Ersch. in: HMD : Praxis der Wirtschaftsinformatik ; 27 (1990), 152. - S. 103-115

Evolution von Datenmodellen

Relational . . . geschachtelt (NF^2) relational . . . objektorientiert?

Marc H. Scholl und Hans-Jörg Schek

In diesem Aufsatz beschreiben wir das Modell der geschachtelten (NF^2 = Non-First-Normal-Form) Relationen; was verstehen wir darunter, wie sind sie aus dem bekannten relationalen Datenmodell entstanden, und wie passen sie in die Diskussion um objektorientierte Datenbanksysteme? Wir verstehen NF^2-Relationen als eine konsequente Weiterentwicklung der „relationalen Idee" — der Konstruktor Relation wird wiederholt angewendet — hin zu komplexeren Datenstrukturen. Der Vorteil des Relationenmodells, nämlich seine mächtige Anfragesprache (SQL) kann dabei beibehalten werden. Zwar gestatten geschachtelte Relationen die direkte Modellierung von hierarchischen Beziehungen, für allgemeinere Beziehungstypen sind sie jedoch immer noch zu eingeschränkt, hierzu benötigt man Referenzen (Surrogates, IDs). Ein weiterer Schritt der Verallgemeinerung („rekursiv geschachtelte Relationen") beseitigt diese Restriktion noch gründlicher. Dieser letzte Schritt ist zugleich auch der Übergang von einem *Daten*modell zu einem *Objekt*modell, schlägt also die Brücke zwischen relationalen Datenbanken und objektorientierten bzw. semantischen Datenmodellen.

Inhaltsübersicht

Was sind relationale Datenbanken?
- Das Relationenmodell
- Grenzen des Relationenmodells

Geschachtelte Relationen
- Hierarchisch geschachtelte Tabellen
- Operationen auf geschachtelten Relationen
- Grenzen des geschachtelten Relationenmodells

Weiterentwicklung in Richtung Objektorientierung
- Semantische und objektorientierte Modelle
- Rekursiv geschachtelte Relationen

Geschachtelte Relationen in DASDBS

Abschlußbemerkungen

Literatur

Was sind relationale Datenbanken?

Das Relationenmodell

Heutige State-of-the-Art-Datenbanksysteme basieren auf dem Relationenmodell [2]. Dieses sehr einfache Datenmodell baut auf einer verständlichen, sauberen mathematischen Grundlage auf. *Relationen* (in SQL heißen sie „tables") können als mathematische Relationen, Teilmengen von kartesischen Produkten, aufgefaßt werden. Man geht von einer gewissen Menge von Grunddatentypen (atomaren *„Domains"*) aus und bildet Mengen von *Tupeln* („rows" in SQL), deren Komponenten aus solchen Domains stammen. Die Reihenfolge der Tupelkomponenten spielt im Gegensatz zu mathematischen Relationen keine Rolle, deshalb vergibt man Namen für sie, die *Attribute* („columns"). Unter einem *Relationenschema* versteht man dann den beschreibenden Teil der Rela-

Konstanzer Online-Publikations-System (KOPS)
URN: http://nbn-resolving.de/urn:nbn:de:bsz:352-230615

tion, also ihren Namen und die Namen der Attribute zusammen mit den ihnen zugeordneten Domains (all das, was sich in einer SQL-„create-table"-Anweisung findet). Oft stellt man ein Schema in der Form *Pers(pnr,name,adr,anr)* dar, wenn die Domains nicht interessieren. Die *Ausprägung* einer Relation, also ihren Wert zu einem bestimmten Zeitpunkt, stellt man in Form einer Tabelle dar, in der jede Spalte einem Attribut und jede Zeile einem Tupel entspricht, die Kopfzeile enthält die Attributnamen. Die einfache Personalrelation mit den Attributen Personalnummer, Name, Adresse und Abteilungsnummer könnte also so aussehen:

Pers	pnr	name	adr	anr
	8888	Xaver	M	1
	4711	Jupp	K	3
	2222	Hein	HH	2

Die *erste Normalform* besagt nichts weiter, als daß (für das Datenbanksystem) jeder Attributwert (also jeder Eintrag in der Tabelle) *atomar* ist, das heißt, es sind zum Beispiel keine Wiederholungsgruppen, zusammengesetzte Felder o. ä. erlaubt. Der Hintergrund dieser Forderung besteht darin, daß man nur *eine einheitliche* Art der Datenstrukturierung haben möchte, nämlich die der Relation. Andernfalls müßte das System auch noch entsprechende Operationen auf solchen konstruierten Datentypen unterstützen. Der wesentliche Vorteil der einfachen Datenstruktur „Relation" liegt aber gerade in der Tatsache, daß sehr einfache, aber dennoch recht mächtige, weil mengenorientierte und orthogonal kombinierbare, Operationen zur Verfügung stehen. So kann etwa die „relationale Vollständigkeit" einer Anfragesprache bereits durch fünf unabhängige Operationen einer „Relationenalgebra" (Vereinigung, Differenz, Produkt, Selektion und Projektion) erzielt werden [2]. Durch die Kombination mehrerer solcher Operationen lassen sich komplexe Anfragen formulieren.

Wir wollen hier nicht eine algebraische Schreibweise für relationale Operationen verwenden, obgleich gerade diese Art typisch für die „Exaktheit" des Relationenmodells wäre. Im Interesse der Lesbarkeit verwenden wir statt dessen eine SQL-artige Notation. Die Sprache SQL ist relational vollständig, d. h., sie kann alle Algebraoperationen nachbilden, geht sogar darüber noch hinaus (z. B. durch Aggregationsoperationen wie Summe, Durchschnitt oder durch Sortierung, Gruppierung usw.). Ein Beispiel für eine SQL-Anfrage an die obige Relation und eine weitere *Abt (anr, aname, aort),* die die Abteilungen beschreibt, ist:

select *aname, name, adr*
from *Abt, Pers*
where *Abt. anr = Pers.anr and*
aort = 'M'

Hier wird nach den Namen aller Abteilungen aus München zusammen mit den Namen und Anschriften ihrer Mitarbeiter gefragt. Die Bedingung *„Abt. anr = Pers. anr"* formuliert (zusammen mit den beiden Relationennamen in der from-Klausel) den *Join,* d. h. die Verknüpfung der Tupel aus den beiden Relationen. *aort = 'M'* stellt eine Selektionsbedingung dar, und die *select*-Klausel beschreibt die Ergebnisrelation (eine Projektion aus dem Joinergebnis).

