XML-Archivierung betriebswirtschaftlicher Datenbank-Objekte*

Bernhard Zeller [1] Axel Herbst [2] Alfons Kemper [1]

[1] Universität Passau
94030 Passau, Germany
⟨Nachname⟩@db.fmi.uni-passau.de

[2] SAP AG
69190 Walldorf, Germany
axel.herbst@sap.com

Abstract: Einer der wichtigsten Einsatzbereiche relationaler Datenbanksysteme liegt in der Verwaltung und Auswertung betriebswirtschaftlicher Daten. Insbesondere dienen relationale Datenbanken als Backend für betriebswirtschaftliche Software. In diesem Umfeld ist das Konzept der Archivierung bekannt. Unter Archivierung versteht man dabei das Verschieben der Daten von selten benötigten betriebswirtschaftlichen Objekten aus den OLTP-Datenbanksystemen auf Tertiärspeichersysteme. Dadurch werden das Datenvolumen der Datenbank reduziert, die Leistung des Datenbanksystems erhöht und Kosten gespart. In SAP-Systemen wurde dieses Konzept unter dem Namen *Datenarchivierung* umgesetzt. Wir schlagen in dieser Arbeit vor, die relationalen Datenbanksysteme um einen XML-Archivierungs-Operator zu erweitern. Der XML-Archivierungs-Operator erlaubt es, den gesamten Archivierungsvorgang auf Datenbank-Ebene auszuführen und die Daten als XML-Dokumente abzulegen. Der Operator erhält die Daten eines betriebswirtschaftlichen Objektes in Form von *temporären Tabellen*. Die temporären Tabellen repräsentieren das relationale Schema des Objektes und beinhalten Verweise auf die zugeordneten Tupel der operativen OLTP-Datenbasis. Ein ebenfalls übergebenes *XML-Schema*-Dokument enthält die genaue Definition des Archivobjektes und gibt an, welche Teile des betriebswirtschaftlichen Objektes archiviert werden sollen. Bei der Archivierung stellt das abschließende Löschen der Tupel wegen der vielen benötigten Schreibsperren eine besonders kritische Phase dar. Deshalb wurde hierfür eine effiziente Technik, bei der Datensätze gemäß ihrer physikalischen Anordnung gelöscht werden, in den XML-Archivierungs-Operator integriert.

1 Einleitung

Einer der wichtigsten Einsatzbereiche relationaler Datenbanksysteme liegt in der Verwaltung und Auswertung betriebswirtschaftlicher Daten (*betriebswirtschaftliche Datenbanksysteme*). Die Datenbanksysteme fungieren dabei als Datenspeicher für betriebswirtschaftliche Anwendungssysteme und dienen als Integrationsplattform aller operationalen Daten eines Unternehmens. Trotz des starken Einsatzes in diesem Bereich bieten die heutigen Datenbanksysteme keine oder nur sehr eingeschränkte Möglichkeiten, rudimentäre betriebswirtschaftliche Vorgänge, wie z.B. das Definieren und Sperren betriebswirtschaftlicher Objekte, auf Datenbank-Ebene auszuführen. Diese Defizite führen dazu, dass Hersteller von betriebswirtschaftlicher Standardsoftware typische Datenbankfunktionen, wie die Sperrverwaltung oder die Überwachung komplexer Integritätsbedingungen,

*Diese Arbeit wurde durch die Firma SAP im Rahmen des sog. Terabyte-Projektes gefördert.

auf Applikations-Ebene in ihre Systeme integrierten und die Datenbank nur als reine Speicherschicht verwenden.

Es ist deshalb notwendig, die Schnittstellen der Datenbanken zu erweitern, anzupassen und zu vereinfachen. Ein erster Schritt in diese Richtung sind die XML-Schnittstellen und die objekt-relationalen Erweiterungen, die fast alle Datenbankhersteller mittlerweile anbieten.

Wir schlagen in dieser Arbeit eine Erweiterung relationaler Systeme um einen XML-Archivierungs-Operator vor. Das Konzept der Archivierung ist im Umfeld betriebswirtschaftlich genutzter Datenbanken bekannt und in SAP-Systemen unter dem Namen *Datenarchivierung* umgesetzt [SBB+02]. Die Archivierung ist ein weiteres Beispiel für eine auf Applikations-Ebene implementierte Datenbank-Funktion. Unter Archivierung versteht man dabei das Verschieben der Daten von selten benötigten betriebswirtschaftlichen Objekten aus den produktiven Datenbanksystemen auf kostengünstigere Tertiärspeichersysteme, bei denen CD's oder Bänder als Speichermedien eingesetzt werden. Die Daten sind auch nach der Archivierung noch von der Anwendung aus zugreifbar. Durch das Verschieben der Daten aus der produktiven Datenbank (die in der Praxis in mehreren (System-) Kopien und zusätzlich oft gespiegelt vorliegt) wird das Datenvolumen verkleinert und die Leistung der Datenbank somit erhöht. Zudem sinken die Kosten für das Gesamtsystem, da Tertiärspeicher in der Regel billiger ist als Sekundärspeicher und der Aufwand für die Administration des Systems wesentlich reduziert wird.

Die Hersteller betriebswirtschaftlicher Anwendungen müssen jedes Jahr hohe Ausgaben für die Wartung und Implementierung der Archivierungssoftware tätigen. Der hohe Aufwand rührt daher, dass jede Anwendung ihre eigene Archivierungskomponente implementiert und die Verarbeitung komplexer betriebswirtschaftlicher Objekte schwierig ist, weil auf Datenbankebene die Daten der Objekte auf sehr viele Tabellen verteilt sind (siehe Abbildung 1). Desweiteren muss ein hoher Aufwand betrieben werden, um bei einem Versionswechsel auf Anwendungs- und Systemseite die Lesbarkeit der Archivdaten zu gewährleisten (z.B. bei Schema-Änderungen in der Datenbank oder der Umstellung des Zeichensatzes). Lediglich bei komponentenbasierten Systemen wie SAP R/3 lassen sich diese Kosten durch den Zugriff mehrerer Anwendungen auf eine gemeinsame Archivierungskomponente etwas senken. Im Falle des Systems SAP R/3 ist diese gemeinsam genutzte Archivierungskomponente das *ADK* (Archive Development Kit) [SBB+02, SR97]. Bei der Archivierung liest die Archivierungskomponente die Daten der zu archivierenden betriebswirtschaftlichen Objekte aus der Datenbank, packt sie in (meist nur für die Anwendung lesbare) Dateien und legt diese auf einem dafür vorgesehenen Ablagesystem ab. Anschließend werden die Daten auf Grundlage der zuvor generierten Archivdateien in der produktiven Datenbank gelöscht. Vor allem dieser Löschvorgang kann die Leistung des Datenbanksystems bei sehr vielen zu löschenden Daten stark beeinträchtigen und stellt deshalb ein großes Problem gerade bei der Archivierung großer Datenbankvolumen dar. Dieses Vorgehen war aber aus folgenden Gründen nötig:

- Die Definition der betriebswirtschaftlichen Objekte ist nur auf Anwendungsseite bekannt. Mit der Definition eines betriebswirtschaftlichen Objektes ist dabei vor allem das Wissen gemeint, welche Tabellen in der Datenbank die Daten welches betriebswirtschaftlichen Objektes speichern. Bisher gab es noch keine standardisierten Techniken, mit denen dieses Wissen in die Datenbank transferiert werden konnte.

Mit der Einführung von XML und XML-Schema [XML00] sind aber entsprechende Formate geschaffen worden.

- Die rechtlichen und betriebswirtschaftlichen Regelungen, wann ein Objekt archiviert werden darf und wann nicht, sind derart komplex, dass sie sich nicht mit SQL-Mitteln abbilden lassen. Es sind dazu komplexe Programme auf Anwendungsseite nötig.

- Es gab noch keinen adäquaten Operator auf Datenbank-Ebene, dem auf einfache Weise mitgeteilt werden konnte, wie ein betriebswirtschaftliches Objekt aussieht, welche Objekte zu archivieren sind und der die Daten im produktiven System auf effiziente Weise, nach dem Erzeugen der Archivdateien, löscht.

