EIN SUBSYSTEM ZUR STABILEN SPEICHERUNG VERSIONENBEHAFTETER, HIERARCHISCH STRUKTURIERTER TUPEL

U. Deppisch, V. Obermeit, H.-B. Paul, H.-J. Schek, M. Scholl, G. Weikum

Technische Hochschule Darmstadt
FB Informatik
FG Datenverwaltungssysteme I
Alexanderstr. 24
D-6100 Darmstadt

Ersch. in: Datenbanksysteme in Büro, Technik und Wissenschaft. GI-Fachtagung, Karlsruhe, 20.–22. März 1985 Proceedings / A. Blaser und P. Pistor [Hg.]. - Berlin [u.a.] : Springer, 1985. - (Informatik-Fachberichte ; 94). - S. 421-440. - ISBN 3-540-15196-6

KURZFASSUNG

Das hier beschriebene Speichersystem unseres Darmstädter Datenbankkernsystems kann als Erweiterung der Dateiverwaltung von Betriebssystemen um Grundfunktionen eines Datenbanksystems betrachtet werden: Es dient der stabilen Speicherung von strukturierten Datensätzen unter Einschluß einer Transaktionsverwaltung und optionaler Versionierung. Zur Strukturierung werden ausschließlich Relationen mit atomaren oder relationenwertigen Attributen zugelassen. Zur Manipulation erlauben wir im wesentlichen eine geschachtelte Projektion und mengenorientierte Änderungsoperationen. Eine solche Schnittstelle relativ weit unten in der Systemarchitektur erlaubt es, gezielt auf einzelne Daten und gleichzeitig effizient auf Datenmengen zuzugreifen. Dementsprechend ist die direkt darunterliegende Pufferschnittstelle (seiten-) mengenorientiert angelegt. Für die Transaktionsverwaltung verwenden wir das Konzept der offenen geschachtelten Transaktionen bereits innerhalb des Speichersystems. Dieses Konzept soll dann auch für die höheren Systemschichten angewandt werden.

ABSTRACT

The storage subsystem of our Darmstadt database kernel system is described here. We regard it as an extension of the file management of operating systems: It supports stable storage of structured records, transaction management, and optionally time versions. As the only data structure we allow relations with atomic or relation valued attributes. For data manipulation we support a nested relational projection and set oriented update as basic functions. Such an interface at a low level of the system architecture will allow efficient access to both single data items and sets of data. Accordingly the next deeper buffer interface supports access to sets of pages. For transaction management we apply the concept of open nested transactions already within the subsystem. This concept will then be extended for higher levels too.

1 ANFORDERUNGEN UND ZIELSETZUNGEN

Die Datenbankforschung versucht augenblicklich, auf neuartige Anwendungsgebiete zu reagieren, die gemeinhin unter dem Begriff "Nicht-Standard-Anwendungen" subsummiert werden (/DKML84/, /HR83/, /SP82/). Dabei werden vor allem Informationssysteme im Ingenieurbereich, geographische Informationssysteme und Büroinformationssysteme oder allgemein die Verwaltung von Dokumenten als Einsatzgebiete zukünftiger Datenbanksysteme genannt. Konventionelle Datenbanksysteme sind für die Verwaltung der in diesen Gebieten auftretenden "komplexen Objekte" (/HL82/) nicht gerüstet.

Ein Ausbau durch eine weitere Schicht, eine sogenannte Preprocessor-Lösung, verbietet sich wegen der damit verbundenen Performance-Einbuße, die wiederholt befürchtet und festgestellt wurde (/Eb84/, /SP82/, /GP83/). Den Grund hierfür kann man sich wie folgt klarmachen: In der Architktur eines aus Preprocessor und konventionellem DBMS bestehenden Gesamtsystem muß die untere Schicht (das konventionelle DBMS also) die Anforderungen der oberen Schicht **effizient** ausführen. Das konventionelle DBMS wird daher zur **internen** Schnittstelle im Sinne der Drei-Schichten-Architektur. Dies war aber nicht die oberste Entwurfsmaxime bei einem System wie System R (/As76/). Im Gegenteil, die übergeordnete Zielsetzung für solche Systeme war die leichte Anwendbarkeit bei klassischen Anwendungen. Nun führt aber die Zielsetzung der **Effizienz** für die Ausführung von Operationen der **Nicht-Standard-Anwendungen** und die Zielsetzung des **Komforts** für Standard-Anwendungen zu unterschiedlicher Systemstruktur.

Unser Ansatz besteht nun darin, ein Datenbank**kern**system, das DArmStädter DatenBankSystem (DASDBS) zu entwickeln (/PSSW84/), das als gemeinsame Basis für verschiedene Anwendungsgebiete geeignet ist und jeweils unterschiedlich ausgebaut werden kann. Ein solches Kernsystem soll wesentliche Komponenten eines Datenbanksystems enthalten, die jeweils um einzelne Aspekte erweitert bzw. auf die speziellen Anforderungen bzw. Einsatzgebiete hin, etwa in Server-Workstation-Umgebungen, optimiert werden können. Dazu ist als oberste Maxime eine Schichtenarchitektur erforderlich, bei der klar definierte Schnittstellen der einzelnen Moduln bei geheimgehaltener Implementierung es erlauben, je nach Anforderung die eine oder andere Implementierungstechnik für die gleichen Funktionen auszuwählen.

Über die Ideen der Modularisierung und des Kernsystemansatzes hinaus erscheint es uns wichtig, die Funktionen von DASDBS nochmals in zwei große Bereiche aufzuteilen. Die innere Ebene ist das Speicherungssubsystem (COS), das im folgenden beschrieben wird, auf dem dann eine Datenmodellschicht (NAS) aufgesetzt ist, vergleichbar mit der Teilung von System R in RDS und RSS (/As76/). Wir betrachten das COS als eine erweiterte idealisierte Dateischnittstelle, ähnlich wie "Database File Systems" (/Ro82/) oder "Speicher Server" (/Mi84/). Im Gegensatz zu einem konventionellen Dateiverwaltungssystem enthält das COS Datenstrukturen und Transaktionsverwaltung, also Grundfunktionen eines Datenbanksystems.

Datenmodelle für die Unterstützung der "Nicht-Standard"-Anwendung werden zur Zeit heftig diskutiert (z.B. /SLTC82/, /Neu83/, /LS83/). In unserem System und in dieser Arbeit wird das NF^2-Relationenmodell (/JS82/) zugrunde gelegt, das durch die relationenwertigen Attributwerte für die Modellierung von komplexen Objekten geeignet erscheint (/HL82/, /SP82/) und in den theoretischen Grundlagen mittlerweile weiter entwickelt wurde (/Ja84/, /SS84/). Charakteristisch für das Modell ist, daß Relationen als Attributwerte auftreten können. In DASDBS sind **alle** Objekte, die verwaltet werden, speziell auch Objekte im Subsystem, als NF^2-Tupel beschrieben und werden mit den Operationen der NF^2-Algebra manipuliert. Dadurch zieht sich durch alle Ebenen des Subsystems die Mengenorientierung der Schnittstellen, vom Pufferverwalter bis hin zum

Versionen-Manager. Auf diese Art sollen sowohl gezielte Zugriffe auf Teile der gespeicherten NF^2-Tupel als auch globale Zugriffe auf Tupel oder gar Relationen (etwa für die Datenextraktion aus dem Server in eine Workstation) in gleichem Maße unterstützt werden.

Ein weiteres wichtiges Charakteristikum von DASDBS ist die Anwendung des Prinzips der "offenen geschachtelten Transaktionen" für die Transaktionsverwaltung durch unsere Schichten (/WS84/, /Wei84/). Im Gegensatz zu den geschachtelten Transaktionen aus /Wa84/, die als Hilfsmittel zur Strukturierung von Anwendungen verstanden werden, oder /Mo82/, wo die speziellen Probleme verteilter Systeme im Vordergrund stehen, ist der hier beschriebene Typus geschachtelter Transaktionen stets starr an eine Schichtenarchitektur gekoppelt. Die unterste Schicht, der stabile Speicher (SMM), enthält bereits ein (Sub-) Transaktionskonzept. Dieses wird angewandt für die Implementierung eines Transaktionskonzeptes auf der CRM-Ebene. Mit jeder Ebene, um die unser Kernsystem erweitert wird, kann dabei auch die Transaktionsverwaltung nach oben gewissermaßen "mitwachsen" (/Sch84/).

Die im Subsystem (s. Abb. 1) integrierten Funktionen werden in den Kapiteln 3 bis 5 näher beschrieben. Dabei handelt es sich im wesentlichen um drei Aspekte: die unterste Schicht im COS umfaßt Pufferverwaltung und den sog. stabilen Speicher SMM (s. Kapitel 5), die zusammen den Unterschied zwischen stabiler (peripherer) und flüchtiger Speicherung (im Hauptspeicher) nach oben hin verbergen und damit einen Beitrag zur Implementierung des Transaktionskonzeptes erbringen.

