Zeit gemessen, die für die Auswertung dieser Anfrage erforderlich ist. Dieses beinhaltet den Aufruf der Anfrage auf dem Accessor-Objekt, welche ein *TResponse*-Objekt für das Anfrageergebnis zurückliefert. Dieses Anfrageergebnis enthält dann entsprechend der Anfrage mehrere XML-Objekte, die iterativ (zeilenweise, vgl. DB2) ausgelesen werden. Die Anfragen werden wie auch bei dem DB2 Datenbanksystem jeweils mehrfach ausgeführt.

4.3.3 DB2 Viper (V9.1)

Die Implementierung der Tests für DB2 Viper ist ähnlich der für DB2 V8.2, da es sich bei DB2 Viper (V9.1) um den direkten Nachfolger von V8.2 handelt. Der entscheidene Unterschied ist jedoch, dass bei Viper keine speziellen Extender mehr erforderlich sind, sondern die XML-Funktionalität direkt in der DB Engine integriert ist (vgl. Kapitel 3.3).

Speichern der XML-Dokumente
Die XML-Dokumente werden hier nicht mehr als CLOB (`XMLCLOB`) abgelegt, sondern als Datentyp `XML`. Die Tabelle der Viper Datenbank wird mit folgender SQL-Anweisung angelegt:

```
CREATE TABLE project
(id integer not null primary key generated always as identity,
documents XML)
```

Sie enthält eine Spalte mit einer systemgenerierten Identität und eine Spalte vom Typ `XML`, die zum Ablegen der XML-Dokumente dient. Wie auch bei DB2 V8 wird die Tabelle für jeden Testlauf neu angelegt. Dies ist eine notwendige Maßnahme, um für jeden einzelnen Testlauf annähernd gleiche Randbedingungen zu schaffen. Die XML-spezifischen Befehle werden in SQL Syntax verpackt und mittels JDBC aufgerufen. Die XML-Dokumente können sowohl als Zeichenkette (String) als auch als Datenstrom (*ByteArrayStream*) übergeben werden. Bei Zeichenketten muss jedoch das Parsen der Dokumente durch die Funktion `XMLPARSE` explizit angeben werden, ansonsten erfolgt das Parsen implizit. Empfohlen wird deshalb vor allem für große Dokumente die Verwendung von *ByteArrayStream*. Deshalb erfolgt das Einfügen der XML-Dokumente analog zu DB2 V8.2 durch den Befehl

```
INSERT INTO project (documents) VALUES (?)
```

Der Platzhalter (?) im PreparedStatement wird durch den *ByteArrayStream* substituiert. An dieser Stelle tritt das gleiches Problem wie DB2 V8.2 mit den kleinen einzelnen Dokumenten auf. Somit wird auch bei DB2 Viper das bereits bei DB2 V8 (vgl. Kapitel 4.3.1) beschriebene „Work-Around“ erforderlich.

Validierung der XML-Dokumente
Die Validierung der Dokumente gegen ein XML-Schema kann beim Einfügen der

Dokumente optional erfolgen. An dieser Stelle eine Anmerkung: Im Gegensatz zu Tamino (oder auch Oracle) ist es damit möglich, Dokumente mit unterschiedlichen Schemata zusammen in einer Spalte (bzw. Collection) zu speichern und trotzdem eine Schemavalidierung durchzuführen. Dies ist beispielsweise ein Vorteil bei Schemaevolution oder ähnlichem. Für die Schemavalidierung ist zunächst das Registrieren des Schemas erforderlich. Dieses erfolgt für die ganze Datenbank und nicht spezifisch für eine Tabelle oder Spalte. Der Befehl für die Registrierung von XML-Schemata ist in Kapitel 3.3.2 nachzulesen. Das veränderte Insert-Kommando für das Einfügen der XML-Dokumente lautet dann:

```
INSERT INTO project (documents)
VALUES (XMLVALIDATE (? ACCORDING TO XMLSCHEMA ID schemaname))
```

Auch bei diesem Befehl dient „?“ als Platzhalter für den übergebenen *ByteArray-Stream*.

Anfragen
Die XQuery-Anfragen können sowohl direkt (Schlüsselwort XQuery) ausgeführt, als auch in SQL Befehlen eingebettet werden. Da der Zugriff auf die DB aus Java via JDBC erfolgt, wurde für die Implementierung die SQL-Variante gewählt:

```
SELECT XMLQUERY('anfrage' PASSING DOCUMENT AS "doc")
FROM project
```
[12]

Der gewählte Bezeichner „doc“ wird dann innerhalb der XQuery-Anfrage zur Referenzierung des Wurzelelements (als $doc) verwendet. Bei der Ausgabe der Ergenisse einer Anfrage wird für jede Zeile der Tabelle ein Dokument (XML Sequence) zurückgegeben, welches das jeweilige Anfrageergebnis enthält. Das zurückgegebene Dokument muss nicht notwendigerweise ein XML-Dokument sein. Wie auch bei DB2 V8 wird für die Ermittlung der Testergebnisse nur die Zeit für das Ausführen des oben beschriebenen SQL-Kommandos und das Auslesen der Rückgabewerte gemessen, nicht die Zeit für das Auf- und Abbauen der JDBC-Verbindung. Außerdem wird auch hier vor jeder einzelnen Anfrage eines Testlaufes das DB-System angehalten und wieder neu gestartet (siehe Kapitel 4.3.1). Die dafür verwendeten Befehlen lauten:

```
db2 terminate
db2stop force
db2start
```

Dieser Vorgang ist ebenfalls notwendig, um für jeden einzelnen Testlauf annähernd gleiche Randbedingungen zu schaffen.

[12] Da jeweils nur das für den aktuellen Testlauf relevantee Dokumente in der Tabelle gespeichert war, kann auf eine Selektion der Dokumente mittels der `WHERE`-Klausel verzichtet werden

5 Durchführung der Tests

In diesem Abschnitt werden zunächst die verwendeten Testsystem überblicksartig vorgestellt und mit einigen Worten die Testumgebung beschrieben. Im zweiten Teil des Kapitels werden die Auswertungen der einzelnen Tests anhand der Diagramme, die auf der Grundlage der Testergebnisse entstanden sind, durchgeführt. Dabei wird kurz noch einmal der Test vorgestellt und anschließend systemabhängig ausgewertet. Das erste System ist jeweils DB2 V8, dann Tamino von der Software AG und abschließend DB2 Viper. Nach der systemabhängigen Betrachtung der Tests wird im letzten Teil eine Bewertung gegeben, die allgemeine Aussagen über die jeweils verglichenen Strukturen beinhaltet.

5.1 Testsysteme

Aufgrund der unterschiedlichen Anforderungen der verwendeten Datenbanksysteme (DB2 und Tamino) sowie zeitlicher Einschränkungen der genutzten Lizenzen konnten nicht alle Tests auf ein und demselben System ausgeführt werden. Die vergleichenden Tests (hinsichtlich der unterschiedlichen Strukturen der Dokumente für ein Datenbanksystem) wurden jedoch stets auf demselben System ausgeführt, so dass diese Ergebnisse aussagekräftig sind. Die ermittelten Zeiten für die verschiedenen Datenbanksysteme sind aus dem oben genannten Grund nicht unbedingt miteinander vergleichbar. Da die einzelnen Systeme aber nicht miteinander verglichen werden sollen, ist dieses auch nicht erforderlich.

IBM DB2 UDB mit XML- und Text-Extender, Version 8.1.8
Prozessor: Intel Pentium 4 mobile, 1700 MHz, 512 MB RAM
Betriebssystem: Microsoft Windows 2000

IBM DB2 UDB, Version 9.1 (IBM DB2 VIPER)
Prozessor: Intel Pentium 4 mobile, 1800 MHz, 1 GB RAM
Betriebssystem: Microsoft Windows XP Professional Version 2002 (Servicepack 1)

Software AG Tamino XML Server, Version 4.2.1
(Evaluierungslizenz, 30 Tage)
Prozessor: Intel Pentium 4 mobile, 1800 MHz, 1 GB RAM
Betriebssystem: Microsoft Windows XP Professional Version 2002 (Servicepack 1)

Neben der genannten DB-Anwendung waren typische Prozesse des Betriebssystems und andere Dienste (Firewall, etc.) ebenfalls aktiv. Da diese Randbedingungen jedoch für die jeweiligen Teststrukturen identisch sind, kann das qualitative Testergebnis als repräsentativ betrachtet werden. Außerdem wurden die einzelnen Tests jeweils mehrfach durchgeführt und anschließend bei der Auswertung der Maximal-

und Minimalwert unberücksichtigt gelassen. Somit ist abgesichert, dass sogenannte Ausreißer, z.B. verursacht durch Aktivitäten eines Systemprozesses, weitgehend ausgeschlossen werden.

5.2 Elemente vs. Attribute

Dieser Test diente dazu herauszufinden, ob die Verwendung von Elementen oder Attributen in XML-Dokumenten bei der Speicherung und beim Anfragen der Werte effizienter ist. Da in vielen Fällen Elemente durch Attribute oder umgekehrt ersetzt werden können, wäre ein Unterschied zwischen der Verwendung von Attributen oder Elementen sehr interessant. Die genaue Erläuterung der Vorgehensweise bei diesem Test ist in Kapitel 4.2.1 nachzulesen.

5.2.1 DB2 V8

Zunächst wird die Auswertung dieses Tests auf dem Datenbanksystem DB2 V8 vorgenommen. Hierzu werden im ersten Teil die Zeiten für die Speicherung der Dokumente (Abb. 16) verglichen und im zweiten Teil die Zeiten für die verschiedenen Anfragen (Abb. 17).

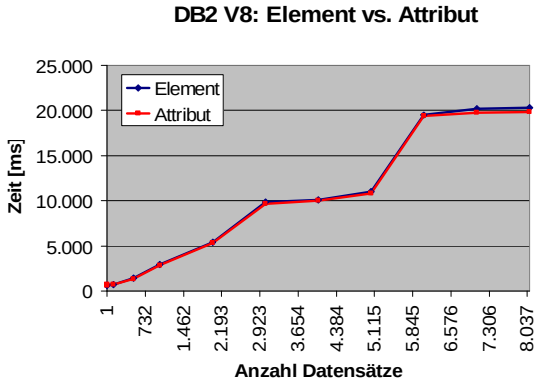

Abbildung 16: Diagramm für die Speicherung der Dokumente

Grundsätzlich ist in dem Diagramm für die Speicherung deutlich zu erkennen, dass die Zeit für das Einfügen sowohl bei den Dokumenten mit Attributen als auch bei denen mit Elementen abhängig von der Dokumentgröße ansteigt. Weiterhin ist im Diagramm bei der Zeit für das Einfügen kein Unterschied zwischen den unterschiedlichen Dokumentstrukturen (*master* als Attribut bzw. Element) zu entdecken. Bei DB2 ist die Speicherung für beide Dokumentarten somit gleich effizient. Dieses Ergebnis ist wie erwartet, da die Dokumente bei DB2 V8, wie in Kapitel 3.1.1 beschrieben, als CLOB's in die Datenbank abgelegt werden und die Zeit für die Speicherung somit nur von der Größe der Dokumente abhängig ist. Dabei macht es also kein Unterschied, ob ein Wert als Element oder Attribut modelliert ist.
Die Sprünge in dem Diagramm (vgl. Abb. 16) sind mit dem Beginn neuer Seiten bei

der Speicherung in DB2 zu erklären.

Bei der Auswertung der XPath-Anfrage (Abb. 17) ist hingegen ein deutlicher Unterschied zwischen den beiden Dokumentstrukturen zu erkennen. Die Anfrage wird bei der Modellierung des *Masters* als Element schneller ausgewertet.

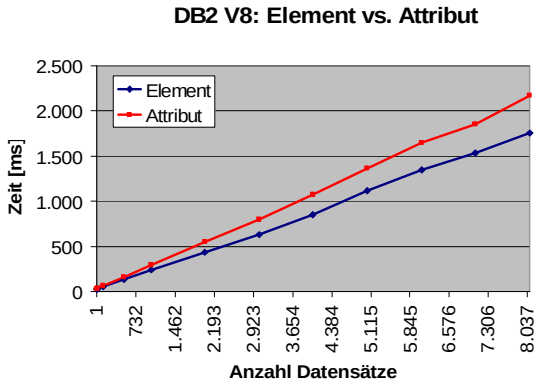

Abbildung 17: Diagramm für die Auswertung der XPath-Anfrage

Das bedeutet, dass bei DB2 V8 XPath-Anfragen auf Attributen langsamer bearbeitet werden als auf Elementen. Außerdem ist zu sehen, dass auch hier die Zeit fürs Anfragen abhängig von der Dokumentgröße linear ansteigt.

5.2.2 Tamino

Als nächstes erfolgt die Auswertung des gleichen Tests auf dem Datenbanksystem Tamino XML Server. Hierbei werden neben der Speicherung ohne Validierung und der XPath-Anfrage auch noch die Speicherung mit Validierung und XQuery-Anfragen zur Bewertung herangezogen.