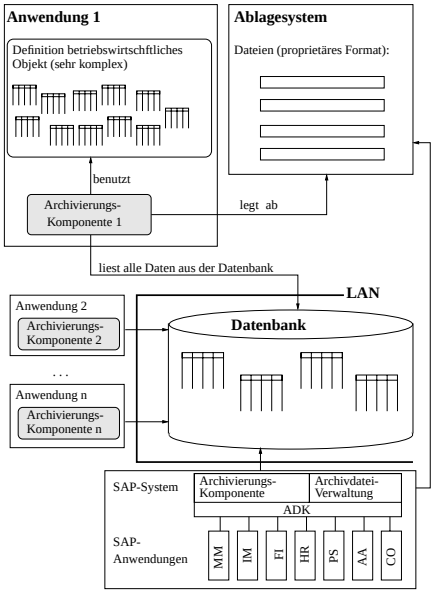


Abbildung 1: Architektur im Bereich der betriebswirtschaftlichen Datenarchivierung

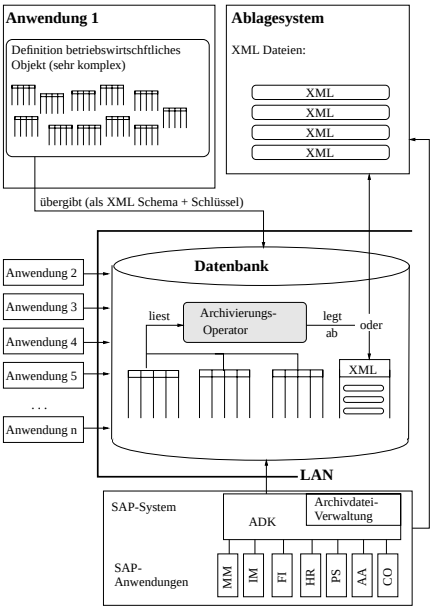


Abbildung 2: Neue Architektur mit XML-Archivierung

Der in dieser Arbeit vorgeschlagene XML-Archivierungs-Operator erlaubt es, im Gegensatz zu den auf Applikations-Ebene implementierten Archivierungskomponenten, den gesamten Archivierungsablauf auf Datenbank-Ebene auszuführen und die Daten als XML-Dokumente abzulegen (siehe Abbildung 2). Dadurch wird der Datenverkehr zwischen Anwendung und Datenbank verringert. Durch die Ablage als XML wird die Portabilität der Daten erhöht und die Implementierung separater Archivierungskomponenten unnötig gemacht. Vor allem die Ablage als XML macht die Daten nicht nur für andere Anwendungen verfügbar, sondern minimiert auch den Aufwand (und damit die Kosten) bei einem Versionswechsel auf Anwendungsseite erheblich.

Desweiteren nutzt der XML-Archivierungs-Operator intelligente Löschalgorithmen, um den Löschvorgang am Ende einer Archivierung zu beschleunigen. Das abschließende Löschen der Tupel ist wegen der vielen benötigten Schreibsperren eine besonders kritische Phase der Archivierung. Die hierfür realisierte effiziente Technik ist eine Variante des von uns in [GKKZ01] vorgestellten *Bulkdelete*-Operators. Wie unsere Messungen zeigen, beschleunigt diese Variante des Bulkdelete-Operators den Löschvorgang um ein Vielfaches (siehe Abschnitt 4).

Die restliche Arbeit ist wie folgt gegliedert: In Abschnitt 2 wird auf verwandte Arbeiten eingegangen. In Abschnitt 3 wird das Konzept der Archivierung genauer erklärt und der neue XML-Archivierungs-Operator vorgestellt. Eine Beschreibung einer Beispiel-Implementierung und die Ergebnisse einiger Messungen sind in Abschnitt 4 zu finden. Abschnitt 5 fasst die vorgestellten Techniken zusammen.

2 Verwandte Arbeiten

Das Konzept eines in die Datenbank integrierten Archivierungs-Operators wurde in [Sch01, Her97, KS98, SBH+98, LS98] erläutert und eine dafür geeignete Erweiterung der SQL-Syntax vorgeschlagen. Im Gegensatz zum XML-Archivierungs-Operator speichern die in diesen Arbeiten vorgeschlagenen SQL-Spracherweiterungen die zu archivierenden Daten in speziellen Archivtabellen und nicht als XML-Dokumente. Außerdem werden die traditionellen Löschmethoden benutzt.

Das Erzeugen von XML-Dokumenten aus relationalen Daten wurde ebenfalls schon detailliert in verschiedenen Projekten untersucht. In [Rys01] sind die XML-Generierungstechniken des Datenbanksystems SQL Server von Microsoft beschrieben. In [CFI+00] wird das Middleware-System XPERANTO, das eine XML-basierte Anfrageschnittstelle zu objekt-relationalen Datenbanken bietet, vorgestellt. In [SSB+01] wurden verschiedene Techniken zum Erzeugen von XML-Dokumenten aus relationalen Daten untersucht. Im System SilkRoute [FTS00] wird eine Kombination aus SQL und XML-QL [DFF+99] verwendet, um das Erzeugen der XML-Dokumente zu steuern. Das Datenbanksystem der Firma Oracle benutzt seit der Version 8i objekt-relationale Typen und Sichten, um die Struktur des zu erzeugenden XML-Dokumentes zu bestimmen [VW02]. Bei der Entwicklung des XML-Archivierungs-Operators lag der Schwerpunkt nicht auf der Untersuchung der XML-Generierung, sondern auf dem verwendeten Löschalgorithmus, der Übergabe der betriebswirtschaftlichen Objekte und dem Fakt, dass die Daten nach der Generierung als XML vorliegen. Wir haben deshalb eine sehr einfache, auf JDOM [JDO] basierende, Art der Generierung angewandt. Prinzipiell können bei der XML-Generierung aber auch alle anderen hier angesprochenen Techniken im XML-Archivierungs-Operator verwendet werden.

In [Moh02] wird ein auf Index Scans basierender Algorithmus zum effizienten Löschen von Tabelleneinträgen vorgestellt, der verwendet werden kann, wenn die zu löschenden Tupel durch eine Bereichsanfrage spezifiziert sind. Der im XML-Archivierungs-Operator verwendete Löschalgorithmus ist eine Variante des in [GKKZ01] vorgestellten Algorithmus. Im Gegensatz zu [Moh02] müssen die zu löschenden Tupel nicht als Bereichsanfrage spezifiziert werden, sondern es kann jede Art von Anfrage benutzt werden.

Eine mögliche Alternative zur Archivierung ist die horizontale Partitionierung der Daten. Auch bei der Partitionierung kann, wie bei der Archivierung, die Wartbarkeit und Leistung der Datenbank erhöht werden. In [Now01] wird gezeigt, welche Partitionierungstechniken in den heute verfügbaren Datenbanksystemen integriert sind und wie diese in der Archivierung eingesetzt werden können [Now99]. In [KN99] wurden Techniken untersucht, um Datenbanken mittels Partitionierung von gewissen Datenbeständen zu maskieren. In [ZK02] wurde untersucht, inwiefern Partitionierung im SAP-Umfeld einsetzbar ist.

Im Rahmen der Entwicklung des XML-Archivierungs-Operators entwickeln wir auch einen XML-Archiv-Browser, der es erlaubt, ein Ablagesystem über Protokolle wie WebDAV oder SOAP [BEK+00] anzusprechen und die abgelegten XML-Archivdokumente mittels benutzerdefinierter Stylesheets darzustellen.

3 Der XML-Archivierungs-Operator

Um es den Programmierern betriebswirtschaftlicher Software zu ermöglichen, mehr Datenbankfunktionen zu nutzen, müssen auf Datenbankseite bessere und angepasstere Schnittstellen und Operatoren zur Verfügung gestellt werden. Der in dieser Arbeit vorgeschlagene XML-Archivierungs-Operator erlaubt es, die Archivierung von betriebswirtschaftlichen Objekten auf Datenbank-Ebene auszuführen. Der XML-Archivierungs-Operator versteht komplexe Objektdefinitionen, kann die Daten in einem geeigneten Austauschformat (XML) ablegen (*Archivdaten-Generierung*) und löscht die Daten anschließend effizient aus der produktiven Datenbasis (*Löschvorgang*). Möglich ist dies zum einen durch die Art und Weise, wie dem XML-Archivierungs-Operator die Daten über die zu archivierenden betriebswirtschaftlichen Objekte übergeben werden und zum anderen durch einen besonderen Algorithmus zum Löschen der Daten aus der produktiven Datenbasis. Diese beiden Techniken bilden das Herzstück des XML-Archivierungs-Operators. Bevor genauer auf diese Techniken eingegangen wird, soll zunächst das Konzept der Archivierung vertieft werden.