Als nächstes folgt der Complex Record Manager (CRM) (s. Kapitel 3), der die Abbildung zwischen NF^2-Tupeln und dem linearen, seitenstrukturierten Adreßraum des SMM leistet. An der Schnittstelle des CRM gibt es nur einen einzigen Typ von Daten, nämlich die NF^2-Relation. Wir haben bewußt diesen puristischen Standpunkt bezogen, alle Daten - auch Verwaltungsinformation, wie etwa den Katalog - im Modell auszudrücken, um mit möglichst wenigen Konzepten zur Klarheit und Handhabbarkeit des Systems beizutragen. Zu dieser "Schlankheit" des System gehören sowohl Datenmodellaspekte (wir erlauben nur relationenwertige nicht etwa auch listen- oder tupelwertige Attribute, vgl. /PHH83/) als auch Implementierungsgesichtspunkte. So gibt es nur einen Seitentyp, also keine Trennung von Daten- und Zugriffspfadseiten. Zugriffspfade sind nicht einmal Bestandteil des Speicherungssystems, sondern werden erst darauf implementiert, dann ebenfalls als NF^2-Relationen (hierin liegt ein wichtiger Unterschied zur Trennung RDS-RSS, da RSS Zugriffspfade wartet!). Dies erlaubt die Anpassung von Zugriffstechniken an die speziellen Anforderungen des jeweiligen Einsatzgebietes.

Die letzte Schicht des Subsystems schließlich, der Versionen-Manager, bietet die Möglichkeit, für einzelne Relationen optional zeitliche Veränderungen aufzuzeichnen. Dazu wird bei Änderungsoperationen der neue Zustand des betroffenen Objekts gespeichert, ohne den jeweiligen alten Zustand zu überschreiben. Die aufgezeichneten Werte bilden eine zeitlich geordnete Folge von **Versionen** eines Objekts; jede Version trägt die Zeitmarke ihrer Entstehung.

Die Notwendigkeit, Versionen zu führen, gibt es u.a. im Bereich des CAD/CAM (/KaLe84/, /MS83/), bei der Speicherung von Dokumenten in Bürosystemen (/Gü83/) und allgemein zur Unterstützung der Revision bei Buchungssystemen jeglicher Art. In konventionellen Datenbanksystemen finden Versionen für Concurrency Control und Recovery Verwendung (z.B. /Re83/). Entsprechend der Gesamtkonzeption unseres Kernsystems werden primär Versionen von NF^2-Tupeln unterstützt, woraus Versionen einer ganzen Relation abgeleitet werden können. Näheres über den Versionen-Manager ist in Kapitel 4 zu finden.

2 EINORDNUNG DES SUBSYSTEMS IN DIE UMGEBUNG

2.1 DAS SUBSYSTEM IN DER GESAMTARCHITEKTUR

Nachdem wir uns einen Überblick über die Funktionen des Speichersubsystems und ihre Implementierungen verschafft haben, geben wir nun zur Orientierung und Einordnung eine Übersicht über die Architektur der zweiten Komponente von DASDBS, des NF^2-Algebra-Systems (NAS). Die Aufgabe des NAS besteht im wesentlichen aus der Erweiterung des Funktionsumfanges des Speichersystems im Hinblick auf die Unterstützung des vollen Spektrums der NF^2-Algebra, ein Anwendungs-Transaktionskonzept, die Wartung von Zugriffspfaden und die Unterstützung von verschiedenen Sichten auf die gespeicherten Daten, insbesondere Datentypen. Da es sich auch an der äußeren Schnittstelle des NAS noch um ein Kernsystem handelt, ist vorgesehen, auf dieser Schicht verschiedene, jeweils anwendungspezifische, Moduln zu implementieren, so daß letztlich vollständige Datenbanksysteme für die jeweiligen Anforderungsprofile entstehen können. Der modulare Aufbau aller drei Komponenten ist in Abb. 1 dargestellt.

Direkt auf der Speicherschnittstelle operiert der "Single-Pass-Query-Processor" (SPQP), der eine Ein-Relationen-Schnittstelle mit Zugriffspfadunterstützung und -wartung bietet, auf der eine eingeschränkte Klasse von algebraischen Abfragen und Änderungsoperationen angeboten werden. Es handelt sich dabei um solche Operationen, die in einem Durchlauf (Scan) durch eine Relation bearbeitbar sind.

Komplexere, insbesondere zweistellige Operationen auf NF^2-Relationen werden durch Iteration auf solche SPQP-Aufrufe abgebildet. Dies ist die Aufgabe des Moduls Query Processor (QP), der außerdem die Auswahl der Zugriffsstrategie zur Laufzeit der Anfragen trifft. Die Auswertung des gewählten Zugriffpfads dagegen ist Sache des SPQP, soweit es sich um Ein-Relationen-Zugriffspfade handelt. Der Query Processor ist außerdem diejenige Schicht von DASDBS, die Datentypen als Domains von Attributen kennt, bzw. aus den Bytefolgen der darunterliegenden Schichten interpretiert. Ähnlich wie RSS in System R kennen also die unteren Schichten in unserer Architektur nur einen Datentyp "Bytestring".

Im weiteren Ausbau soll die Möglichkeit geboten werden, zwischen konzeptuellem und internem Schema einer Datenbank zu unterscheiden, evtl. sogar noch externe Sichten (Views) auf das konzeptuelle Schema ermöglichen (3-Schichten Architektur nach ANSI/SPARC). Durch diese Schichtenbildung können häufig vorkommende, teure Operationen - wie etwa Joins - "materialisiert" werden, was zu wesentlichen Performanceverbesserungen führt. Für den physischen Datenbankentwurf wird es eine eigene Komponente geben, die das Schema der internen Relationen inklusive den Zugriffspfaden allein aus Performanceaspekten festlegt. Das Subsystem arbeitet dann nur noch mit diesen internen Relationen. Eine Voraussetzung für die nichttrivialen Abbildungen zwischen den Schichten ist allerdings, daß die Anfragen, die sich durch Einsetzen der Transformationsgleichungen ergeben, effizient algebraisch optimiert werden können (vgl. /SS83/, /Scho82/). Die beiden Teilgebiete Schichtentransformation und Algebraische Optimierung sind im Modul NF^2-Algebra-Transformation und Optimierung (NATO) zusammengefaßt.

Das Prinzip der offenen geschachtelten Transaktionen wird im NAS entsprechend den Operationen des Query Processors durch eine prädikatenorientierte Concurrency Control fortgesetzt.

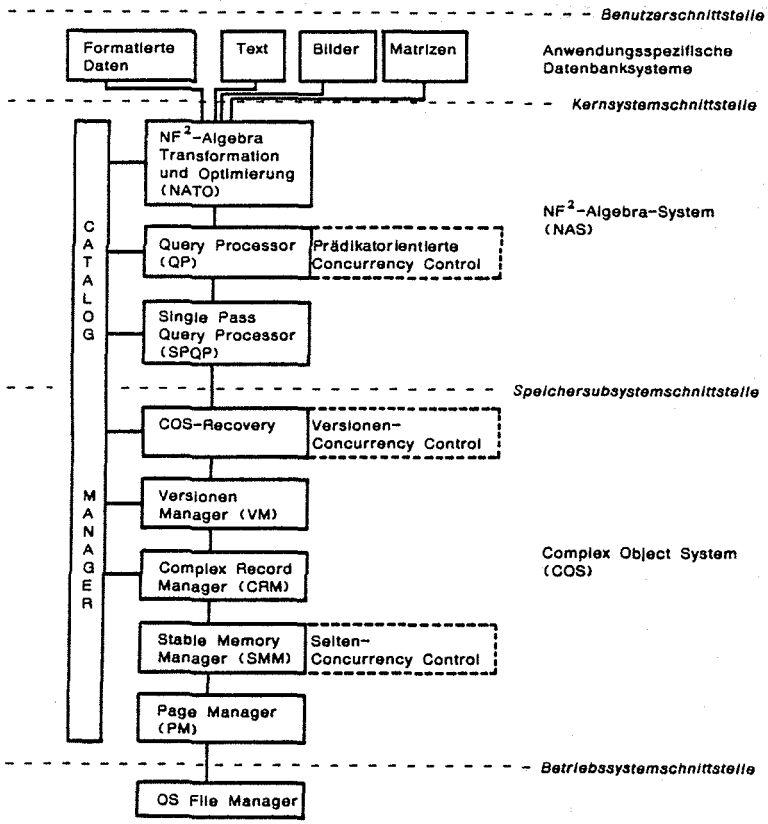

Abb. 1: Architektur von DASDBS (DArmStädter DatenBankSystem)

2.2 DAS NF²-RELATIONENMODELL

Das in DASDBS verwendete erweiterte sog. NF²-Relationenmodell baut auf dem bekannten Relationenmodell auf. Es wird jedoch, wie der Name NF² = Non-First-Normal-Form schon sagt, die Erste-Normalform-Bedingung aufgegeben, d.h. als Domains für Attribute werden nicht nur Mengen mit atomaren Werten sondern wiederum Relationen, also Elemente von Potenzmengen, zugelassen. Als einfaches Beispiel dient folgendes Abteilungsschema:

ABT(ANR, ABTNAME,
PERS(PNR, TBEZ, NAME, LEHRGANG(LNR, JAHR), KIND(NAME, GEBDAT)),
VERW(PNR, TBEZ, NAME))

ABT											
ANR	ABTNAME	PERS							VERW		
		PNR	TBEZ	NAME	LEHRGANG		KIND		PNR	TBEZ	NAME
					LNR	JAHR	NAME	GEBDAT			
1	Forschung	511	Programmierung	Knut	1 78 4 79 6 81		Helga 160781 Tom 010479		112	Sekretariat	Rüber
		552	Grundlagen	Bond	1 81 2 79		Julia 030878		121	Bibliothek	Meier
		648	Planung	Hurtig	1 80 2 81 3 82						

Abb. 2: Beispiel für eine Ausprägung der Relation ABT

Jedes Abteilungstupel (s. Abb. 2) hat die beiden atomaren Attribute Abteilungsnummer (ANR) und -name (ABTNAME) sowie als relationenwertige Attribute das technische Personal (PERS) und das Verwaltungspersonal (VERW). Jede Subrelation PERS ihrerseits besteht aus den atomaren Attributen Personalnummer (PNR), Tätigkeitsbezeichnung (TBEZ) und Name sowie aus den relationenwertigen Attributen LEHRGANG - mit Lehrgangsnummer (LNR) und Jahr - und KIND - mit Name und Geburtsdatum des Kindes. Die Subrelation VERW schließlich besteht aus den atomaren Attributen Personalnummer, Tätigkeitsbezeichnung und Name.