Ein Charakteristikum vieler relationaler Anfragen ist, daß die gewünschten Informationen aus mehreren Tabellen „zusammengesucht" werden müssen; es werden also häufig mehrere Joins erforderlich. Dies ist um so mehr der Fall, je komplexer die in der relationalen Datenbank beschriebenen Zusammenhänge aus der Anwendung sind. Maßgeblich hierfür ist die Einhaltung von höheren Normalformen (z. B. dritte Normalform), ein Ziel des logischen Datenbankentwurfs zur Vermeidung von Redundanzen und damit verbundenen Anomalien bei Änderungsoperationen. Dagegen wäre unsere Beispielanfrage einfacher zu formulieren, wenn die Daten nicht in zwei, sondern nur in einer einzigen Relation zusammengefaßt wären. Diese Relation hätte dann das Schema *AbtPers(anr, aname, aort, pnr, name, adr).* Es fällt auf, daß nunmehr die eine Abteilung beschreibende Information *(aname, aort)* wiederholt — mit jedem ihrer Angestellten — abgespeichert wäre. Folglich müßte etwa bei Namensänderung einer Abteilung nicht nur ein Tupel, sondern entsprechend viele geändert werden.

Grenzen des Relationenmodells

Die Zerlegung in einzelne Relationen, die jeweils nur einen einzigen Entity- oder Objekttyp beschreiben, ist durchaus ein erstrebenswertes Ziel. Es gibt jedoch auch Beispiele, bei denen ein Objekt etwa durch eine Menge von verschiedenen Werten eines „Attributs" beschrieben werden soll (Hobbies von Personen, Autoren oder Deskriptoren von Dokumenten usw.). Werden solche Sachverhalte in Relationen dargestellt, so wird „zuviel zerlegt", d. h ., ein Objekt muß in mehr als einem Tupel (in verschiedenen Relationen) repräsentiert werden; im Dokumenten-Beispiel etwa in den Relationen *Dokument (dnr, titel, . . .), Autor (dnr, name), Deskriptor (dnr, keyword).* Hierin drückt sich einfach die Tatsache aus, daß im Relationenmodell keine Beziehungen direkt unterstützt werden. Diese Beobachtung auf der Datenmodellebene war ein Ausgangspunkt unserer Untersuchungen über geeignete Erweiterungen des Relationenmodells [18].

Eine zweite Motivation bestand schlicht in der Beobachtung, daß die häufige Berechnung von Joins in Datenbankanfragen eine aufwendige Operation und einen wesentlichen Anteil an den gesamten Ausführungskosten darstellt. Neben dem Datenmodell-Aspekt auf der logischen Ebene kommt also noch der Performance-Aspekt der internen Ebene hinzu. Die Darstellung in Form von NF^2-Relationen benötigt weniger Relationen, viele der häufigsten Joins sind bereits „materialisiert".

Eine andere Limitierung der Darstellungsmöglichkeiten des Relationenmodells ist die Tatsache, daß lediglich eine sehr kleine — üblicherweise fixe — Menge von Basisdatentypen (verschiedene Zahlen und Strings) zur Verfügung stehen. In nichtkonventionellen Anwendungsbereichen, etwa Büro, Geodaten, alle Arten von Entwurfsumgebungen (CAx), gibt es jedoch Objekte, die durch ganz andere „Basisdatentypen" beschrieben werden sollen (geometrische Beschreibungen, textuelle, Graphik, . . .). Wie das Datenbanksystem auf den eingebauten Grunddatentypen bestimmte Operationen (z. B. Vergleiche) ausführen kann, so soll das auch bei solchen anwendungsspezifischen Datentypen möglich sein. Hierzu benötigt man ein *erweiterbares Datenbanksystem,* das es gestattet, neue Datentypen mit darauf arbeitenden Operationen (im Sinne von ADTs) zu definieren und fortan wie atomare Basisdatentypen zu verwenden. Die Erweiterbarkeit ist ein zweites wichtiges Anliegen, auf das wir hier allerdings nicht im gleichen Maße eingehen; die Weiterentwicklung von Relationen zum NF^2-Modell ist orthogonal zur Erweiterbarkeit. Es wurde das Konzept der extern definierten Datentypen (EDTs) entwickelt [14, 25], das die Verwendung von EDT-Operationen in den tieferen Schichten des Datenbanksystems erlaubt.

Geschachtelte Relationen

Hierarchisch geschachtelte Tabellen

Die Idee der NF²-Relationen ist einfach: Wir erlauben die Verwendung des Typkonstruktors „Relation" mehrfach. Das bedeutet, ausgehend von einer Zuordnung von Basistypen zu Attributnamen darf nun der Konstruktor Relation nicht nur genau einmal, sondern (hierarchisch geschachtelt) mehrmals angewendet werden. Ein Attributwert kann also, anders ausgedrückt, wiederum eine ganze Relation sein. Wir erhalten geschachtelte Tabellen: Tabellen in Tabellen in Tabellen . . . Wesentlich ist jedoch, daß wir weiterhin an einem fixen Relationenschema festhalten, d. h., die Schachtelungstiefe ist ausprägungsunabhängig im Schema der Relation festgelegt. Somit können wir mengenwertige „Attribute" tatsächlich als solche darstellen, auch die Modellierung von hierarchischen („1:n")-Beziehungen ist unmittelbar möglich, z. B. kann das Personal direkt bei der Abteilung oder können sowohl Autoren als auch Deskriptoren direkt beim Dokument gespeichert werden: *Abt (anr, aname, aort, Pers (pnr, name, adr))* bzw. *Dokument (dnr, Titel, . . ., Autor (name), Deskriptor (keyword))*. Auf der Ausprägungsebene erhalten wir eine entsprechend geschachtelte Tabellendarstellung.

