Proc. GI-Fachtagung BTW,
Kaiserslautern, März 1991

Auswertung komplexer Anfragen an hierarchisch strukturierte Objekte mittels Pfadindexen

Ullrich Keßler*, Peter Dadam

Universität Ulm
Fakultät für Informatik
Abt. Betriebliche Informationssysteme/CIM
Postfach 4066, D-7900 Ulm

Zusammenfassung

Die Diskussion über die Verwendung von Indexen zur Anfrageauswertung in objekt-orientierten Datenbanksystemen wird immer noch relativ weit am Rande geführt. Insbesondere dem für die Konzeption eines Datenbanksystems wichtigen Aspekt der Abgrenzung der Funktionalität des Indexmanagers gegen die Auswertungsstrategien wurde bisher wenig Beachtung geschenkt. In diesem Beitrag wird gezeigt, wie mehrfach geschachtelte Anfragen zum Zugriff auf komplexe hierarchisch strukturierte Objekte, die komplexen Tupeln in NF^2-Relationen ähneln, formuliert und unter Verwendung von Pfadindexen ausgewertet werden können. Dabei werden die Operationen auf Indexen und deren Verwendung zur Auswertung von Anfragen klar gegeneinander abgegrenzt. Dazu wird zuerst die Funktionalität eines Indexmanagers für Pfadindexe beschrieben, die zum Teil erheblich von derjenigen abweicht, die von relationalen Systemen her bekannt ist. Auf der Basis dieses Indexmanagers wird dann gezeigt, wie auch komplexe Anfragen, die gleichzeitig Objekte aus mehreren Klassen selektieren, unter Ausnutzung der Möglichkeiten, die Pfadindexe bieten, geeignet ausgewertet werden können. Ein wichtiges Konzept hierbei ist die Zerlegung von Anfragen in sogenannte unabhängige und abhängige Teilanfragen, für die jeweils unabhängige alternative Auswertungsstrategien entwickelt werden.

1. Einleitung

In den letzten Jahren ist immer deutlicher geworden, daß es notwendig ist, in Datenbanken komplex strukturierte Daten und nicht nur Tupel in erster Normalform wie in relationalen Systemen zu verwalten. Aus dieser Erkenntnis entstanden viele Systeme, in denen komplexe Daten direkt gespeichert werden können. Zu nennen sind einmal Systeme wie AIM-P [Dada86] und DASDBS [ScWe87], in denen die komplexen Daten in NF^2-Relationen abgelegt werden. Zum anderen gibt es Systeme, die komplexe Daten in miteinander verbundenen einzelnen Objekten speichern. Typische Vertreter sind GemStone, Orion und O_2 [MaSt87], [Kim89], [Banc88]. Außer einem geeigneten Datenmodell wird auch eine auf die Struktur der gespeicherten Daten abgestimmte Anfragesprache benötigt. Mit ihr müssen die komplexen Daten zugegriffen und verwaltet werden können. Dazu wurden u.a. in [RoKS88],

---

* Teile dieser Arbeit entstanden während des Aufenthalts als Gastwissenschaftler am Wissenschaftlichen Zentrum Heidelberg der IBM Deutschland GmbH

[ScSc86], [PiTr86] der Relationenkalkül, die Relationenalgebra und SQL so erweitert, daß diese Sprachen für NF^2-Relationen anwendbar werden. Zum Zugriff auf beliebig strukturierte Objekte wurde z. B. in [MaSt87] die Sprache Opal vorgestellt.

Ein Datenbanksystem für komplexe Daten wird letztlich jedoch nur dann auf breiter Basis akzeptiert werden, wenn die Anfragen mit einer genügenden Effizienz ausgewertet werden. Hierzu wird man in der Regel Indexe in die Auswertung der Anfragen mit einzubeziehen haben. Wesentlich hierbei ist, daß die Indexe bei der Auswertung von Anfragen automatisch ausgewählt und in die Auswertung integriert werden können. Dies wiederum verlangt Auswertungsstrategien, die auch dann noch automatisch vom System ausgewählt und eingesetzt werden können, wenn komplexe Anfragen gestellt werden.

An dieser Stelle setzt die vorliegende Arbeit auf. In ihr wird ein Konzept vorgestellt, das im Zusammenhang zeigt, wie komplexe Anfragen mit Pfadindexen ausgewertet werden können. Die darin betrachteten komplexen Anfragen selektieren aus einer Menge von komplexen hierarchisch strukturierten Objekten in einer einzelnen Anfrage ganze Substrukturen dieser Objekte. Die dazu notwendige gezielte Betrachtung komplexer Objekte mit einer hierarchischen Struktur erscheint aus Effizienzgründen angemessen, da solch strukturierte Objekte eine wichtige praktische Bedeutung haben. Mit ihnen lassen sich in einer objekt-orientierten Datenbank die in nicht unerheblichem Maße auftretenden 1:n Beziehungen zwischen Einzelobjekten modellieren. Außerdem können sie, im Gegensatz zu beliebig strukturierten komplexen Objekten, effizient durch Pfadindexe invertiert werden. In diesem Konzept setzt sich ein komplexes hierarchisch strukturiertes Objekt - im folgenden kurz als hierarchisches Objekt bezeichnet -, ähnlich wie ein komplexes Tupel einer NF2-Relation, aus einem Wurzelobjekt und davon direkt und indirekt abhängigen Subobjekten zusammen. Hierarchische Objekte können hierbei beliebig tief strukturiert sein.

Zur Formulierung der Anfragen verwenden wir eine Kalkülschreibweise, in der komplexe Anfragen aus sogenannten unabhängigen und abhängigen Teilanfragen aufgebaut werden. Dabei ist die Syntax dieser Kalkülschreibweise (verglichen z.B. mit [RoKS88]) so gewählt, daß sich der für die folgenden Ausführungen wichtige Unterschied zwischen unabhängigen und abhängigen Teilanfragen direkt widerspiegelt. Mit dieser Kalkülschreibweise lassen sich komplexe Anfragen formulieren, die gleichzeitig Wurzelobjekte und deren direkt und indirekt abhängigen Subobjekte selektieren. Diese Anfragen führen dann in einem einzigen Datenbankaufruf beliebige Selektionen und Projektionen auf hierarchischen Objekten aus. Hierin unterscheidet sich die verwendete Kalkülschreibweise von den derzeit diskutierten typischen Anfragesprachen für objekt-orientierte Datenbanksysteme, wie z.B. Opal [MaSt87] oder die in [KeMo90],[KeMo90a] zugrunde gelegte Sprache, in denen solche Selektionen nur schrittweise durch Formulierung von Einzelanfragen mit entsprechend vielen Datenbankaufrufen ausgeführt werden können.

Die Auswertung solcher komplexen Anfragen kann sehr gut durch Indexe unterstützt werden, die Attribute einzelner Objektklassen invertieren und gleichzeitig die hierarchischen Abhängigkeiten zwischen den einzelnen Objekten enthalten. Solche Indexe lassen sich entweder als Pfadindexe oder als eine Kombination aus normalen und Link-Indexen realisieren. Pfadindexe, die ein Attribut einer Subobjektklasse indexieren, speichern für jedes invertierte Subobjekt den gesamten hierarchischen Identifier-Pfad. Dieser beginnt bei dem Wurzelobjekt, von dem das Subobjekt direkt oder indirekt abhängt, und endet bei dem Subobjekt selbst. Pfadindexe ähneln damit den für NF^2-Relationen in [ScSc83] oder [Dada86] beschriebenen Indexen. Bei der Kombination aus normalen und Link-Indexen wird ein normaler Index über das indexierte Attribut einer Subobjektklasse aufgebaut. Dieser liefert bei Aufruf nur Identifier der indexierten Subobjekte zurück. Die Identifier der zugehörigen Wurzelobjekte und der Subobjekte, die auf dem Pfad von den Wurzelobjekten zu den indexierten Subobjekten liegen, werden durch Anfragen an die Link-Indexe ermittelt. Diese enthalten, ähnlich wie die

in [Vald87] beschriebenen Join-Indexe, den Zusammenhang zwischen Objekten und ihren Subobjekten. In [MaSt86] und [BeKi89] werden beide Varianten diskutiert. Das Ergebnis dieser Diskussion ist, daß die einzelnen Indexe einer Kombination aus normalen und Link-Indexen stets mit akzeptablen Kosten gewartet werden können, auch wenn beliebig komplex strukturierte Objektstrukturen zu invertieren sind. Dahingegen kann die Wartung von Pfadindexen in solchen Situationen, besonders wenn dabei auch noch beliebige Update-Operationen zugelassen werden, praktisch unmöglich werden. Der Nachteil der Kombination aus normalen und Link-Indexen ist dabei jedoch, daß mehr Indexe aufgebaut und gewartet werden müssen, als wenn Pfadindexe eingesetzt werden. Außerdem verlangen die durch diese Indexe bedingten und in [MaSt86] und [BeKi89] beschriebenen Auswertungsstrategien, daß zur Auswertung eines Prädikates auf mehrere Indexe zugegriffen werden muß. Pfadindexe hingegen erfordern zur Auswertung eines Prädikates einer Anfrage jeweils nur einen Indexzugriff, da sie genau die benötigte Identifier-Menge - und nicht lediglich eine Obermenge davon - zurückliefern. Berücksichtigt man, daß jeder Indexzugriff zu festen Verwaltungskosten, wie Öffnen des Indexes und Anlegen eines Kontrollblocks, führt, ist eine Lösung, die weniger Indexe benötigt, erstrebenswert. Deshalb bietet es sich an, die Verwendung von Pfadindexen, dort wo es möglich ist, zu bevorzugen. Dies ist zum Beispiel der Fall, wenn Mengen von hierarchischen Objekten invertiert werden sollen, insbesondere wenn diese stets, wie in diesem Beitrag angenommen, von den Wurzelobjekten aus zugegriffen werden (siehe dazu auch Abschnitt 3.1). In diesem Fall tritt das in [MaSt86] diskutierte Problem, daß Pfadindexe nur mit hohen Kosten zu aktualisieren sind, nicht auf.