An diesem Beispiel sollen auch noch einige in den nachfolgenden Kapiteln beschriebenen Begriffe erläutert werden: Elemente einer NF^2-Relation werden im folgenden Strukturierte Tupel (Kosename: Strupel) oder NF^2-Tupel genannt. ABTNAME ist ein atomares Attribut (AV-Attribut), PERS ein relationenwertiges Attribut (RV-Attribut) von ABT. PERS ist damit eine Subrelation und die Tupel von PERS heißen Subtupel. Das gleiche gilt für LEHRGANG und KIND, nur eine Hierarchiestufe tiefer.

Die Operationen auf NF^2-Relationen sind ebenfalls eine Erweiterung der Operationen auf flachen Relationen und in /SS84/ formal beschrieben.

3 SPEICHERVERWALTUNG STRUKTURIERTER TUPEL

Die zentrale Komponente in unserem Subsystem ist der "Complex Record Manager" (CRM). Er führt die Abbildung zwischen den internen NF^2-Relationen und den Seiten des "Stable Memory Managers" durch, d.h. alle Module oberhalb des CRM kennen nur noch Relationen bzw. Tupel, aber keine Seiten mehr.

3.1 KONZEPTION DER SPEICHERUNG

Über den CRM können ausschließlich Daten verwaltet werden, die eine NF^2-Tupel-Struktur besitzen und auf Seiten gespeichert werden. Für Schichten weiter oben dient der CRM daher zur Verwaltung von internen Relationen, Zugriffspfadrelationen, Versionenrelationen sowie Katalogrelationen. Dies hat den Vorteil, daß man gleiche Funktionen für verschiedene Typen von Relationen nur einmal implementieren muß, was zu weniger Moduln führt. Die Gleichbehandlung der Relationen führt dazu, daß der CRM als eine Art Speicherserver dient, der über die Bedeutung und die Beziehungen zwischen den Relationen nichts weiß.

Bei der Behandlung komplexer Tupel gibt es im wesentlichen 3 Möglichkeiten, bis zu welcher Ebene diese bekannt sind: nur auf Benutzerebene, auch auf konzeptueller Ebene, bis zur internen Ebene.

Die erste Möglichkeit erreicht man durch eine Preprocessor-Lösung: Auf ein vorhandenes Datenbanksystem, das nur flache Tupel kennt, wird ein System aufgesetzt, das Operationen auf komplexen Objekten verarbeiten kann und diese auf Operationen für flache Tupel abbildet. Überlegungen und Untersuchungen hierzu werden u.a. mit System R gemacht (/Lo84/).

Bei der zweiten Möglichkeit kennt zwar das System komplexe Objekte, verwendet dann aber weiter unten bei der Abbildung auf Seiten flache Tupel bzw. IMS-artige Segmente. Der Zusammenhang zwischen diesen atomaren Fragmenten und dem gesamten NF^2-Tupel wird dann über ein Directory, das mit einer Linkstruktur vergleichbar ist, hergestellt. Diese Richtung wird z.Zt. im AIM-Projekt verfolgt (/AIM84/, /Lu85/).

Bei der dritten Möglichkeit dagegen, die wir weiter verfolgen wollen, werden NF^2-Tupel direkt ohne die Zwischenstufe der flachen Tupel auf Seiten abgebildet. Dabei sollen ganze NF^2-Tupel möglichst dicht zusammen abgespeichert werden, unter Einhaltung folgender Regeln:

- Wenn ein Tupel kleiner als eine Seite ist, dann soll es auch auf einer Seite abgespeichert sein.
- Wenn ein Tupel mehrere Seiten umfaßt, sind auf diesen Seiten nicht noch andere Tupel , d.h. ein seitenübergreifendes Tupel soll auf möglichst wenigen Seiten stehen.
- Auf einer Seite stehen immer nur Tupel bzw. Tupelteile von **einer** NF^2-Relation.
- Die ersten beiden Regeln sollen rekursiv auch auf Tupel von Subrelationen, d.h. relationenwertigen Attributen, angewendet werden.

Die Adressierung und Speicherung Strukturierter Tupel ist so ausgelegt, daß sowohl komplette Tupel bzw. Subtupel als auch Mengen von Tupeln adressiert werden können. Ein (ganzes) Tupel einer NF^2-Relation belegt im allgemeinen eine Menge von Seiten, die als dynamischer Adreßraum des NF^2-Tupels betrachtet wird. Die erste Seite (Wurzelseite), die u.a. den Header eines Tupels enthält, identifiziert diese Seitenmenge. Der strukturelle Zusammenhang der einzelnen Subtupel soll am Beginn eines NF^2-Tupels bereits ersichtlich sein. Dies hat den Vorteil, daß bei einem Zugriff auf ein Subtupel eine Traversierung vieler Seiten vermieden wird. Die Seitennummern der Seitenmenge werden ebenfalls am Tupelanfang gespeichert. Auf diese Weise kann man häufig mit zwei Aufrufen des SMM auf das komplette Strupel zugreifen. Der erste Aufruf liefert die benötigte Strukturinformation und die Nummern der betroffenen Seiten, die im zweiten Aufruf (via chained I/O) vom mengenorientierten SMM bereitgestellt werden.

Eine detaillierte Darstellung findet sich in /DGW85/. Die Adressierung komplexer Strupel

orientiert sich am TID-Konzept, während für den inneren Zugriff auf Subtupel eine relative Adressierung in Art des Database-Key-Konzepts verwendet wird.

3.2 FUNKTIONEN DES COMPLEX-RECORD-MANAGERS (CRM)

3.2.1 LESEOPERATIONEN DES CRM

Die Leseoperation des CRM muß eines unserer Hauptziele, nämlich schnellen Zugriff (d.h. möglichst wenig Aufrufe des SMM) für vollständige Strupel als auch direkten Zugriff auf Teilbäume bis hin zu den Blattknoten des hierarchischen Tupels verwirklichen. An der Schnittstelle äußert sich dies durch die geschachtelte Projektion als konsequenter Verallgemeinerung der im flachen Relationenmodell angebotenen Projektion zum Auswählen von Attributwerten. Diese π-π-Operation ist charakteristisch für den CRM. Angereichert wird die Schnittstelle durch einfache Selektionsfilter, die auf jeder Ebene des hierarchischen Tupel im Stile von "searchable arguments" des System R mitgegeben werden können. An der Schnittstelle zum SMM wird dann mit der π-π-Spezifikation und den Adressen im Tupel-Header die Menge der Seiten berechnet, auf die zugegriffen werden muß, um die π-π-Operation zu befriedigen.

In der entsprechenden Implementierung in DASDBS sind folgende Operationen vorgesehen:

RETRIEVE
NEXT
CLOSE

Es werden immer Tupelmengen von einer Relation des CRM in den Übergabebereich (s. Abschnitt 3.3) übertragen. Ist genügend Platz für die Ergebnismenge vorhanden, wird der gesamte Lesebebefehl mit einer RETRIEVE-Operation ausgeführt. Ist dies nicht der Fall, wird mit RETRIEVE implizit ein Scan geöffnet. Darauf kann man sich dann in einer anschließenden Folge von NEXT-Operationen beziehen. Selbstverständlich können andere Scans auf diese oder weitere Relationen gleichzeitig geöffnet sein. Mit CLOSE wird ein geöffneter Scan abgebrochen.

Grundlage der RETRIEVE-Operation ist die NF^2-Algebra, wie sie in /SS84/ detailliert beschrieben ist. Das allgemeine Format der RETRIEVE-Operation sieht so aus:

π[<A-Liste>] (σ[<σ-Formel>] (π[<π-Liste>] (Ψ[<Adreß-Relation>] (<Relation>))))

Dabei ist <Relation> der Name einer im Katalog beschriebenen NF^2-Relation. Die Ψ-Operation schränkt die überhaupt in Frage kommenden Strupel (und die dazugehörigen Sub-Tupel) durch die Vorgabe von Adressen ein. Die innere π-Operation dient dazu, um in der Hierarchie des NF^2-Tupels "hinabzusteigen", d.h. in der π-Liste werden Operationen auf Attributen von RV-Attributen der <Relation> (Subrelationen) angegeben. Die Selektionsbedingung - spezifiziert durch die σ-Formel - und die Attributliste der Projektion schränkt das Ergebnis in bekannter Weise ein.