3.1 Archivierung betriebswirtschaftlicher Datenbank-Objekte

Der wichtigste Grund für eine Archivierung sind Performance-Probleme der Datenbank, die aus zu groß gewordenen Tabellen resultieren. Im SAP-Umfeld gibt es bereits Systeme, deren Datenbasen eine Größe von mehreren Terabyte erreicht haben und kontinuierlich weiter wachsen. Ein Problem sind z.B. die Beeinträchtigungen beim Pflegen der Datenbasis. Arbeiten, wie das Aufbauen von Statistiken und Indexen, dauern bei großen Tabellen sehr lange und können den dafür vorgesehenen (zeitlichen) Rahmen sprengen. Dies wiederum kann dazu führen, dass diese Arbeiten gar nicht mehr durchführbar sind und sich die Leistung des Systems deshalb immer mehr verschlechtert. Die Archivierung kann solche Probleme lösen, indem sie die Daten nicht mehr oder nur noch selten benötigter betriebswirtschaftlicher Objekte von den Tabellen der produktiven Datenbasis auf Tertiärspeichersysteme verschiebt und damit die Tabellen im produktiven Datenbanksystem wieder verkleinert. Beim Archivieren in betriebswirtschaftlichen Anwendungen sollen aber nicht einzelne Tabelleneinträge archiviert werden, sondern immer die gesamten Daten eines betriebswirtschaftlichen Objektes, also z.B. eines Beleges. Dadurch sind

die archivierten Daten in sich konsistent und können grundsätzlich auch außerhalb des Datenbanksystems weiter bearbeitet werden.

Wie bereits erwähnt, werden nur die betriebswirtschaftlichen Objekte archiviert, die voraussichtlich nur noch selten (oder gar nicht mehr) benötigt werden. Dies kann z.B. der Fall sein, wenn die Daten aus früheren Jahren stammen und deshalb veraltet sind oder die Objekte aus betriebswirtschaftlicher Sicht abgeschlossen sind (z.B. ausgeglichene Finanzbuchhaltungsbelege).

Das Archivieren betriebswirtschaftlicher Objekte geschieht in 2 Schritten:

1. Zuerst werden die Daten der zu archivierenden betriebswirtschaftlichen Objekte aus der Datenbank gelesen und in speziellen (komprimierten) Dateien - sog. *Archivdateien* - gespeichert (**Archivdaten-Generierung**).

2. Anschließend werden die Daten auf Grundlage der zuvor generierten Archivdateien in der produktiven Datenbank gelöscht (**Löschvorgang**).

Damit die Anwendung auf die Archivdateien zugreifen kann, werden sie von einer speziellen Applikation verwaltet. Diese Applikation (das sog. *Ablagesystem*) funktioniert vereinfacht gesagt wie ein Dokumenten Management System. Die archivierende Anwendung übergibt die Archivdateien und einen entsprechenden Schlüssel an das Archivsystem und das Archivsystem liefert auf Anfrage die eingestellte Archivdatei zurück.

Die in diesem Abschnitt vorgestellte Archivierung wurde bisher immer auf Applikations-Ebene implementiert. Der in den folgenden Abschnitten beschriebene XML-Archivierungs-Operator arbeitet dagegen auf Datenbank-Ebene.

3.2 Die Aufruf-Syntax des XML-Archivierungs-Operators

In diesem Abschnitt wird gezeigt, wie dem XML-Archivierungs-Operator die Definition des betriebswirtschaftlichen Objektes übergeben wird und wie der Operator erfährt, welche Objekte zu archivieren sind.

Welche Tabellen in der Datenbank die Daten welches betriebswirtschaftlichen Objektes speichern, ist, wie bereits erwähnt, nur auf Anwendungsseite bekannt. Desweiteren kann nur die Anwendung entscheiden, welche betriebswirtschaftlichen Objekte archiviert werden können. Die Anwendung trifft diese Entscheidung aufgrund juristischer und betriebswirtschaftlicher Regeln, die in Form von Programmen in der Anwendung enthalten sind. Dem XML-Archivierungs-Operator werden diese Daten mittels zweier Parameter übergeben (siehe Abbildung 3): Die Definition des betriebswirtschaftlichen Objektes wird mittels eines *XML-Schema-Dokumentes* übergeben (Parameter **schema**) und das Wissen, welche Objekte zu archivieren sind, mittels spezieller *temporärer Tabellen* (**from**-Klausel).

Die Parameter sowie die restlichen Klauseln der Aufruf-Syntax des XML-Archivierungs-Operators werden in den folgenden Abschnitten genauer erklärt.

3.2.1 Der Parameter *schema*

Mit Hilfe des Parameters **schema** wird dem XML-Archivierungs-Operator mitgeteilt, wie das betriebswirtschaftliche Objekt als XML-Dokument abzulegen ist. In dem übergebenen

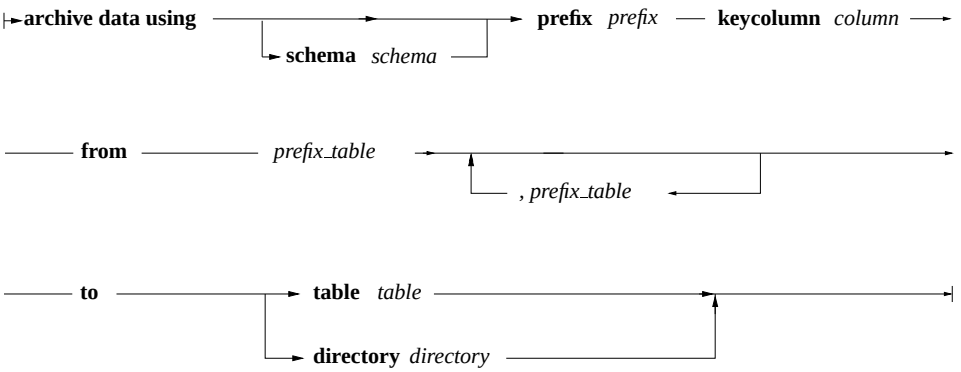


Abbildung 3: Aufrufsyntax des XML-Archivierungs-Operators

XML-Schema ist angegeben, welche Tabellen zu dem betreffenden betriebswirtschaftlichen Objekt gehören, welche Spalten dieser Tabellen relevant sind und wie diese Daten in das zu erzeugende XML-Dokument eingebunden werden sollen. Das übergebene XML-Schema muss dazu den im Folgenden beschriebenen Bedingungen genügen, um korrekt vom Operator bearbeitet werden zu können.

Tabellendaten werden mittels *Annotationen* in das XML-Schema eingebunden. Annotationen sind laut XML-Schema-Spezifikation vorgesehen, um Applikationen Informationen zukommen zu lassen, wie das jeweilige Schema zu verarbeiten ist [XML00]. Im XML-Schema muss dazu an der Stelle, wo die Tabellendaten integriert werden sollen, folgende Annotation eingefügt werden:

```
<xsd:annotation>
   <xsd:appinfo>
      sql:[Schema].[Tabelle].[Spalte]
   </xsd:appinfo>
</xsd:annotation>
```

Bei der XML-Generierung wird dann an dieser Stelle ein XML-Element erzeugt, das den Attributwert eines Tupels enthält. Der XML-Archivierungs-Operator wandelt dabei die Datenbank-Datentypen in passende XML-Datentypen um. Die XML-Datentypen sind mittels sog. *SimpleType*-Definitionen vordefiniert und brauchen nur noch benutzt zu werden. Der von uns implementierte Prototyp (siehe Abschnitt 4) unterstützt z.B. alle im SAP R/3 System verwendeten Datentypen.

Die XML-Elemente aller Attribute eines Tupels ergeben ein in sich konsistentes XML-Fragment. Werden mehrere Tupel einer Tabelle archiviert, so werden mehrere dieser XML-Fragmente hintereinander erzeugt. Abbildung 4 zeigt einen entsprechenden Teil eines XML-Schemas, die dazugehörige Tabelle und die daraus erzeugten XML-Fragmente. Aus Gründen der Übersichtlichkeit ist die Darstellung vereinfacht.