Will man Pfadindexe für hierarchische Objekte effizient einsetzen, so genügt es nicht, den Indexmanager wie in relationalen Systemen [Seli79], [JaKo84] zu implementieren und beim ersten Zugriff auf einen Index den ersten Eintrag zu suchen, der dem auszuwertenden Prädikat entspricht, und dann bei jedem weiteren Zugriff einen beliebigen nächsten Eintrag zu liefern, der dem Prädikat entspricht. Vielmehr ist es notwendig, daß der Indexmanager die Menge der gefundenen Indexeinträge aufbereitet und die Einträge in einer Form zurückliefert, die der Struktur der hierarchischen Objekte entspricht. Dazu wird nachfolgend eine Schnittstelle für einen Indexmanager vorgeschlagen, die - inspiriert von der Idee in [ScSc83], Pfadindexe als NF^2-Relationen zu repräsentieren - die Ergebnisse von Indexanfragen in Form von virtuellen NF^2-Relationen zurückliefert. Einhergehend damit wird explizit beschrieben, welche Funktionalität und Freiheitsgrade der Indexmanager anbieten sollte.

Allein das Vorhandensein eines mächtigen Indexmanagers ist jedoch noch kein Garant für eine effiziente Anfrageauswertung. Bei der Anfrageauswertung muß vielmehr von den Möglichkeiten, die ein Indexmanager bietet, auch sinnvoll Gebrauch gemacht werden. Ob und wie ein Index sinnvoll eingesetzt werden kann, hängt von dem Typ einer Anfrage und der Datenverteilung ab. Für verschiedene Anfragetypen und Datenverteilungen müssen demzufolge unterschiedliche Strategien vorgesehen werden. Da es jedoch unmöglich ist, für jede denkbare Kombination ein spezielles Auswertungsverfahren vorzusehen, werden allgemeingültige Verfahren benötigt, mit denen beliebige komplexe Anfragen ausgewertet werden können. Ein solches Verfahren wird im zweiten Teil der Arbeit entwickelt. Bei diesem werden komplexe Anfragen in sogenannte unabhängige und abhängige Teilanfragen zerlegt, für die dann, bei Beachtung eines ebenfalls beschriebenen Sonderfalls, unabhängig voneinander die am besten geeigneten Zugriffsmethoden gewählt werden können. Anschließend werden die dabei erzeugten Teilpläne zu einem Gesamtplan zusammengefügt. Bei diesem Vorgehen werden nur noch alternative Auswertungsstrategien für die wesentlich einfacher strukturierten Teilanfragen benötigt.

Der Rest dieses Papiers gliedert sich wie folgt: In Abschnitt 2 werden die betrachteten hierarchischen Objekte und die Anfragesprache genauer beschrieben. Daran schließt sich in Abschnitt 3 die Beschreibung der Funktionalität des Indexmanagers an. Dieser Abschnitt enthält auch die Ergebnisse einiger typischer Indexanfragen, auf die dann in den Abschnitten 4

und 5 Bezug genommen wird. Abschnitt 4 zeigt, wie beliebig strukturierte Anfragen mit und ohne Indexe ausgewertet werden können. Abschnitt 5 behandelt abschließend einen Sonderfall, der bedingt, daß die Teilanfragen nicht in jedem Fall vollkommen unabhängig behandelt werden sollten. Abschnitt 6 faßt das Geschriebene noch einmal kurz zusammen und gibt einen kleinen Ausblick.

2. Objektstruktur und Anfragesprache

2.1. Komplexe hierarchisch strukturierte Objekte

Ein komplexes hierarchisch strukturiertes Objekt setzt sich - ähnlich wie komplexe Tupel im NF^2-Relationenmodell - aus einem Wurzelobjekt und direkt und indirekt hierarchisch abhängigen Subobjekten zusammen. Als Wurzelobjekte werden die Objekte bezeichnet, die keinen Vater haben. Dagegen hat jedes Subobjekt genau ein Vaterobjekt. Dies kann entweder ein Wurzelobjekt oder ein anderes Subobjekt sein, so daß sich Objekthierarchien beliebig tief schachteln lassen. Wie im NF^2-Relationenmodell kann ein Subobjekt weder zwei Väter haben, noch kann es ohne Vater existieren.

Jedes Wurzelobjekt und jedes Subobjekt gehört genau einer Objektklasse an, wobei alle Objekte einer Klasse die gleiche Struktur haben. Die Attribute können entweder einen skalaren Typ wie Integer oder String oder sie können Mengen von Subobjekten als Wert haben, wobei dann alle Subobjekte einer Menge der gleichen Klasse angehören müssen. Damit werden Beziehungen zwischen Objekt- und Subobjektklassen über typgebundene Attribute der Vaterobjekte realisiert.

Jedes Wurzelobjekt und jedes Subobjekt soll wie jedes Objekt eines objekt-orientierten Datenbanksystems einen eindeutigen, systemverwalteten Identifier haben, der für den Anwender allerdings unsichtbar bleibt.

Ein Beispiel für die Struktur eines hierarchischen Objektes ist in Abbildung 1 dargestellt (K_ID, A_ID, S_ID und E_ID seien die oben beschriebenen Identifier). Drei Beispielobjekte mit dieser Objektstruktur sind in Abbildung 2 gegeben.