Die Elemente der π-Liste können eines der beiden folgenden Formate haben (E steht jeweils als vorgegebener Name für das Ergebnisschema):

<Attribut> : E
Name eines AV- oder RV-Attributs der Relation, das vollständig übernommen wird.

π[<A-Liste>] (σ[<σ-Formel>] (π[<π-Liste>] (<RV-Attribut>))) : E
Hier gilt das gleiche wie allgemein auf der Relationsebene nur eine Hierarchiestufe

tiefer für die Subrelation ‹RV-Attribut›. Da π-Liste rekursiv wieder eines der beiden Formate haben kann, kann man so in der gesamten NF^2-Relation bis zu RV-Attributen, die nur noch aus AV-Attributen bestehen, herunterprojizieren. Das Format von A-Liste und σ-Formel ist das gleiche wie in der gesamten RETRIEVE-Operation und wird unten erklärt. Jedes der drei Elemente A-Liste, σ-Formel und π-Liste kann weggelassen werden. Sind die Selektionsformel und die Attributliste nicht angegeben, handelt es sich um die einleitend erwähnte π-π-Operation.

Als Beispiel hierfür liefert der π-π-Ausdruck

Q: =π[ABTNAME, π[NAME, KIND](PERS) : PK](ABT)

der sich auf das Schema von Abschnitt 2.2 bezieht, den Abteilungsnamen von jeder Abteilung sowie von deren technischen Angestellten den Namen und alle Kinder. Das Ergebnisschema ist Q(ABTNAME, PK(NAME, KIND(NAME, GEBDAT))).

Ist die π-Liste nicht spezifiziert, so können weder in der A-Liste noch in der σ-Formel Attribute von Subrelationen erscheinen, d.h. diese können nur komplett oder überhaupt nicht übernommen werden.

Die σ-Formel entspricht der üblichen Selektionsformel der flachen Relationenalgebra. Für die Selektionsatome der Form 'A op B' sind zwei Fälle zu unterscheiden:

A ist AV-Attribut oder ein Ergebnisschema mit atomarem Domain. Dann gilt:
op $\in \{=, \neq, <, \leq, >, \geq\}$ und
B ist AV-Attribut oder eine Konstante $\in$ domain(A)

A ist RV-Attribut oder ein Ergebnisschema mit relationenwertigem Domain. Dann gilt:
op $\in \{=, \neq\}$ und
$B = \phi$.

Es sind also der übliche Vergleich von atomaren Attributen mit Konstanten sowie bei Subrelationen der Vergleich mit der leeren Menge zur Realisierung eines Existenzquantors möglich. Allgemeine Vergleiche zwischen Subrelationen und nichtleeren Mengen können hier nicht zugelassen werden, da sie keine Single-Pass-Ausdrücke darstellen. Ist die Selektionsliste nicht spezifiziert, so werden von allen ausgewählten Tupeln die angegebenen Attribute ohne weitere Selektion übertragen.

Für die Elemente der A-Liste gibt es zwei Möglichkeiten: zum einen können es Attribute der Relation bzw. im π-π-Fall des RV-Attributs sein. Andererseits sind Ergebnisschemata, die in der nachfolgenden π-Liste definiert werden, zugelassen. Ist die A-Liste nicht spezifiziert, so werden alle Attribute aller Hierarchiestufen der ausgewählten und selektierten Tupel übernommen.

Die Adreßrelation wird angegeben, um damit direkt bestimmte Tupel und Subtupel zu adressieren. Mit dem Ψ-Operator wird also eine Adreßselektion durchgeführt, was in /SS83/ auch als "TID-Join" bezeichnet wird. Die Adreßrelation kann z.B. aus einem Zugriffspfad gewonnen werden und hat einen hierarchischen Aufbau analog zur Schachtelung der spezifizierten Relation. Der CRM-Aufruf

Ψ[{‹14, 2, {‹1, 1›, ‹1, 3›}›}](ABT)

liefert beispielsweise den ersten und dritten technischen Angestellten inklusive den Lehrgängen und Kindern der Abteilung mit dem TID ‹14, 2›.

Ist die Adreßrelation bei der RETRIEVE-Operation nicht spezifiziert, so wird ein sequentieller Scan über die gesamte Relation durchgeführt.

Ein weiteres Beispiel zeigt die Mächtigkeit der RETRIEVE-Operation:

```
π[ANR,
   π[NAME,π[LNR](σ[JAHR ≥81](LEHRGANG)),KIND]
        (σ[TBEZ='Programmierung'](PERS)),
   σ[TBEZ='Sekretariat'](VERW)]
                (σ[ABTNAME='Forschung'] (ABT))
```

Über die Forschungsabteilung soll neben der Abteilungsnummer folgende Information gewonnen werden: Von den technischen Angestellten, die programmieren, interessieren deren Name, die Lehrgangsnummer ihrer seit 1981 besuchten Lehrgänge sowie Name und Geburtsdatum aller ihrer Kinder. Außerdem sollen die Personalnummer, die Tätigkeitsbezeichnung und der Name aller Verwaltungsangestellten im Sekretariat dieser Abteilung übergeben werden.

Gegenüber der allgemeinen NF^2-Projektion in /SS84/ sind hier deutliche Beschränkungen auferlegt: Nestung, Entnestung, Vereinigung, Differenz und Kartesisches Produkt sowie Hintereinanderschalten sind hier nicht erlaubt; diese sollen in weiter oben liegenden Schichten mittels der o.g. Bausteine implementiert werden. Verglichen mit konventionellen Dateiverwaltungssystemen und im Hinblick auf das intendierte Subsystem eines Datenbankkerns erscheint die zur Verfügung gestellte Mächtigkeit angemessen. Genauere Überlegungen zu den Beschränkungen befinden sich in /Sch85/.

3.2.2 AENDERUNGSOPERATIONEN DES CRM

Der CRM bietet die Änderungsoperationen INSERT, DELETE und UPDATE an. Bei der INSERT-Operation wird eine Menge von NF^2-Tupeln mit allen ihren Subrelationen, die im Übergabebereich bereitgestellt wird, in die angegebene Relation eingefügt und die Adreß-Relation für die neuen Tupel zurückgegeben. Mit der DELETE-Operation werden alle NF^2-Tupel komplett gelöscht, die über eine Adreß-Relation adressiert werden und sich über eine Selektionsformel qualifiziert haben. Die hier notwendigen Operationen ψ und σ entsprechen denen der RETRIEVE-Operation. Diese beiden Operationen INSERT und DELETE beziehen sich nur auf Mengen von kompletten NF^2-Tupeln. Wenn man jedoch an bestehenden Tupeln etwas ändern will, z.B. ein Subtupel hinzufügen, muß man die UPDATE-Operation benutzen. Dabei werden wieder über eine Adreß-Relation und eine Selektions-Formel die NF^2-Tupel ausgewählt, bei denen etwas geändert werden soll. Eine Änderungsliste beschreibt, welche Attribute (einschließlich den relationenwertigen) wie geändert werden sollen. Es gibt dabei die Möglichkeit, eine Menge von Subtupeln einer Subrelation hinzuzufügen bzw. zu löschen oder atomare Attribute durch eine Konstante zu überschreiben. Diese Änderungsoperationen kann man rekursiv auf allen Hierarchieebenen anwenden.

Es gibt also zwei Ziele für die Änderungsoperationen des CRM: Zum einen soll man durch eine geschachtelte UPDATE-Operation analog zum π-π-Konstrukt der RETRIEVE-Operation schnell und gezielt einzelne Teile eines Tupels direkt ändern können. Zum anderen soll es aber durch eine Mengenorientierung auch möglich sein, mit einem Aufruf sehr viele Tupel bzw. Subtupel einzuspeichern, zu löschen oder zu ändern. Detaillierte Beschreibungen sowie Beispiele der Änderungsoperationen sind in /DVSI85/ angegeben.

3.3 UEBERGABEBEREICHE FUER STRUKTURIERTE TUPEL

Eine wichtige Zielsetzung unseres Systems ist die konsequente Mengenorientierung. Es sollen nicht nur Mengen von Seiten vom externen Speicher auf einmal in den Hauptspeicher gebracht werden können, sondern auch zwischen den einzelnen Moduln sollen Mengen von Strupeln übergeben werden. Dies erscheint besonders wichtig in einer (verteilten) Server-Workstation-Umgebung, wo zu den Kosten für eine Schnittstellenüberquerung noch die Kommunikationskosten hinzukommen.

Herkömmliche DBMS bieten i.a. eine "One-tuple-at-a-time"-Schnittstelle an. Bei CODASYL-DBMS ist dies schon durch das Datenmodell festgelegt, aber auch relationale Systeme wie SQL/DS erlauben nur, einzelne Tupel zu übergeben. Ergebnismengen werden durch einen vom Benutzer verwalteten Scan ("open cursor", "fetch",...,"close cursor") abgearbeitet, wobei immer nur ein Tupel die Schnittstelle passiert.