Die Speicherung in XML bietet durch geeignete Wahl der Element-Namen (mnemonische Namen) die Möglichkeit, zusätzliche Information abzuspeichern (z.B. Spalte ANZ speichern mit Element-Namen <Menge>). Die Eindeutigkeit der Element-Namen für die Spaltendefinitionen innerhalb einer Tabelle muss dabei gewährleistet sein.

```
...
<element name="Position"
     type="bestellposition" />
...
<complexType name="bestellposition">
  <sequence>
    <element name="BID" type="NUMBER">
      <annotation><appinfo>
         sql:SAPR3.Bestellposition.BID
      </appinfo></annotation>
    </element>
    <element name="Zeile" type="NUMBER">
      <annotation><appinfo>
         sql:SAPR3.Bestellposition.Zeile
      </appinfo></annotation>
    </element>
    <element name="Ware" type="CHAR">
      <annotation><appinfo>
         sql:SAPR3.Bestellposition.Ware
      </appinfo></annotation>
    </element>
    <element name="Menge" type="NUMBER">
      <annotation><appinfo>
         sql:SAPR3.Bestellposition.Anz
      </appinfo></annotation>
    </element>
    <element name="Preis" type="NUMBER">
      <annotation><appinfo>
         sql:SAPR3.Bestellposition.Preis
      </appinfo></annotation>
    </element>
  </sequence>
</complexType>
```

```
...
<Position>
  <BID>1</BID>
  <Zeile>1</Zeile>
  <Ware>Fernseher</Ware>
  <Menge>1.0</Menge>
  <Preis>1000.0</Preis>
</Position>
<Position>
  <BID>1</BID>
  <Zeile>2</Zeile>
  <Ware>Videorecorder</Ware>
  <Menge>1.0</Menge>
  <Preis>200.0</Preis>
</Position>
...
```

Bestellposition				
BID	**Zeile**	Ware	Anz.	Preis
1	1	Fernseher	1	1000
1	2	Videorecorder	1	200
2	1	Radio	1	57
3	1	1,5 V Batterie	10	0,10
3	2	Lautsprecher	2	23
3	3	CD's	2	30
...	...	...	...	...

Abbildung 4: XML-Schema Ausschnitt mit erzeugten XML-Fragmenten

Durch Zusammenfügen der einzelnen Bausteine für die Tupel eines betriebswirtschaftlichen Objektes ergibt sich ein XML-Schema für dieses Objekt. Dabei ist es möglich, durch geeignete Schachtelung zusätzliche Information innerhalb des erzeugten XML-Dokumentes zu speichern. So können z.B. Fremdschlüsselbeziehungen durch geeignetes Schachteln der Tabellendaten dargestellt werden (siehe hierzu auch das Beispiel in Abschnitt 3.3).

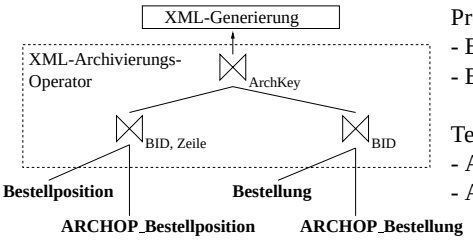
Wird kein Schema angegeben, so erzeugt der Archiv-Operator selbst ein Default-XML-Schema. Er untersucht dazu die Schemata der zu archivierenden Tabellen. Allerdings geht der Operator dabei davon aus, dass alle Attribute dieser Tabellen archiviert werden sollen. Eine Einschränkung auf bestimmte Attribute ist hier nicht möglich. Das erzeugte XML-Schema ist außerdem sehr einfach gehalten und beinhaltet keine zusätzlichen Informationen wie z.B. Fremdschlüsselbeziehungen.

3.2.2 Die *from*-Klausel

Mittels der **from**-Klausel wird dem XML-Archivierungs-Operator mitgeteilt, welche betriebswirtschaftlichen Objekte zu archivieren sind. Dies geschieht durch die Angabe der zu verwendenden *temporären Tabellen* in der **from**-Klausel. Jeder produktiven Tabelle entspricht dabei genau eine temporäre Tabelle. Die Verbindung zwischen temporärer Tabelle und produktiver Tabelle wird über den Namen hergestellt: Der Name der temporären Tabelle ist zusammengesetzt aus dem im Parameter **prefix** angegebenen Präfix und dem Namen der entsprechenden produktiven Tabelle. Die temporären Tabellen bilden das Bin-

deglied, mit dessen Hilfe die Daten eines betriebswirtschaftlichen Objektes identifiziert werden können. Dieses Bindeglied ist nötig, da in der produktiven Datenbasis oft keine direkte Verbindung zwischen den Tabellen eines betriebswirtschaftlichen Objektes (z.B. mittels Fremdschlüssel-Beziehungen) hergestellt werden kann. Die meisten betriebswirtschaftlichen Anwendungen wurden in den letzten Jahren ständig erweitert und nicht alle diese Erweiterungen konnten durch Änderungen im Datenmodell korrekt abgebildet werden. Deshalb sind die Tabellen eines betriebswirtschaftlichen Objektes auf Datenbank-Ebene nicht immer mittels eines Verbundes verknüpfbar. Stattdessen wird das in den Anwendungen vorhandene, zusätzliche Wissen genutzt, um die Tabellen auf Anwendungs-Ebene zu verknüpfen.

Um es trotzdem möglich zu machen, die Daten der produktiven Tabellen zu verbinden, enthalten die temporären Tabellen zum einen eine Spalte mit einem eindeutigen Objektschlüssel und zum anderen Spalten mit den Schlüsselattributen der jeweiligen produktiven Tabelle. Die Spalte mit den Objektschlüsseln trägt den mittels des Parameters **keycolumn** angegebenen Namen. Die Spalten mit den Schlüsselwerten der produktiven Tabelleneinträge tragen den selben Namen wie die entsprechenden Spalten in den produktiven Tabellen. Durch den eindeutigen Objektschlüssel können nun die temporären Tabellen mittels eines natürlichen Verbundes verknüpft werden. Da die temporären Tabellen neben dem Objektschlüssel auch noch die Schlüsselwerte der Einträge der produktiven Tabellen enthalten, können nun auch die produktiven Tabellen verknüpft werden. Für die Beispiel-Tabellen aus Abschnitt 3.3 ist diese Verknüpfung der temporären mit den produktiven Tabellen in Abbildung 5 dargestellt[1].



Produktive Tabellen:
- Bestellung (**BID**, Kunde, Datum, Summe)
- Bestellposition (**BID**, **Zeile**, Ware, Anz, Preis)

Temporäre Tabellen:
- ARCHOP_Bestellung (ArchKey,BID)
- ARCHOP_Bestellposition (ArchKey, BID, Zeile)

Abbildung 5: Verknüpfung der produktiven Tabellen mit Hilfe der temporären Tabellen

Sind die Tabellen entsprechend verknüpft, können die Daten der betriebswirtschaftlichen Objekte ausgelesen und archiviert werden. Die **to**-Klausel gibt dabei an, wohin die Daten gespeichert werden sollen. Ist in der **to**-Klausel eine Tabelle angegeben, so darf diese Tabelle nur 2 Spalten enthalten. In der ersten Spalte werden die Objektschlüssel abgelegt (um die XML-Dokumente eindeutig identifizieren zu können) und in der 2. Spalte die XML-Dokumente. Der Typ der 2. Spalte muss es erlauben, in einer Zeile ein komplettes XML-Dokument zu speichern. Die meisten Datenbankhersteller bieten für solche Zwecke spezielle XML-Datentypen an, aber auch ein CLOB (Character Large Object) kann verwendet werden. Wird das Dateisystem als Ablageort gewählt, wird für jedes Objekt eine

[1] Wegen der Einfachheit des Beispiels wäre hier ein direkter Join der produktiven Tabellen möglich. Dies ist aber bei komplexeren Objekten in der Regel nicht der Fall

Datei erzeugt. Der Dateiname enthält dabei den Objektschlüssel des enthaltenen Objektes und dessen Typ (z.B. ”Beleg4711.xml”).

Im folgenden Abschnitt wird der Ablauf einer Archivierung nochmals genau erläutert.