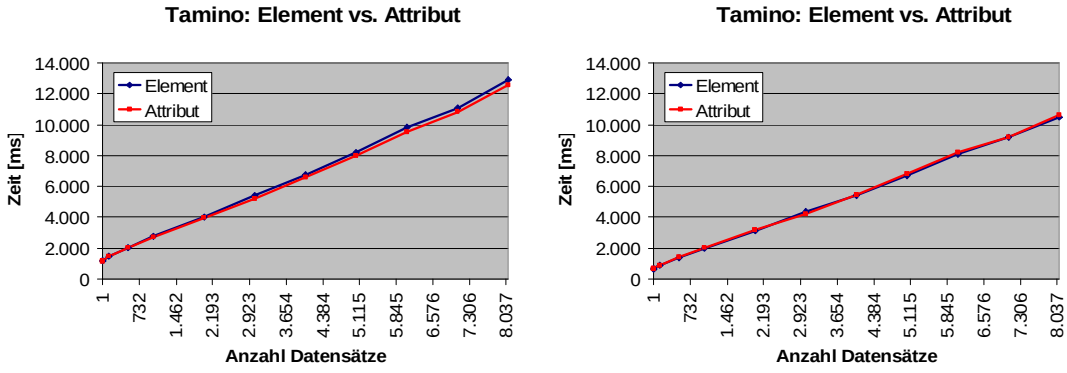

Abbildung 18: Diagramm für das Speichern ohne (links) und mit Validierung (rechts)

Die Speicherung der XML-Dokumente erfolgt bei Tamino in nativer Form. Das bedeutet, die Dokumente werden in ihrer hierarchischen Struktur im Datenbanksystem

abgelegt. Die Speicherung wurde wie bereits angedeutet einmal mit und einmal ohne Schemavalidierung getestet. Wie links in Abbildung 18 zu erkennen ist, ist die Modellierung des *Masters* als Attribut bei der Speicherung minimal schneller als als Element. Der Unterschied zwischen den beiden Dokumentstrukturen ist allerdings vernachlässigbar gering, so dass an dieser Stelle keine konkrete Aussage darüber möglich ist, ob die Speicherung von Dokumenten mit Elementen oder Attributen deutlich effizienter ist. Diese Feststellung wird auch durch das Ergebnis für die Speicherung der XML-Dokumente mit Validierung (Abb. 18 rechts) bestätigt. Auch hier ist kein Unterschied bei der zum Einfügen benötigten Zeit zu erkennen. Im Gegensatz zu DB2 V8 gibt es bei Tamino bei der Speicherung einen größeren Offset (ungefähr 500 msec), aber sonst sind die Ergebnisse identisch. Auffällig ist jedoch, dass Tamino für die Speicherung mit Schemavalidierung weniger Zeit benötigt als ohne.

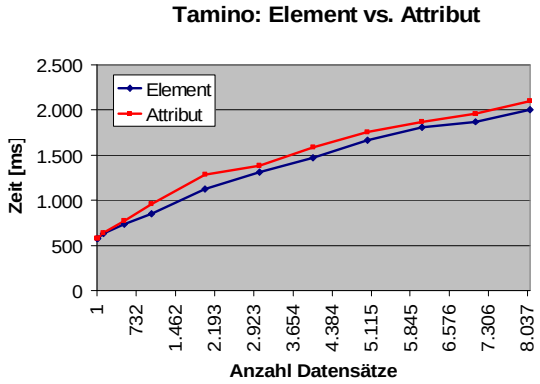

Abbildung 19: Diagramm für die Bearbeitung der XPath-Anfrage

Auch bei der Auswertung der XPath-Anfrage liefert der Tamino XML Server das gleiche Ergebnis wie DB2 V8. Die Verarbeitung der Anfrage erfolgt für die Modellierung des *Masters* als Element effizienter. Der Unterschied ist allerdings nicht so deutlich wie bei DB2 V8, aber trotzdem sehr konstant. Es ist also festzuhalten, dass bei Tamino die Auswertung von XPath-Ausdrücke für Elemente effizienter ist als für Attribute. Der Offset für die Anfrage liegt bei beiden Dokumentstrukturen bei ungefähr 500 Millisekunden.
Im Gegensatz zur XPath-Anfrage liefert die `COUNT`-XQuery ein entgegengesetztes Ergebnis. Das bedeutet, bei der `COUNT`-Anfrage werden die Dokumente, bei denen der *Master* als Attribut modelliert ist, schneller ausgewertet. Bei der `FOR-LET-WHERE-RETURN`-Query verhalten sich beide Strukturen (Elemente und Attribute) äquivalent. Der Offset bei beiden XQueries ist ähnlich der XPath-Anfrage bei ungefähr 600 ms.

Für den Test „Elemente vs. Attribute“ mit Tamino ist somit festzuhalten, dass bei der Speicherung der verschiedenen Dokumente kein Unterschied festzustellen ist und bei der Auswertung der Anfragen gegensätzliche Ergebnisse erzielt wurden. Das

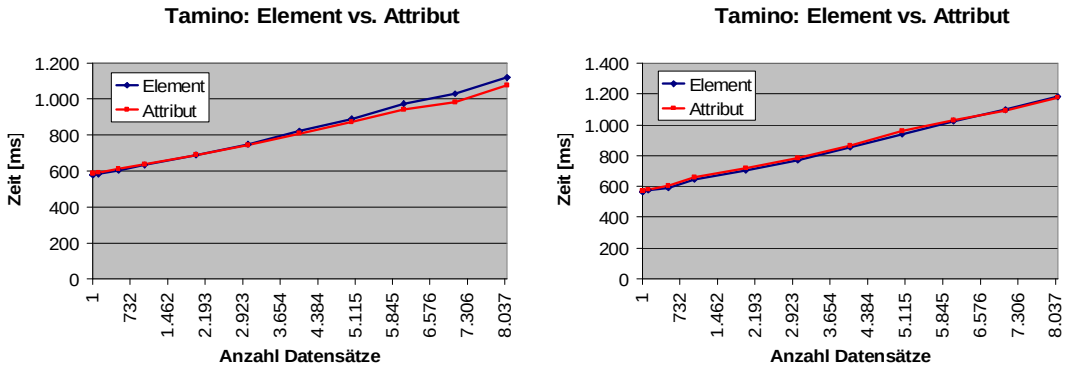

Abbildung 20: Diagramm für die XQuery-Anfragen: `COUNT` (links) und `FLWR` (rechts)

heißt, die Effizienz ist abhängig von den verschiedenen Anfragen und deshalb kann keine allgemeine Aussage abgeleitet werden, ob die Verwendung von Elementen oder Attributen effizienter ist.

5.2.3 Viper

Abschließend für den Test „Elemente vs. Attribute“ erfolgt die Auswertung der beiden Dokumentstrukturen auf dem Datenbanksystem DB2 Viper.

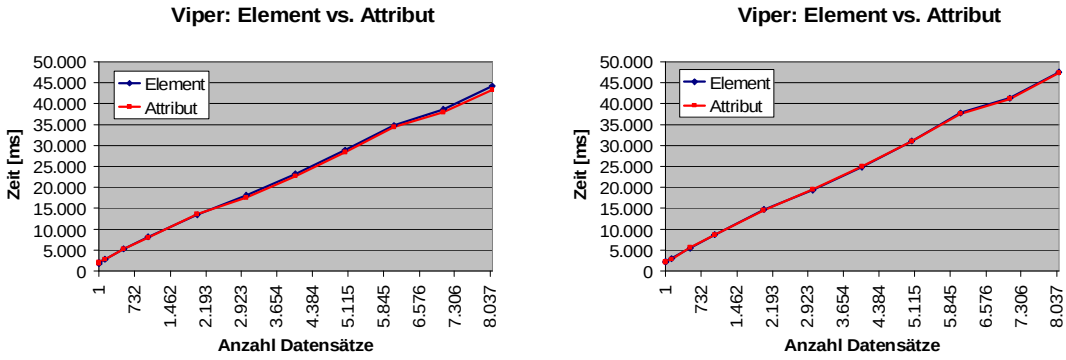

Abbildung 21: Diagramm für das Speichern ohne (links) und mit Validierung (rechts)

Beim Vergleich der unterschiedlichen Strukturen bezüglich der Speicherung ist weder bei der Speicherung ohne Validierung noch bei jener mit XML Schemavalidierung ein Unterschied festzustellen. Das bedeutet, für die Speicherung bei Viper ist es uninteressant, ob Werte als Elemente oder Attribute modelliert sind. Im Gegensatz zu Tamino benötigt Viper für die Speicherung mit Schemavaliderung mehr Zeit.

Sehr interessant jedoch ist das Ergebnis für die verschiedenen Anfragen bei DB2 Viper. Hier ist sowohl bei der XPath-Anfrage als auch bei der `COUNT`- Anfrage die Verwendung von Attributen deutlich effizienter. Hervorzuheben ist, dass der Un-

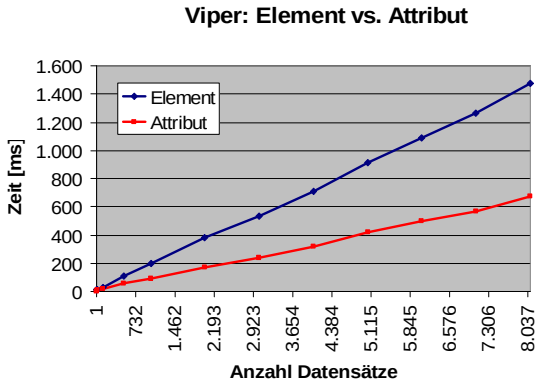

Abbildung 22: Diagramm für die Bearbeitung der XPath-Anfrage

terschied zwischen Attributen und Elementen unerwartet groß ist. Die Dokumente mit Attributen sind bei der Verarbeitung des XPath-Ausdruckes etwa doppelt so schnell wie die Dokumente mit ausschließlich Elementen. Bei der `COUNT`-Anfrage ist die Differenz nicht ganz so groß. Unabhängig von den obigen Beobachtungen ist auch bei DB2 Viper die Auswertung der `FLWR`- Anfrage für beide Strukturen gleich schnell. Zusammengefasst ist bei DB2 Viper die Effizienz für die Speicherung der XML Dokumente unabhängig davon, ob Werte als Elemente oder Attribute modelliert werden. Bei Anfragen hingegen ist die Effizienz für Attribute größtenteils besser als für Elemente.

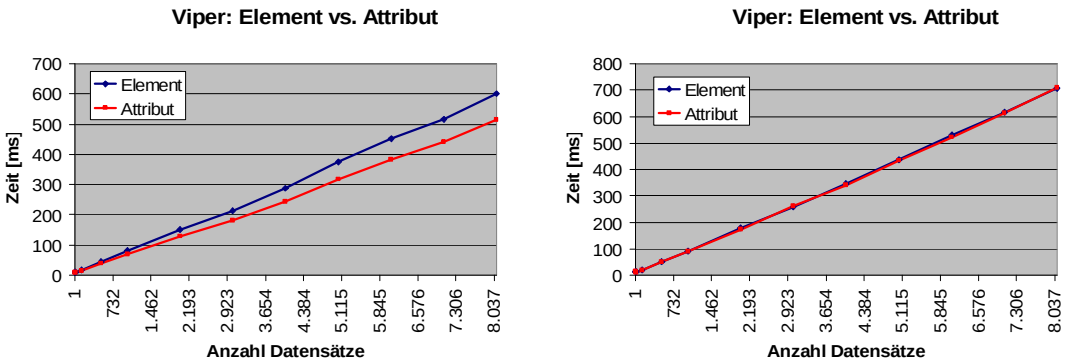

Abbildung 23: Diagramm für die XQuery-Anfragen: `COUNT` (links) und `FLWR` (rechts)

5.2.4 Auswertung

Allgemein lässt sich für den Test „Attribute versus Elemente“ sagen, dass es auf keinem der getesteten Systeme einen entscheidenden Unterschied bei der Speicherung zwischen der Verwendung von Elementen bzw. Attributen gab. Dies trifft sowohl auf die Speicherung mit als auch ohne XML Schemavalidierung zu. Zu der Effizienz bei der Verarbeitung der Anfragen kann keine allgemeine Aussage abgeleitet werden.

Hier wurden systemabhängige Unterschiede zwischen der Verwendung von Elementen und Attributen festgestellt. Auch innerhalb eines Systems ist es bei verschiedenen Anfragen nicht eindeutig, ob Attribute oder Elemente effizienter in der Verarbeitung sind. Sowohl bei DB2 V8 als auch bei Tamino ist bei der XPath-Anfrage die Verwendung von Elementen effizienter als von Attributen. Bei dem Datenbanksystem DB2 Viper ist es genau umgekehrt. Letztlich bleibt nur zu sagen, dass die Effizienz der beiden Dokumentstrukturen auf der einen Seite vom verwendeten System und auf der anderen Seite von der Art der modellierten Anfragen abhängig ist.

5.3 Länge der Elementnamen

Bei diesem Test ging es darum, den Einfluss der Länge von Elementnamen zu untersuchen. Das Ergebnis für diesen Test soll Aufschluss darüber geben, ob es einen Unterschied macht lange oder kurze Elementnamen zu verwenden. Die exakte Erläuterung des Tests ist in Kapitel 4.2.2 zu finden. Die Auswertung erfolgt zunächst wieder systemspezifisch und letztendlich sollen allgemeine Aussagen abgeleitet werden.

5.3.1 DB2 V8

In dem Diagramm für die Speicherung der Dokumente (Abb. 24) ist deutlich zu erkennen, dass die Elementnamenlänge keinen Einfluss auf die für die Speicherung benötigte Zeit hat. Es sind keine signifikanten Unterschiede zwischen der Verwen-

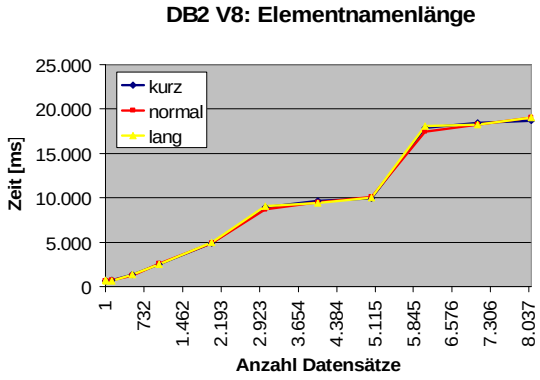

Abbildung 24: Diagramm für das Speichern der beiden Dokumente

dung von kurzen oder sehr langen Elementnamen ersichtlich. Dieses Ergebnis ist, wie auch beim vorherigen Test, mit der Speicherung der XML-Dokumente als CLOB zu erklären. Bei dieser Speicherungsform beeinflusst nur die Größe der Dokumente die Zeit für die Speicherung und da alle drei verschiedenen Dokumente jeweils gleich groß sind, ist auch die Zeit für die Speicherung gleich.

Auch bei der Auswertung der XPath-Anfrage auf DB2 V8 sind keine entscheidenden Unterschiede zwischen den verschiedenen Längen zu erkennen (vgl. Abbildung 25).

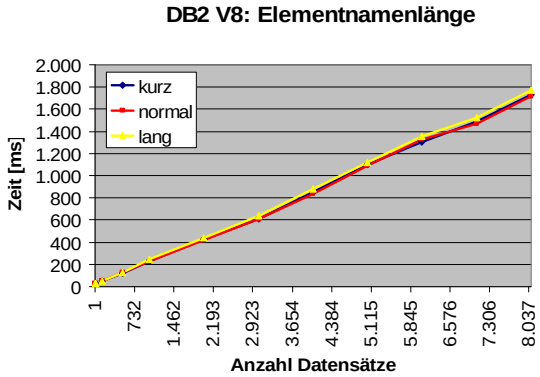

Abbildung 25: Diagramm für die Bearbeitung der XPath-Anfrage

Es wird aber deutlich, dass die langen Elementnamen minimal langsamer sind als die Kurzen. Die Differenz ist jedoch vernachlässigbar gering, sodass die Länge von Elementnamen bei DB2 weder einen Einfluss auf die Zeit für die Speicherung noch einen signifikanten Einfluss auf die Zeit für das Auswerten von XPath-Anfragen hat.