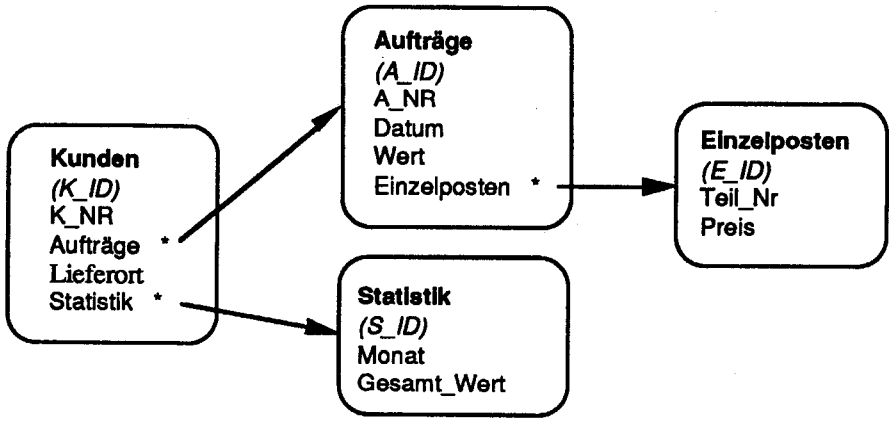

Abb. 1: Beispiel für einen komplexen hierarchisch strukturierten Objekttyp

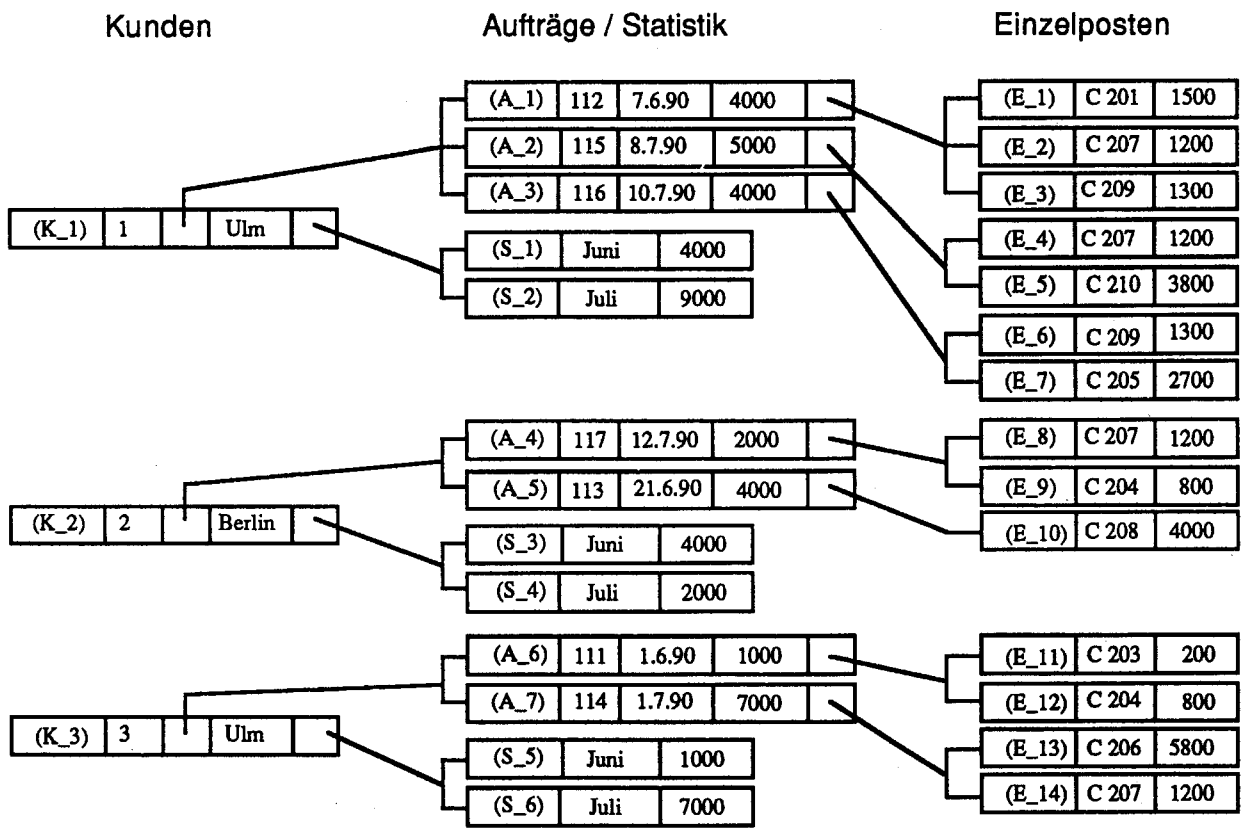

Abb. 2: Mögliche Ausprägungen des hierarchischen Objekttyps in Abbildung 1

2.2. Anfragesprache

In der Anfragesprache können gleichzeitig mit der Selektion von Wurzelobjekten deren direkt oder indirekt abhängigen Subobjekte mit selektiert werden. Es ist nicht notwendig, daß der Benutzer oder ein von ihm geschriebenes Anwendungsprogramm, wie bei einigen objektorientierten Systemen üblich (siehe z.B. [MaSt87]), zuerst Wurzelobjekte aus einer Klasse selektieren muß, um dann erst in einem zweiten Schritt für jedes selektierte Objekt einzeln die benötigten Subobjekte zu bestimmen. Dazu können Anfragen des Grundtyps

{[Projektionsliste] | Variablenbindung und Prädikat}

ineinandergeschachtelt werden. In der "Variablenbindung" wird eine Objektvariable an eine Klasse gebunden. Aus dieser Klasse selektiert die Anfrage alle Objekte, die dem "Prädikat" genügen. In der "Projektionsliste" können die Attribute eines Objektes, die im Ergebnis sichtbar sein sollen, angegeben werden. Außerdem können durch Einfügen von geschachtelten Anfragen in die Projektionsliste gleichzeitig mit der Selektion von Objekten deren Subobjekte mit selektiert werden. Beispielanfrage 1 zeigt eine solche typische Schachtelung.

Beispielanfrage 1:

Zur Vorbereitung einer Werbeaktion werden alle Kunden benötigt, die aus Ulm beliefert werden. Von diesen Kunden sollen die Aufträge, die einen Wert zwischen 2000 und 4000 haben, mit ausgewählt werden. Dabei werden bewußt auch die Kunden mit ausgegeben, die keinen Auftrag zwischen 2000 und 4000 haben.

```
{[ K_Nr:      K.K_Nr,
   Aufträge:  {A | A in K.Aufträge and 2000 ≤ A.Wert ≤ 4000},
   Lieferort: K.Lieferort,
   Statistik: K.Statistik ]

 | K in Kunden and K.Lieferort = 'Ulm'}
```

In dieser Anfrage werden beide Arten der Variablenbindung verwendet, die benötigt werden, um beliebige Selektionen auf komplexen Objekten, wie sie hier definiert sind, auszudrücken. In der äußeren Anfrage wird die Variable an eine Wurzelobjektklasse (hier: Kunden) gebunden. In der inneren Anfrage werden Subobjekte (hier: Aufträge) in Abhängigkeit ihrer Vaterobjekte selektiert.

Anfragen von dem äußeren Typ werden im folgenden als **unabhängige Anfragen** bezeichnet. Sie zeichnen sich dadurch aus, daß Objekte aus solchen Klassen selektiert werden, in denen die Objekte keine Väter haben. Syntaktisch sind diese Anfragen daran zu erkennen, daß kein Vaterobjekt in der Anfrage spezifiziert wird.

Beispiel für die Syntax einer unabhängigen Anfrage:

{[...] | **K in Kunden** and ... }

Anfragen von dem inneren Typ werden als **abhängige Anfragen** bezeichnet. Sie selektieren Subobjekte aus Subobjektklassen. Da Subobjekte stets von Vaterobjekten abhängen, muß auf diese in der Anfrage Bezug genommen werden. Syntaktisch zeigt sich das daran, daß in diesen Anfragen eine Variable angegeben wird, die das Vaterobjekt enthält, von dem die Subobjekte abhängen.

Beispiel für die Syntax einer abhängigen Anfrage:

{[...] | **A in K.Aufträge** and ... }

Diese beiden Anfragetypen reichen aus, um hierarchische Objekte zu selektieren und dabei gleichzeitig gezielt Subobjekte auszuwählen. Abhängige Anfragen können dabei beliebig tief geschachtelt werden, so daß sich beliebig tief strukturierte hierarchische Objekte bearbeiten lassen.

3. Architektur des Datenbanksystems

Wie in vielen objekt-orientierten Systemen wird auch im folgenden zwischen dem Objekt- und dem Indexmanager unterschieden. Der Objektmanager speichert die komplexen Objekte persistent, während der Indexmanager die zusätzlichen Zugriffspfade verwaltet.

3.1. Objektmanager

Eine typische systeminterne, funktionale Schnittstelle eines Objektmanagers bietet zum sequentiellen Lesen einzelner Objekte einer Klasse Funktionen der folgenden Art an:

- get_first_object (object_class, projection, out: cursor): object
- get_next_object (object_class, projection, in_out: cursor): object

Nachdem ein Objekt gelesen wurde, steht der zurückgelieferte Cursor noch auf dem Objekt. Er kann anschließend in weiteren Funktionsaufrufen, z. B. von get_next_object, verwendet werden.

Für den direkten Zugriff auf Objekte benötigt man eine Funktion der folgenden Art:

- get_object_by_id (object_class, projection, out: cursor): object

Zusätzlich werden noch Funktionen zum Lesen von Subobjekten in Abhängigkeit zu ihren Vaterobjekten angenommen:

- get_first_subobject (parent_object_cursor, sub_object_class, projection, out: cursor): subobject
- get_next_subobject (parent_object_cursor, sub_object_class, projection, in_out: cursor): subobject
- get_subobject_by_id (parent_object_cursor, sub_object_class, projection, out:cursor): subobject

Wie an den Parametern zu erkennen, soll auf Subobjekte nur zugegriffen werden können, nachdem das zugehörige Vaterobjekt gelesen wurde. Hierdurch wird sichergestellt, daß Pfadindexe auf hierarchischen Objekten leicht zu warten sind, da stets ausgehend von einem Wurzelobjekt auf ein Subobjekt zugegriffen wird. Der Pfad eines veränderten Subobjektes, der in einen Index einzutragen ist, ist damit stets bekannt.

Diese Funktionen werden natürlich nicht von einem Anwender oder seinem Programm aus aufgerufen, sondern nur systemintern zur Auswertung der Anfragen verwendet.

3.2. Indexmanager

Im Gegensatz zum Objektmanager, dessen Aufgaben nur angedeutet wurden, muß die Funktionalität des Indexmanagers etwas ausführlicher beschrieben werden, damit die in den Abschnitten 4 und 5 behandelten Auswertungsstrategien verständlich bleiben.

3.2.1. Pfadindexe

Wie in der Einleitung ausgeführt, sind Pfadindexe zur Invertierung von komplexen hierarchisch strukturierten Objekten besonders interessant, da in ihnen auch die Abhängigkeiten zwischen den einzelnen Objekten gespeichert werden. Hierdurch können Anfragen mit wenigen Indexzugriffen ausgewertet werden. Ein Pfadindex, aufgebaut über einem Attribut einer Subobjektklasse, speichert im Gegensatz zu einfachen Indexen nicht nur den Identifier eines indexierten Subobjektes, sondern den ganzen Identifier-Pfad von dem Wurzelobjekt des Subobjektes bis hin zu dem Subobjekt selbst.