Solche hierarchischen Darstellungen sind im übrigen gar nicht so unbekannt: viele „Forms"-basierten Benutzerschnittstellen relationaler Datenbanksysteme bieten Vergleichbares zur Aufbereitung von Ergebnissen an. Wir sprechen von relationenwertigen Attributen oder *Subrelationen* bzw. von darin enthaltenen *Subtupeln*. Jedes Tupel *t* der NF²-Relation *Abt* hat eine Subrelation *t(Pers)*, die aus mehreren Subtupeln besteht. Bei tieferer Schachtelung kann es sinnvoll sein, auch von Subsubtupeln usw. zu sprechen. Mit Wurzeltupel meinen wir das äußere Tupel, mit Supertupel das nächstübergeordnete.

Auf der Datenmodellebene erlauben also NF²-Relationen die direkte Modellierung komplexerer Strukturen, als das mit gewöhnlichen (flachen) Relationen möglich ist. Wir können also dem abstrakten Entwurfsziel, ein Objekt der realen Welt durch genau ein Tupel darzustellen, deutlich näher kommen. Zum Entwurf mittels NF²-Relationen gibt es eine Reihe von Untersuchungen über Normalformen, auf die wir nicht näher eingehen wollen, z. B. [7, 12].

Operationen auf geschachtelten Relationen

Mit der wiederholten Verwendung von Relationen in einer hierarchischen Schachtelung haben wir kein essentiell neues Konzept in das Modell eingeführt, sondern lediglich ein bereits vorhandenes in allgemeinerer Art verwendet. Dies sollte sich auch bei den Operationen auf NF²-Relationen widerspiegeln. Wenn wir bei der Datenstruktur der Relation anstelle von Attributen nun auch wieder Relationen vorfinden können, so erlauben wir entsprechend in SQL (oder Relationenalgebra) die Verwendung von relationalen Anfragen überall dort, wo in Anfragen Attribute vorkommen dürfen (in SQL also in der *select*- und der *where*-Klausel, in der Algebra in Selektionsbedingungen und in Projektionslisten). Auch hier kann man wegen der präziseren, kompakteren Schreibweise eine Algebra-Notation verwenden [19]. Wir lehnen uns hier aber wiederum an Arbeiten über SQL-Erweiterungen für NF²-Relationen an [10, 11, 13]. Wir benötigen also eine geschachtelte Anfragesprache, die an verschiedenen Stellen „nested subqueries" enthalten darf (diese Art der Schachtelung ist von den subqueries in Standard-SQL, die eine alternative Art der Formulierung von Joins darstellen, zu unterscheiden).

Bei den nun folgenden Beispielen beziehen wir uns immer auf folgende zwei Relationen:

Abt (anr, aname, aort,
Pers (pnr, name, adr, Kurse (knr, jahr), Kinder (kname, gebjahr))),

Kursinfo (knr, titel, referent).

Eine entsprechende (flache) relationale Datenbank bestünde aus fünf Relationen *Abt, Pers, Kurse, Kinder, Kursinfo*. Die den Subrelationen entsprechenden flachen Relationen wären um die Schlüsselattribute ihrer jeweils übergeordneten Relation erweitert, um die Verknüpfung darzustellen (z. B. *Pers* um *anr, Kurse* um *pnr* usw.).

Zunächst sei bemerkt, daß jede relationale Anfragesprache unverändert auf NF²-Relationen anwendbar ist, allerdings kann dann die Anfrage immer nur die äußere Hierarchieebene ansprechen, die Subrelationen werden wie „long fields" behandelt, d. h., ihre interne Struktur kann nicht ausgenutzt werden. Damit dieses möglich wird, brauchen wir die geschachtelten Operationen. Zum Beispiel können wir eine Auflistung von Abteilungsorten mit den Anschriften der Mitarbeiter bekommen, indem wir eine „geschachtelte Projektion" durchführen. Wir wollen von der Relation *Abt* nur das (atomare) *aort*-Attribut sowie etwas von der *Pers*-Subrelation behalten. Dieses „Etwas" spezifizieren wir wiederum als Projektion auf der Subrelation:

select aort, (select adr from Pers)
from Abt.

Zur Verdeutlichung der Semantik der Schachtelung kann man explizite Tupelvariablen benutzen:

select a.aort, (select p.adr from p in a.Pers)
from a in Abt.

Das Ergebnis dieser Anfrage ist eine NF²-Relation mit dem Schema *Abt' (aort, Pers'_(adr))*. Man beachte, daß eine Gruppierung der Mitarbeiter nach Abteilungen bereits auch in Standard-SQL möglich ist (mit „group by"), jedoch ist das Ergebnis nicht in einer Relation darstellbar (Abgeschlossenheit), weshalb anschließend in der *select*-Klausel nur noch Attribute von Abteilungen oder Aggregatfunktionen auf Personalattributen aufgeführt werden dürfen. Ferner sei angemerkt, daß Ausdrücke — wenn auch nur arithmetische, da nur skalare Attributdomains zugelassen sind — ebenfalls bereits in Standard-SQL in der *select*-Klausel vorkommen dürfen (z. B. *„select name, gehalt * 13, . . . from . . ."*).