5.3.2 Tamino

Die Auswertung des Tests „Länge der Elementnamen“ erfolgt bei Tamino bezüglich der Speicherung mit und ohne Schemavalidierung und bezüglich einer XPath-Anfrage und zwei XQueries.

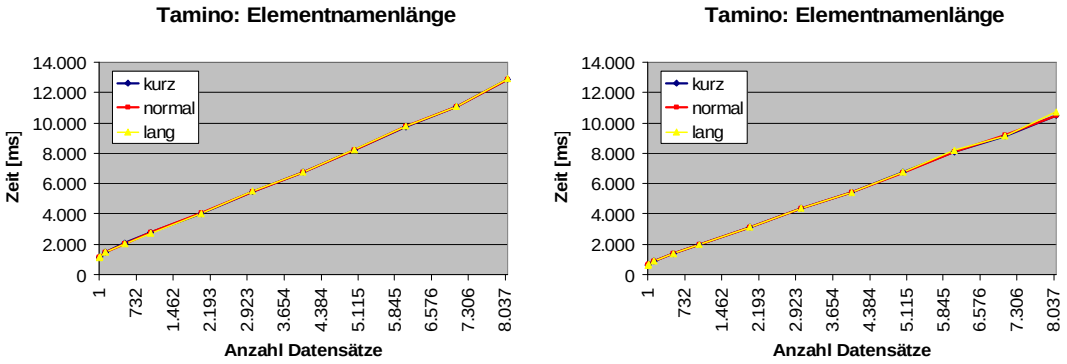

Abbildung 26: Diagramm für das Speichern ohne (links) und mit Validierung (rechts)

Trotz der nativen Speicherungsform der XML-Dokumente ist bei Tamino, wie auch bei DB2 V8, kein Unterschied beim Einfügen der verschiedenen Dokumente in die Datenbank erkennbar. Die Dokumente mit verhältnismäßig langen Elementnamen benötigen annähernd die gleichen Zeiten wie die Dokumente mit durchschnittlich langen (5-10 Zeichen) und kurzen Elementnamen. Diese Beobachtung trifft sowohl für die Speicherung mit Schemavalidierung als auch für jene ohne Schemadefinition und -validierung zu. Bezüglich der Effizienz für die Speicherung ist es dem Nut-

zer bei Tamino somit freigestellt, wie lang er die Elementnamen wählt, solange sie überschaubar (bis ungefähr 10 Zeichen) bleiben, da keine konkrete Aussage über die Effizienz möglich ist. Auch bei diesem Test ist ein gewisser Offset (ungefähr 1 sec bei der Speicherung ohne Validierung und 500 msec mit Validierung) bei der Speicherung der XML-Dokumente in eine Tamino Datenbank festzustellen.

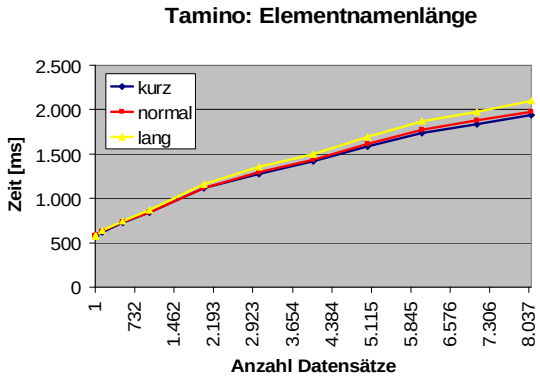

Abbildung 27: Diagramm für die Bearbeitung der XPath-Anfrage

Beim Anfragen der Dokumente ist ein Unterschied zwischen einerseits der XPath-Anfrage (vgl. Abb. 27) und andererseits den XQueries (vgl. Abb. 28) zu erkennen. Die Zeit für das Bearbeiten der XPath-Anfrage scheint geringfügig von der Länge der Elementnamen abhängig zu sein. Das bedeutet, kurze Elementnamen sind effizienter in der Verarbeitung als Längere. Da die Unterschiede aber auch an dieser Stelle nur minimal sind, müssen sie nicht besonders berücksichtigt werden.

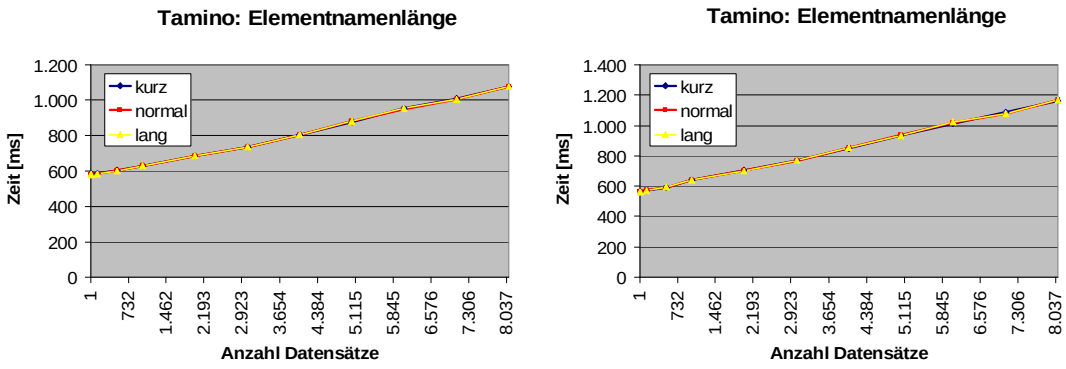

Abbildung 28: Diagramm für die XQuery-Anfragen: `COUNT` (links) und `FLWR` (rechts)

Entgegen der festgestellten Unterschiede bei der XPath-Anfrage verhalten sich bei den XQuery-Anfragen (Abb. 28) alle drei Namenslängen gleich. Die Zeit für das Auswerten der XQueries ist somit unabhängig von der Länge der Elementnamen. Dass bedeutet, dass bei Tamino abgesehen von den kleinen Differenzen bei der Anfrage mittels des XPath-Ausdruckes keine Effizienzunterschiede bei der Verwendung von

langen, normalen oder sehr kurzen Elementnamen festzustellen sind. Dies wiederum sagt aus, dass die Länge keinen Einfluss auf die Performanz hat.

5.3.3 Viper

Analog zu den Auswertungen des Tests auf den beiden vorhergehenden Datenbanksystemen ist auch bei Viper die Zeit für die Speicherung der XML-Dokumente unabhängig von der Länge der Elementnamen.

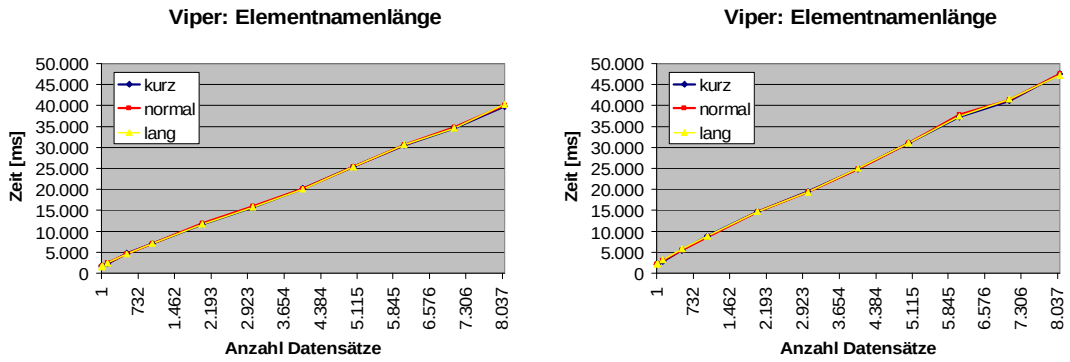

Abbildung 29: Diagramm für das Speichern ohne (links) und mit Validierung (rechts)

Diese Aussage trifft auch auf beide Insert-Varianten zu. Die zugehörigen Diagramme für die Speicherung mit und ohne Schemadefinition und Validierung sind in Abbildung 29 zu sehen.

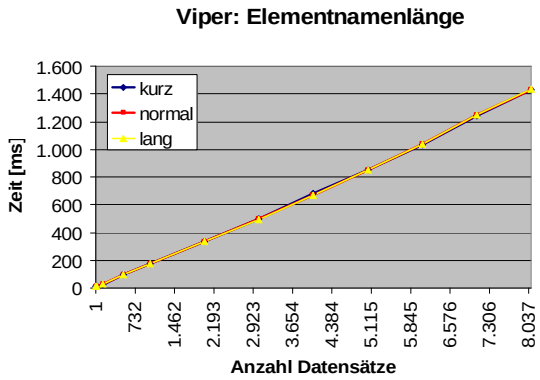

Abbildung 30: Diagramm für die Bearbeitung der XPath-Anfrage

Bei den verschiedenen Anfragen auf die XML-Dokumente sind in den Diagrammen ebenfalls keine zeitlichen Unterschiede zu erkennen. Sowohl die Anfrage mittels eines XPath-Ausdruckes (vgl. Abb. 30) als auch die XQueries (vgl. Abb. 31) werden bei Viper unabhängig von der Namenslänge verarbeitet. Das bedeutet, dass bei dem Datenbanksystem DB2 V9.1 nicht nur bei der Speicherung sondern auch beim Anfragen die Bearbeitungszeit der unterschiedlichen Dokumente etwa gleich lange dauert. Es

hat somit keinen Einfluss auf die Effizienz, ob lange oder kurze Namen verwendet werden.

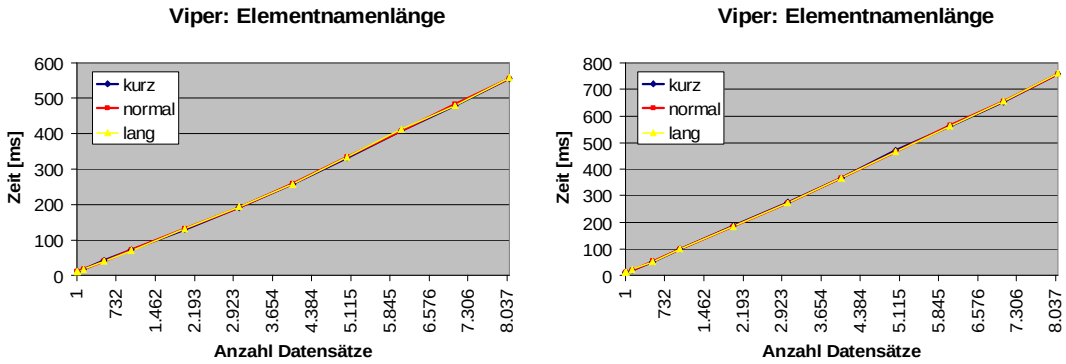

Abbildung 31: Diagramm für die XQuery-Anfragen: `COUNT` (links) und `FLWR` (rechts)

5.3.4 Auswertung

Das Resultat dieses Tests ist, dass keines der verwendeten Datenbanksysteme signifikante Unterschiede zwischen den einzelnen Namenslängen aufweist. Bei DB2 V8 und bei Tamino sind geringfügige Unterschiede bei der XPAth-Anfrage festzustellen. Diese Unterschiede sind aus Sicht der Effizienz jedoch verhältnismäßig klein. Alle anderen Tests zur Untersuchung des Einflusses der Elementnamenlänge liefern das gleiche Ergebnis. Die Performanz ist nahezu komplett unabhängig von der Länge der Elementnamen. Somit liegt es im Ermessen der Nutzers, wie lang er seine Namen wählt, solange sie in einem überschaubaren (ungefähr bis 10 Zeichen) Rahmen bleiben. An dieser Stelle soll darauf hingewiesen werden, dass die Namen oder Bezeichner der Elemente aus Gründen der Übersichtlichkeit weder zu kurz noch unnötig lang sein sollten. Wichtig ist, dass sie sowohl aussagekräftig als auch eindeutig sind.

5.4 Einfluss von Kommentaren

Mit Hilfe dieses Tests wurde untersucht, ob und wenn ja wie groß der Einfluss von Kommentaren innerhalb eines XML-Dokumentes auf die Performanz bei der Verarbeitung selbiger ist. Der Unterschied dieses Tests zu allen anderen ist, dass das Dokument mit Kommentaren entsprechend größer war (7,5 MB zu ungefähr 5,8 MB) und somit allein aus diesem Grund beim Einfügen und Anfragen etwas mehr Zeit benötigt wird. Trotz dieser Tatsache ist die Bewertung der Performanz bezüglich der `INSERT`- und `QUERY`-Operationen möglich. Die Beschreibung des Tests „Einfluss von Kommentaren“ ist in Kapitel 4.2.3 nachzulesen.

5.4.1 DB2 V8

Auf dem Datenbanksystem DB2 V8 wird auch bei diesem Test nur das Einfügen der Dokumente in die Datenbank und das Anfragen mittels eines XPath-Ausdruckes zur Bewertung herangezogen.

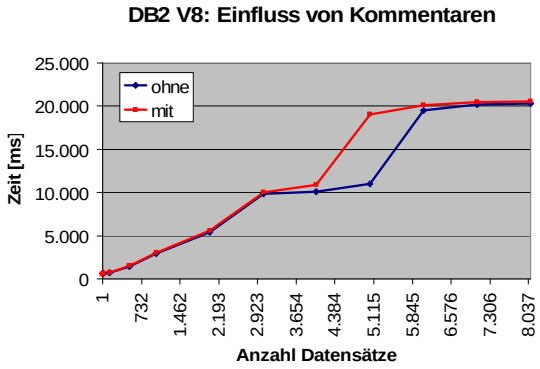

Abbildung 32: Diagramm für die Speicherung der Dokumente

Im Diagramm für die Speicherung der jeweiligen Dokumente (Abbildung 32) ist der Einfluss der unterschiedlichen Größen deutlich zu sehen. Die Dokumente mit Kommentaren benötigen etwas mehr Zeit für die Speicherung und erreichen auch eher ein Seitenende im Hauptspeicher, sodass der Sprung auf eine neue Seite dementsprechend zeitiger erfolgt. In diesem konkreten Fall ist für die Dokumente mit Kommentaren das Seitenende bereits nach ungefähr 4000 Datensätzen erreicht und für jene ohne Kommentare erst nach etwa 5200 Datensätzen. Ansonsten ist zu erkennen, dass der Einfluss von Kommentaren auf die Zeit zur Speicherung der XML-Dokumente vernachlässigbar gering ist. Es wird nur geringfügig mehr Zeit benötigt, da die Dokumente mit Kommentaren größer sind. Wie auch bei den vorhergehenden Tests auf DB2 V8 ist dies mit der Speicherung der Dokumente als CLOB zu erklären.