Viele Indexstrukturen sind geeignet, Pfadindexe zu implementieren. Und obwohl das Nachfolgende weitestgehend für alle Indexstrukturen gilt, sei hier angenommen, daß B*-Bäume

verwendet werden, so daß Annahmen über die Reihenfolge der Indexeinträge in den Indexen gemacht werden können.

Graphisch können Pfadindexe durch Tabellen oder auch als NF^2-Relationen repräsentiert werden. Als Beispiel ist in Abbildung 3 ein Pfadindex für das Attribut "Wert" der Subobjektklasse "Aufträge" sowohl als Tabelle wie auch als NF^2-Relation dargestellt.

Wert_Index		
Wert	K_ID	A_ID
1000	K_3	A_6
2000	K_2	A_4
4000	K_1	A_1
4000	K_1	A_3
4000	K_2	A_5
5000	K_1	A_2
7000	K_3	A_7

Wert_Index		
Wert	K_ID	{A_IDs}
		A_ID
1000	K_3	A_6
2000	K_2	A_4
4000	K_1	A_1
		A_3
	K_2	A_5
5000	K_1	A_2
7000	K_3	A_7

Abb. 3: Pfadindex, dargestellt als Tabelle und NF^2-Relation

Natürlich werden auch Indexe für atomare Attribute einer Wurzelobjektklasse benötigt, wie etwa für das Attribut "Lieferort" der Wurzelobjektklasse "Kunden". Obwohl solche Indexe für Wurzelobjektklassen keine Pfadindexe im eigentlichen Sinne sind, können sie wie in Abbildung 4 als Pfadindexe mit einer Pfadlänge 1 aufgefaßt und ebenfalls als Tabelle oder NF^2-Relation dargestellt werden.

Lieferort_Index	
Lieferort	K_ID
Berlin	K_2
Ulm	K_1
Ulm	K_3

Lieferort_Index	
Lieferort	{K_IDs}
	K_ID
Berlin	K_2
Ulm	K_1
	K_3

Abb. 4: Index für eine Wurzelobjektklasse

3.2.2. Ergebnis einer Indexanfrage

Bevor der erste Indexeintrag gelesen werden kann, muß dem Indexmanager bekanntgegeben werden, auf welchen Index zuzugreifen und welches Prädikat auszuwerten ist. Der Indexmanager öffnet daraufhin den Index und bereitet das Lesen der Identifier vor. Für das Lesen der Identifier gibt es zwei mögliche Vorgehensweisen. Bei der einen wird bei jedem Aufruf des Indexmanagers der erste bzw. nächste Identifier entsprechend dem vorgegebenen Prädikat bestimmt. Bei der anderen Vorgehensweise werden bereits beim ersten Aufruf des Indexmanagers gleich alle relevanten Identifier ermittelt und zwischengespeichert. Man sagt, das Ergebnis der Indexanfrage wird "materialisiert". Die erste Variante hat den Vorteil, daß das Ergebnis eines Indexzugriffs nicht materialisiert zu werden braucht, wodurch sich die Auswertungskosten reduzieren. Sie wird daher nach Möglichkeit eingesetzt. Die effiziente

Auswertung von Bereichsprädikaten und die in den Abschnitten 4.1.3 und 4.2.3 diskutierten speziellen Auswertungstechniken erfordern jedoch, daß das Ergebnis einer Indexanfrage auch materialisiert werden kann, sollen diese Strategien nicht von vornherein ausgeschlossen werden. Deshalb sollten beide Varianten in einem System vorgesehen werden.

Um die Auswertungsstrategien aber nicht unnötig zu komplizieren, sollte außerhalb des Indexmanagers nicht unterschieden werden müssen, ob ein Indexergebnis materialisiert ist oder nicht. Deshalb wird hier vorgeschlagen, daß der Indexmanager das Ergebnis beide Male in Form einer **virtuellen Relation** zur Verfügung stellt. Hierzu eignen sich insbesondere NF^2-Relationen, da sich in ihnen die Identifier-Pfade bezüglich gemeinsamer Präfixe gruppieren lassen. Damit spiegelt sich in ihnen automatisch auch die Objektstruktur wider. Die NF^2-Relationen, die nach einem Indexzugriff die gefundenen Identifier-Pfade enthalten, werden im folgenden als **Indexergebnisrelationen** (index access result) bezeichnet. Eine virtuelle Indexergebnisrelation könnte nach Zugriff auf den Index für das Attribut "Wert" wie folgt aussehen:

index_access result	
K_ID	{A_IDs}
	A_ID
K_1	A_1
	A_3
K_2	A_5

Abb. 5: Beispiel einer Indexergebnisrelation

Eine Indexergebnisrelation kann sequentiell mit Funktionen wie "get_first_object_id", "get_next_object_id", "get_first_subobject_id" und "get_next_subobject_id" gelesen werden. Sofern die Indexergebnisrelation nicht materialisiert zu werden braucht, simuliert der Indexmanager die NF^2-Ergebnisstruktur durch Verschieben entsprechender Zeiger im Index. Zusätzlich verlangen die in den Abschnitten 4.1.3 und 4.2.3 beschriebenen Auswertungsstrategien die assoziative Suche in einer Indexergebnisrelation. Es bietet sich daher an, diese gegebenenfalls hauptspeicherresident in einer Hashtabelle zu materialisieren, so daß eine zusätzliche Funktion "find_identifier" effizient implementiert werden kann.

3.2.3. Parameter einer Indexanfrage

Um einen Pfadindex voll auszunutzen, reicht es nicht aus, beim Zugriff auf einen Index nur ein einfaches Vergleichsprädikat angeben zu können. Die später entwickelten Strategien setzen voraus, daß auch Adreßprädikate angegeben und die Struktur der Indexergebnisrelationen explizit beeinflußt werden können. Hierzu kann, wie in [ScSc83], eine systeminterne Algebra zur Manipulation von NF^2-Relationen verwendet werden. Mit ihr läßt sich jeder beliebige komplexe Indexzugriff spezifizieren. Die Mächtigkeit des Indexmanagers ist damit allerdings auch wesentlich größer als benötigt.

Um die Funktionalität des Indexmanagers möglichst exakt auf die Bedürfnisse der Auswertungsstrategien abzustellen, wird in diesem Beitrag eine funktionale Schreibweise verwendet. Dazu wird die Funktion **index_access** definiert, deren Parameter so gewählt sind, daß sich durch sie alle Indexergebnisrelationen, die eine der im folgenden beschriebenen Auswertungsstrategien verwenden, beschreiben lassen. Durch diese kompakte Schnittstellendefinition wird erreicht, daß im Indexmanager genau die tatsächlich benötigten Operationen effizient implementiert werden können.

Die Parameter der Funktion "index_access" sind:

```
index_access ( index:              zugegriffener Index,
               predicate:          auzuwertendes Prädikat,
               address_predicate:  zusätzliches Adreßprädikat,
               path_projection:    (Projektion des Adreßpfades),
               associative_access: assoziativer Zugriff auf die Indexergebnisrelation)
```

Ihre Bedeutung wird nun an kurzen Beispielen beschrieben, welche die wichtigsten Anwendungsfälle repräsentieren. Dabei kann ihre Verwendung jedoch teilweise erst gezeigt werden, wenn diskutiert wird, wie Anfragen ausgewertet werden. Gleichzeitig wird noch gezeigt, warum eine Indexergebnisrelation bei Angabe eines Bereichsprädikats materialisiert werden muß.

Der Standardzugriff auf einen Index, bei dem nur ein Prädikat sepzifiziert wird, ohne dabei ein Adreßprädikat anzugeben oder den Identifier-Pfad zu verkürzen, wird mit Indexanfragen des Typs I formuliert.

Indexanfragetyp Ia: Zugriff auf einen einfachen Index mit einfachem Prädikat

Beispielindexanfrage 1:

Es werden alle Identifier der Objekte der Klasse "Kunden" gesucht, die in dem Attribut "Lieferort" den Wert "Ulm" besitzen.

```
index_access ( index:              Lieferort_Index,
               predicate:          Lieferort = Ulm,
               address_predicate:  nil,
               path_projection:    (K_ID),
               associative_access: no);
```

index_access result
K_ID
K_2
K_3

Indexanfragetyp Ib: Zugriff auf einen Pfadindex mit einfachem Prädikat

Beispielindexanfrage 2:

Es werden aus der Subobjektklasse "Aufträge" alle Aufträge gesucht, die einen Wert von 4000 haben. Dabei soll der gesamte Adreßpfad der Subobjekte ausgegeben werden.

```
index_access ( index:              Wert_Index,
               predicate:          Wert = 4000,
               address_predicate:  nil,
               path_projection:    (K_ID, A_ID),
               associative_access: no);
```

index_access result	
K_ID	{A_IDs}
	A_ID
K_1	A_1
	A_3
K_2	A_5

In Indexanfragen des Typs Ib wird von jedem Indexeintrag der gesamte Adreßpfad ausgegeben. Dazu wird in dem Parameter "path_projection" der gesamte Identifier-Pfad spezifiziert. Wird hingegen bei der Anfrageauswertung nur ein Teil des Identifier-Pfades benötigt, weil z.B. ein Existenzprädikat ausgewertet wird, brauchen nur die benötigten Teile des Adreßpfades

angegeben zu werden. Insbesondere wenn das Indexergebnis materialisiert werden muß, kann dadurch Speicherplatz eingespart werden.