Ein neuerer Ansatz sind die "Database Portals" (/SR84/), die es erlauben, ganze Relationen oder Ausschnitte daraus auf einmal zur Verfügung zu stellen und zu manipulieren. Das Portal ist quasi ein Puffer mit darauf definierten Funktionen wie Vor- und Zurückblättern, Ändern, Einfügen usw.. Die in /SR84/ vorgeschlagene Realisierung als array [1..k] of record reicht für unsere Anforderungen nicht aus.

Die CRM-Schnittstelle ist nämlich doppelt mengenorientiert: Es werden nicht nur Mengen von Strupeln übergeben, sondern auch einzelne Strupel können, falls sie relationenwertige Attribute haben, eine Menge von Subtupeln enthalten. Es ist demnach nicht möglich, den Übergabebereich exakt der Struktur des Ergebnisses entsprechend im voraus anzulegen. Es bietet sich daher an, die Tupel als Bytestrings darzustellen und Interpretationsinformationen mitzugeben. Eine mögliche Kodierung, die sich an den Speicherstrukturen des CRM orientiert, wird in /HW84/ beschrieben.

Das Wissen über diese Kodierung kann man in einem Interpretationsmodul verbergen, der den Übergabebereich nach oben zum abstrakten Datentyp (ADT) werden läßt. Objekte dieses ADT sind Strupel (oder Subtupel) und darauf sind vor allem Retrievalfunktionen definiert, z.B. "gebe Strukturbeschreibung für nächste(s) Strupel","lese das (die) nächste(n) Strupel (oder Subtupel)", "bestimme Kardinalität des Ergebnisses", "extrahiere bestimmte(s) Subtupel".

Zur Realisation dieses "logischen" Übergabebereichs muß man auf die realen Übergabebereiche (d.h. physische Puffer) zurückgreifen. Da Ergebnismengen oder gar einzelne Tupel so groß werden können, daß sie nicht auf einmal in den größten verfügbaren Bereich passen, ist es nötig, die Ergebnismenge zu portionieren und die Größe der Übergabebereiche mit dieser Portionierung abzustimmen. Zur Extraktion dieser Portionen benutzt man die CRM-Funktionen RETRIEVE, NEXT, CLOSE (siehe Abschnitt 3.2). Die erste Portion wird sofort als Ergebnis von RETRIEVE zurückgegeben, die weiteren Pakete können durch Aufrufen von NEXT erhalten werden. Ist die letzte Portion verarbeitet oder benötigt der Verbraucher das Restergebnis nicht mehr (z.B. weil bei einer "exists"-Anfrage schon ein Treffer gefunden wurde), so ruft er CLOSE auf. Der Erzeuger kann nun den Übergabebereich freigeben und sämtliche damit verbundene Verwaltungsinformation (Scan-Block) wegwerfen.

Bei der Frage der Portionierung wird in DASDBS versucht, soweit als möglich, an den Grenzen logischer Einheiten (Strupel, Subtupel) zu zerteilen; wenn aber ein bestimmter Füllgrad unterschritten wird, spaltet man. Atomare Attribute werden nur im Extremfall gespalten, wenn sie alleine nicht mehr in eine Bereichsfüllung passen.

3.4 DER KATALOG

Für die hier beschriebene Speicherverwaltungskomponente dient der Katalog der Speicherung derjenigen Metadaten, die von Versionenmanager und Complex Record Manager (CRM) benötigt werden. Dazu zählen vor allem die Beschreibung des Schemas der gespeicherten Relationen, die gewählten Speicherungsstrukturen , Angaben über die Versionierung der Relationen sowie im Hinblick auf ein Gesamtdatenbanksystem die Erfassung von Performanceparametern, etwa Zugriffsstatistiken für das Tuning durch Änderungen des physischen Datenbankentwurfs.

Schon bei (flachen) relationalen Datenbanksystemen werden auch die Metadaten in Form von Katalogrelationen geführt, um redundante Implementierungen von Funktionen zur Manipulation von Benutzer- und Systemdaten zu vermeiden und dynamische Änderungen der Katalogeinträge zu unterstützen. Gemäß unserer Maxime, ein "schlankes" System möglichst ohne Spezialkonstrukte zu implementieren, wollen wir folglich einen Katalog in Form von NF^2-Relationen implementieren, die - wie alle anderen Relationen - vom CRM verwaltet und mittels NF^2-Algebra manipulierbar werden (siehe hierzu das später folgende Beispiel).

Um endlose Rekursionen durch wechselseitiges Aufrufen von Katalogverwalter und CRM beim Lesen von Kataloginformationen zu vermeiden, muß das Schema mindestens einer Katalogrelation fest im Programmcode verankert werden, andernfalls wäre folgendes Szenario denkbar: Wenn der CRM einen Auftrag ausführen soll, so holt er sich dazu aus dem Katalog die Beschreibung der zugrundeliegenden Relation. Der Katalog ist nun aber ebenfalls als Relation in der Datenbank gespeichert, d.h. zunächst ergeht vom Katalogverwalter ein Leseauftrag an den CRM zurück. Kennt nun der CRM nicht die Struktur dieser "Urstrupel" des Katalogs, so würde er wiederum im Katalog nachsehen, usw.

Im folgenden werden drei Varianten für die feste Struktur der zentralen Katalogrelation diskutiert, dabei kommt es uns weniger auf eine vollständige Auflistung der Attribute im Katalog an, als vielmehr auf die Darstellung der grundsätzlich verschiedenen Strukturierungsmöglichkeiten. Jede dieser Varianten erlaubt die Abspeicherung der Schemainformation aller weiteren vom DBS verwalteten Relationen:

```
1. CATALOG1 ( RELNAME, ATOMIC-ATT ( ANAME, SEQNO, DOMAIN ),
                       RV-ATT ( RVANAME, SEQNO, LEVELNO ) )

2. CATALOG2 ( RELNAME, ATOMIC-ATT ( ANAME, SEQNO, DOMAIN ),
                       RV-ATT ( RVANAME, FATHERNAME, SEQNO, LEVELNO,
                                ATOMIC-ATT ( ANAME, SEQNO, DOMAIN ) ) )

3. CATALOG3 ( RELNAME, ATOMIC-ATT0( ANAME0, SEQNO0, DOMAIN0 ),
                       RV-ATT0 ( RVANAME0, ATOMIC-ATT1 (...),
                                           RV-ATT1 ( ...,
                                                     RV-ATT2 ( ... ))))
```

Ein Kontrollstrupel dieser Relation enthält jeweils bei Variante 1: die Beschreibungen einer (Sub-) Relation einer bestimmten Schachtelungstiefe mit atomaren Attributen (ATOMIC-ATT) sowie "Verweise" (RVANAME) auf die Beschreibungen ihrer relationenwertigen Attribute. Jede Relation und jede Subrelation (aller Schachtelungstiefen) der Datenbank ist in CATALOG1 durch ein Strupel beschrieben (vgl. /AIM84/). Bei Variante 2 gibt es nur ein Strupel je (äußerer) Relation in CATALOG2, dieses enthält jeweils alle atomaren Attribute (ATOMIC-ATT) der ersten Ebene, sowie alle

relationenwertigen Attribute, jeweils mit ihren atomaren Unterattributen, aller Schachtelungstiefen (RV-ATT). Die Struktur des Schemas wird durch Verweise (FATHERNAME) innerhalb der RV-Attribute dargestellt. Auch in der letzten Variante ist eine Relation durch genau ein Strupel aus CATALOG3 beschrieben. Dieses enthält, in gleicher Weise geschachtelt wie die beschriebene Relation, zu jeder Schachtelungsebene die atomaren Attribute und die RV-Attribute, jeweils als eine Subrelation.

Ein Nachteil dieser letzten Strukturierung ist offensichtlich: die maximale Schachtelungstiefe ist durch die vorgegebene Struktur des Katalogs begrenzt. Gemeinsam mit der Variante 2 hat sie gegenüber Variante 1 den Vorteil, daß die Beschreibung einer Relation hier kompakt in einem Strupel enthalten ist, also auch mit einem einzigen CRM-Aufruf selektiert werden kann, was speziell in Server-Workstation-Umgebungen - je nach funktionaler Verteilung - an Bedeutung noch gewinnt. Die ersten beiden Varianten haben gegenüber der dritten jedoch den großen Nachteil, daß zur Rekonstruktion der Beschreibung einer (Sub-) Relation die Bildung der transitiven Hülle über die Abarbeitung von Verweisketten erforderlich ist, eine Operation, die nicht mittels NF^2-Algebra beschreibbar, also auch nicht vom CRM durchzuführen ist. Daher erscheint uns die dritte Möglichkeit trotz ihrer Einschränkung - bei geeigneter Wahl der maximalen Schachtelungstiefe ist sie praktisch irrelevant - am geeignetsten, zumal sie sehr einfach erlaubt, aus der Beschreibung einer Relation und einem spezifizierten π-π-Ausdruck die Beschreibung der Ergebnisstrupel abzuleiten. Diese Operation wird sicher bei der Übergabe von Zwischenergebnissen die Bearbeitung beschleunigen können. So kann etwa zu dem CRM-Aufruf

π[ABTNAME, π[NAME](PERS)](ABT) mittels

π[σ[ANAME0='ABTNAME'](ATOMIC-ATT0), π[σ[ANAME1='NAME'](ATOMIC-ATT1)]
(σ[RVANAME0='PERS'](RV-ATT0))](σ[RELNAME='ABT'](CATALOG3))

die Beschreibung der Ergebnisstrupel aus der Beschreibung des ABT-Strupels berechnet werden.