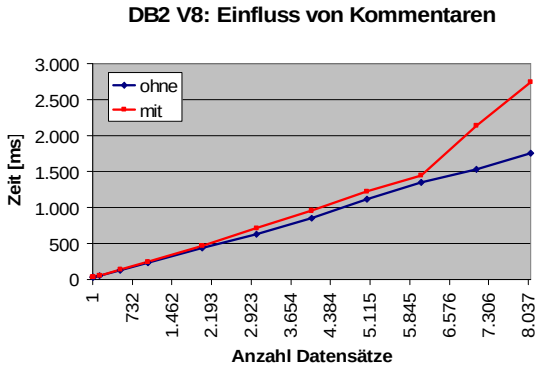

Abbildung 33: Diagramm für die Bearbeitung der XPath-Anfrage

TODO @Meike: An dieser Stelle muss noch die Auswertung der XPath-Anfrage erfolgen. Ich wollte gerne deine Meinung dazu hören, da in dem Diagramm für die Dokumente mit Kommentaren am Ende eine interessante Erscheinung auftritt, die ich nicht wirklich interpretieren kann. Entweder du hast eine Erklärung dafür oder ich vernachlässige diese Erscheinung und bewerte den Test auf Grundlage der anderen Werte.

5.4.2 Tamino

Die Auswertung der Zeiten für die Speicherung bei Tamino ergibt folgendes Ergebnis: Sowohl für die Speicherung ohne Schemadefinition und Validierung (Abb. 32

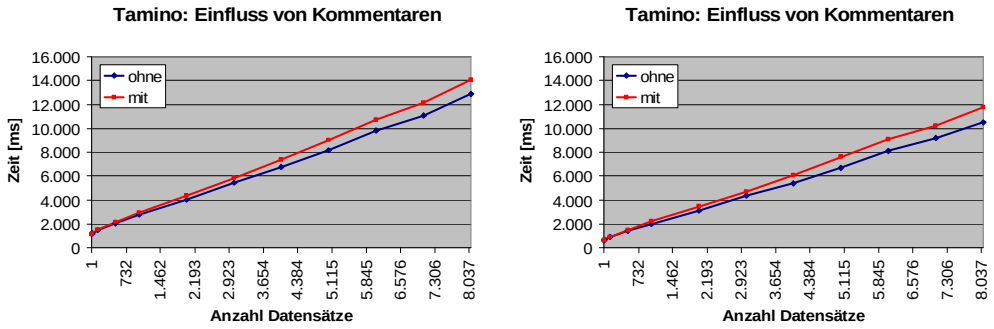

Abbildung 34: Diagramm für das Speichern ohne (links) und mit Validierung (rechts)

links) als auch für die Speicherung mit Schemavalidierung (Abb. 32 rechts) sind die Kommentare in den Dokumenten bedeutend für die Effizienz der Operation. Der Unterschied zwischen den beiden Graphen ist jedoch größtenteils auf die unterschiedlichen Dokumentgrößen zurückzuführen. Durch die Verwendung von Kommentaren in XML-Dokumenten ergibt sich somit weder eine entscheidende Effizienzverbesserung (was auch sehr unerwartet gewesen wäre) noch eine signifikante Verschlechterung bezüglich des Einfügens. Zu beachten ist jedoch, dass der Informationgehalt innerhalb eines Dokumentes bei gleicher Größe der Dokumente bei den Dokumenten mit Kommentaren geringer wird.

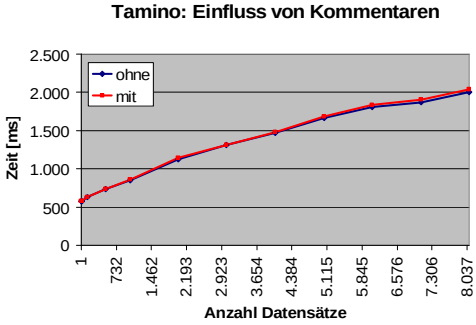

Abbildung 35: Diagramm für die Bearbeitung der XPath-Anfrage

Für Anfragen ergibt sich ein ähnliches Bild wie bei der Speicherung der XML-Dokumente. Auch die Auswerung der XPath-Anfrage (vgl. Diagramm in Abb. 35) benötigt bei den Dokumenten mit Kommentaren etwas mehr Zeit als bei denen ohne Kommentare. Bei den XQuery-Anfragen (Abb. 36) werden die Dokumente ohne

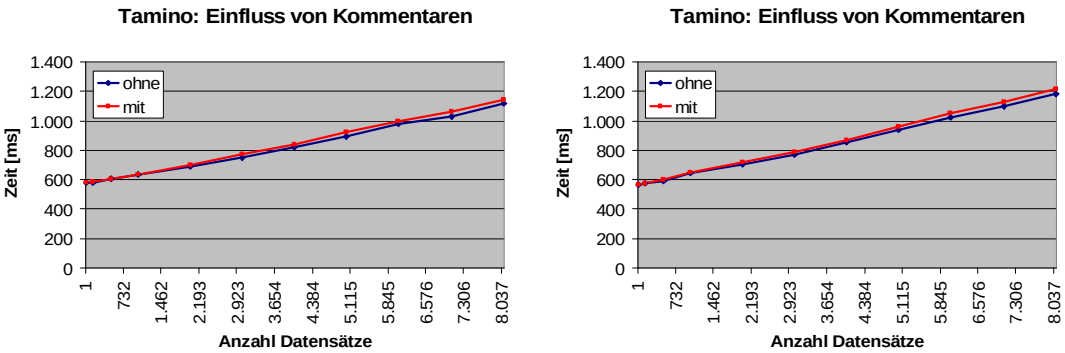

Abbildung 36: Diagramm für die XQuery-Anfragen: `COUNT` (links) und `FLWR` (rechts)

Kommentare ebenfalls schneller verarbeitet. Dies liegt jedoch wiederum an den unterschiedlichen Dokumentgrößen. Als Ergebnis der Auswertung des Einflusses von Kommentaren auf die Effizienz bei der Verarbeitung von XML-Dokumenten ergibt sich somit für Tamino, dass die Verwendung von Kommentaren unbedeutend ist, solange die Dokumente durch die Kommentare nicht wesentlich größer werden. Mit Verarbeitung ist hierbei sowohl die Speicherung als auch das Anfragen der Dokumente gemeint.

5.4.3 Viper

Die Auswertung des Tests für Kommentare in XML-Dokumenten liefert auf dem Datenbanksystem DB2 V9.1 annähernd das gleiche Ergebnis wie bei Tamino.

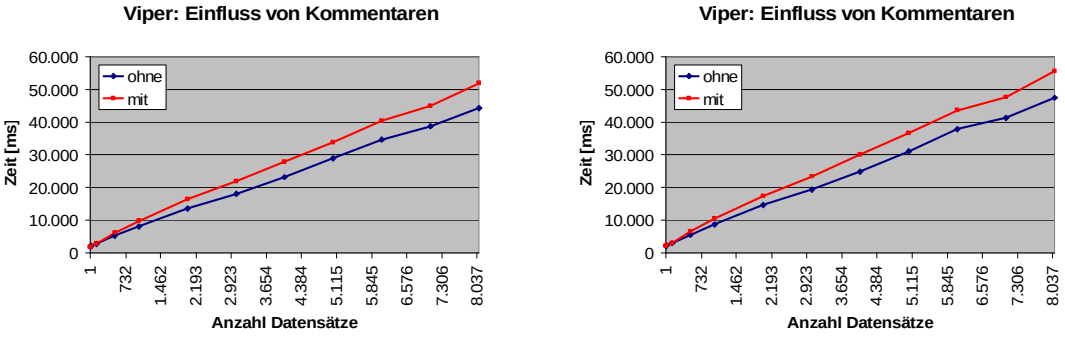

Abbildung 37: Diagramm für das Speichern ohne (links) und mit Validierung (rechts)

Auch auf diesem Testsystem resultiert der Unterschied zwischen den Dokumenten mit und ohne Kommentare bei beiden Speicherungsformen (vgl. Abb. 37) hauptsäch-

lich aus den verschiedenen Dokumentgrößen. Das bedeutet, beim Einfügen von XML-Dokumenten in das Datenbanksystem macht es keinen signifikanten Unterschied, ob das Dokument Kommentare besitzt oder nicht, solange die Dokumente ungefähr gleich groß sind. Diese Beobachtung gilt, wie bereits angedeutet und auch in den beiden Diagrammen in Abbildung 37 zu sehen, sowohl für die Speicherung mit und ohne XML-Schemavalidierung. Zu beachten ist jedoch, sobald die Kommentare einen entscheidenen Einfluss auf die Größe der Dokumente ausüben, wird auch die Effizienz beim Einfügen beeinträchtigt.

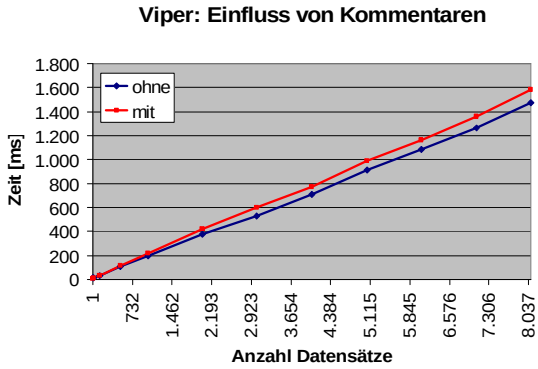

Abbildung 38: Diagramm für die Bearbeitung der XPath-Anfrage

Die drei unterschiedlichen Queries zeigen abgesehen von anfragebedingten, zeitlichen Unterschieden das gleiche Ergebnis. Der XPath-Ausdruck benötigt für das Anfragen des großen Dokumentes ungefähr 1,5 sec (Abb. 38), die `COUNT`-Anfrage 700 msec und die `FLWR`-Anfrage 800 msec (Abb. 39). Wie auch bei Tamino zeigen die Diagramme jediglich einen Unterschied, der hauptsächlich auf die Dokumentgröße zurück zu führen ist, auch wenn die Differenz zwischen den beiden Graphen bei DB2 Viper etwas größer ist.

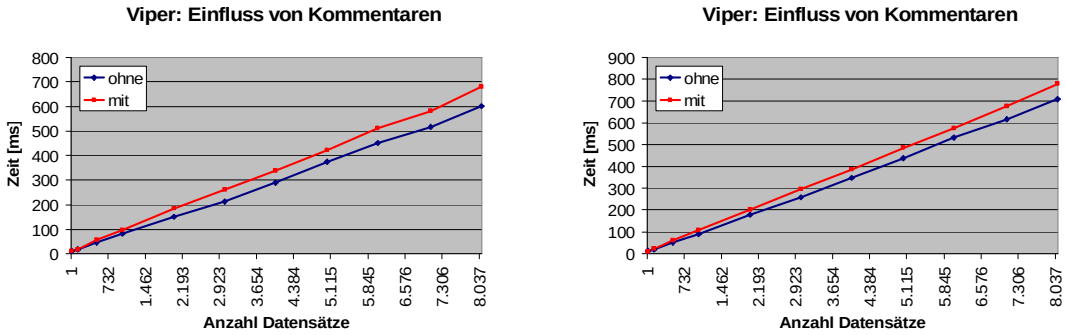

Abbildung 39: Diagramm für die XQuery-Anfragen: `COUNT` (links) und `FLWR` (rechts)

Das heißt, weder bei Anfragen mittels eines XPath-Ausdruckes (Abb. 38) noch bei XQuery-Anfragen (Abb. 39) wird die Effizienz positiv oder negativ durch die Verwendung von Kommentaren beeinflusst. Es ist kein signifikanter Unterschied zwischen

beiden Strukturen (XML-Dokumente mit und ohne Kommentaren) zu erkennen. Der aber auftretende Unterschied ist mit den verschiedenen Dokumentgrößen zu erkären und vergrößert sich entsprechend der Differenz der Dokumentgrößen.

5.4.4 Auswertung

Die Untersuchungen des Einflusses von Kommentaren in einem XML-Dokumenten haben ergeben, dass die Effizienz vom Vorhandensein von Kommentaren abhängig ist. Das bedeutet, auf alle drei Systemen, auf denen der Tests durchgeführt wurde, war die Verarbeitung der Dokumente ohne Kommentare effizienter als die Verarbeitung der Dokumente mit Kommentaren. Diese Erscheinung ist jedoch im Wesentlichen auf die unterschiedlichen Dokumentgrößen zurückzuführen. Das Dokument mit 8076 Datensätzen war ohne Kommentare ungefähr 5,8 MB und mit Kommentaren 7,5 MB groß. Hervorzuheben ist, dass bei den Tests mit Tamino die zeitlichen Unterschiede zwischen den beiden Strukturen beim Anfragen der Dokumente verhältnismäßig gering waren. Das heißt, bei Tamino war die Bearbeitung der Anfragen für die Dokumente mit Kommentaren trotz des Größenunterschiedes annähernd gleich schnell. Es bleibt außerdem festzuhalten, solange die Dokumente mit und ohne Kommentare ungefähr gleich groß sind, ist der Einfluß der Kommentare auf die Effizienz sowohl hinsichtlich des Einfügens als auch hinsichtlich des Anfragens vernachlässigbar gering. Sobald das Dokument mit Kommentaren jedoch wesentlich größer ist, erhöht sich auch der negative Einfluss der Kommentare auf die Effizienz bei der Verarbeitung der Dokumente. An dieser Stelle sei noch einmal darauf hingewiesen, dass der Informationsgehalt von Dokumenten vergleichbarer Größe bei dem Dokument mit Kommentaren dementsprechend geringer ist. Aus Effizienzsicht sollten Kommentare somit möglichst sparsam verwendet werden.