3.3 Archivieren mit dem XML-Archivierungs-Operator

Beim Einsatz des XML-Archivierungs-Operators läuft eine Archivierung wie folgt ab:

1. Es werden von der Anwendung die temporären Tabellen erzeugt (Gemäß den Vorgaben durch die Definition des betriebswirtschaftlichen Objektes).
2. Die temporären Tabellen werden mit den Schlüsselwerten der Tabelleneinträge des betriebswirtschaftlichen Objektes gefüllt.
3. Der XML-Archivierungs-Operator wird aufgerufen. Dieser verarbeitet die Daten wie folgt:
   (a) Die temporären Tabellen werden komplett gesperrt. Dies ist notwendig, um die Konsistenz der erzeugten XML-Daten zu gewährleisten. Dieses Vorgehen stellt aber keine Problem dar, da die temporären Tabellen nur vom XML-Archivierungs-Operator verwendet werden.
   (b) Der Operator sperrt die Daten aus den produktiven Tabellen mittels einer Lesesperre. Welche Daten zu sperren sind, ist in den temporären Tabellen festgelegt. Anschließend werden die Daten gelesen.
   (c) Die Daten und das mittels des Parameters **schema** übergebene XML-Schema werden verknüpft, und es werden die XML-Dokumente erzeugt.
   (d) Die XML-Dokumente werden an dem in der **to**-Klausel angegebenen Ort abgelegt.
   (e) Die Lesesperren werden in Schreibsperren umgewandelt, und die Daten werden in den produktiven Tabellen gelöscht.
   (f) Der Operator gibt die Sperren frei.
4. Nun können die Daten in den temporären Tabellen gelöscht werden und eventuell die XML-Dokumente weiterverarbeitet werden.

Diese Vorgehensweise soll an folgendem Beispiel verdeutlicht werden: Wir gehen von einem Elektronikfachgeschäft aus. Aus der Datenbasis des Unternehmens sollen Belege archiviert werden. Die Datenbasis aus Abbildung 7 soll dabei als Grundlage dienen (Schlüsselfelder sind fett gedruckt).

Es sollen nun die beiden ersten Bestellungen archiviert werden. Dies geschieht mit folgendem Befehl:

```
archive data using schema /home/zeller/Beleg.xsd
                   prefix ARCHOP_
                   keycolumn ArchKey
from          ARCHOP_Bestellung, ARCHOP_Bestellposition
to            table BelegArchiv
```

Die Archivierung wird nun gemäß des oben dargestellten Algorithmus durchgeführt (siehe Abbildung 6). Bei der Archivierung werden dann die in Abbildung 9 dargestellten XML-Dokumente erzeugt. Diese XML-Dokumente enthalten die Daten der beiden betriebswirtschaftlichen Objekte. Das bei der Archivierung verwendete Schema *Beleg.xsd* ist in Abbildung 8 dargestellt. Sind die XML-Dokumente erzeugt, werden die Daten in den produktiven Tabellen gelöscht. Der dabei verwendete Löschalgorithmus wird im folgenden Abschnitt erklärt.

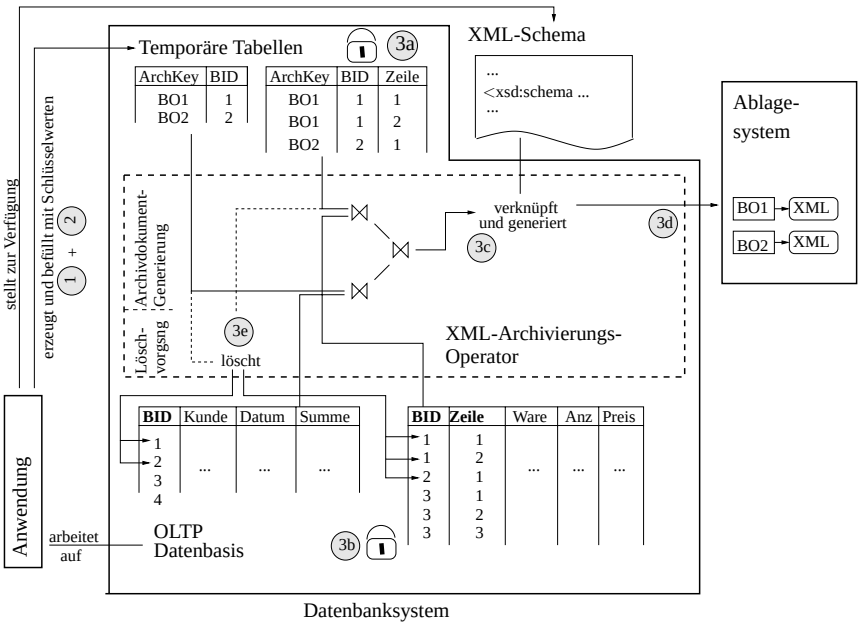


Abbildung 6: Ablauf einer Archivierung gemäß Algorithmus

Bestellung			
BID	Kunde	Datum	Summe
1	4711	3.5.2002	1200
2	5678	3.5.2002	57
3	3456	4.5.2002	107
4	5678	4.5.2002	10
...	...	...	...

Bestellposition				
BID	**Zeile**	Ware	Anz.	Preis
1	1	Fernseher	1	1000
1	2	Videorecorder	1	200
2	1	Radio	1	57
3	1	1,5 V Batterie	10	0,10
3	2	Lautsprecher	2	23
3	3	CD's	2	30
...	...	...	...	...

Abbildung 7: Beispiel-Tabellen eines Elektronikfachgeschäftes

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.sap.com/archive/axml/document" elementFormDefault="qualified"
targetNamespace="http://www.sap.com/archive/axml/document">
  <xsd:element name="Beleg" type="ItemListType" />
  <xsd:complexType name="ItemListType">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="BelegKopf" type="Bestellung" />
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="BelegPositionen">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="Position" type="Bestellposition" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Bestellung">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" name="BID" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellung.BID</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="KundenNummer" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellung.Kunde</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="BestellDatum" type="CHAR">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellung.Datum</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="GesamtSummeInEuro" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellung.Summe</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Bestellposition">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="1" name="BID" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellposition.BID</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="Zeile" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellposition.Zeile</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="Ware" type="CHAR">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellposition.Ware</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="Menge" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellposition.Anz</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="1" name="Preis" type="NUMBER">
        <xsd:annotation>
          <xsd:appinfo>sql:SAPR3.Bestellposition.Preis</xsd:appinfo>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="NUMBER">
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
  <xsd:simpleType name="CHAR">
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
</xsd:schema>
```

Abbildung 8: Beleg.xsd: XML-Schema für die Beispiel-Belege

```
Dokument BO1:
=============

<?xml version="1.0" encoding="UTF-8"?>
<Beleg xmlns="http://www.sap.com/archive/axml/document"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.sap.com/archive/axml/document schema.xsd">
  <BelegKopf>
    <BID>1</BID>
    <KundenNummer>4711</KundenNummer>
    <BestellDatum>20020503</BestellDatum>
    <GesamtSummeInEuro>1200.0</GesamtSummeInEuro>
  </BelegKopf>
  <BelegPositionen>
    <Position>
      <BID>1</BID>
      <Zeile>1</Zeile>
      <Ware>Fernseher</Ware>
      <Menge>1.0</Menge>
      <Preis>1000.0</Preis>
    </Position>
    <Position>
      <BID>1</BID>
      <Zeile>2</Zeile>
      <Ware>Videorecorder</Ware>
      <Menge>1.0</Menge>
      <Preis>200.0</Preis>
    </Position>
  </BelegPositionen>
</Beleg>