Zur Unterstützung des Zugriffs auf die Relation CATALOG wird eine Bootstrapping-Tabelle mit den Adressen des CATALOG-Strupels jeder Relation angelegt, um sequentielle Scans zu vermeiden. Diese Tabelle steht an einer festen Adresse in einem Systemsegment.

Neben der Katalogrelation gibt es keine weiteren "festverdrahteten" Relationen. Weitere Systemrelationen werden wie Benutzerrelationen im Katalog beschrieben. Zur Freispeicherverwaltung werden etwa die folgenden Relationen angelegt:

FPA (SEGMENTNO, PAGESIZE, FREEPAGES (PAGENO))

RELFPA (RELID, FREESPACE (PAGENO, FREEBYTES))

FPA enhält zu jedem Segment die Menge der (ganz) leeren Seiten, RELFPA gibt zu jeder Relation den freien Platz (unscharf!) auf den Wurzelseiten; nur in diesen kann ja evtl. noch ein weiteres Tupel eingefügt werden. Die zweite Relation wird natürlich auch für Relationenscans benutzt, so daß nie ein Segmentscan erforderlich wird. Die Freispeicherinformation für die einzelnen Seiten befindet sich jeweils im Page-Header.

4 KOMPLEXE OBJEKTE: VERSIONEN STUKTURIERTER TUPEL

4.1 OPERATIONEN AUF UND MIT HILFE VON VERSIONEN

Der Versionen-Manager unseres Kernsystems soll kein allgemeines "Zeit-Verständnis" etwa im Sinne von /KILo83/ haben. Er kennt eine einzige, linear geordnete Zeit bis zur jeweiligen Gegenwart. Die Operationen des Versionen-Managers bilden keine temporale Sprachschnittstelle (/KI83/, /MS83/), sondern stellen lediglich Grundfunktionen im Umgang mit (Zeit-) Versionen von Strupeln dar.

Eine Alternative zu Strukturierten Tupeln als Granulat wäre es, Versionen jeweils des atomaren Teils eines Subtupels zu bilden (/AIM84/). Prinzipiell ist dieser Ansatz genauso mächtig wie der von uns verfolgte, jedoch ist dabei ein größerer Aufwand zu erwarten, wenn oft auf Versionen eines gesamten NF^2-Tupels zugegriffen wird.

Wir nehmen an, daß jedes Objekt des Versionen-Managers, im folgenden "Komplexes Objekt" genannt, einen für seine Lebensdauer stabilen, eindeutigen "Complex Object Identifier" (Abk.: COID) hat. Der CRM als Basisimplementierungsschicht des Versionen-Managers kennt nur TID's als Identifizierungskriterium. Aufgabe des Versionen-Managers ist es folglich, ein Komplexes Objekt auf Strupel des CRM abzubilden. Wenn **COID** die Menge aller COID's, **TS** die Menge aller Zeitmarken und **TID** die Menge aller TID's bezeichnet, so hat er die Funktion

$$\tau\colon \mathbf{COID} \times \mathbf{TS} \rightarrow \mathbf{TID}$$

zu implementieren. Diese Funktion τ kann sowohl als Mechanismus verstanden werden, Versionen abzuspeichern, als auch als Retrievaloperation. Sie ist partiell, d.h. außerhalb der "Lebensspanne" eines Komplexen Objekts undefiniert. Zwischen zwei Objektänderungen interpoliert τ durch Annahme eines konstanten Objektzustandes im dazwischenliegenden Zeitraum (vgl. /KILo83/).

Alle Änderungsoperationen sind mit denen des CRM identisch, jedoch wird als zusätzlicher Parameter jeweils eine Zeitmarke verlangt, die mit den erzeugten Versionen bzw. einem Löscheintrag verknüpft wird. Beim Einfügen eines neuen Objekts gibt der Versionen-Manager einen COID zurück. Die Retrievalfunktion des Versionen-Managers ist die RETRIEVE-Operation des CRM, angereichert durch eine Zeitangabe, d.h. entweder einen Zeitpunkt oder ein Zeitintervall. Im Falle einer Zeitpunktangabe wird der Zustand der ausgewählten Objekte gemäß der o.a. Interpolationsvorschrift ermittelt. Bei Angabe eines Intervalls sind diejenigen Objektversionen Treffer, die innerhalb der Intervallgrenzen gültige Zustände waren und die Selektionsbedingung erfüllt haben.

4.2 SPEICHERUNG VON VERSIONEN

Oberste Maxime für den Versionen-Manager ist es, keinen Overhead für den Zugriff auf aktuelle Versionen zu verursachen. Insbesondere muß eine vorgesehene Clusterung aktuell gültiger Strupel zu jedem Zeitpunkt erhalten bleiben. Diese Forderung führt zwangsläufig dazu, aktuelle und alte Versionen separat zu speichern (/KaLe84/). Overhead erhält man allerdings trotzdem, wenn der Zusammenhang zwischen den verschiedenen Versionen eines Objekts über einen expliziten COID hergestellt wird und der Zugriff auf die aktuelle Version über eine COID-TID-Umsetzungstabelle erfolgt (z.B. /KaLe84/). Vermeiden können wir diese Indirektion, indem wir den COID eines Objekts als TID seiner aktuellen Versionen definieren; d.h. wir vereinbaren:

$$\tau(\text{Coid}, \text{NOW}) = \text{Tid} \Rightarrow \text{Coid} = \text{Tid} \quad (\text{Coid} \in \textbf{COID}, \text{Tid} \in \textbf{TID})$$

Möglich ist diese Festsetzung, da wir bislang keinerlei Annahmen bezüglich der Natur eines COID's getroffen haben!

Eine weitere Forderung, die sich mit dem Konzept impliziter COID's leicht verbinden läßt, ist, daß COID's nach dem "logischen" Löschen eines Objekts wiederverwendbar sein sollen, d.h. daß insbesondere der Speicherplatz des entsprechenden TID's wieder verfügbar wird (vgl. /Lu84/). Wir schließen damit "Reinkarnationen" eines Objekts aus (z.B. Angestellte, die nach einer Kündigung später wieder in die Firma eintreten). Diese sind sauberer auf einer semantischen Ebene, z.B. über gleiche Primärschlüssel, handzuhaben.

Die zweite Maxime für den Versionen-Manager in DASDBS ist Flexibilität bezüglich des Tradeoffs zwischen Speicherplatz und Zugriffszeit. Das naheliegendste Speicherungsschema für Versionen eines Objekts ist eine zeitlich rückwärts verkettete Liste mit der aktuellen Version als Kettenanker (/Re83/). Diese Struktur hat die wünschenswerte Eigenschaft, daß der Zugriff auf eine Objektversion umso länger dauert, je älter diese ist. Das Schema ist auch vom Speicherplatz her recht günstig, da außer der aktuellen Version alle Versionen komprimiert in Form sogenannter Rückwärtsdifferenzen (UNDO-Information für die entsprechende Änderungsoperation) gespeichert werden können (/DLW84/, /Wei83/).

Ein alternatives Speicherungsschema, das gleichschnellen Zugriff auf alle alten Versionen unterstützt, basiert auf der Struktur eines Pointerarrays mit zeitlich geordneten Verweisen auf die einzelnen Versionen. Die Pointerarrays der verschiedenen COID's sind dabei in Form eines B*-Baums organisiert (/KaLe84/). Diese Speicherorganisation hat allerdings den Nachteil, daß sie sich nicht mit der angesprochenen Differenzentechnik kombinieren läßt.

Es liegt darum nahe, die beiden skizzierten Verfahren zu verbinden, um die angestrebte Flexibilität bei der Speicherverwaltung von Versionen zu erreichen. Aus den im folgenden "HIST-INDEX" genannten Pointerarrays wird ausschließlich auf vollständig gespeicherte Versionen verwiesen, während in "Deltaform" vorliegende Versionen untereinander rückwärts verkettet sind. Einträge im HIST-INDEX, die sich auf solche Versionen beziehen, verweisen auf den jeweiligen Kettenanker. Wenn COID's wiederverwendet wurden, so enthält der HIST-INDEX natürlich entsprechende Löschvermerke. Die gesamte Speicherungsstruktur eines Objekts ist im folgenden Bild exemplarisch dargestellt.