5.5 Schachtelungstiefe der Dokumente

Dieser Test diente zur Untersuchung des Einflusses unterschiedlicher Hierarchietiefen in XML-Dokumenten. Dabei wurden auf der einen Seite Dokumente mit einer Schachtelungstiefe von 2 und auf der anderen Seite Dokumente mit einer Schachtelungstiefe von 4 miteinander verglichen. Aber auch bei dieser relativ geringen Differenz wurden aussagekräfte Ergebnisse erzielt. Für ausf"hrliche Informationen über diesen Test, sei auf Kapitel 4.2.4 verwiesen, in dem der Test detailiert vorgestellt wird. Auch bei diesem Test wurden die Untersuchungen auf allen drei Datenbanksystemen durchgeführt und diese werden im folgenden ausgewertet.

5.5.1 DB2 V8

Zu Beginn erfolgt wie üblich die Auswertung hinsichtlich des Systems DB2 V8. Für die Speicherung der Dokumente ergibt sich das bei DB2 V8 bereits bekannte Bild. Wie schon bei den vorherigen Tests ist bei der Speicherung der verschiedenen Dokumente in die Datenbank kein Unterschied zu erkennen (vgl. Diagramm in Abbildung

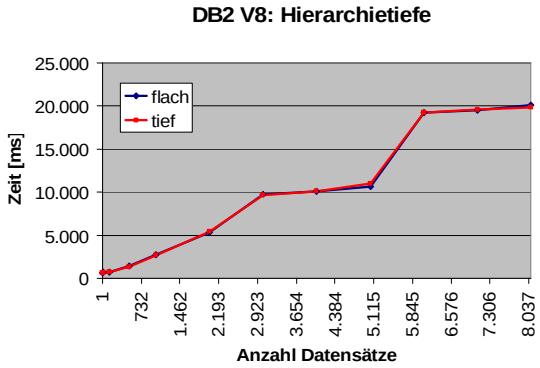

Abbildung 40: Diagramm für die Speicherung der Dokumente

40). Sowohl die Dokumente mit flacher Hierarchie als auch die mit verschachtelter Hierarchie liefern nahezu die gleichen Zeiten für die Speicherung, was auf das Einfügen als Datentyp CLOB zurückzuführen ist. Hierbei werden die Dokumente unabhängig von der Hierarchietiefe hintereinander eingelesen und die Zeit für das Einfügen der Dokumente variiert nur in Abhängigkeit von ihrer Größe. Ebenfalls bekannt ist der Effekt der vollen Seiten bei der Speicherung der Dokumente. Nach ungefähr 5000 Datensätzen ist bei diesem Test eine Seite im Hauptspeicher gefüllt und das System ist gezwungen, eine neue Seite zu beginnen. Aus diesem Grund ist der Sprung bei den Zeiten für die Speicherung im Diagramm (Abb. 40) sichtbar.

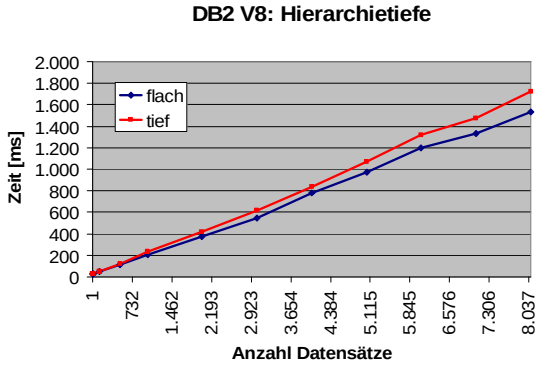

Abbildung 41: Diagramm für die Bearbeitung der XPath-Anfrage

Das Diagramm für das Anfragen der Dokumente mit einem XPath-Ausdruck (vgl. Abb. 41) zeigt ebenfalls ein eindeutiges Ergebnis. Aufgrund der unterschiedlichen Struktur der Dokumente ist die Effizienz beim Anfragen der Dokumente mit weniger Leveln besser als bei denen mit vielen Schachtelungsleveln. Selbstverständlich wird der Unterschied abhängig von der Dokumentgröße immer auffälliger. Das bedeutet, um so größer die verarbeiteten Dokumente sind um so größer wird der Einfluss der Schachtelungstiefe auf die Effizienz beim Anfragen der Dokumente. Bei DB2 V8 ist entsprechend den Testergebnissen die Effizienz für das Anfragen verschachtelter

Dokumente im Gegensatz zur Effizienz beim Einfügen jener von der Hierarchietiefe abhängig. Stark geschachtelte Dokumente weisen eine schlechtere Performanz beim Anfragen mit XPath-Ausdrücken auf.

5.5.2 Tamino

Auf diesem System wurde im Gegensatz zu DB2 V8 erneut auch die Speicherung mit vorhergehender XML-Schemavalidierung und die Auswirkungen bei XQueries getestet. Zunächst erfolgt aber die Auswertung der Speicherung für die bei verschiedenen Strukturen.

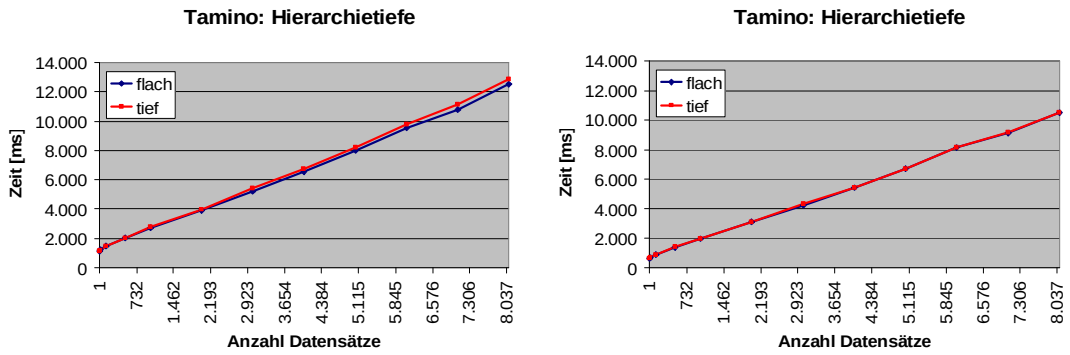

Abbildung 42: Diagramm für das Speichern ohne (links) und mit Validierung (rechts)

Bei den beiden gegenüberstehenden Diagrammen in Abbildung 42 ist ein leichter Unterschied zu erkennen. Beim Einfügen der Dokumente ohne Schemavalidierung (links) verhalten sich die Dokumente mit einer flachen Hierarchietiefe minimal effizienter. Die Differenz zwischen beiden Strukturen ist allerdings unbedeutend gering und bei der Speicherung mit Schemavalidierung (rechts) gar nicht mehr vorhanden. Bei dieser Art der Speicherung benötigen beide Strukturen annähernd die gleichen Zeiten. Das bedeutet für die Speicherung der verschiedenen Dokumente, dass diejenigen mit einer flacheren Struktur auf Tamino geringfügig effizienter gespeichert werden, dieser Unterschied ist jedoch vernachlässigbar.

Das Ergebnis für die drei Query-Operationen ist ähnlich dem Ergebnis des Einfügens der Dokumente ohne XML-Schemavalidierung und Definition. Das bedeutet, sowohl bei der XPath-Anfrage (vgl. Diagramm in Abbildung 43) als auch bei beiden XQuery-Anfragen (siehe Abb. 44) werden die Dokumente mit einer geringeren Schachtelungstiefe effizienter verarbeitet als die übrigen. Der Unterschied zwischen beiden Strukturen ist jedoch bei allen drei Anfragen sehr gering.
Beim Anfragen mittels des XPath-Ausdruckes (Abb. 43) ist die Differenz kaum erkennbar und auch bei den übrigen Anfragen ist sie sehr klein, aber kontinuierlich. Für die Auswertung des Tests heißt das, um so größer die Dokumente werden, um

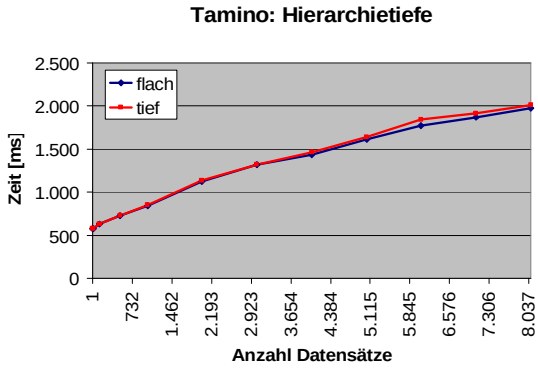

Abbildung 43: Diagramm für die Bearbeitung der XPath-Anfrage

so größer wird auch der Einfluss der Schachtelungstiefe. Somit ist neben der Effizienz beim Einfügen auch die Effizienz beim Anfragen bei Tamino von der Anzahl der Hierarchielevel abhängig. Der Unterschied ist allerdings bei kleinen Dokumenten unbedeutend und gewinnt mit zunehmender Größe der Dokumente an Einfluss.

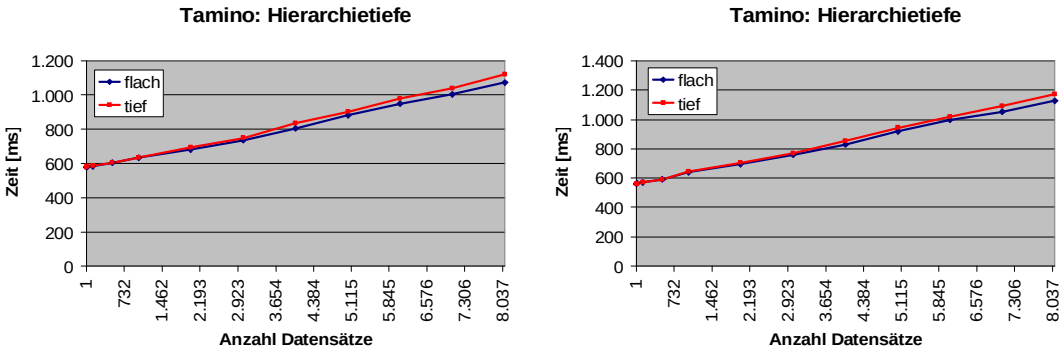

Abbildung 44: Diagramm für die XQuery-Anfragen: `COUNT` (links) und `FLWR` (rechts)

5.5.3 Viper

Entgegen der Beobachtungen bei Tamino sind bei Viper sowohl bei der Speicherung als auch beim Anfragen der XML-Dokumente deutliche Unterschiede erkennbar. Zunächst aber die Erläuterungen zu den Ergebnissen für die Speicherung der Dokumente. Bei der Auswertung der einzelnen Zeiten für den Test verhalten sich beide Abspeicherungsvarianten (mit und ohne Schemavalidierung) äquivalent. Sowohl bei der Speicherung mit Schemavalidierung als auch bei der Speicherung ohne werden die Dokumente mit einer flachen Hierarchie schneller verarbeitet (vgl. Diagramme in Abbildung 45). Die Speicherung dieser Dokumente benötigt ungefähr 10 Prozent weniger Zeit. Wie auch bei den meisten vorhergehenden Tests ist in beiden Diagrammen für die Speicherung eine lineare Abhängigkeit zwischen der Speicherungszeit und der Anzahl der Datensätze zu erkennen. Weiterhin ist zu erkennen,

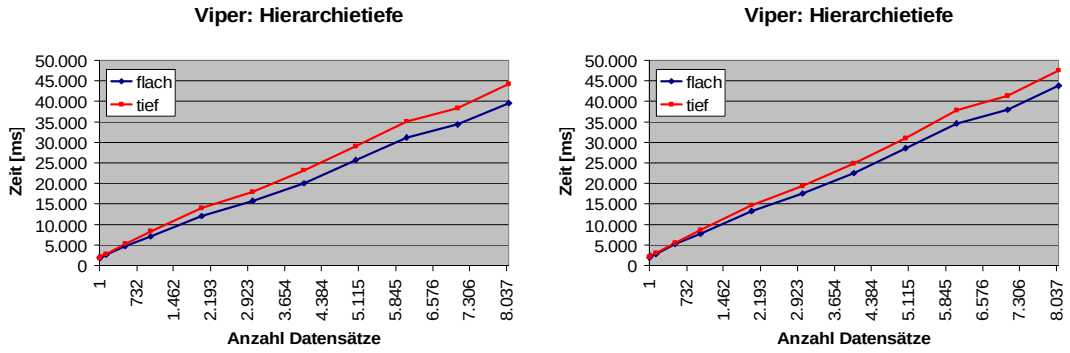

Abbildung 45: Diagramm für das Speichern ohne (links) und mit Validierung (rechts)

desto größer die Dokumente werden, desto offensichtlicher wird der Einfluss der Hierarchietiefe. Dies gilt sowohl für die Speicherung mit Schemavalidierung als auch für jene ohne XML-Schemavalidierung. Verallgemeinernd lässt sich für Viper somit sagen, dass die Zeit für die Speicherung direkt von der Anzahl der Hierarchielevel im zu speichernden Dokument abhängt.

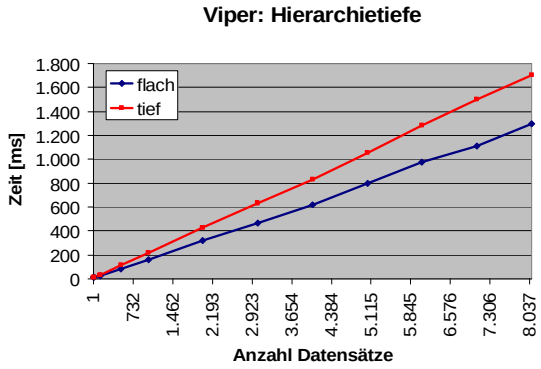

Abbildung 46: Diagramm für die Bearbeitung der XPath-Anfrage

Für das Anfragen der verschiedenen XML-Dokumente ergibt sich bei DB2 Viper genau das gleiche Ergebnis mit dem Unterschied, dass die prozentuale Differenz zwischen den Zeiten für beide Strukturen größer ist. Bei der XPath-Anfrage (siehe Abb. 46) beträgt die Differenz zwischen den Zeiten für das Anfragen der flachen Dokumente gegenüber den geschachtelten Dokumenten annähernd 25%. Das bedeutet, für diese Anfrage ist die Differenz deutlich größer als beim Einfügen der Dokumente. Im Vergleich zum XPath-Ausdruck sind die Abweichungen bei den beiden XQueries (`COUNT` und `FLWR`) nicht ganz so groß (vgl. Abb. 47). Es handelt sich bei den beiden Anfragen entsprechend der Diagramme um eine Abweichung von ca. 15 Prozent. Abgesehen von den prozentualen Unterschieden ist die Verarbeitung der flachen Dokumente sowohl bezüglich der Speicherung als auch bezüglich der Anfragen bei DB2

Viper entscheidend effizienter.