Natürlich können auch andere Operationen in die *select*-Klausel geschachtelt werden als nur reine Projektionen. Wenn wir etwa für jede Abteilung (deren Name sowie) eine Liste derjenigen Mitarbeiter (wiederum deren Namen) erstellen wollen, die keine Kinder haben, so brauchen wir eine Selektion. Außerdem zeigt dieses Beispiel die Notwendigkeit für Mengenvergleichsoperationen:

```
select  a.aname, ( select p.name
                   from p in a.Pers
                   where p.Kinder <> {} )
from    a in Abt.
```

Dabei bezeichne „<>" die Ungleichheit und „{}" die leere Menge.

Die Schachtelung in der *where*-Klausel zeigt das folgende Beispiel, bei dem wir solche Abteilungen suchen, in denen (mindestens) ein Mitarbeiter mit Namen Fritzchen arbeitet:[1])

```
select   a.name, a.Pers
from     a in Abt
where    {} <>( select *
                from p in a.Pers
                where p.name = 'Fritzchen' ).
```

Im Unterschied zu Selektionen in der *select*-Klausel, bei denen nur eine Teilmenge der Subtupel im Ergebnis enthalten ist, dienen Selektionen in der *where*-Klausel lediglich als Filterprädikat für die übergeordneten Tupel. Das bedeutet, daß bei der letzten Anfrage die Subrelation *Pers* der Ergebnisstupel jeweils *alle* Mitarbeiter der ausgewählten Abteilungen enthält, nicht nur Fritzchens. Wollen wir nur Abteilungen mit Fritzchens und von diesen auch nur die Fritzchens sehen, so ist die Anfrage umzuformulieren. In der SQL-artigen Schreibweise muß dabei der geschachtelte Unterausdruck zweifach verwendet werden (in der NF²-Algebra ist dies nicht nötig), daher verwenden wir eine separate Vereinbarung (siehe auch [11]):

```
let FP = ( select * form p in a.Pers
           where p.name = 'Fritzchen' )
in {
     select   a.name, FP
     from     a in Abt
     where    {} <> FP
   }.
```

Ein letztes Beispiel verknüpft die Subrelation *Kurse* der zweiten Schachtelungstiefe mit der Relation *Kursinfo;* wir erhalten eine NF²-Relation *Abt' (aname, Pers'_(name, Kurse' (titel)))*, bei der zu jedem Abteilungsmitarbeiter die Titel der besuchten Kurse aufgeführt sind. Wir führen also auf einer tieferen Ebene einen Join durch:

```
let  Kurse' = ( select ki.titel
                from k in p.Kurse, ki in Kursinfo
                where k.knr = ki.knr )
in   {
       let Pers' = ( select p.name, Kurse'
                     from p in a.Pers )
       in {
            select a.name, Pers'
            from a in Abt
          }
     }.
```

Diese Anfrage zeigt auch die Schachtelungsmöglichkeiten über mehrere Ebenen hinweg. Deutlich wird ebenfalls, daß solche Anfragen keineswegs immer einfach zu formulieren oder zu verstehen sind. Hier lohnt es sich durchaus, Überlegungen zur Unterstützung des Benutzers, etwa durch graphische Oberflächen, anzustellen.

1) Vgl. „where exists . . ." in Standard-SQL! Hier ist allerdings der Join implizit, d. h., die Verknüpfungsbedingung taucht nicht explizit auf.

Wir wollen die vielfältigen Möglichkeiten der NF²-Anfragesprachen nicht weiter vertiefen, wichtig ist jedoch festzuhalten, daß wir mit relationalen Operationen auch auf Subrelationenebene in gleicher Art operieren können, wie wir das von (flachen) relationalen Sprachen gewohnt sind. In der lokalen Sicht eines (Sub-)Tupels *(Pers* im Beispiel) stellt jede Subrelation *(Kurse)* eine Relation im gleichen Sinne dar wie andere äußere Relationen auch *(Kursinfo)*.

Grenzen des geschachtelten Relationenmodells

Selbstverständlich besteht keine Anwendung aus rein hierarchischen Beziehungen zwischen ihren Objekttypen. Die Frage ist also, inwieweit die hierarchischen NF²-Relationen zur Darstellung geeignet sind. Hierbei sollten zwei Ebenen unterschieden werden: logisches Datenmodell und interne Speicherungsstrukturen [20]. Zunächst zur internen Ebene: Hier geht es letztlich um das Layout der Datenelemente auf dem (blockstrukturierten) linearen Adreßraum der Platte. Solange keine Redundanz (mindestens in Form von Verweisen) eingeführt wird, lassen sich aber höchstens hierarchische Strukturen linearisieren. Daher ist für diesen Zweck das NF²-Modell allgemein genug. In [24] haben wir ausführlich diskutiert, wie mit dem geschachtelten Relationenmodell alle bekannten (schemaorientierten) Clusterungsstrategien auch für n:m-Beziehungen beschrieben werden können (insbesondere auch die verschiedenen Alternativen der CODASYL-SSL).

Auf der Ebene des logischen Datenmodells sieht es anders aus. Hierfür haben wir das NF²-Modell gar nicht propagiert — wenngleich erste Überlegungen auch aus dieser Richtung kamen [18]. Es wurden bereits diverse Erweiterungen von NF²-Relationen vorgeschlagen, die es eher als logisches Datenmodell geeignet machen. Im AIM-Projekt [3] zum Beispiel wurde ein „erweitertes NF²-Modell" implementiert, das nicht nur geschachtelte Relationen, sondern die orthogonale Konstruktion von Tupeln, Mengen und Listen unterstützt [11]. Dort, wie auch im LauRel-Projekt [4], wurden außerdem Tupel-Identifier eingeführt, so daß *Referenzen* auf Tupel möglich werden. Anders als bei Referenzen mittels Fremdschlüssel werden diese speziellen **„ref"**-Attribute vom System gewartet, die IDs werden automatisch vergeben. Auf diese Art können netzwerkartige Beziehungen modelliert werden. So kann etwa jeder Lieferant eine Subrelation mit Referenzen auf die lieferbaren Teile haben (Attributtyp **„set_of ref_to** *Teile")* und umgekehrt jedes Teil eine Verweismenge zu den Lieferanten. Unterschiede zwischen den beiden Ansätzen finden sich z. B. bei der Dereferenzierung dieser Verweise. In der Sprache HDBL von AIM gibt es eine explizite Operation **„MAT"** (für materialize), in LauRel werden Referenzattribute automatisch dereferenziert, sobald weiter mit ihnen gearbeitet wird.