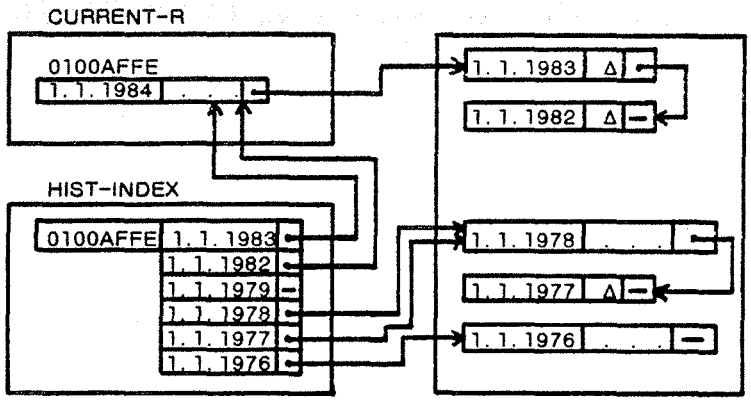

Abb. 3: Speicherungsschema für Versionen

Die Entscheidung, ob eine Version vollständig oder in Form einer Differenz gespeichert wird, wollen wir vor allem abhängig machen von der Größe einer vollständigen Version im Vergleich zur Größe des entsprechenden Deltas. Weitere Einzelheiten zur gewählten Speicherungsform von Versionen sind /DVSI85/ zu entnehmen.

5 GESCHACHTELTE TRANSAKTIONEN IN EINER SCHICHTENARCHITEKTUR

DASDBS bietet an seiner Schnittstelle die Transaktionsklammern BOT und EOT an, mit denen Operationsfolgen kenntlich gemacht werden, die unteilbar, dauerhaft und isoliert von anderen Transaktionen ablaufen sollen. Die übliche Implementierung der Transaktionsverwaltung findet in einer niedrigen Systemschicht statt. Die Vorteile, die man sich davon verspricht, sind zum einen potentielle Performancegewinne bedingt durch bessere Anpassung an die I/O-Struktur eines Systems und zum anderen natürlich die vereinfachte Programmierung höherer Schichten.

Es liegt daher nahe, Recoveryfunktionen und - sofern auch eine Mehrbenutzerumgebung unterstützt werden soll - Concurrency Control bereits im Kernsystem zu realisieren und Anwendungstransaktionen auf diese Basisimplementierung in geeigneter Weise abzubilden. Derartige Basis-Transaktionskonzepte wurden in jüngster Zeit sowohl für Programmiersprachen als auch für Betriebssysteme (z. B. /SSp84/, /Tr83/) vorgeschlagen. Der entscheidende Nachteil aber ist, daß Betriebssysteme oder allgemeiner Systemkerne notgedrungen einen semantisch ärmeren Objekt- und Operationsbegriff haben als anwendungsnahe Schichten. Konkret bedeutet dies in der Regel Seiten als Granulat für Concurrency Control und Recovery. Seitenorientierte Sperrverfahren wiederum führen zumindest bei klassischen Datenbankanwendungen zu vergleichsweise hohen Konfliktwahrscheinlichkeiten und damit zu einer potentiellen Einschränkung der Parallelität. Dieselben Überlegungen gelten auch für Optimistische Verfahren.

Wenn aber schon mit relativ vielen Behinderungen gerechnet werden muß, so besteht die einzige Chance zur Erhöhung der Parallelität in der Verkürzung der Sperrzeiten. Es gibt eine Reihe von Vorschlägen, in Verbindung mit speziellen Zugriffspfadstrukturen vom Zweiphasenprotokoll abzuweichen und bestimmte Sperren frühzeitig freizugeben. Wenn man indes die jeweiligen Freigabezeitpunkte als "Commitment" bestimmter Verarbeitungsphasen

interpretiert, so erhält man einen fließenden Übergang zu einem anderen Ansatz, den wir als allgemeineren Rahmen ansehen: den der geschachtelten Transaktionen.

Eine einfache Variante dieses Mechanismus findet sich in System R, bei dem Seitensperren nur für die Dauer jeweils einer RSS-Aktion gehalten werden (/As76/). RSS-Aktionen sind also gewissermaßen Subtransaktionen eines Anwendungsprogramms. Wegen der Freigabe von Seitensperren vor EOT der eigentlichen Benutzertransaktion wird dieser Ansatz als "offene geschachtelte Transaktion" bezeichnet (/Tr83/). Tupelsperren werden bis EOT gehalten, so daß Serialisierbarkeit gewährleistet ist.

Die Frage stellt sich, wie unser Kernsystem aus derartigen Überlegungen Nutzen ziehen kann. Bisherige Studien zum Performancevergleich verschiedener Concurrency Control-Verfahren (z.B. /PR83/) zeigen, daß die Güte eines Verfahrens stark von den Lastcharakteristika, d.h. vom Referenzverhalten eines Systems abhängt. Da Referenzmuster Funktionen der jeweils betrachteten Objekte und Operationen, d.h. schichtenspezifisch sind, ist die Qualität einer Concurrency Control-Methode folglich eng mit der Systemebene verknüpt, auf der sie angewendet werden soll.

Die Konsequenz daraus ist, daß jede Ebene einer Schichtenarchitektur ihre eigene Concurrency Control durchführen sollte. Um Anomalien wie "Lost Update" oder "Domino-Effekt" zu verhindern, hat jede Systemschicht auch selbst Recovery durchzuführen - eine Idee, die bereits in /Ve79/ propagiert wurde. Im Gesamtbild erhalten wir damit pro Schicht eine Transaktionsverwaltungskomponente, deren Transaktionen jeweils Subtransaktionen der nächsthöheren Schicht sind. Diese Architektur gestattet den Austausch der jeweils implementierten Concurrency Control- und Recoveryalgorithmen, was im Hinblick auf die Anpaßbarkeit unseres Kernsystems wichtig erscheint.

In dem hier beschriebenen Subsystem von DASDBS gibt es zwei wohlunterscheidbare Objektebenen, die der Seiten und die der Strukturierten Tupel bzw. deren Versionen. Wir wollen deshalb bereits innerhalb dieses Kerns den Mechanismus der offenen geschachtelten Transaktionen anwenden. Mittels eines durch den "Stable Memory Manager" (SMM) implementierten Transaktionskonzepts auf der Seitenebene werden einzelne Operationen auf Strukturierten Tupeln "stabil" gemacht. Mehrere CRM-Operationen aber sollen ebenfalls transaktionsartig geklammert werden können. Dies kann nun einerseits klassisch durch Benutzung des SMM, d.h. mit seitenorientierter Recovery und Concurrency Control oder andererseits mit der Einführung einer CRM-spezifischen Transaktionsverwaltung zweistufig realisiert werden. Da der Versionen-Manager bereits eine Art Logdatei führt, ist es sinnvoll, diese zweite Transaktionsebene oberhalb davon anzusiedeln. Aufgrund der Charakterisierung von Strukturierten Tupeln als interne Objektrepräsentationen eines Datenbanksystems (siehe Kapitel 2) hat schließlich auch das über diesem Subsystem geplante NAS eine eigene Objektsicht mit spezifischen Operationen und konsequenterweise auch eine eigene Transaktionsverwaltung. Wir wollen dafür die Tauglichkeit sogenannter Signatursperren (/DPS83/) in künftigen Arbeiten untersuchen.

Die Vorteile der mehrstufigen Transaktionsverwaltung sind näher in /WS84/ erläutert; weitere Entwurfsüberlegungen zur Implementierung in DASDBS sind in /Wei84/ und /DVSI85/ zu finden.

6 AUSBLICK

Bei der Beschreibung eines Kernsystems, in unserem Fall sogar des Subsystems eines solchen, kommt naturgemäß die Diskussion der Anwendungen etwas kurz. Die Diskussion einer Schicht, die zwischen Anwenderebene und Kernsystemebene eingeschoben werden muß, wurde hier bewußt ausgeklammert.

Eine solche Schicht scheint im Bereich der konventionellen Datenbankanwendungen realisierbar. Abbildungen vom klassischen Relationenmodell, das für konventionelle Anwendungen konzipiert und erprobt ist, auf unser Subsystem werden durch die NF^2-Relationenschnittstelle direkt unterstützt. Die flexiblere Gestaltung von internen Datenstrukturen als NF^2-Relationen müßte zu höherer Effizienz für konventionelle Anwendungen führen. Dies wäre bereits ein Fortschritt.

Inwieweit auch die Nicht-Standard-Anwendungen von unserem Kernsystem profitieren können, müssen weitere Untersuchungen zeigen. Es muß ja immerhin nachgewiesen werden, daß die Verwendung eines Datenbankkernsystems einfach und effizient ist. Gemessen werden wir anhand vieler Spezialentwicklungen aus dem Nicht-Standardbereich. Es ist zu hoffen, daß die Vorteile bei der Verwendung eines (Kern-) Datenbanksystems (einheitliche Datenstrukturen und Datenbeschreibung, Transaktionskonzept, Stabiler Speicher) größer sind als die befürchteten Nachteile, nämlich daß ein allgemein verwendbares System zwangsläufig umständlicher in der Handhabung und unbefriedigend im Laufzeitverhalten sein müsse.

Die zur Klärung dieser Fragen notwendigen Pilotimplementierungen und -Erprobungen scheinen spannend zu werden !

Danksagung

Zur Architektur eines zukünftigen Datenbanksystems haben viele Diskussionen mit Herrn V. Lum und seiner Gruppe stattgefunden, wofür wir uns herzlich bedanken möchten. Unser Vorhaben wird auch in seinem für "Datenverwaltung in Arbeitsplatzrechnern" relevanten Teil durch einen Forschungsauftrag des Wissenschaftlichen Zentrums der IBM Heidelberg gefördert. Viele Anregungen, speziell auch zur "Zeit in Datenbanken", erhielten wir durch Herrn T. Härder, dem wir ebenfalls herzlich danken möchten.