Dokument BO2:
=============

<?xml version="1.0" encoding="UTF-8"?>
<Beleg xmlns="http://www.sap.com/archive/axml/document"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.sap.com/archive/axml/document schema.xsd">
  <BelegKopf>
    <BID>2</BID>
    <KundenNummer>4711.0</KundenNummer>
    <BestellDatum>20020503</BestellDatum>
    <GesamtSummeInEuro>57.0</GesamtSummeInEuro>
  </BelegKopf>
  <BelegPositionen>
    <Position>
      <BID>2</BID>
      <Zeile>1</Zeile>
      <Ware>Radio</Ware>
      <Menge>1.0</Menge>
      <Preis>57.0</Preis>
    </Position>
  </BelegPositionen>
</Beleg>
```

Abbildung 9: Erzeugtes XML-Dokument eines Beleg-Objektes

3.4 Der Löschalgorithmus des XML-Archivierungs-Operators

Das massenhafte Löschen von Daten in relationalen Datenbanksystemen kann zu erheblichen Performance-Problemen führen, da die traditionell verwendete Löschmethode die Zugriffe auf den Hintergrundspeicher nicht optimiert. Bei der traditionellen Löschmethode werden die Einträge in den Tabellen einzeln und separat voneinander gelöscht. Dadurch kommt es zu vielen zufälligen Zugriffen auf den Hintergrundspeicher (Random IO), die wiederum wesentlich mehr Zeit in Anspruch nehmen als ein sequenzieller Zugriff (Sequential IO). Wie unsere Messungen in einem kommerziellen Datenbanksystem zeigen, dauert das Löschen von 15% der Einträge aus einer 500 MB großen Tabelle mit 1.000.000 Einträge bereits fast 3 Stunden (siehe Abbildung 10).

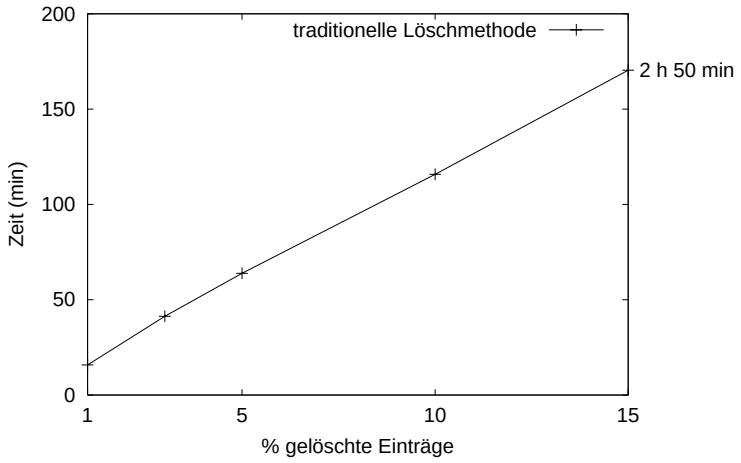


Abbildung 10: Löschen aus einer 500 MB großen Tabelle in einem kommerziellen Datenbanksystem

Wir haben deshalb in [GKKZ01] einen neuen Operator für Massenlöschungen entwickelt: den *Bulkdelete*-Operator. Dieser neue Bulkdelete-Operator vermeidet den zufälligen Zugriff auf den Hintergrundspeicher und erzwingt stattdessen einen sequenziellen Zugriff. Dies geschieht durch geschicktes Ändern der Löschreihenfolge, je nach zu bearbeitender Datenbankstruktur. So wird beim Löschen von Tabelleneinträgen in der Reihenfolge der physikalischen Speicheradresse (RID) gelöscht, und bei B-Baum-Indexen werden die Zugriffe bzgl. der indizierten Werte sortiert. Abbildung 11 zeigt einen möglichen Ausführungsplan für einen solchen Bulkdelete-Löschvorgang. Ein Pfeil neben dem Join-Symbol bedeutet dabei, dass in der Eingaberelation auf der Seite des Pfeiles während des Joinvorgangs auch gleichzeitig Einträge gelöscht werden. Die Menge *D* gibt an, welche Einträge in der Tabelle T gelöscht werden sollen (Delete-Set).

In [GKKZ01] wurden noch weitere Auswertungs- und Optimierungsmöglichkeiten untersucht und auch Messungen mit einem Prototypen durchgeführt. Diese Messungen zeigen, dass der Bulkdelete-Operator um ein vielfaches schneller arbeitet als die herkömmlichen Löschalgorithmen. Der Bulkdelete-Operator ist jedoch in keinem kommerziellen Datenbanksystem verfügbar. Um dessen Kernidee dennoch für den XML Operator zu nutzen,

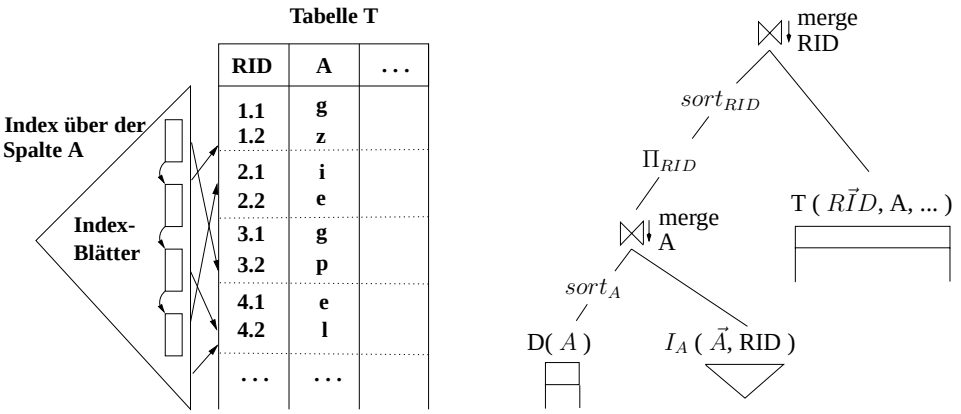


Abbildung 11: Möglicher Auswertungsplan eines Bulkdelete-Operators

haben wir eine leicht zu implementierende *light*-Variante des Bulkdelete-Operators realisiert. Auch die light-Variante erzwingt einen sequenziellen Zugriff durch das Verändern der Löschreihenfolge, allerdings nur beim Zugriff auf die Tabelle. Beim Bearbeiten der Indexe wird auch hier zufällig auf den Hintergrundspeicher zugegriffen.

Die genaue Funktionsweise dieser light-Variante soll an folgendem Beispiel verdeutlicht werden. Gegeben sei eine Tabelle *T*. *T* enthält eine Spalte A, über die ein Index I_A definiert ist (siehe Abbildung 12). Außerdem ist zu jedem Tabelleneintrag die physikalische Speicheradresse abrufbar[2] (hier durch die Spalte *RID* dargestellt). Ein Benutzer möchte nun mehrere Einträge aus *T* löschen und speichert dazu die A-Werte dieser Einträge in einer Tabelle *D* ab (siehe ebenfalls Abbildung 12). Um nun einen sequenziellen Zugriff

T			
RID	...	A	...
001	...	4	...
002	...	7	...
003	...	5	...
004	...	9	...
005	...	3	...
006	...	5	...

D
A
3
7
5

Abbildung 12: Beispieltabellen zur Verdeutlichung der Arbeitsweise der light-Variante

auf *T* zu erzwingen, werden die Einträge der Tabelle *D* bzgl. der physikalischen Adressen der korrespondierenden Einträge aus *T* sortiert. Anschließend werden die Einträge in *T* gelöscht. Dies geschieht mit folgendem SQL-Aufruf:

```
delete from T where T.A in
      (select T.A
       from   D, T
       where  D.A = T.A
       order by T.RID)
```

[2]Die meisten Datenbankhersteller bieten eine entsprechende Funktionalität an. Im nächsten Absatz wird aufgezeigt, welche Tuningmaßnahmen möglich sind, wenn eine entsprechende Funktionalität fehlt.

Dabei ist zu beachten, dass der Join $T \bowtie D$ ausgeführt werden kann, ohne die Tupel aus *T* wirklich zu lesen. Stattdessen kann der Index I_A benutzt werden, um die entsprechenden (T.A, RID) Paare zu finden. Dadurch kann die Unteranfrage sehr effizient beantwortet werden. Ist der Index I_A als B-Baum realisiert und liegt eine Ballung (Clusterung) der Daten in der Tabelle *T* bzgl. der Spalte A vor, so werden durch die light-Variante auch die Blätter des Indexes I_A sequenziell bearbeitet. Sollte kein geeigneter Index existieren (d.h. es existiert kein Index, der ein in *D* gegebenes Attribut indiziert), so kann die light-Variante nicht verwendet werden. In diesem Fall muss aber ohnehin die ganze Tabelle durchsucht werden, da es ohne einen geeigneten Index keine andere Möglichkeit gibt, die zu den Einträgen in *D* korrespondierenden Einträge in *T* zu finden.

Sollte es desweiteren nicht möglich sein, die physikalischen Speicheradresse eines Tabelleneintrages herauszufinden und werden B-Bäume als Indexstrukturen verwendet, so müssen die Einträge in der Tabelle *D* gemäß ihren Attributwerten sortiert werden. Dadurch werden zumindest die Blätter des Indexes I_A sequenziell gelesen. Da die Daten in betriebswirtschaftlichen Datenbanksystemen wegen der Verwendung von künstlichen, aufsteigend nummerierten Schlüsselwerten oft auch sehr gut bzgl. der indizierten Tabellenspalten geballt sind, wird auf diese Weise auch die Tabelle *T* sequenziell gelesen.

Bei der Archivierung mit dem XML-Archivierungs-Operator wird die light-Variante des Bulkdelete-Algorithmus in Verbindung mit den temporären Tabellen eingesetzt, um die Daten in den produktiven Tabellen zu löschen. Dabei werden delete-Ausdrücke der Form