Handelt es sich also bei der Unterstützung von netzwerkartig verknüpften Strukturen um nichts weiter als die Einführung des altbekannten Pointerkonzeptes in Datenmodelle? Es steht außer Zweifel, daß wir im Kern so etwas wie Pointer, d. h. das Unterscheiden von Kopier- und Referenzsemantik benötigen. Die Frage stellt sich jedoch, auf welchem Abstraktionsniveau das in einem (relationalen) Datenmodell, also an der Benutzerschnittstelle, geschehen sollte. Es war unsere Absicht, möglichst ohne dem Benutzer explizit im Modell bekanntgemachte Pointer und Dereferenzierungsoperation auszukommen. Der Ansatz von LauRel geht mit der automatischen Dereferenzierung ebenfalls in diese Richtung. Können also sog. „Shared Subobjects" ohne explizite Referenzen unterstützt werden? Wie gehen andere Modelle mit dieser und verwandten Fragen um?

Weiterentwicklung in Richtung Objektorientierung

Semantische und objektorientierte Modelle

In der Terminologie objektorientierter Modelle brauchen wir zur Unterstützung von Shared Subobjects die Objektidentität. Der Übergang von der Modellierung konkreter Datenstrukturen zur Beschreibung des Verhaltens (d. h. der Schnittstelle) von Instanzen abstrakter Objekttypen (im Sinne von ADTs) ist ein weiteres Charakteristikum objektorientierter Modelle. Um darüber hinaus die Anforderungen bzgl. im Modell semantisch bekannter Beziehungen zu analysieren, müssen semantische Modelle und Wissensrepräsentationssprachen einbezogen werden [15]. Es erscheint wichtig, ausgewählte Grundkonzepte dieser Modelle, mindestens den Abstraktionsmechanismus der Generalisierung mit Vererbung, in eine Erweiterung mit aufzunehmen. Das Wissensrepräsentationsmodell KL-ONE [1] ist geeignet für eine Synthese mit den NF²-relationalen Konzepten. Eine Reihe weiterer Vorschläge auch aus anderen Bereichen (ERM-Erweiterungen, objektorientierte Programmiersprachen und Datenmodelle, Frames, Komplex-Objekt-Modelle, Funktionale Datenmodelle) unterscheidet sich weniger in wesentlichen Grundkonzepten als vielmehr in der Art der Darstellung oder der Terminologie. Insofern ist ein evolutionärer Weg, ausgehend vom Relationenmodell, ein Versuch, vermeintlich weit auseinanderliegende Richtungen wieder zusammenzuführen.

Rekursiv geschachtelte Relationen

Die Grundidee unseres Ansatzes besteht darin, daß das Schema einer Objektbank in Form eines KL-ONE-Netzes beschrieben wird, also im wesentlichen netzartige Beziehungen zuläßt. Für die Formulierung von Operationen nehmen wir, ausgehend von einem jeweils geeignet auszuwählenden Startpunkt, dynamisch eine spezifische hierarchische Sicht in das Netz ein und arbeiten wie mit NF²-Relationen auf dieser hierarchischen Sicht. Konzeptuell kann man sagen, daß ein Netz viele sich überlappende Hierarchien beinhaltet. Auf diesen arbeiten die Operationen. Das bedeutet, wir behalten den Vorteil der deskriptiven und damit optimierbaren Anfragesprache auch in dieser netzartigen Umgebung bei. (Ein ähnliches Vorgehen findet sich bei sog. High-Level-Sprachen für CODASYL-Netzwerk-Datenbanksysteme.) Die Sprache muß ferner mit ererbten Eigenschaften von Objekten in der Generalisierungshierarchie umgehen können und diese zu manipulieren gestatten. Genaueres zu dem Ansatz kann [5, 6, 21, 22] entnommen werden, wir wollen es hier bei einem Beispiel belassen.

Objekttypen („Generic Concepts") werden beschrieben durch eine Reihe von Funktionen („Generic Roles") und deren Resultattyp („Rollenfüller"), der wiederum ein Objekttyp ist. Funktionen (Rollen) können mengenwertig sein, sie können über sog. Number Restrictions in der Kardinalität (min, max) eingeschränkt werden. Diese Rollen entsprechen in Entity-Relationship-Terminologie den Relationships, allerdings jeweils nur einer Richtung derselben. Die umgekehrte Richtung einer Relationship kann durch eine Rolle in umgekehrter Richtung (d. h. mit einem separaten Namen) als „inverse Rolle" eingeführt werden. Die Generalisierung wird mittels „IS_A"-Kanten zwischen Concepts dargestellt. Neben dieser einfachen Form der Generalisierung werden eine Reihe von Integritätsbedingungen auch über vererbte Rollen unterstützt. (Beispiele: Die Rolle „WOrt" von Personen zu Städten ist invers zu „Bewohner" von Städten zu Personen; Bundesländer haben Landtagsmitglieder — MdLs —, die Politiker sind. Einer von diesen ist spezieller, der Ministerpräsident, nur er darf in der Rolle „Hat_MP" zum Bundesland stehen. „MdL" ist eine ererbte Rolle bei Ministerpräsidenten, „MP" ist eine differenziertere Rolle.) (vgl. Abb. 1)