7 LITERATUR

/AIM84/ AIM-Projektdarstellung, IBM Wissenschaftliches Zentrum Heidelberg, Februar 1984

/As76/ M. M. Astrahan et al., System R: Relational Approach to Database Management, TODS Vol. 1 No. 2, 1976

/DGW85/ U. Deppisch, J. Günauer, G. Walch, Speicherungsstrukturen und Adressierungstechniken für komplexe Objekte des NF^2-Relationenmodells, GI-Fachtagung Datenbanksysteme für Büro, Technik und Wissenschaft, Karlsruhe 1985, Springer Verlag 1985

/DKML84/ K. Dittrich, A. Kotz, J. Müller, P. Lockemann, Datenbankkonzepte für Ingenieuranwendungen: eine Übersicht über den Stand der Entwicklung, in: H.-D. Ehrich (Hrsg.), Proc. GI-14. Jahrestagung Braunschweig 1984, IFB 88, Springer Verlag 1984

/DLW84/ P. Dadam, V. Lum, H.-D. Werner, Integration of Time Versions into a Relational Database System, VLDB 1984

/DPS83/ P. Dadam, P. Pistor, H.-J. Schek, A Predicate Oriented Locking Approach for Integrated Information Systems, Proc. of the IFIP Conference, Paris 1983

/DVSI85/ U. Deppisch, V. Obermeit, H.-B. Paul, H.-J. Schek, M. Scholl, G. Weikum, The Storage Component of a Data Base Kernel System, Technical Report DVSI-1985-T1, FB Informatik, TH Darmstadt 1985

/Eb84/ W. Eberlein, Architektur technischer Datenbanken für Integrierte Ingenieursysteme, Dissertation, Arbeitsberichte des IMMD, Band 17, Nr. 1, Universität Erlangen, 1984

/GP83/ L. Gründig, P. Pistor, Flächenbezogene Informationssysteme und ihre Anforderungen an Sprachschnittstellen, in: J. W. Schmidt (Hrsg.), Sprachen für Datenbanken, IFB 72, Springer Verlag 1983

/Gü83/ K. D. Günther, Database Requirements of Computer-Aided Office Procedures, GMD-Arbeitspapier Nr. 54, 1983

/HL82/ R. L. Haskin, R. Lorie, On Extending the Functions of a Relational Database System, ACM SIGMOD Conf., 1982

/HR83/ T. Härder, A. Reuter, Database Systems for Non-Standard Applications, Proc. Int. Computing Symposium, 1983

/HW84/ D. Horn, A. Wolf, Strukturen zur Darstellung von NF^2-Tupeln in internen Übergabepuffern von Datenbanksystemen, TH Darmstadt, Arbeitsbericht DVSI-1984-A11, 1984

/Ja84/ G. Jaeschke, Recursive Algebra for Relations with Relation Valued Attributes, Technical Report 84.01.003, IBM Heidelberg, 1984

/JS82/ G. Jaeschke, H.-J. Schek, Remarks on the Algebra of Non First Normal Form Relations, ACM Symposium on Principles of Database Systems 1982

/KaLe84/ R. H. Katz, T. J. Lehman, Database Support for Versions and Alternatives of Large Design Files, IEEE Transactions on Software Engineering Vol. SE-10 No. 2, 1984

/Ki83/ H. Kinzinger, Erweiterungen einer Datenbank-Anfragesprache zur Unterstützung des Versionenkonzepts, in: J. W. Schmidt (Hrsg.), Sprachen für Datenbanken, IFB 72, Springer-Verlag 1983

/KILo83/ M. R. Klopprogge, P. C. Lockemann, Modelling Information Preserving Databases: Consequences of the Concept of Time, VLDB 1983

/Lo84/ R. Lorie et al., User Interface and Access Techniques for Engineering Databases, IBM Research Report RJ 4155, San Jose, 1984

/LS83/ W. Lamersdorf, J. W. Schmidt, Rekursive Datenmodelle, in: J. W. Schmidt (Hrsg.), Sprachen für Datenbanken, IFB 72, Springer Verlag 1983

/Lu84/ V. Lum et al., Designing DBMS Support for the Temporal Dimension, SIGMOD Conference 1984

/Lu85/ V. Lum et al., Design of an Integrated DBMS to Support Advanced Applications, GI-Fachtagung Datenbanksysteme für Büro, Technik und Wissenschaft, Karlsruhe 1985, Springer Verlag 1985

/Mi84/ B. Mitschang: Überlegungen zur Architektur von Datenbanksystemen für Ingenieuranwendungen, in: H.-D. Ehrich (Hrsg.), Proc. GI-14. Jahrestagung Braunschweig 1984, IFB 88, Springer Verlag 1984

/Mo82/ J. Moss, Nested Transactions and Reliable Distributed Computing, Proc. 2nd IEEE Symposium on Reliability of Distributed Software and Database Systems, 1982

/MS83/ T. Müller, D. Steinbauer, Eine Sprachschnittstelle zur Versionenkontrolle in CAM-Datenbanken, in: J. W. Schmidt (Hrsg.), Sprachen für Datenbanken, IFB 72, Springer-Verlag 1983

/Neu83/ T. Neumann, On Representing the Design Information in a Common Data Base, Proc. Database Week - Engineering Design Applications, San Jose 1983

/PHH83/ P. Pistor, B. Hansen, M. Hansen, Eine Sequelartige Sprachenschnittstelle für das

NF^2-Modell, in: J.W.Schmidt (Hrsg.), Sprachen für Datenbanken, IFB 72, Springer Verlag 1983

/PR83/ P.Peinl, A.Reuter, Empirical Comparison of Database Concurrency Control Schemes, VLDB 1983

/PSSW84/ H.-B.Paul, H.-J.Schek, M.Scholl, G.Weikum, Überlegungen zur Architektur eines "Non-Standard"-Datenbanksystems, Arbeitsbericht Nr. DVSI-1984-A2, Technische Hochschule Darmstadt, 1984

/Re83/ D.P.Reed, Implementing Atomic Actions on Decentralized Data, ACM Transactions on Computer Systems Vol.1 No.1, 1983

/Ro82/ M.J.Rochkind, Structure of a Database File System for UNIX Operating Systems, The Bell System Technical Journal Vol. 61 No.9, 1982

/Sch84/ H.-J.Schek, Nested Transactions in a Combined IRS-DBMS Architecture, in: Proc. 3rd BCS/ACM Symp. on Research and Development in Information Retrieval, Cambridge 1984

/Sch85/ H.-J.Schek, Towards a Basic Relational NF^2-Algebra Processor, Proc. International Conference on Foundations of Data Organization, Kyoto, Japan, 1985

/Scho82/ M.Scholl, Algebraische Frageoptimierung in Datenbanksystemen mit nichttrivialen Abbildungen zwischen konzeptuellem und internem Datenmodell, Diplomarbeit, Technische Hochschule Darmstadt, 1982

/SLTC82/ N.C.Shu, V.Y.Lum, F.C.Tung, C.L.Chang, Specification of Forms Processing and Business Procedures for Office Automation, IEEE Transactions on Software Engineering Vol. SE-8, No. 5, 1982

/SP82/ H.-J.Schek, P.Pistor, Data Structures for an Integrated Data Base Management and Information Retrieval System, VLDB 1982

/SR84/ M.Stonebraker, L.A.Rowe, Database Portals: A New Application Programm Interface, VLDB 1984

/SS83/ H.-J.Schek, M.Scholl, Die NF^2-Relationenalgebra zur einheitlichen Manipulation externer, konzeptueller und interner Datenstrukturen, in J.W.Schmidt (Hrsg.), Sprachen für Datenbanken, IFB 72, Springer Verlag 1983

/SS84/ H.-J.Schek, M.Scholl, An Algebra for the Relational Model with Relation-Valued Attributes, Technical Report DVSI-1984-T1, Technische Hochschule Darmstadt, 1984

/SSp84/ P.M.Schwarz, A.Z.Spector, Synchronizing Shared Abstract Types, ACM Transactions on Computer Systems, Vol. 2 No. 3, 1984

/Tr83/ I.L.Traiger, Trends in Systems Aspects of Database Management, Proc. 2nd Int. Conf. on Databases (ICOD-2), Cambridge 1983

/Ve79/ J.S.M. Verhofstad, Recovery Based on Types, in: G.Bracchi/G.M.Nijssen (eds.), Data Base Architecture, North-Holland Publ. 1979

/Wa84/ B.Walter, Nested Transactions with Multiple Commit Points: An Approach to the Structuring of Advanced Database Applications, VLDB 1984

/Wei83/ G.Weikum, Entwurfsüberlegungen für einen Versionen-Manager zur Realisierung eines Temporalen Datenbanksystems, TH Darmstadt, Arbeitsbericht DVSI-1983-A1

/Wei84/ G.Weikum, Transaktions-Recovery in Datenbanksystemen mit Schichtenarchitektur: Neue Ansätze zu einer Systematik, TH Darmstadt, Arbeitsbericht DVSI-1984-A1

/WS84/ G.Weikum, H.-J.Schek, Architectural Issues of Transaction Management in Multi-Layered Systems, VLDB 1984