```
delete from <produktive Tabelle T> where <T.Schlüssel> in
      (select <T.Schlüssel>
       from   <produktive Tabelle T>, <temporärer Tabelle D>
       where  <T.Schlüssel>=<D.Schlüssel>
       order by <T.RID>)
```

verwendet. Die light-Variante ist im Idealfall[3] genauso effizient wie der Bulkdelete-Operator und ist auch in allen übrigen Fällen den traditionellen "tuple-at-a-time"-Ansätzen überlegen. Dies belegen die im folgenden Abschnitt beschriebenen Messungen.

4 Beispiel-Implementierung und Messungen

Wir haben eine Reihe von Tests mit einer Beispiel-Implementierung des XML-Archivierungs-Operators durchgeführt. Für unsere Beispiel-Implementierung haben wir den XML-Archivierungs-Operator als *java stored procedure* realisiert. Als zugrundeliegende Datenbank benutzten wir ein sehr weit verbreitetes, kommerzielles Datenbanksystem. Wir haben 2 Varianten des XML-Archivierungs-Operators implementiert:

- Traditionell (**trad**): Als Löschmethode wurde hier die traditionelle Löschmethode verwendet.
- Bulkdelete *light* (**light**): Als Löschmethode wurde hier die light-Variante des Bulkdelete-Algorithmus verwendet.

Bei den Messungen wurde untersucht, wie hoch der Performancegewinn beim Einsatz der light-Variante des Bulkdelete als Löschmethode bereits ist. Als Messdaten wurden die

[3] Im Idealfall ist nur ein Index vorhanden und es liegt eine Ballung der Daten bzgl. der indizierten Spalte vor.

Daten eines von uns im Rahmen einer Kooperation mit SAP entwickelten Archivierungsbenchmarkes verwendet. Die Daten sind an Finanzbuchhaltungsdaten eines SAP-Systems (R/3 4.6C) angelehnt. Die Datenbank lief auf einer SUN Enterprise 450 mit 2 GB Hauptspeicher und 4 Prozessoren a 400 MHz. Die Datenbank belegte 512 MB Hauptspeicher. Als Speichermedium wurde ein SUN A1000 RAID mit 500 GB Speicherkapazität und RAID Level 5 verwendet.

Um die Auswirkungen der Komplexität des betriebswirtschaftlichen Objektes auf den Archivierungsvorgang zu untersuchen, haben wir die Laufzeiten bei 4 unterschiedlichen Objekten gemessen. Die Objekte unterscheiden sich sowohl in der Anzahl der Tupel pro Tabelle, als auch in der Breite der Tupel, d.h. im Verhältnis Daten zu Null-Werten innerhalb eines Tupels (siehe Tabelle 1).

	Tupel stark gefüllt	Tupel wenig gefüllt
wenige Tupel pro Tabelle	Objekt 1	Objekt 2
viele Tupel pro Tabelle	Objekt 3	Objekt 4

Tabelle 1: Definition der Objekttypen

Bei den ersten Messungen wurde die Anzahl der archivierten Objekte variiert. Die Datenbasis hatte eine Größe von 10.000 Objekten. Die Daten jedes der Objekte waren auf 7 Tabellen verteilt. Die Abbildungen 13 und 14 zeigen die Laufzeiten der Löschoperationen bei der Archivierung der verschiedenen Objekttypen.

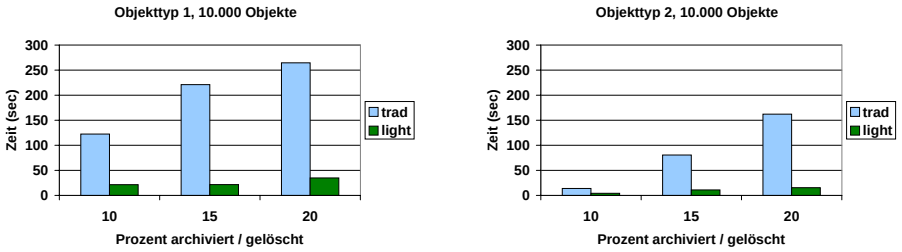


Abbildung 13: Laufzeiten der Löschoperationen für Objekttyp 1 und 2

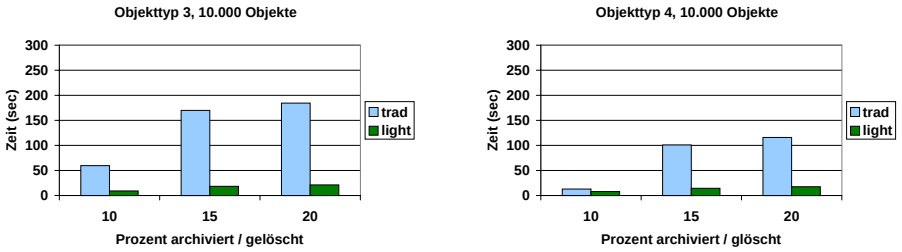


Abbildung 14: Laufzeiten der Löschoperationen für Objekttyp 3 und 4

Wie zu sehen ist, nimmt die Laufzeit der Löschoperation bei der traditionellen Löschmethode (trad) mit zunehmender Menge zu löschender Tupel stark zu. Bei der light-Variante

sind die Laufzeiten wesentlich kleiner, und sie steigen auch mit zunehmender Anzahl gelöschter Einträge weniger stark an. Dies liegt daran, dass die light-Variante jede Hintergrundspeicherseite maximal einmal liest. Die Laufzeit ist deshalb durch die Anzahl der Hintergrundspeicherseiten beschränkt. Bei der traditionellen Variante kann jede Löschung eines Eintrages zu einem Zugriff auf eine andere Hintergrundspeicherseite führen (Random IO). Die Laufzeit ist also hier nur durch die Anzahl der Tupel in der Tabelle beschränkt, die in der Regel sehr viel größer ist als die Anzahl der von der Tabelle belegten Hintergrundspeicherseiten.

In Abbildung 15 sind die Laufzeiten für die XML-Dokument-Generierung zu sehen. Wie zu sehen ist, sind die Zeiten für die XML-Dokument-Generierung wesentlich höher als die Löschzeiten (Minuten gegenüber Sekunden). Obwohl die Zeit für die Datengenerierung dominiert, sind die Löschzeiten keineswegs vernachlässigbar, denn während der Löschphase ist die Tabelle in der Regel komplett mittels einer Schreibsperre gesperrt und daher für die zeitkritischen OLTP-Anwendungen nicht verfügbar. Viele Datenbankensysteme wechseln nämlich bei derart datenaufwendigen Transaktionen, wie einer Massenarchivierung, von einer kleineren Sperr-Granularitäten (einzelner Eintrag, Seite) zur nächst höheren (Seite, komplette Tabelle), um den Aufwand für die Verwaltung der Sperren zu minimieren. Bei der Datengenerierung ist zwar auch die ganze Tabelle gesperrt, aber nur mittels einer Lesesperre. Andere Transaktionen können die Tabellendaten also während der XML-Generierung lesen.

Einige Datenbankhersteller bieten die Möglichkeit an, Datenbankprozesse auf dem rufenden Client zu starten. Dadurch wird der Datenbankrechner entlastet und die Ressourcen des Clients besser ausgenutzt. Wir haben deshalb eine Variante des XML-Archivierungs-Operators implementiert, bei der die speicherintensive XML-Generierung auf dem rufenden Client ausgeführt wird (**client**) und sie mit der serverseitigen Ausführung (**server**) verglichen. Wie in Abbildung 16 zu sehen ist, profitierte die **client**-Variante von der exklusiven Nutzung der Ressourcen des Clients sehr stark.

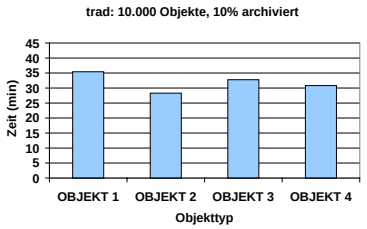


Abbildung 15: Laufzeiten der XML-Dokument-Generierung

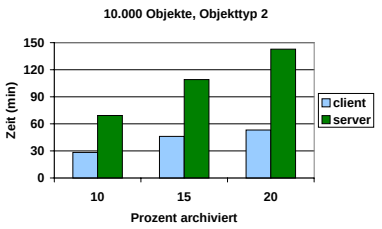


Abbildung 16: Vergleich serverseitige mit clientseitiger XML-Generierung

5 Zusammenfassung

Einer der wichtigsten Einsatzbereiche relationaler Datenbanksysteme liegt in der Verwaltung und Auswertung betriebswirtschaftlicher Daten. Die Datenbanksysteme dienen dabei als Datenspeicher für betriebswirtschaftliche Anwendungssysteme, wie SAP R/3. Anwen-