Indexanfragen vom Typ Ia und Ib können vom Indexmanager, sofern die gefundenen Identifier wie in den Beispielindexanfragen 1 und 2 nur sequentiell gelesen werden sollen (dies wurde durch den Parameter "associative_access: no" angegeben), bearbeitet werden, ohne die Indexergebnisse zu materialisieren. Es reicht aus, wenn der Indexmanager den B*-Baum vorwärts durchläuft und bei jedem Zugriff auf das Indexergebnis interne Positionszeiger in dem B*-Baum vorwärts schiebt. Soll auf das Indexergebnis jedoch assoziativ zugegriffen werden, um z.B. einen Hash-Merge zwischen zwei Indexergebnissen oder eine im folgenden vorgestellte spezielle Technik zur Auswertung von abhängigen Anfragen zu verwenden, erzwingt der Parameter "associative_access: yes", daß der Indexmanager die Indexergebnisrelation in einer Hashtabelle materialisiert. Damit kann auf die Indexergebnisrelation auch mit der oben erwähnten Funktion "find_identifier" zugegriffen werden.

Es gibt auch Fälle, in denen der Indexmanager die Indexergebnisrelationen automatisch materialisieren muß. Dies ist immer dann nötig, wenn die Indexeinträge beim einfachen Durchlaufen des Indexes nicht entsprechend der Präfixe der Identifier-Pfade gruppiert werden können. Diese Situation ist beispielsweise gegeben, wenn Pfadindexe als B*-Bäume realisiert werden und mit ihnen, wie in Beispielindexanfrage 3, ein Bereichsprädikat ausgewertet werden soll. In dem Ergebnis dieser Beispielanfrage müssen die Identifier-Pfade (K_1, A_1), (K_1, A_3) und (K_2, A_4), (K_2, A_5) jeweils zu einer Gruppe zusammengefaßt werden. Beim einfachen Vorwärtsdurchlaufen des B*-Baumes ist dies jedoch nicht möglich, da zwischen den Indexeinträgen (2000, K_2, A_4) und (4000, K_2, A_5) die Indexeinträge (4000, K_1, A_1) und (4000, K_1, A_3) liegen.

Beispielindexanfrage 3:

Es werden alle Auftragssubobjekte mit einem Wert zwischen 2000 und 4000 gesucht. Wie bei Indexanfragen vom Typ I üblich, wird der gesamte Adreßpfad ausgegeben.

```
index_access ( index:              Wert_Index,
               predicate:          2000 ≤ Wert ≤ 4000,
               address_predicate:  nil,
               path_projection:    (K_ID, A_ID),
               associative_access: no);
```

index_access result	
K_ID	{A_IDs}
	A_ID
K_1	A_1
	A_3
K_2	A_4
	A_5

Als letztes ist noch der Parameter "address_predicate" zu behandeln. Dieser wird in Indexanfragen vom Typ II verwendet. Pfadindexe sind über ganze Subobjektklassen aufgebaut. Damit werden bei der Suche mit einem einfachen Prädikat die Identifier aller Subobjekte, die dem Prädikat genügen, gefunden, und zwar unabhängig davon, welches Vaterobjekt sie haben. Werden die Identifier von Subobjekten eines bestimmten Vaters benötigt, muß nach diesen anschließend im Indexergebnis assoziativ gesucht werden. Alternativ kann auch bei der Eröffnung des Indexes die Suche gleich auf die Subobjekte eingeschränkt werden, die diesen Vater haben. Dazu wird in dem Parameter "address_predicate" der Identifier des Vaterobjektes spezifiziert.

Indexanfragetyp II: Zugriff auf einen Index mit zusätzlichem Adreßprädikat

Beispielindexanfrage 4:

Gesucht werden alle Auftragssubobjekte, die Sohn des Kundenobjektes mit dem Identifier "K_1" sind und einen Wert zwischen 2000 und 4000 haben.

```
index_access ( index:              Wert_Index,
               predicate:          2000 ≤ Wert ≤ 4000,
               address_predicate:  K_ID = K_1,
               path_projection:    (A_ID),
               associative_access: no);
```

index_access result
A_ID
A_1
A_3

Da bei Anfragen dieses Typs die Identifier der Vaterobjekte bekannt sind, wird der Identifier-Pfad entsprechend verkürzt. Dazu wird der benötigte Teil des Identifier-Pfades in dem Parameter "path_projection" spezifiziert.

4. Prinzip der Auswertung geschachtelter Anfragen

Nachdem im vorigen Abschnitt der Indexmanager diskutiert wurde, werden nun Auswertungsstrategien, die diesen Indexmanager nutzen, vorgestellt. Dazu wird gezeigt, wie beliebige komplexe Anfragen ausgewertet werden können. Das Grundprinzip ist, jede Anfrage in die unabhängigen und abhängigen Teilanfragen, aus denen sie aufgebaut ist, zu zerlegen und für diese jeweils einzelne Auswertungspläne zu erzeugen. Dabei wird für jede Teilanfrage, egal ob unabhängig oder abhängig, eine Programmschleife konstruiert. Die so erzeugten Programmschleifen werden anschließend entsprechend der Struktur der Gesamtanfrage ineinandergeschachtelt.

Bei der Bestimmung der Strategie, mit der eine konkrete Anfrage ausgewertet werden soll, sind im konkreten Fall natürlich auch Kostenabschätzungen anzustellen. Dazu werden im folgenden jedoch nur einige allgemeine Überlegungen angestellt, aber keine expliziten Kostenformeln entwickelt, da diese insbesondere von den gewählten Speicherungsstrukturen für die Objekte und Indexe beeinflußt werden, von denen hier bewußt abstrahiert wurde.

4.1. Auswertung unabhängiger Anfragen

Die Strategien zur Auswertung unabhängiger Anfragen sind dem Prinzip nach von relationalen Systemen her bekannt [Seli79], [JaKo84].

Eine typische unabhängige Anfrage ist in der Beispielanfrage 1 aus Abschnitt 2.2 gegeben:

```
{[ K_Nr:      K.K_Nr,
   Aufträge:  ... ,
   Lieferort: K.Lieferort,
   Statistik: K.Statistik ]

 | K in Kunden and K.Lieferort = 'Ulm'}
```

Diese Anfrage besitzt ein einfaches Prädikat, das durch einen Index, soweit vorhanden, ausgewertet werden kann. Prinzipiell gibt es drei Möglichkeiten vorzugehen.

4.1.1. Sequentielle Suche

Die sequentielle Suche kommt ohne einen Index aus. In einer Schleife werden alle Objekte einer Objektklasse gelesen und geprüft, ob sie das angegebene Prädikat erfüllen. Dazu werden die beiden Funktionen "get_first_object" und "get_next_object", wie in Abschnitt 3.1 beschrieben, verwendet. Diese Methode ist angemessen, wenn nur Prädikate mit geringer Selektivität spezifiziert wurden oder kein Index vorhanden ist.

4.1.2. Index Scan

Ist ein passender Index vorhanden und soll das Prädikat in der Anfrage mit diesem ausgewertet werden, so wird eine entsprechende Indexanfrage vom Typ Ia spezifiziert. In dem Beispiel würde die Funktion "index_access" wie in Beispielindexanfrage 1 aufgerufen werden. Anschließend werden die Identifier sequentiell aus der Indexergebnisrelation gelesen, und parallel dazu wird mit der Funktion "get_object_by_id" auf die Wurzelobjekte zugegriffen. Sofern das Prädikat es nicht erfordert, wird bei diesem Verfahren das Indexergebnis nicht materialisiert. Dieses Verfahren bietet sich insbesondere bei selektiven Prädikaten an.

4.1.3. Indexzwischenergebnis

Enthält die Anfrage zwei oder mehr Prädikate, die durch Indexe ausgewertet werden sollen, so werden alle Indexergebnisrelationen bis auf eine materialisiert. Dazu wird in den entsprechenden Indexanfragen der Parameter "associative_access" auf "yes" gesetzt. Anschließend werden die Identifier in dem nicht materialisierten Indexergebnis sequentiell gelesen und durch assoziativen Zugriff auf die übrigen Indexergebnisse getestet, ob der Identifier auch in ihnen auftritt. Verschränkt dazu wird auf die Objekte direkt zugegriffen.

4.1.4. Auswertung von Existenzprädikaten

Bis hierhin wurden stets einfache Vergleichsprädikate, die durch Indexe ausgewertet werden sollen, betrachtet. Daneben gibt es noch eine zweite Prädikatsklasse, die Existenzprädikate, die sich ebenfalls sehr gut durch Indexe auswerten lassen. Ein typisches Existenzprädikat wird in Beispielanfrage 2 verwendet.

Beispielanfrage 2:

Zeige alle Kunden, die mindestens einen Auftrag mit einem Wert zwischen 2000 und 4000 erteilt haben.