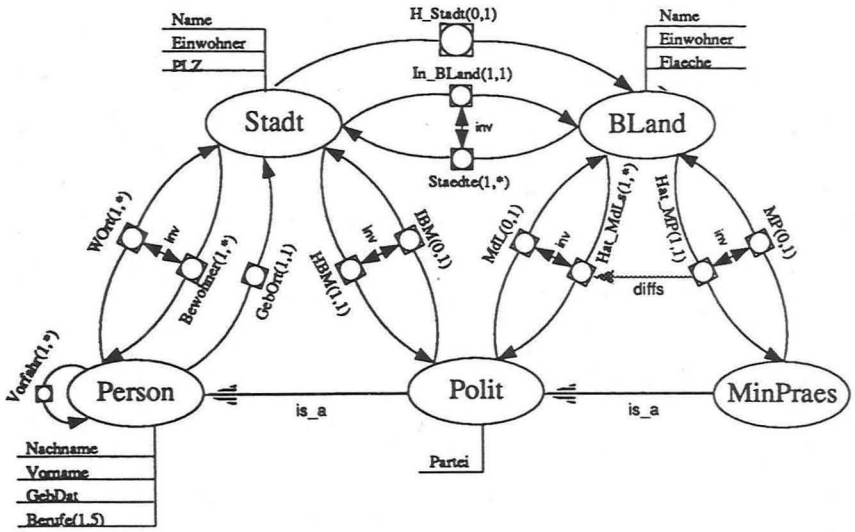

Abb. 1 KL-ONE-Schema

Wir verstehen nun dieses KL-ONE-Schema als eine Menge von wechselseitig rekursiv definierten NF²-Relationen, z. B.

Stadt (Name, Einwohner, PLZ, Bewohner (<eine „Subrelation" vom Typ Person>), . . .)
Person (Nachname, Vorname, GebDat, . . .,
Wohnort (<eine „Subrelation" vom Typ Stadt>)),

Wenn wir diese „Relationenschemata" als Typdefinitionen verstehen wollen, so haben wir es mit rekursiv definierten Typen zu tun. Zur Semantik sei wiederum auf [21, 22] verwiesen.

Wir können nun aber auf jeder dieser rekursiv geschachtelten NF²-Relationen NF²-artige Anfragen stellen. So können wir gleichermaßen bei Städten wie auch bei Personen einsteigen und den jeweils anderen Objekttyp als untergeordnet (d. h. wie eine Subrelation) betrachten: „Gib zu jeder Stadt mit mehr als 200 000 Einwohnern eine Liste aller Bewohner, die vor der Jahrhundertwende geboren wurden."

```
let Oldies = ( select p.vorname, p.nachname
               from p in s.Bewohner
               where p. GebDat < 1.1.1900 )
in {select s.name, Oldies
    from s.in Stadt
    where s. Einwohner > 200 000
   }.
```

Oder ausgehend von Personen: „Gib zu allen Personen mit Nachnamen „Weizsäcker" eine Liste der Wohnorte im PLZ-Bereich 5*** aus" —

```
let Fuenf = ( select s.name
              from s in p.w.ort
              where s.PLZ > 4999 and s.PLZ < 6000 )
in {
     select p.vorname, p.GebDat, Fuenf
     from p in Person
     where p.nachname = 'Weizsäcker'
   }.
```

Was hat der Benutzer durch dieses Modell gegenüber NF²-Relationen gewonnen? Im Gegensatz zu NF²-Relationen muß mit diesem Modell keine Entscheidung getroffen werden, welcher Objekttyp als äußerer und welcher als untergeordnet betrachtet werden soll; beide Sichten sind symmetrisch im Schema enthalten. (Für die Implementierung, d. h. die Auswahl der Speicherungsstrukturen, läßt das logische Schema noch alle Alternativen offen — ebenso wie im Zusammenhang mit n:m-Beziehungen zwischen Relationen der logischen Ebene [24]). Das gestattet die Formulierung von Anfragen in der jeweils am besten geeigneten hierarchischen Sicht. Die Transformation auf die jeweilige interne Speicherungsstruktur leistet das System. Das System wartet außerdem die n:m-Beziehungen bei Änderungen, d. h., Shared Subobjects werden unterstützt. Dafür wollen wir ein Beispiel betrachten. Zunächst ist aber zu klären, wie die Instanzen dieser rekursiv definierten Objekttypen konkret in einem Programm manipuliert werden. Dazu benötigen wir *Variablen* vom Typ dieser Datenbankobjekte. Durch die *Zuweisung* des Werts einer Objektvariablen an eine „Komponente" eines anderen Objekts wird dann eine Beziehung etabliert (Referenzsemantik), so als hätte der Programmierer mit Pointern gearbeitet.

```
var S: Stadt,
    P1, P2: Person;
S := create [name='XDorf', . . .] (Stadt); /* erzeuge eine Stadt */
/* erzeuge zwei neue Personen, beide wohnen in der neuen Stadt */
P1 := create [name='Franz', . . ., WOrt=S ] (Person);
P2 := create [name='Fritz', . . ., WOrt=S ] (Person);
/* ändere die Einwohnerzahl von XDorf */
update [Einwohner = 2] (S);
```

Bei diesem einfachen Beispiel sind mehrere Dinge anzumerken. Erstens sehen wir, daß die *create*-Anweisung als Resultat das neu erzeugte Objekt liefert, daher kann sie als rechte Seite einer Zuweisung verwendet werden. Zweitens kann anschließend eine solche Objektvariable in anderen Operationen weiterverwendet werden, z. B. um eine Beziehung auszudrücken. Drittens wird die zu „WOrt" inverse Beziehung (Rolle) „Bewohner" durch die Änderung bei „WOrt" automatisch mit gewartet, das heißt, beide, Fritz und Franz, sind von XDorf aus über diese Beziehung erreichbar. Viertens sind alle Änderungen an XDorf von beiden Personen aus (über die Beziehung WOrt) sichtbar, d. h., sowohl *„Einwohner (Wort(P1))"* wie auch *„Einwohner (Wort (P1))"* ergäben nach dem *update*-Statement den Wert 2. Städte sind bzgl. der WOrt-Beziehung Shared Subobjects von Personen. Wesentlich für unseren Ansatz ist, daß *mengenorientiert* auf der Objektbank gearbeitet werden kann, so wie das bei Relationen auch möglich ist (*update . . . where . . .* in Standard-SQL). Wir können auf diesen Punkt hier nicht näher eingehen und verweisen auf [21, 22].