dungsspezifische Vorgänge, wie z.B. das Definieren und Sperren von betriebswirschaftlichen Objekten, werden von heutigen Datenbanksystemen praktisch nicht unterstützt. Es ist deshalb notwendig, die Schnittstellen der Datenbanken zu erweitern, anzupassen und zu vereinfachen. Ein erster Schritt in diese Richtung sind die XML-Schnittstellen und die objekt-relationalen Erweiterungen, die in fast allen Datenbanksystemen inzwischen zu finden sind.

Wir haben in dieser Arbeit eine Erweiterung relationaler Systeme vorgeschlagen, für die noch keine entsprechende Schnittstelle auf Datenbankebene existiert: den XML-Archivierungs-Operator. Unter Archivierung versteht man dabei das Verschieben der Daten von selten benötigten betriebswirtschaftlichen Objekten aus den produktiven Datenbanksystemen auf kostengünstigere Tertiärspeichersysteme. Dadurch werden die Datenbasis der produktiven Datenbank verkleinert, die Leistung der Datenbank erhöht und letztlich Kosten gesenkt.

Der vorgeschlagene XML-Archivierungs-Operator erlaubt es, im Gegensatz zu den auf Applikations-Ebene implementierten Archivierungskomponenten, den gesamten Archivierungsablauf auf Datenbank-Ebene auszuführen und die Daten als XML-Dokumente abzulegen. Dadurch wird der Datenverkehr zwischen Anwendung und Datenbank verringert. Durch die Ablage als XML wird die Portabilität der Daten erhöht und die Implementierung separater Archivierungskomponenten unnötig gemacht. Vor allem die Ablage als XML macht die Daten nicht nur für andere Anwendungen verfügbar, sondern minimiert auch den Aufwand (und damit die Kosten) bei einem Versionswechsel auf Anwendungsseite erheblich. Der XML-Archivierungs-Operator nutzt intelligente Löschalgorithmen, was den Löschvorgang erheblich beschleunigt und somit den Durchsatz des Gesamtsystems erhöht.

Danksagungen

Wir danken Herrn Klaus Zimmermann, Herrn Thomas Wondrak, Herrn Stefan Krompaß und Herrn Constantin Holzner für die Implementierung des Prototypen und des XML-Archiv-Browsers sowie für die Durchführung der Messungen. Desweiteren danken wir Herrn Wolfgang Becker und Herrn Dr. Ulrich Marquard für die gute Zusammenarbeit im Rahmen des Terabyte-Projektes. Unser Dank gilt außerdem den anonymen Gutachtern für die hilfreichen Kommentare.

Literaturverzeichnis

[ABS99] S. Abiteboul, P. Buneman, and D. Suciu. *Data On The Web, From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1999.

[BEK+00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. `http://www.w3.org/TR/SOAP`, May 2000.

[CFI+00] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Publishing Object-Relational Data as XML. In *Proceedings of the Third International Workshop on the Web and Databases*, pages 105–110, Dallas, USA, May 2000.

[DFF+99] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. In *Proceedings of the Eighth International World-Wide Web Conference*, 1999.

[FTS00] M.F. Fernandez, W.-C. Tan, and D. Suciu. "SilkRoute: Trading between Relations and XML". In *Int'l World Wide Web Conf. (WWW)*, Amsterdam, Netherlands, May 2000.

[GKKZ01] A. Gärtner, A. Kemper, D. Kossmann, and B. Zeller. Efficient Bulk Deletes in Relational Databases. In *Proc. IEEE Conf. on Data Engineering*, pages 183–192, Heidelberg, Germany, 2001.

[Her97] A. Herbst. *Anwendungsorientiertes DB-Archivieren. Neue Konzepte zur Archivierung in Datenbanksystemen*. Springer Verlag, 1997.

[JDO] JDOM Projekt. `www.jdom.org`.

[KN99] K. Küspert and J. Nowitzky. Partitionierung von Datenbanktabellen (Aktuelles Schlagwort). *Informatik Spektrum*, 22(2):146–147, April 1999.

[KS98] K. Küspert and R. Schaarschmidt. Archivierung in Datenbanksystemen (Das aktuelle Schlagwort). *Informatik Spektrum*, 21(5):277–278, October 1998.

[LS98] J. Lufter and R. Schaarschmidt. Auswirkungen von Schemaänderungen einer Datenbank auf die datenbanksystem-integrierte Archivierung. Forschungsergebnisse der Fakultät für Mathematik und Informatik Math/Inf/98/07, Institut für Informatik, Friedrich-Schiller-Universität Jena, March 1998.

[Moh02] C. Mohan. An Efficient Method for Performing Record Deletions and Updates Using Index Scans. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 940–949, HongKong, China, August 2002.

[Now99] J. Nowitzky. Partitionierung relationaler Datenbanktabellen und deren Anwendung für die Datenarchivierung. In *Tagungsband des 11. Workshop „Grundlagen von Datenbanken", Luisenthal/Thüringen, Mai 1999*, Jenaer Schriften zur Mathematik und Informatik, Math/Inf/99/16, pages 77–81, Friedrich-Schiller-Universität Jena, May 1999.

[Now01] J. Nowitzky. Partitionierungstechniken in Datenbanksystemen: Motivation und Überblick. *Informatik Spektrum*, 24(6):345–356, December 2001.

[Rys01] M. Rys. State-of-the-art XML Support in RDBMS: Microsoft SQL Server's XML Features. In *IEEE Data Engineering Bulletin, Vol 24, No 2*, pages 3–11. June 2001.

[SBB+02] H. Stefani, B. Brinkmöller, G. Buchmüller, G. Fischer, M. Fischer, R. Gentinetta, A. Herbst, J. Nolte-Bömelburg, T. Pferdekämper, G. Scherer, and P. Zimmerer. *Datenarchivierung mit SAP*. SAP PRESS, 2002.

[SBH+98] R. Schaarschmidt, K. Bühnert, A. Herbst, K. Küspert, and R. Schindler. Konzepte und Implementierungsaspekte anwendungsorientierten Archivierens in Datenbanksystemen. *Informatik Forschung und Entwicklung*, 13(2):79–89, June 1998.

[Sch01] R. Schaarschmidt. *Archivierung in Datenbanksystemen: Konzept und Sprache*. Teubner-Reihe Wirtschaftsinformatik. Verlag B. G. Teubner, Stuttgart, Leipzig, Wiesbaden, 2001.

[SR97] R. Schaarschmidt and W. Röder. Datenbankbasiertes Archivieren im SAP System R/3. *Wirtschaftsinformatik*, 39(5):469–477, October 1997.

[SSB+01] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald. Efficiently publishing relational data as XML documents. *The VLDB Journal*, 10(2-3):133–154, 2001.

[VW02] V. Votsch and M. Walter. Oracle XML DB, 2002. www.oracle.com/ip/deploy/database/oracle9i/collateral/xmldb_buswp.pdf.

[XML00] XML Schema, April 2000. `http://www.w3.org/xml/Schema`.

[ZK02] B. Zeller and A. Kemper. Experience Report: Exploiting Advanced Database Optimization Features for Large-Scale SAP R/3 Installations. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, HongKong, China, August 2002.