{K | K in Kunden and ∃ (A in K.Aufträge): 2000 ≤ A.Wert ≤ 4000}

Hierbei werden alle Objekte gesucht, die mindestens ein Subobjekt mit einer bestimmten Eigenschaft besitzen. Ist ein Index über die entsprechende Subobjektklasse aufgebaut, kann dieser verwendet werden, um eben die Subobjekte mit der Eigenschaft zu selektieren. Zur Auswertung dieser Anfrage kann das Ergebnis der Beispielindexanfrage 3 verwendet werden. Dieses enthält die Identifier aller Kundenobjekte zusammen mit den Auftragssubobjekten, die einen Wert zwischen 2000 und 4000 haben. Die weitere Bearbeitung kann dann mittels des Index-Scans, wie oben beschrieben, erfolgen.

Da bei der Auswertung eines Existenzprädikats mit einem Pfadindex nicht auf die Subobjekte zugegriffen werden muß, bietet es sich an, die Indexergebnisrelation durch Angabe eines entsprechenden Projektionsvektors zu verkleinern. Dies ist insbesondere dann sinnvoll, wenn das Indexergebnis wegen Angabe eines Bereichsprädikats in dem Existenzausdruck

materialisiert werden muß. Aus diesem Grund wird ein Existenzprädikat besser mit einer Indexanfrage vom Typ III ausgewertet. Zur Auswertung der Anfrage 2 würde dann die Beispielindexanfrage 5 verwendet werden.

Indexanfragetyp III: Zugriff auf einen Pfadindex mit Verkürzung der Identifier-Pfade

Beispielindexanfrage 5:

Es werden alle Kundenobjekte mit mindestens einem Auftragssubobjekt, das einen Wert zwischen 2000 und 4000 hat, gesucht.

```
index_access ( index:              Wert_Index,
               predicate:          2000 ≤ Wert ≤ 4000,
               address_predicate:  nil,
               path_projection:    (K_ID),
               associative_access: no);
```

index_access result
K_ID
K_1
K_2

Selbstverständlich lassen sich auch Indexanfragen vom Typ III mit dem Parameter "associative_access: yes" formulieren. Dann kann auf diese Indexergebnisrelationen ebenfalls assoziativ zugegriffen werden, so daß sie bei der Auswertung von mehreren Prädikaten mit mehreren Indexen, wie sie in Abschnitt 4.1.3 beschrieben wurde, orthogonal verwendet werden können.

4.2. Auswertung abhängiger Anfragen

Es folgt nun die Diskussion des wesentlich interessanteren Falls, und zwar die Auswertung abhängiger Anfragen. Diese selektieren stets Subobjekte aus Subobjektklassen in Abhängigkeit von Vater(sub)objekten. Eine typische abhängige Anfrage ist in der Beispielanfrage 1 in Abschnitt 2.2 enthalten:

{A | A in K.Aufträge and 2000 ≤ A.Wert ≤ 4000}

Die Anfrage selektiert zu einem gegebenen Vaterobjekt K alle Subobjekte A, die ein bestimmtes Prädikat erfüllen. Da das Vaterobjekt durch eine Variable spezifiziert ist, hängt es von der Treffermenge der äußeren Anfrage ab, wie oft die innere (abhängige) Anfrage auszuwerten ist.

Es werden wieder 3 Strategien zur Auswertung einer abhängigen Anfrage vorgestellt. Welche auszuwählen ist, hängt von der Selektivität des Prädikates in der abhängigen Anfrage ab. Zusätzlich ist aber bei Auswahl der zweiten und dritten Strategie auch die jeweilige Anzahl der qualifizierten Vaterobjekte zu beachten.

Die einfachste Strategie ist wieder die sequentielle Suche ohne Verwendung eines Indexes.

4.2.1. Sequentielle Suche

Jedesmal wenn bei Auswertung der äußeren Anfrage ein Objekt K gefunden wurde, wird die abhängige Anfrage ausgewertet. Dazu werden in einer Schleife alle von K abhängigen Subobjekte A sequentiell gelesen und das Prädikat ausgewertet. Implementiert wird diese Schleife unter Verwendung der Funktionen "get_first_subobject" und "get_next_subobject". Nachdem ein Cursor auf das Vaterobjekt K gesetzt wurde, können die Subobjekte mit diesen

Funktionen sequentiell gelesen werden. Dieses Verfahren sollte angewendet werden, wenn in der abhängigen Anfrage ein wenig selektives Prädikat angegeben ist oder kein Index zur Verfügung steht.

Ist das Prädikat in der abhängigen Anfrage hingegen sehr selektiv und existiert ein Pfadindex, der das entsprechende Attribut indexiert, so sollte dieser eingesetzt werden. Dazu kann eine der beiden folgenden Strategien verwendet werden.

4.2.2. Indexzugriff mit Adreßprädikat

Um die nachfolgend beschriebene Strategie besser zu motivieren, werden zunächst zwei äquivalente Schreibweisen für die abhängige Anfrage aus Beispielanfrage 1 in Abschnitt 2.2 gegeben. Diese führen dann unmittelbar zu der beschriebenen Auswertungsstrategie.

(1) {A | A in K.Aufträge and 2000 ≤ A.Wert ≤ 4000}

(2) <=> {A | A in Aufträge and parent(A) = K and 2000 ≤ A.Wert ≤ 4000}

(3) <=> {A | A in Aufträge and parent(A).K_ID = K.K_ID and 2000 ≤ A.Wert ≤ 4000}

In Zeile (1) ist die abhängige Anfrage in der Kalkülschreibweise, wie sie in diesem Beitrag verwendet wird, wiedergegeben. Das Vaterobjekt K, von dem die selektierten Subobjekte A abhängen sollen, wird dabei in einem Term "A in K.Subobjektklasse" spezifiziert. Es ist stets in einer äußeren Anfrage gebunden. In der zweiten und dritten Darstellungsform ist dieser Zusammenhang zwischen dem Vaterobjekt und seinen Subobjekten explizit als Prädikat ausgedrückt. Diese Darstellungen implizieren, daß Subobjekte unabhängig von ihrem Vaterobjekt aus Subobjektklassen gelesen werden, wobei dann beide Prädikate explizit überprüft werden müssen.

Nach diesem Prinzip wird vorgegangen, wenn die Auswertungsstrategie **Indexzugriff mit Adreßprädikat** verwendet wird. In dieser Strategie wird eine Indexanfrage vom Typ II verwendet. In ihr wird neben dem normalen Prädikat zusätzlich der Identifier des Vaterobjektes, für das die abhängige Anfrage aktuell auszuwerten ist, als Adreßprädikat mit angegeben. Beispielsweise würde zur Auswertung der obigen abhängigen Anfrage für das Vaterobjekt "K_1" die Beispielindexanfrage 4 formuliert werden. Diese Indexanfrage selektiert genau die Identifier der Subobjekte, die dem Prädikat genügen und die das Vaterobjekt "K_1" haben. Die Verwendung des Identifiers des Vaters als zusätzliches Prädikat ist notwendig, da Pfadindexe über ganze Subobjektklassen aufgebaut sind. Daher liefern die Indexe prinzipiell die Identifier aller Subobjekte zurück, die das angegebene Prädikat erfüllen, unabhängig davon, welchen Vater die Subobjekte haben.

Hier zeigt sich der entscheidende Vorteil von Pfadindexen gegenüber solchen Indexen, die nur den Attributwert und den Identifier eines Subobjektes speichern. Da in diesen Indexen der Zusammenhang zwischen den Subobjekten und ihren Vaterobjekten verlorengeht, muß nach einem Indexzugriff noch geprüft werden, welche der gefundenen Subobjekte den Vater besitzen, für den gerade die abhängige Anfrage ausgewertet wird. Hier gibt es entweder die Möglichkeit, einen weiteren Link-Index zu führen, der den Zusammenhang zwischen Objekt und Subobjekt enthält, oder explizit durch Zugriff auf das Vaterobjekt oder die Subobjekte zu prüfen, ob die entsprechende Beziehung besteht. Im Regelfall ist dabei davon auszugehen, daß die Mehrzahl der gefundenen Subobjekte nicht dem Vater angehören, für den die abhängige Anfrage ausgewertet wird. Dadurch kommt es zu vielen negativen Tests.

Der Vorteil der Strategie **Indexzugriff mit Adreßprädikat** unter Verwendung eines Pfadindexes ist, daß eine abhängige Anfrage, die für ein Vaterobjekt K ausgewertet werden soll, mit einem Zugriff auf einen Index ausgewertet werden kann. Der Index liefert genau die Identifier der Subobjekte zurück, die das Prädikat erfüllen und die Subobjekte des Vaters sind, für den die Anfrage ausgewertet wird. Es ist nicht nötig, das Ergebnis der Indexanfrage weiter aufzuarbeiten. Der Nachteil ist, daß der Index für jedes Vaterobjekt K, für das die abhängige Anfrage auszuwerten ist, einzeln angefragt werden muß. Dies ist insbesondere dann unökonomisch, wenn sich in der äußeren Anfrage viele oder gar alle Objekte qualifizieren, also dort ein wenig selektives Prädikat verwendet wird. In diesem Fall kommt es zu sehr vielen Indexaufrufen. Deshalb sollte dieses Verfahren nur dann eingesetzt werden, wenn sich in der übergeordneten Anfrage wenige Vaterobjekte qualifizieren und damit die abhängige Anfrage nicht zu häufig auszuwerten ist.