Geschachtelte Relationen in DASDBS

An der Technischen Hochschule Darmstadt wurde von Anfang 1983 bis Ende 1988 das DASDBS-Projekt[2]) durchgeführt, das seither an der ETH Zürich fortgesetzt wird. Es handelt sich um die Entwicklung eines neuartigen Datenbanksystems. Die Architektur von DASDBS sieht vor, daß auf einem gemeinsamen Kern, dem Speicherverwalter, eine Reihe von auf bestimmte Anwendungsklassen hin zugeschnittenen „Frontends" aufgesetzt werden. Wir sprechen daher auch von der DASDBS-Familie [8, 16]. Speziell wurden im Rahmen des Projektes solche Frontends untersucht für die Anwendungsklassen Geo-Datenverwaltung [14, 25], Büro-Dokumentverwaltung [9], standardrelationale Datenbanken [24] und objektorientierte Datenbanken [22, 21]. Der Schwerpunkt der Untersuchungen im Geo-Bereich liegt bei der Erweiterbarkeit, im Gebiet Büro-Dokumente bei intelligenten Suchverfahren für Anfragen in gemischtem Datenbank- und Information-Retrieval-Sinne, bei der Objektorientierung auf dem Aspekt der Datenmodell-Evolution und schließlich im Bereich der Standard-(relationalen)-Datenbankschnittstellen auf der Optimierung. Erfahrungen mit der Implementierung von DASDBS sowie erste Meßergebnisse sind in [17] zusammengestellt.

NF²-Relationen werden im DASDBS-Projekt primär nicht als *Datenmodell* im herkömmlichen Sinne, also für den logischen Datenbankentwurf, betrachtet, sondern zur formalen Beschreibung von *internen Datenstrukturen,* als Zielsprache des physischen Datenbankentwurfs, verwendet. Der Vorteil dieses Vorgehens liegt darin, daß so die Operationen der internen Ebene (die ausführbaren Einheiten) in Form von deskriptiven Operationen einer NF²-Algebra beschrieben werden können [19]. Mehr noch, auch die Abbildung der logischen auf die internen Strukturen und somit auch der logischen (Benutzer-)Operationen auf die internen kann in diesem formalen Rahmen beschrieben werden. Dies ermöglicht den Einsatz von algebraischen Optimierungstechniken bei der Abbildung und Optimierung der Operationen [23]. Eine besonders wichtige Technik des physischen Datenbankentwurfs dient der Beschleunigung von Joins, die zu den teuersten Operationen zählen und von daher großen Einfluß auf die gesamte Effizienz haben: Anstelle der beiden flachen Relationen *Abt* und *Pers* der logischen Ebene können wir zum Beispiel (ohne Einführung von Redundanz) intern die NF²-Relation *Abt(. . ., Pers(. . .))* speichern. Dies entspricht einer relationen-übergreifenden Clusterung; zu jedem Abteilungstupel finden wir in unmittelbarer Nachbarschaft die zugehörigen Personaltupel. Diese Art der Speicherung bedeutet also deutlich verringerte Kosten für Join-Anfragen. (Die *Berechnung* des Join entfällt, das Ergebnis muß lediglich noch *gelesen* werden.)

Abschlußbemerkungen

Wir haben versucht aufzuzeigen, wie ausgehend vom einfachen Relationenmodell durch systematische Erweiterung um wenige wichtige Grundbausteine zunächst NF²-Relationen (hierarchisch geschachtelte Relationen) mit zugehörigen Operationen abgeleitet werden können. Dieses Modell ist bereits erfolgreich implementiert worden [3, 17] und wird in verschiedenen Anwendungsgebieten intensiv evaluiert. Das NF²-Modell hat sich zur Beschreibung der internen Ebene eines Datenbanksystems als hervorragend geeignet erwiesen. Um auch auf der logischen Datenmodellebene einen wesentlichen Schritt gegenüber flachen Relationen voranzuschreiten, ist es nötig, all-

2) DASDBS=DArmStädter DatenBankSystem

gemeinere, nichthierarchische Beziehungstypen zu unterstützen. Diese Forderung führt auf identifizierbare Objekte hin und damit weiter verallgemeinert in den Bereich der objektorientierten Datenmodelle. Im Gegensatz zu manchem Verfechter solcher objektorientierter Modelle sind wir jedoch der Auffassung, daß sich Objektorientierung sehr wohl mit den Konzepten des relationalen Datenmodells, insbesondere des geschachtelten, in Einklang bringen läßt.

Die Untersuchungen zu einem „relationalen Objektmodell" wurden im Rahmen des DFG-Schwerpunktprogramms „Objektbanken für Experten" von 1988–1989 gefördert und werden seither in Zürich im Projekt COCOON[3]) fortgesetzt. Eine erste Prototyp-Implementierung auf der Basis eines relationalen DBMS ist fertiggestellt, sie wird zu Vergleichszwecken mit der DASDBS-basierten Realisierung, die gerade begonnen hat, herangezogen werden. Schwerpunkte der weiteren Arbeiten in diesem Zusammenhang sind die Anfrageoptimierung, der Vergleich verschiedener Realisierungen auf der Ebene der Speicherungsstrukturen sowie die Wahrung der Integrität bei Änderungsoperationen.

Literatur

[1] *Brachman, R. J.; Schmolze, J. G.:* An overview of the KL-ONE knowledge representation system. Cognitive Science, 9, S. 171–216, 1985.

[2] *Codd, E. F.:* A relational model for large shared data banks. Commun. ACM, 13(6), S. 377–387, Juni 1970.