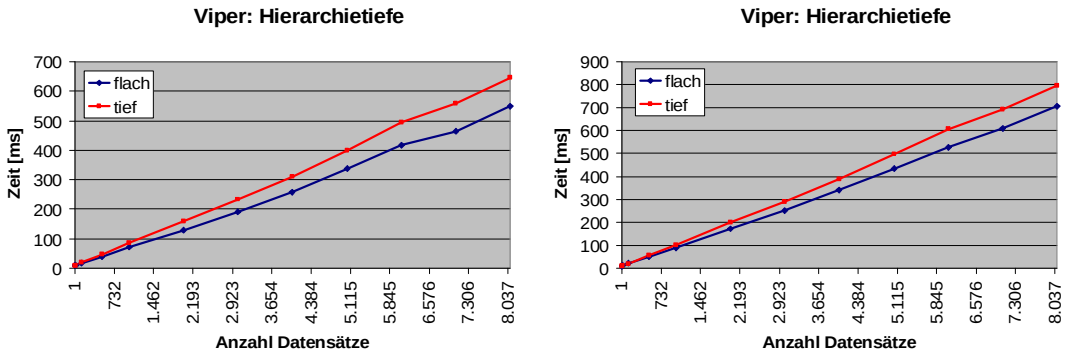

Abbildung 47: Diagramm für die XQuery-Anfragen: `COUNT` (links) und `FLWR` (rechts)

5.5.4 Auswertung

Aus den Beobachtungen zu diesem Test geht hervor, dass die Hierarchietiefe der verarbeiteten Dokumente direkten Einfluss auf die Effizienz für das Speichern und vor allem für das Anfragen hat. Bei Tamino ist der Unterschied nicht ganz so auffällig wie bei DB2 V8 (XPath-Anfrage) und Viper, aber trotzdem vorhanden. Das bedeutet, aus Effizienzsicht sollte beim Entwurf von XML-Dokumenten auf ihre Schachtelungstiefe geachtet werden, denn um so größer die Anzahl der Level ist, um so ineffizienter werden die Dokumente bei der Verarbeitung. Natürlich ist damit nicht die Schlussfolgerung verbunden, dass Dokumente keine Verschachtelungen enthalten sollten, aber bezogen auf die Effizienz bei der Verarbeitung der Dokumente sollte die Tiefe möglichst gering sein.

Die Beobachtungen konnten aus den in Kapitel 4.2.4 genannten Gründen nur für zwei und vier Level gemacht werden. Es ist aber zu erwarten, dass die Effizienz noch weiter abnimmt, wenn die Anzahl der Schachtelungslevel steigt.
Abschließend soll nochmal darauf hingewiesen werden, dass die Ergebnisse für diesen Test von den verschiedenen Systemen abhängig waren. Beim Tamino XML Server waren die Unterschiede zwischen den beiden Strukturen beispielsweise nicht so auffällig wie bei Viper.

5.6 Kompakte vs. verteilte Speicherung

Mit Hilfe dieses Tests sollte herausgefunden werden, ob wenig große Dokumente oder viele kleine Dokumente effizienter in der Verarbeitung sind. Zur Durchführung des Tests wurde ein großen Dokument in viele kleine mit gleichem Inhalt unterteilt. Der exakte Ablauf des Tests ist in Kapitel 4.2.5 beschrieben.

5.6.1 DB2 V8

Auch bei diesem Test wird auf dem Datenbanksystem DB2 V8 nur die Speicherung ohne Schemavalidierung und das Anfragen mit einem XPath-Ausdruck untersucht.

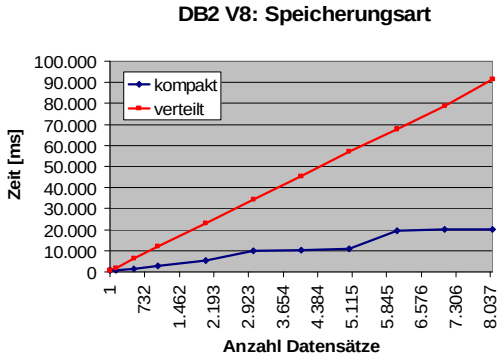

Abbildung 48: Diagramm für die Speicherung der beiden Dokumente

Für die Speicherung der XML-Dokumente ergibt sich nach der Auswertung der benötigten Zeiten das Diagramm aus Abbildung 48. Es ist ganz deutlich zu erkennen, dass die Speicherung derselben Informationen in einem großen Dokument wesentlich effizienter ist als in vielen kleinen Dokumenten. Die Speicherung in vielen kleinen Dokumenten benötigt größtenteils viermal soviel Zeit wie in einem großen Dokument. Weiterhin ist in dem Diagramm (Abb. 48) gut zu erkennen, dass bei der Speicherung vieler kleiner Dokumente der Effekt der vollen Seiten im Hauptspeicher nicht so deutlich wie in den vorherigen Tests auftritt. Bei der Speicherung des großen Dokumentes ist er hingegen erneut sehr offensichtlich. Dies ist bei DB2 V8 auch ein entscheidener Unterschied der beiden verschiedenen Strukturen. Mit dieser Erscheinung lässt sich auch der lineare Verlauf des Graphen zwischen der Dokumentanzahl und Speicherungszeit für die kleinen Dokumente erklären und dementsprechend der nicht lineare Verlauf für die Speicherung der großen Dokumente. Hier verhindern die bereits beschriebenen Sprünge auf neue Seiten einen solchen Verlauf.

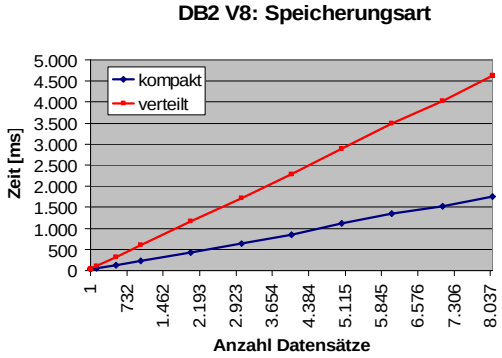

Abbildung 49: Diagramm für die Bearbeitung der XPath-Anfrage

In dem Diagramm für die Anfrage der verschieden XML-Dokumente mit dem XPath-Ausdruck (vgl. Abb. 49) ist der Unterschied zwischen den beiden Dokumentstrukturen ebenso deutlich wie bei der Speicherung zu sehen. Das große Dokument ist wesentlich effizienter bei der Auswertung der XPath-Anfrage. Im Gegensatz zur Speicherung der Dokumente liefern bei der Anfrage beide Strukturen annähernd einen linearen Verlauf zwischen der Anzahl der Datensätze (bzw. einzelnen Dokumente) und der für das Auswerten benötigten Zeit. Das große Dokument wird ungefähr dreimal so schnell verarbeitet, wie die einzelnen kleinen Dokumente. Aus Sicht dieser Untersuchungsergebnisse wäre die Schlussfolgerung, dass große Dokumente bei DB2 V8 wesentlich effizienter in der Verarbeitung sind als viele kleine, möglich. Dies gilt dann sowohl für die Speicherung als auch für das Anfragen mit XPath-Ausdrücken. An dieser Stelle sei jedoch darauf hingewiesen, dass in diesem konkreten Testfall (verteilte Speicherung vs. kompakte Speicherung der Informationen) noch andere Faktoren eine entscheidende Rolle für die Effizienz spielen. Mehr Informationen dazu sind in der allgemeinen Auswertung dieses Tests in Kapitel 5.6.4 zu finden.

5.6.2 Tamino

Zur Bewertung des Tests „kompakte versus verteilte Speicherung“ wird im Gegensatz zu den anderen Tests auch bei Tamino und Viper nur die Speicherung ohne Schemadefinition und Validierung herangezogen. Der Grund hierfür ist in Kapitel 4.2.5 nachzulesen.

Auf Grund der Beschränkung von 20 MB Datenbankspeicher bei Tamino konnte der Test für die einzelnen kleinen Dokumente nur bis zu einer Anzahl von 4038 Dokumenten durchgeführt werden. Die Beschränkung des Datenbankspeichers hängt damit zusammen, dass es sich bei der verwendeten Version 4.2.1 des Tamino XML Servers um eine Testversion handelt. Da aber auch bei der getesteten Menge von Dokumenten ein deutliches Ergebnis zu erkennen ist, stellt die Beschränkung des Speichers keine Einschränkung der Bewertung des Tests dar.

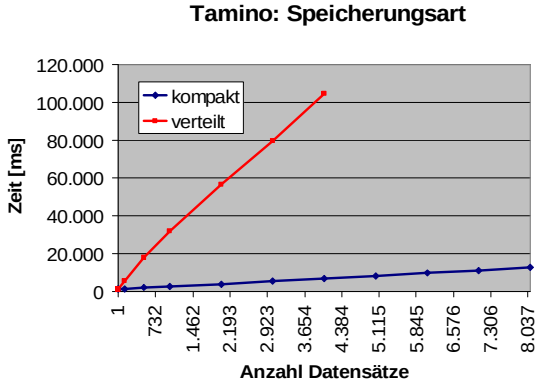

Abbildung 50: Diagramm für das Speichern ohne (links) und mit Validierung (rechts)

Bei der Speicherung der verschiedenen Dokumente ist wie auch bei DB2 V8 ein enormer Vorteil des großen Dokumentes gegenüber den vielen kleinen Dokumenten festzustellen. Die Speicherung des großen Dokumentes mit denselben Informationen benötigt ungefähr ein Zehntel der Zeit für die Speicherung der vielen kleinen Dokumente. Somit ist die Speicherung großer Dokumente laut dieses Testergebnisses wesentlich effizienter als die Speicherung vieler kleiner Dokumente.

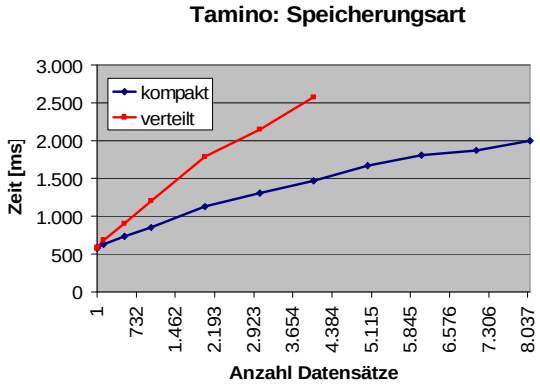

Abbildung 51: Diagramm für die Bearbeitung der XPath-Anfrage

Auch die Ergebnisse der drei verschiedenen Anfragen auf die Dokumente zeigen dieses Ergebnis. Sowohl die Ergebnisse der XPath-Anfrage (Abb. 51) als auch die der beiden XQueries (vgl. Diagramme in Abbildung 52) zeigen entscheidende Effizienzvorteile bei der Verarbeitung der großen Dokumente. Bei der Anfrage mittels eines XPath-Ausdruckes benötigen die kleinen Dokumente fast die doppelte Zeit wie das große Dokument. Im Gegensatz dazu benötigen die beiden XQuery-Anfragen (`COUNT` und `FLWR`) bei dem einzelnen großen Dokument ziemlich genau die Hälfte der Zeit. Trotz dieses minimalen Unterschiedes ist auch das Ergebnis der verschiedenen Anfragen sehr deutlich und eindeutig.

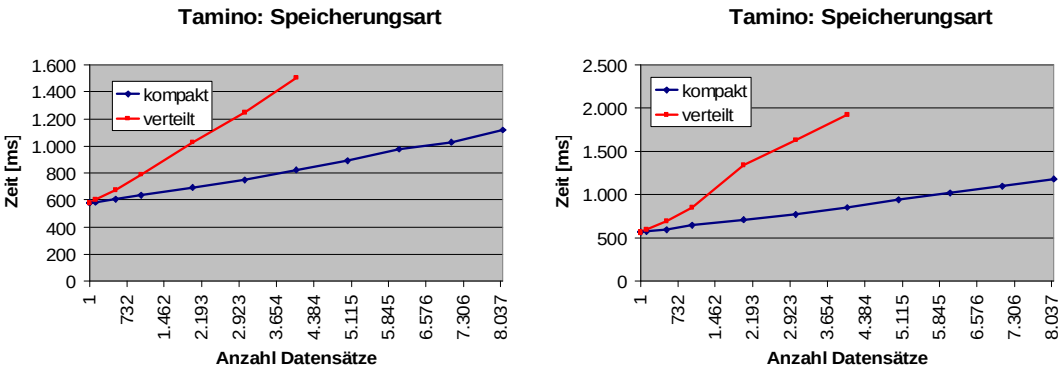

Abbildung 52: Diagramm für die XQuery-Anfragen: `COUNT` (links) und `FLWR` (rechts)

x Somit ist das Resultat des Tests „kompakte versus verteilte Speicherung“ für Ta-

mino identisch zu dem Ergebnis von DB2 V8. Die Effizienz für die Verarbeitung (Speicherung und Anfragen) großer Dokumente ist wesentlich besser als die für viele kleine Dokumente. Aber auch hier sei wie schon bei DB2 V8 und später bei DB2 Viper auf andere Faktoren hingewiesen, die die Effizienz bei vielen kleinen Dokumenten im Vergleich zu einem großen Dokument entscheidend beeinflussen. Eine genaue Erläuterung dazu erfolgt in Kapitel 5.6.4.

5.6.3 Viper

Abschließend erfolgt die Auswerung der verschiedenen Testläufe zu dem Test „kompakte versus verteilte Speicherung“ auf DB2 Viper.