Qualifizieren sich in der äußeren Anfrage hingegen viele Objekte, ist es besser, die als nächstes beschriebene Strategie zu verwenden.

4.2.3. Indexauswertung mit Indexzwischenergebnis

Diese Methode verwendet wieder Indexanfragen vom Typ I. Der Pfadindex wird nur mit dem normalen Prädikat ohne das Adreßprädikat aufgerufen. Er liefert dann die Identifier-Pfade aller Subobjekte zurück, die das normale Prädikat erfüllen, und zwar unabhängig davon, welchem Vaterobjekt die Subobjekte angehören. Für die betrachtete abhängige Anfrage würde der Indexmanager entsprechend Beispielindexanfrage 3 aufgerufen werden. Allerdings muß der Parameter "associative_access" mit "yes" initialisiert werden.

Soll nun die abhängige Anfrage für ein bestimmtes Vaterobjekt K ausgewertet werden, müssen in der Indexergebnisrelation die Subobjekte, deren Vater K ist, gesucht werden. Dieser Suchvorgang ist sehr schnell, wenn die Indexergebnisrelation in einer Hashtabelle materialisiert wurde. Sofern diese eine NF^2-Struktur hat, werden mit einem assoziativen Zugriff alle Identifier der Subobjekte gefunden, die einen gemeinsamen Vater haben.

Der Vorteil dieses Verfahrens ist, daß der Pfadindex nur einmal vor der ersten Auswertung der abhängigen Anfrage angefragt werden muß. Danach muß zur Auswertung der abhängigen Anfragen nur noch assoziativ auf die Indexergebnisrelation zugegriffen werden. Der Nachteil ist, daß die Indexergebnisrelation auch Identifier solcher Subobjekte enthält, deren Väter sich in der äußeren Anfrage nicht qualifizieren. Es ist daher insbesondere dann sinnvoll einzusetzen, wenn sich in der äußeren Anfrage viele oder alle Objekte der angefragten Klasse qualifizieren.

4.3. Transformation komplexer Anfragen

Nachdem in den Abschnitten 4.1 und 4.2 Strategien zur Auswertung unabhängiger und abhängiger Anfragen beschrieben wurden, wird nun gezeigt, wie für die einzelnen Teilanfragen, aus denen komplexe Anfragen aufgebaut sind, schrittweise Strategien ausgewählt werden können. Dazu betrachten wir eine Erweiterung der Beispielanfrage 1.

Beispielanfrage 3:

Von den Kunden, die aus Ulm beliefert werden, sollen die Aufträge mit einem Wert zwischen 2000 und 4000 und die bei diesen Aufträgen bestellten Teile "C 207" ausgegeben werden. Außerdem werden von diesen Kunden nur die Statistiken des Monats Juli benötigt.

```
{[ K_Nr:      K.K_Nr,
   Aufträge:  {[ A_Nr:        A.A_Nr,
                 Datum:       A.Datum,
                 Wert:        A.Wert,
                 Einzelposten: {E | E in A.Einzelposten and E.Teil_Nr = 'C 207'}4]
              | A in K.Aufträge and 2000 ≤ A.Wert ≤ 4000}2,
   Lieferort: K.Lieferort,
   Statistik: {S | S in K.Statistik and S.Monat = 'Juli'}3]

 | K in Kunden and K.Lieferort = 'Ulm'}1
```

Diese Beispielanfrage ist aus der äußeren unabhängigen Teilanfrage 1 und den drei abhängigen Teilanfragen 2, 3 und 4 aufgebaut. Entsprechend wird die Auswertungsstrategie für die Gesamtanfrage in vier Schritten festgelegt.

Schritt 1: Auswahl einer Strategie für die unabhängige Teilanfrage 1 entsprechend Abschnitt 4.1.

Schritt 2: Auswahl einer Strategie für die abhängige Teilanfrage 2 entsprechend Abschnitt 4.2.

Schritt 3: Auswahl einer Strategie für die abhängige Teilanfrage 3. Dabei stehen die Strategien "Indexzugriff mit Adreßprädikat" und "Indexauswertung mit Indexzwischenergebnis" nur zur Verfügung, wenn ein weiterer Pfadindex für das Attribut "Monat" der Statistikobjekte existiert.

Schritt 4: Bestimmung einer Strategie für Teilanfrage 4. Auch zur Auswertung von Teilanfragen in der zweiten und jeder weiteren Schachtelungstiefe können die drei Strategien für abhängige Anfragen eingesetzt werden, und zwar unabhängig davon, welche Strategien für die übergeordneten Teilanfragen gewählt wurden. In dem Beispiel erfordert dies einen Pfadindex auf dem Attribut "Teil_Nr" der Einzelpostenobjekte, soll nicht nur die sequentielle Suche eingesetzt werden können. Wird die Teilanfrage 4 schließlich mit der Strategie "Indexzugriff mit Adreßprädikat" ausgewertet, wird in den entsprechenden Indexanfragen vom Typ II neben dem normalen Prädikat noch der Identifier des Auftragssubobjektes A, für das die Teilanfrage 4 aktuell auszuwerten ist, angegeben. Wird hingegen für die Teilanfrage 4 die Strategie "Indexauswertung mit Indexzwischenergebnis" gewählt, wird einmal eine Indexanfrage vom Typ I ausgewertet. Bei der anschließenden Auswertung der Teilanfrage 4 für ein konkretes Auftragsobjekt A wird ganz normal in dem Indexergebnis assoziativ nach den Identifiern der Einzelpostenobjekte gesucht, die von dem Auftragsobjekt A abhängen. Dabei braucht das Kundenobjekt, von dem das Auftragsobjekt wiederum abhängt, nicht spezifiziert zu werden. Da Subobjekte stets nur einen Vater haben, ist die assoziative Suche in dem Ergebnis eindeutig. Analoges gilt auch bei Auswertung der Teilanfrage mit der Strategie "Indexauswertung mit Adreßprädikat".

5. Hierarchische Selektion

In Abschnitt 4 wurde beschrieben, wie sich komplexe Anfragen mit und ohne Pfadindexen auswerten lassen. Dabei konnte unabhängig für jede Teilanfrage einer Gesamtanfrage eine Auswertungsstrategie gewählt werden. Erst nachdem für alle Teilanfragen Strategien gewählt wurden, wird der Auswertungsplan für die Anfrage erzeugt. Der Vorteil dieses Vorgehens ist, daß es auch beim Vorliegen komplexer Anfragen angewendet werden kann. Ein Spezialfall wird dabei jedoch nicht hinreichend gut behandelt. Es kann passieren, daß ein Index zur Auswertung von zwei ineinandergeschachtelten Anfragen zweimal unabhängig voneinander mit dem gleichen Prädikat zugegriffen wird. Dies ist immer dann der Fall, wenn in der äußeren von beiden Anfragen ein Existenzprädikat verwendet wird, welches das gleiche Vergleichsprädikat wie die innere Anfrage enthält. Ein typisches Beispiel hierfür ist die Beispielanfrage 4:

Beispielanfrage 4:

Zeige alle Kunden mit mindestens einem Auftrag mit einem Wert zwischen 2000 und 4000. Von diesen Kunden zeige genau die Aufträge mit einem Wert zwischen 2000 und 4000.

```
{[ K_Nr:      K.K_Nr,
   Aufträge: {A | A in K.Aufträge and 2000 ≤ A.Wert ≤ 4000},
   Lieferort: K.Lieferort,
   Statistik: K.Statistik ]

 | K in Kunden and ∃ (A' in K.Aufträge): 2000 ≤ A'.Wert ≤ 4000 }
```

Dieser Anfragetyp sei hier als hierarchische Selektion bezeichnet. Syntaktisch ist die hierarchische Selektion leicht daran zu erkennen, daß in der äußeren Anfrage ein Existenzprädikat

$$\exists\ (A' \text{ in } K.\text{Subobjektklasse}):\ P(A')$$

benutzt wird, in dem das gleiche Prädikat "P" wie in der inneren abhängigen Anfrage

$$\{A \mid A \text{ in } K.\text{Subobjektklasse and } P(A)\}$$

verwendet wird, wobei die Subobjektvariablen A und A' an die gleichen Subobjektklassen gebunden sind.

Diese Anfragen können entsprechend den Strategien aus Abschnitt 4 ausgewertet werden. Zur Auswertung der äußeren Anfrage würde dann die Beispielindexanfrage 5 vom Typ I formuliert werden. Die innere Anfrage würde mit dem Ergebnis von Beispielindexanfrage 3 oder 4 ausgewertet werden. Dieses Vorgehen ist allerdings insofern ungeschickt, da sich die Beispielindexanfrage 5 und die Beispielindexanfrage 3 bzw. 4 nur in den Projektionsvektoren und eventuell dem Adreßprädikat unterscheiden, nicht aber in dem eigentlichen auszuwertenden Prädikat. Zur Auswertung der beiden Indexanfragen muß der Indexmanager zweimal auf die gleichen Indexeinträge zugreifen.