[3] *Dadam, P.; Küspert, K.; Andersen, F.; Blanken, H.; Erbe, R.; Günauer, J.; Lum, V.; Pistor, P.; Walch, G.: A DBMS prototype to support extended NF² relations: An integrated view on flat tables and hierarchies. In: Proc. ACM SIGMOD Conf. on Management of Data,* S. 356–366, Washington, 1986, ACM, New York.

[4] *Larson, P.-A.:* The data model and query language of LauRel. IEEE Database Engineering Bulletin, 11(3), S. 23–30, Sept. 1988, Special Issue on Nested Relations.

[5] *Lausen, G.; Schek, H.-J.:* Semantic specification of complex objects. In: Proc. IEEE CS Symp. on Office Automation, Gaithersburg, 1987.

[6] *Lorie, R.; Schek, H.-J.:* On dynamically defined complex objects and SQL. In: K. Dittrich (Hrsg.) Proc. 2nd Int. Workshop on Object-Oriented Database Systems, Bad Münster, Sept. 1988, Springer LNCS 334.

[7] *Ozsoyoglu, Z. M.; Yuan, L. Y.:* A normal form for nested relations. In: Proc. ACM SIGACT/SIGMOD Symp. on Principles of Database Systems, S. 251–260, Portland, März 1985, ACM, New York.

[8] *Paul, H.-B.; Schek, H.-J.; Scholl, M. H.; Weikum, G.; Deppisch, U.:* Architecture and implementation of the Darmstadt database kernel system. In: Proc. ACM SIGMOD Conf. on Management of Data, San Francisco, 1987, ACM, New York.

[9] *Paul, H. B.; Söder, A.; Schek, H.-J.; Weikum, G.:* Supporting the Office Filing Service by a database kernel system. In: H.-J. Schek, (Hrsg.) Proc. GI Conf. on Database Systems for Office, Engineering, and Scientific Applications, S. 196–211, Darmstadt, 1987, Springer (in German).

[10] *Pistor, P.; Andersen, F.:* Designing a generalized NF² model with an SQL-type language interface. In: Proc. Int. Conf. on Very Large Databases, S. 278–285, Kyoto, Aug. 1986.

[11] *Pistor, P.; Traunmüller, R.:* A data base language for sets, lists, and tables. Information Systems, 11(4), S. 323–336, Dezember 1986.

3) COCOON = rekursives Akronym für „COcoon . . . Complex-Object-Orientation based on Nested relations". Neben der Synthese aus komplexen Objekten und Objektorientierung verdeutlicht die Rekursion, wie dies basierend auf NF²-Relationen erreicht wird.

[12] *Roth, M. A.; Korth, H. F.:* The design of NON–1 NF relational databases into nested normal form. In: Proc. ACM SIGMOD Conf. on Management of Data, S. 143–159, San Francisco, Mai 1987, ACM, New York.

[13] *Roth, M. A.; Korth, H. F.; Batory, D. S:* SQL/NF: A query language for NON–1 NF relational databases. Information Systems, 12(1), S. 99–114, März 1987.

[14] *Schek, H.-J.:* Database systems for the management of geometric objects. In: Proc. 16. GI-Jahrestagung, Berlin, 1986, Springer.

[15] *Schek, H.-J.:* Frames, KL-ONE Concepts and NF² relations: A comparison. Manuscript GI Workshop on Databases and Expertsystems. Dortmund, Mai 1986.

[16] *Schek, H.-J.:* A database kernel system supporting application-specific layers — architecture of the DASDBS family. *Informationstechnik* **it**, 29(3), S. 153-164, März 1987.

[17] *Schek, H.-J.; Paul, H.-B.; Scholl, M.; Weikum, G.:* The DASDBS project: Objectives, experiences and future prospects. *IEEE Trans. on Knowledge and Data Engineering,* 2(1), März 1990. Special Issue on Prototype Systems. Also available as Tech.Rep. Nr. 118 from ETH Zürich, Dept. of Computer Science.

[18] *Schek, H.-J.; Pistor P.:* Data structures for an integrated database management and information retrieval system. In: *Proc. Int. Conf. on Very Large Databases,* S. 197–207, Mexiko, 1982.

[19] *Schek, H.-J.; Scholl, M. H.:* The Nf² relational algebra for a uniform manipulation of external, conceptual, and internal data structures. In: J. Schmidt (Hrsg.), *Sprachen für Datenbanken,* S. 113–133. IFB 72, Springer, Berlin, Heidelberg, 1983 (in German).

[20] *Schek, H.-J.; Scholl, M. H.:* The two roles of nested relations in the DASDBS project. In: S. Abiteboul, P. C. Fischer, H.-J. Schek (Hrsg.), *Nested Relations and Complex Objects in Databases.* Springer LNCS 361, Heidelberg, 1989.

[21] *Scholl, M.; Schek, H.-J.:* A relational object model, submitted for publication, Dezember 1989.

[22] *Scholl, M.; Schek, H.-J.:* A synthesis of complex objects and object-orientation. In: *Proc. GI Workshop on Foundations ob Object Models and Languages,* Lessach, Austria, Sept. 1989. Technical Report, Dept. of Computer Science, TU Clausthal, Clausthal, West Germany.

[23] *Scholl, M. H.:* Theoretical foundation of algebraic optimization utilizing unnormalized relations. In: ICDT '86: *Int. Conf. on Database Theory,* Rome. Italy, 1986, S. 380–396, LNCS 243, Springer, Berlin, Heidelberg.

[24] *Scholl, M. H.; Paul, H.-B.; Schek, H.-J.:* Supporting flat relations by a nested relational kernel. In: *Proc. Int. Conf. on Very Large Databases,* S. 137–146, Brighton, Sept. 1987. Morgan Kaufmann, Los Altos, Ca.

[25] *Wolf, A.:* The DASDBS-Geo Kernel: Concepts, experiences, and the second step. In: *Proc. Int. Symp. on the Design and Implementation of Large Spatial Databases,* Santa Barbara, Ca., 1989. Springer LNCS.