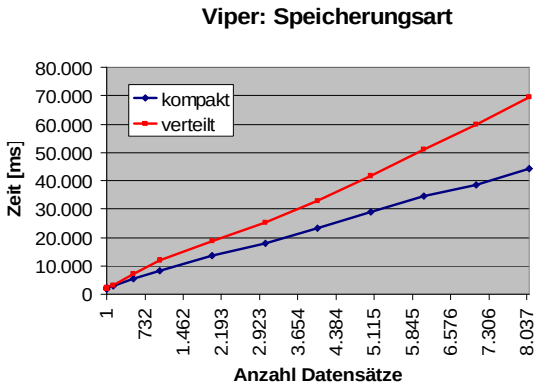

Abbildung 53: Diagramm für die Speicherung der Dokumente

Auch bei diesem Datenbanksystem ist die Speicherung eines großen Dokumentes effizienter als die Speicherung vieler kleiner XML-Dokumente. Das Diagramm zur Auswertung dieses Tests ist in Abbildung 53 zu sehen. Entgegen den Ergebnissen auf den beiden anderen Systemen ist der Unterschied zwischen beiden Abspeicherungsarten (verteilt und kompakt) jedoch nicht ganz so gravierend. Die Speicherung der kleinen Dokumente benötigt bei DB2 V9.1 nicht einmal doppelt so viel Zeit wie die Speicherung des großen Dokumentes (zum Vergleich: bei DB2 V8 benötigten die kleinen Dokumente ungefähr viermal so viel Zeit und bei Tamino zehnmal so viel). Dies ist jedoch auch darauf zurückzuführen, dass die Speicherung großer Dokumente bei Viper im Allgemeinen verhältnismäßig viel Zeit benötigt. Festzuhalten bleibt jedoch auch bei Viper, dass die Speicherung großer Dokumente im Vergleich deutlich effizienter ist.
Die gleiche Beobachtung wie bei der Speicherung der Dokumente ist bei Viper auch für die Anfrage mit dem XPath-Ausdruck zu machen. Das bedeutet, die Anfrage wird für das große Dokument deutlich schneller ausgewertet. Die Differenz zwischen den beiden Abspeicherungsarten ist jedoch nicht so gravierend wie bei Tamino und DB2 V8. Davon aber abgesehen sind bei der XPath-Anfrage signifikante Nachteile

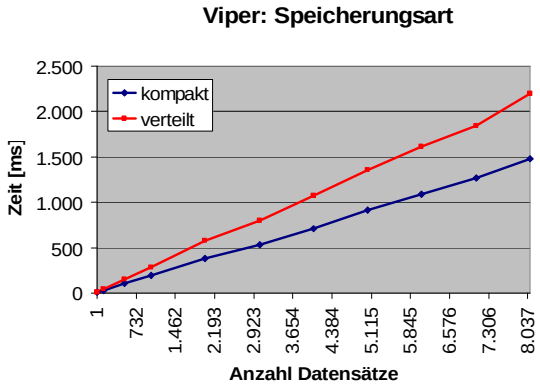

Abbildung 54: Diagramm für die Bearbeitung der XPath-Anfrage

in der Effizienz von vielen kleinen Dokumenten zu erkennen. Dies gilt nicht nur für die XPath-Anfrage sondern auch für die beiden verschiedenen XQueries (vgl. Abb. 55).

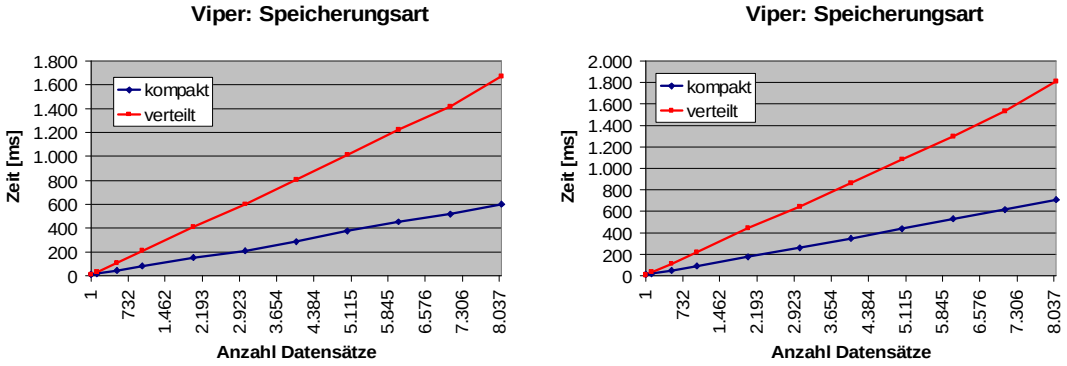

Abbildung 55: Diagramm für die XQuery-Anfragen: `COUNT` (links) und `FLWR` (rechts)

Hier ist die Differenz zwischen den beiden Strukturen wieder deutlich größer. Die Bearbeitung der beiden XQuery-Ausdrücke benötigt für die kleinen Dokumente ungefähr dreimal soviel Zeit wie für die großen Dokumente.

Entsprechend den Testergebnissen ist auch bei DB2 Viper die Effizienz hinsichtlich der Speicherung und der Anfragen für große Dokumente besser als für viele kleine Dokumente. Jedoch müssen auch bei diesem Datenbanksystem die bereits angedeuteten, effizienzbeeinflussenden Faktoren beachtet werden. Mehr zu diesen Faktoren ist in der folgenden allgemeinen Auswertung des Tests „kompakte versus verteilte Speicherung“ zu finden [Kapitel 5.6.4].

5.6.4 Auswertung

Der Test „kompakte versus verteilte Speicherung“ hat auf allen drei verwendeteten Systemen zu dem gleichen Ergebnis geführt. Die Untersuchungen haben gezeigt, dass die Verarbeitung von großen Dokumenten entscheidend effizienter ist als von vielen kleinen. Das bedeutet, der gleiche Inhalt in einem großen Dokument entgegen vielen kleinen Dokumenten wird sowohl effizienter in die Datenbank eingefügt als auch effizienter angefragt. An dieser Stelle wäre die Schlussfolgerung, große XML-Dokumente zu verwenden anstatt vieler kleine, möglich. Wie bereits in den vorherigen Auswertungen zu den einzelnen Systemen angedeutet, gibt es aber noch andere Faktoren, die die Effizienz der verteilten bzw. kompakten Speicherung von XML-Inhalten entscheidend beeinflussen. Beispiele für solche Faktoren sind zum einen Indexe oder zum anderen auch das Debugging[13] von Dokumenten. Bei der Verwendung von Indexen ist es beispielsweise effizienter viele kleine Dokumente in der Datenbank zu haben, da nur so die Indexe der Datenbanksysteme ausgenutzt werden können. Wie bereits in Abschnitt 4.2 beschrieben, können diese vor allem dazu genutzt werden, um die relevanten Dokumente zu selektieren und so die gesamte Anfrage effizienter zu bearbeiten. Allerdings ist auch zu beachten, dass die Effizienz mit Indexen stark von der Selektivität der Anfrage abhängig ist. Das bedeutet, um so höher die Selektivität der Anfrage um so besser ist auch die Effizienz. Bei Anfragen mit geringer Selektivität nützen Indexe somit nicht sehr viel. Wie bereits angedeutet sind auch für die Fehlersuche viele kleine Dokumente besser geeignet als ein einziges großes Dokument. Trotz des Testergebnisses, dass gezeigt hat, dass große Dokumente sowohl beim Einfügen als auch Anfragen effizienter sind, sollten in der Praxis aus oben genannten Gründen XML-Dokumente durchnittlicher Größe bevorzugt werden.

5.7 IDs vs. KEYs

Dieser Test wurde nur auf den Datenbanksystemen DB2 Viper und Tamino durchgeführt. Der Grund dafür ist, dass für diesen Test nur die Zeit für die Speicherung mit Validierung der verschiedenen Dokumente benötigt wird und in Verbindung mit XML Column unterstützt DB2 V8 eine solche Validierung nicht (vgl. Kapitel 3.1.1). Der Test diente zur Untersuchung der Effizienzeigenschaften bei der Verwendung von ID's und IDREF's oder KEY's und KEYREF's. Die genaue Erläuterung des Vorgehens bei diesem Test ist in Kapitel 4.2.6 zu finden.

5.7.1 Tamino

Zunächst erfolgt die Auswertung des Tests auf dem nativen Datenbanksystem Tamino. Wie in Kapitel 4.2.6 bereits beschrieben, unterstützt der Tamino XML Server Version 4.2.1 nur die Schemavalidierung für ID's/IDREF's und nicht für KEY's/ KEYREF's. Aus diesem Grund kann lediglich ein Vergleich zwischen der Speicherung mit der Überprüfung der ID/IDREF-Eigenschaften und der Speicherung ohne

[13] aus dem Englischen: Fehlerbeseitigung

Überprüfung dieser Eigenschaften erfolgen. Das bedeutet, es kann festgestellt werden, wie groß der zeitliche Unterschied für die Speicherung mit Validierung zwischen den oben genannten Fällen ist.

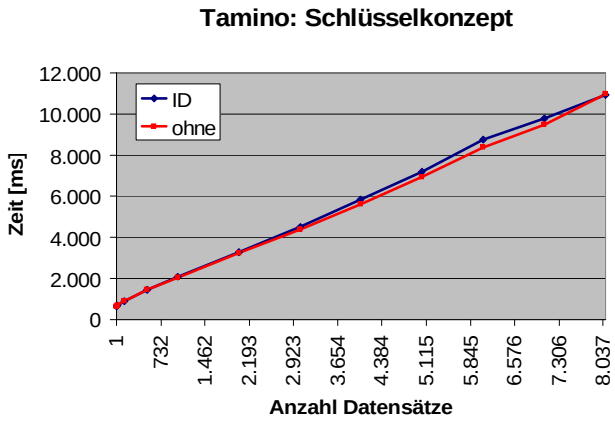

Abbildung 56: Diagramm für die Speicherung mit unterschiedl. Schlüsselkonzepten

Das Diagramm in Abbildung 56 zeigt, dass das Einfügen mit Überprüfung der Schlüsseleigenschaften bei Tamino nur unwesentlich mehr Zeit beansprucht. Das heißt, es macht aus Effizienzsicht keinen signifikanten Unterschied, ob die ID/IDREF-Eigenschaften beim Einfügen eines Dokumentes überprüft werden oder nicht.

5.7.2 DB2 Viper

Mit Hilfe von DB2 Viper konnte das Testszenario ID's vs. KEY's im Gegensatz zu Tamino komplett getestet werden. Das bedeutet, der Test umfasste sowohl das Einfügen mit Validierung der Dokumente ohne Schlüsselkonzept als auch das Einfügen der Dokumente mit ID's/IDREF's bzw. KEY's/KEYREF's.

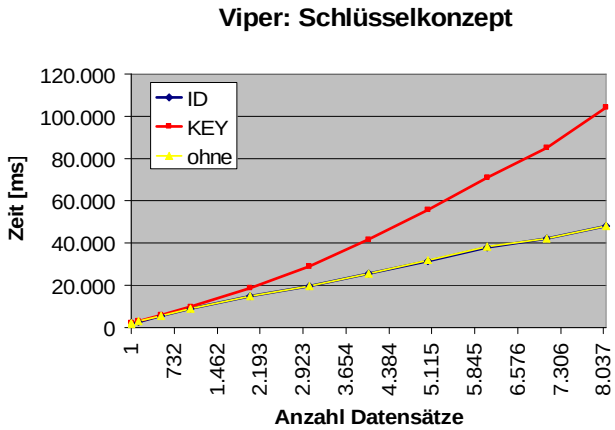

Abbildung 57: Diagramm für die Speicherung mit unterschiedl. Schlüsselkonzepten

In dem Diagramm aus Abbildung 57 ist sehr deutlich zu erkennen, dass es einen signifikanten Unterschied zwischen einerseits Dokumente mit ID's und andererseits Dokumente mit KEY's als Schlüsselkonzept gibt. Das Einfügen mit Validierung, was eine Überprüfung aller Schlüsseleigenschaften erfordert, benötigt bei der Verwendung von KEY's deutlich mehr Zeit wie bei der Verwendung von ID's. Neben diesem signifikanten Unterschied ist in dem Diagramm ebenfalls zu sehen, dass das Einfügen der Dokumente ohne Schlüsselkonzept genauso lange dauert, wie mit ID's. Das ist das gleiche Ergebnis, welches auch Tamino bei diesem Test geliefert hat. Es bleibt somit festzuhalten, dass die Effizienz für das Einfügen von XML-Dokumenten mit Schlüsselkonzept bei Viper stark von der Art des Schlüssels abhängt. Dokumente mit ID's werden wesentlich schneller in die Datenbank eingefügt als Dokumente mit KEY's.

5.7.3 Auswertung

Die Tests mit Tamino und DB2 Viper haben gezeigt, dass das Einfügen mit Validierung für ID-Dokumente keinen negativen Einfluss auf die Performanz hat. Das bedeutet, aus Effizienzsicht können in einem XML-Dokument ID's und IDREF's bedenkenlos definiert werden. Zu beachten ist dabei allerdings, dass auch alle Eigenschaften dieses Schlüsselkonzeptes im Dokument eingehalten werden müssen, denn ansonsten scheitert das Dokument bei der Schemavalidierung und wird nicht in die Datenbank eingefügt. Die wesentlichen Eigenschaften des Schlüsselkonzepts sind das zwingende Vorhandensein jeder ID, auf die durch ein IDREF verwiesen wird oder auch die Eindeutigkeit der ID's. Für die Performanz beim Einfügen von KEY-Dokumenten ist das Ergebnis ein komplett anderes. Bei Viper hat sich gezeigt, dass die Validierung dieses Schlüsselkonzeptes sehr viel mehr Zeit in Anspruch nimmt. Das bedeutet, die Dokumente werden entsprechend langsamer in die DB eingefügt. Ein Grund dafür ist sicherlich die unterschiedliche Mächtigkeit der beiden Konzepte. KEY's stellen wesentlich mehr Funktionalität als ID's zur Verfügung und geben dem Nutzer somit mehr Anwendungsspielraum. Die Erfordernisse der Anwendung bestimmen, ob KEY's oder ID's benötigt werden. Sobald jedoch ID's für die Anwendung genügen, sollte aus Effizienzgründen auf jene zurückgegriffen werden.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Aufbauend auf dem Wissen über die Funktionalität verschiedender Benchmarks und dem Wissen über die einzelnen Datenbanksysteme wurde in dieser Studienarbeit die Untersuchung verschiedener XML-Konstrukte hinsichtlich ihrer Effizienz bei der Verarbeitung durchgeführt. Als Systeme zum Testen dienten das native XML-Datenbanksystem Tamino von der Software AG, DB2 UDB Version 8, ein relationales DBS von IBM und die Weiterentwicklung der Version 8 zu DB2 Viper. DB2 Version 9.1 (Viper) ermöglicht wie auch Tamino eine native Speicherung von XML-Dokumenten. Es wurden drei unterschiedliche Datenbanksysteme gewählt, um herauszufinden, ob sich die Ergebnisse der einzelnen Tests verallgemeinern lassen. Die Untersuchungen der Effizienz verschiedener Dokumente umfassten sechs Testszenarien: den Einfluss der Verwendung von Attributen oder Elementen in XML-Dokumenten, den Einfluss der Länge von Elementnamen und von Kommentaren in Dokumenten, den Einfluss der Schachtelungstiefe der XML-Dokumente und der Verwendung von IDREF's oder KEYREF's als Schlüsselkonzept, sowie ein Vergleich zwischen kompakter und verteilter Speicherung der XML-Inhalte.