Die bessere Strategie ist im Fall der hierarchischen Selektion, das Existenzprädikat und das Prädikat in der inneren Anfrage mit einem einzigen Indexzugriff vom Typ I auszuwerten. In dem Beispiel wäre vor Auswertung der äußeren Anfrage der Index entsprechend Beispielindexanfrage 3 zuzugreifen. Das Ergebnis der Indexanfrage enthält genau die Identifier der Wurzel- und Subobjekte, die zur Auswertung der gesamten Anfrage benötigt werden. Die äußere Anfrage wird bei diesem Vorgehen entsprechend dem Index-Scan-Verfahren ausge-

wertet. Nachdem ein Kundenobjekt-Identifier gelesen wurde, wird auf das Kundenobjekt zugegriffen. Die innere Anfrage muß jeweils ausgewertet werden, nachdem ein Kundenobjekt gelesen wurde. Hierbei wird entsprechend dem Prinzip "Indexauswertung mit Indexzwischenergebnis" vorgegangen. Es ist jedoch nicht nötig, assoziativ auf die Indexergebnisrelation zuzugreifen, da durch das Lesen des Kunden-Identifiers bereits bekannt ist, welche Auftrags-Identifier in der Indexergebnisrelation zu verwenden sind. An dieser Stelle wird ein weiterer Vorteil dieses Vorgehens deutlich. Da auf die Indexergebnisrelation nicht assoziativ zugegriffen wird, braucht diese, sofern es das auszuwertende Prädikat erlaubt, nicht materialisiert zu werden.

6. Zusammenfassung und Ausblick

Sollen objekt-orientierte Datenbanksysteme in der Praxis akzeptiert werden, müssen komplexe Anfragen, die beliebige Teile komplexer Objekte selektieren, formuliert und dann vom System automatisch transformiert und effizient ausgewertet werden können. Dazu wurde in diesem Beitrag ein abgestimmtes Zusammenspiel zwischen dem Indexmanager, der Pfadindexe verwaltet, und geeigneten Auswertungsstrategien, die diesen Indexmanager effizient nutzen, vorgestellt.

In dem vorgestellten Konzept spielt die Funktionalität des Indexmanagers eine wesentlich größere Rolle als in relationalen Systemen. Der Indexmanager sucht nicht nur Indexeinträge, die einem einfachen Prädikat genügen, sondern wertet auch Adreßprädikate aus und stellt das Resultat einer Indexanfrage in einer situationsabhängigen, internen NF^2-Repräsentation dar, die bei Bedarf neben dem sequentiellen auch den assoziativen Zugriff auf Indexergebnisse erlaubt.

Darauf aufbauend wurde dann gezeigt, wie komplexe Anfragen unter Verwendung des Indexmanagers effizient ausgewertet werden können. Das Prinzip ist, komplexe Anfragen in die wesentlich einfacher strukturierten unabhängigen und abhängigen Teilanfragen zu zerlegen und für diese jeweils separat Auswertungsstrategien auszuwählen. So ist sichergestellt, daß bei der Anfragetransformation automatisch erkannt werden kann, welche der diskutierten Strategien in den einzelnen Fällen eingesetzt werden können. Dabei konnten zur Auswertung unabhängiger Anfragen Strategien aus relationalen Systemen übernommen werden. Zur Auswertung von Existenzprädikaten und abhängigen Anfragen wurden neue Strategien wie "Indexzugriff mit Adreßprädikat" und "Indexauswertung mit Indexzwischenergebnis" vorgestellt. Das Ergebnis dieses Gesamtkonzepts ist, daß auch für komplexe Anfragen Auswertungspläne, die Indexe verwenden, vom System automatisch erzeugt werden können.

Ziel der weiteren Arbeiten ist die Entwicklung geeigneter Kostenformeln in Abhängigkeit der gewählten Speicherungsstrukturen für Objekte und Indexe für die Bestimmung der optimalen Ausführungsstrategie im Kontext allgemeiner, objekt-orientierter Datenmodelle.

Danksagung

Wichtige Vorarbeiten für diesen Beitrag entstanden während der Mitarbeit von U. Keßler als Gastwissenschaftler im AIM-Projekt des Wissenschaftlichen Zentrums Heidelberg der IBM Deutschland GmbH. In diesem Zusammenhang sei allen Mitarbeitern dieser Abteilung für die gute Zusammenarbeit und die vielen Anregungen gedankt. Ein ganz besonderer Dank gilt Herrn K. Küspert für die vielen fruchtbaren Diskussionen und guten Ideen, die halfen, das beschriebene Konzept zu entwickeln.

Literatur

[Banc88] F. Bancilhon, G. Barbedette, V. Benzaken, C. Delobel, S. Gamerman, C. Lécluse, P. Pfeffer, P. Richard, F. Velez: *The Design and Implementation of O2, an Object-Oriented Database System.* Proc. of 2nd Int. Workshop on Object-Oriented Database Systems, Bad Münster, September, in: Advances in Object-Oriented Database Systems, (K. R. Dittrich, Ed.), Springer-Verlag, Lecture Notes in Computer Science 334, pp. 1-22, 1988.

[BeKi89] E. Bertino, W. Kim: *Indexing Techniques for Queries on Nested Objects.* IEEE Transactions on Knowledge and Data Engineering, June, Vol. 1, No. 2, pp. 196-214, 1989.

[Dada86] P. Dadam, K. Küspert, F. Andersen, H. Blanken, R. Erbe, J. Günauer, V. Lum, P. Pistor, G. Walch: *A DBMS Prototype to Support Extended NF^2-Relations: An Integrated View on Flat Tables and Hierarchies.* ACM SIGMOD Proc. of Int. Conference on Management of Data, Washington, May 28 - 30, pp. 356-367, 1986.

[JaKo84] M. Jarke, J. Koch: *Query Optimization in Database Systems.* ACM Computing Surveys, Vol. 16, No. 2, June, pp. 111-152, 1984.

[KeMo90] A. Kemper, G. Moerkotte: *Access Support in Object Bases.* ACM SIGMOD Proc. of Int. Conference on Management of Data, Atlantic City, May 23 - 25, pp. 364-374, 1990.

[KeMo90a] A. Kemper, G. Moerkotte: *Advanced Query Processing in Object Bases Using Access Support Relations.* Manuskript Universität Karlsruhe.

[Kim89] W. Kim, N. Ballou, H.-T. Chou, J.F. Garza, D. Woelk: *Features of the Orion Object-Oriented Database.* Object-Oriented Concepts, Databases, and Applications, Addison-Wesley Publishing Company, (W. Kim, F. H. Lochovsky, Eds.), pp. 251-282, 1989.

[MaSt86] D. Maier, J. Stein: *Indexing in an Object-Oriented DBMS.* Proc. of Int. Workshop on Object-Oriented Database Systems, (K. Dittrich, U. Dayal, Eds.), California, September, pp. 171-182, 1986.

[MaSt87] D. Maier, J. Stein: *Development and Implementation of an Object-Oriented DBMS.* Research Directions in Object-Oriented Programming, MIT Press, (B. Shriver, P. Wegner, Eds.), pp. 355-392, 1987.

[PiTr86] P. Pistor, R. Traunmüller: *A Database Language for Sets, Lists and Tables.* Information Systems, Vol. 11, No. 4, pp. 323-336, 1986.

[RoKS88] M.A. Roth, H.F. Korth, A. Silberschatz: *Extended Algebra and Calculus for Nested Relational Databases.* ACM Transactions on Database Systems, Vol. 13, No. 4, pp. 389-417, 1988.

[ScSc83] H.-J. Schek, M. Scholl: *Die NF^2-Relationenalgebra zur einheitlichen Manipulation externer, konzeptueller und interner Datenstrukturen.* Sprachen für Datenbanken, (J. W. Schmidt, ed.), Springer-Verlag, Informatik-Fachberichte 72, pp. 113-133, 1983.

[ScSc86] H.-J. Schek, M. H. Scholl: *The Relational Model with Relation-Valued Attributes.* Information Systems, Vol. 11, No. 2, pp. 137-147, 1986.

[ScWe87] H.-J. Schek, G. Weikum: *DASDBS Konzepte und Architektur eines neuartigen Datenbanksystems.* Informatik Forschung und Entwicklung, (A. Endres, Ed.), Springer Verlag, No. 2, pp. 105-121, 1987.

[Seli79] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, T.G. Price: *Access Path Selection in a Relational Database Management System.* ACM SIGMOD Proc. of Int. Conference on Management of Data, Boston, May 30 - June 1, pp. 23-34, 1979.

[Vald87] P. Valduriez: *Join Indices.* ACM Transactions on Database Systems, Vol. 12, No. 2, pp. 218-246, 1987.