Zunächst werden einige allgemeine Feststellungen angeführt, die im Zusammenhang mit den verschiedenen Tests gemacht wurden und somit abhängig von den jeweiligen Testbedingungen sind. Bei den Tests hat sich herausgestellt, dass das native Datenbanksystem Tamino am schnellsten beim Einfügen der XML-Dokumente ist und DB2 Viper hingegen am schnellsten bei der Bearbeitung der unterschiedlichsten Anfragen. Weiterhin haben die Tests ergeben, dass Tamino bei der Speicherung der XML-Dokumente mit Schemavalidierung weniger Zeit benötigt als bei der Speicherung ohne Schemavalidierung. Dies deutet daraufhin, dass bei Tamino bereits zum Zeitpunkt der Schemadefinition bestimmte Templates[14] bzw. Speicherstrukturen angelegt werden, die die eigentliche Speicherung dann beschleunigen. An dieser Stelle sei jedoch vermerkt, dass das Vergleichen der Systeme nicht Ziel dieser Studienarbeit war und daher auch keine Performanzoptimierung auf einem der Systeme durchgeführt wurden. Die oben angeführten Ergebnisse sind dennoch sehr interessant und auch aussagekräftig.

Nach diesen allgemeinen Betrachtungen sollen nun die Ergebnisse für die Effizienzbewertung der unterschiedlichen Dokumentstrukturen gegeben werden. Zuerst wird darauf aufmerksam gemacht, dass die Ergebnisse sowohl systemabhängig als auch abhängig von den einzelnen Anfragen waren. Die Tests wurden einerseits hinsichtlich des Einfügens der XML-Dokumente mit und ohne Schemavalidierung und andererseits hinsichtlich einer XPath-Anfrage, einer `COUNT`- und einer `FLWR`-XQuery durchgeführt.

Für drei der sechs Testszenarien haben alle drei Datenbanksysteme die gleichen

[14] aus dem Englischen: Schablone, Vorlage

Ergebnisse geliefert. Das bedeutet, auf allen drei Systemen hat sowohl die Länge der Elementnamen als auch das Vorhandensein von einigen Kommentaren keinen signifikanten Einfluss auf die Effizienz gemacht. Außerdem haben alle drei Systeme bestätigt, dass die kompakte Speicherung bei Vernachlässigung anderer Performanzfaktoren, wie beispielsweise Indexe, effizienter als die verteilte Speicherung der XML-Inhalte ist. Die Tests für die Verwendung von Attributen bzw. Elementen und für den Einfluss der Schachtelungstiefe der Dokumente haben stark systemabhängige Ergebnisse erbracht. Aus diesem Grund kann für diese beiden Szenarien keine allgemeine Auswertung bezüglich der unterschiedlichen Dokumentstrukturen abgeleitet werden. Zum Schluss das Ergebnis für die Untersuchung des Einflusses der verschiedenen Schlüsselkonzepte auf die Effizienz bei der Verarbeitung der XML-Dokumente. Da dieser Test nur auf einem System komplett durchgeführt wurde, können die Ergebnisse nicht verallgemeinert werden. Es ist jedoch zu vermuten, dass auch auf anderen Datenbanksystemen neben DB2 Viper die Überprüfung der KEY-Eigenschaften langsamer ist als die der ID-Eigenschaften, da das KEY-Konzept wesentlich mächtiger ist.

Wie an den unterschiedlichen Tests zu erkennen ist, ist es schwierig allgemeine Aussagen in Bezug auf die Effizienz der unterschiedlichen Dokumentstrukturen abzuleiten. Mit Hilfe dieser Studienarbeit ist dies jedoch in einigen Fällen gelungen. Eine Tendenz, die sich speziell bei den Tests hinsichtlich der Schachtelunstiefe und der Schlüsselkonzepte andeutet, ist, dass sich besonders die komplexen Modellierungsmöglichkeiten in XML negativ auf die Effizienz auswirken.

6.2 Ausblick

Die Untersuchungen im Rahmen dieser Studienarbeit haben ergeben, dass die Effizienz unterschiedlich strukturierter XML-Dokumente entscheidend von den zum Testen verwendeten Datenbanksystemen anhängt. Das bedeutet, dass sich die Untersuchungsergebnisse mit der Weiterentwicklung und Verbesserung der einzelnen Systeme durchaus verändern können. Die Herstellerfirmen der DBS sind sich über die Nachteile ihres Systems bewusst und arbeiten ständig daran, Ineffizienzen zu verringern bzw. zu beseitigen.

Weiterhin kann diese Arbeit keinesfalls den Anspruch auf Vollständigkeit erheben, da es sich bei dem untersuchten Thema um ein sehr komplexes Gebiet handelt. So sind beispielsweise weitere Testszenarien denkbar. Ein Vorschlag für ein solches Szenario wäre der Vergleich des Einflusses bei der Verwendung von `include`, `import` oder `redefine` in einem XML-Dokument. Mit Hilfe von `include` können Typinformationen innerhalb eines Namensraumes, die auf mehreren Schemabeschreibungen verteilt sind, zusammengefasst werden. Auf diese Weise können mehrere Schema inkludiert werden, solange sie im Namensraum übereinstimmen. Der Tag `import` hingegen erlaubt es Elemente aus anderen Namensräumen zu importieren und somit Schemabestandteile aus unterschiedlichen Namespaces zu verwenden. `redefine` ermöglicht es Typen, die aus externen Schemabeschreibungen stammen, umzudefi-

nieren. Dieser Mechanismus ist jedoch wie auch `include` nur im selben Namensraum zulässig. Weitere Tests könnten sich mit den unterschiedlichsten Möglichkeiten der Typdeklaration mit Vererbungsmechanismen (durch Erweiterung bzw. durch Restriktion) auf der einen Seite und durch „Substitution Groups“ auf der anderen Seite befassen. Entsprechend der Vielzahl an Modellierungsmöglichkeiten in XML sind mehrere Szenarien für weitere Tests denkbar.

Neben neuen Szenarien besteht auch die Möglichkeit die unterschiedlichen Strukturen der Dokumenten mit Hilfe weiterer, komplexer XPath- und XQuery-Anfragen zu testen. Solche Untersuchungen dienen eventuell auch zur weiteren Verallgemeinerung der Ergebnisse für die Effizienz der XML-Dokumente.

Weiterhin sind Test mit verschiedenen Indexen denkbar, sobald die Systeme diese auch zur Auswerung der XQuery selbst verwenden und nicht nur zur Selektierung der Dokumentmenge mit Hilfe der `WHERE`-Klausel.

Literatur

[BDL+01] Bressan, S.; Dobbie, G.; Lacroix, Z.; Lee, M.; Li, Y. G.; Nambiar, U.; Wadhwa, B.: XOO7: Applying OO7 Benchmark to XML Query Processing Tool. In Proc. of CIKM, S. 167-174, 2001.

[Bou05] Bourret, R.: XML and Databases. 2005. http://www.rpbourret.com/xml/XMLAndDatabases.htm

[BR01] Böhme, T.; Rahm, E.: XMach-1: A Benchmark for XML Data Management. Proc. 9. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft, S. 264-273, Springer, Berlin, 2001.

[Bre05] Breutmann, B.: IT-Kompaktkurs Datenbanken; Folge 13: Objektorientierte Datenbanksysteme und neuere Entwicklungen. Fachhochschule Würzburg-Schweinfurt, 2005. http://www.bw.fh-deggendorf.de/kurse/db/skripten/skript13.pdf

[Cav01] Cavadini, P.: XML Database Benchmarking. http://www.inf.uni-konstanz.de/dbis/teaching/ss01/data-on-the-web/local/XML-Benchmarking.PDF

[CDN93] Carey, M. J.; DeWitt, D. J.; Naughton, J. F.: The OO7 Benchmark. In Proc. of SIGMOD Conference, S. 12-21, 1993.

[CX00] Cheng, J.; Xu, J.: IBM DB2 XML Extender brochure. ICDE '00 Conference, San Diego, Februar 2000. ftp://ftp.software.ibm.com/software/data/db2/extenders/ xmlext/xmlextbroch.pdf

[EDO+00] Ennser, L.; Delporte, T.; Oba, M; Sunil, K. M.: Integration XML with DB2 XML Extender and DB2 Text Extender. Dezember 2000. http://www.redbooks.ibm.com/redbooks/pdfs/sg246130.pdf

[Gra93] Gray, J.: The Benchmark Handbook. Morgan Kaufmann, San Mateo, 1993.

[Ham] Hammori, M. R.: XML-Benchmarks. http://www.hammori.de/files/XML-Benchmarks.pdf

[IBM03] Text Extender Administration and Programming. Version 8 for Windows, USA, Juni 2003.

[IBM04] XML Extender Administration and Programming. Version 8.2 for Windows, USA, Oktober 2004. ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/ pdf/en_US/db2sxe81.pdf

[KM02] Klettke, M.; Meyer, H.: XML & Datenbanken - Konzepte, Sprachen und Systeme. dpunkt, Heidelberg, 2002.

[NL05a] Nicola, M; Van der Linden, B.: Native XML Support in DB2 Universal Database. IBM, San Jose, USA, 2005. www.vldb2005.org/program/paper/thu/p1164-nicola.pdf

[NL05b] Nicola, M; Van der Linden, B.: Native XML Support in DB2 Universal Database. IBM, San Jose, USA, 2005. www.vldb2005.org/program/paper/thu/s1164-nicola.pdf

[NLB+01] Nambiar, U.; Lacroix, Z.; Bressan, S.; Lee, M.; Li, Y. G.: XML Benchmarks Put to the Test. In Proc. of the Third International Conference on Information Integration and Web-based Applications and Services (IIWAS), Linz, Austria, 2001.

[RB02] Rahm, E.; Böhme, T.: Proc. 9. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft, S. 264-273, Springer, Berlin, 2001.

[RV03] Rahm, E.; Vossen, G. (Hrsg): Web & Datenbanken. dpunkt, Heidelberg, 2003.

[Sch03] Schöning, H.: XML und Datenbanken. Hanser, München, 2003.

[SAGa] Software AG: Tamino API for Java. http://developer.softwareag.com/tamino/taminoAPI4J/default.htm

[SAGb] Software AG: XML:DB API. http://developer.softwareag.com/tamino/xmldb/Default.htm

[SWK+01] Schmidt, A.; Waas, F.; Kersten, M. L.; Florescu, D.; Carey, M. J.; Manolesco, I.; Busse, R.: The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, Niederlande, 2001.

[SWK+02] Schmidt, A.; Waas, F.; Kersten, M. L.; Carey, M. J.; Manolesco, I.; Busse, R.: XMark: A Benchmark for XML Data Management. In Proc. of the International Conference on Very Large Data Bases (VLDB), S. 974-985, Hong Kong, China, August 2002.

[Tam02a] Software AG: Java API Migration Cookbook (for Tamino). Darmstadt, 2002. http://developer.softwareag.com/tamino/knowhow/apimigcookbook.htm

[Tam02b] Software AG: Tamino XML:DB API Programmers Cookbook. Darmstadt, 2002. http://developer.softwareag.com/tamino/knowhow/progguide_xmldb.htm

[Tam03] Tamino XML Server Documentation (Software AG). Tamino XML Server Version 4.1.4., Darmstadt, 2003.

[VdL05] Van der Linden, B.: DB2 Viper's native XML datastore. San Jose, 2005. http://www.idealliance.org/proceedings/xml05/slides/vanderlinden.pdf

[XML:DB] XML:DB Initiative for XML Databases. http://xmldb-org.sourceforge.net/

Abkürzungen

	Bedeutung	**eingeführt in Abschnitt**
BLOB	Binary Large Object	3
CLOB	Character Large Object	3
DAD	Document Access Definition	3.1.1.1
DB	Datenbank	3
DB2 UDB	DB2 Universal Database	3.1.1
DBIS	Lehrstuhl für Datenbank- und Informationssysteme	0
DBS	Datenbanksystem	3
DTD	Document Type Definiton	1.1
FLWR	FOR-LET-WHERE-RETURN	3.2.4
FLWU	FOR-LET-WHERE-UPDATE	3.2.5
HTML	Hypertext Markup Language	3.1.2
HTTP	Hypertext Transfer Protokoll	3.2.1
RDB	Rational Database	3.1.1.2
SGML	Standard Generalized Markup Language	1.1
SQL	Structured Query Language	3
TPC	Transaction Processing Performance Counsil	2
UDF	User Defined Function	3.1.1
UDT	User Defined Type	3.1.1
URL	Uniform Resource Locator	3.2.1
W3C	World Wide Web Consortium	1.1
XML	eXtensible Markup Language	0
XSR	XML Schema Repository	3.3.2

Abbildungsverzeichnis

Inhalt der CD

- Implementierung
  - DB2 Version 8
  - Tamino
  - DB2 Viper
- Dokumente
  - Originaldokumente
  - Element als Attribut
  - Kommentar
  - Kurze Elementnamen
  - Lange Elementnamen
  - Schachtelungstiefe
  - IDREF
  - KEYREF
- Testprotokolle
  - DB2 Version 8
  - Tamino
  - DB2 Viper
- Diagramme
  - DB2 Version 8
  - Tamino
  - DB2 Viper
- Schemata (.xsd)
- Tamino Schemata (.tsd)
- Quellen