Rekursive Strukturen

Modellierung und Implementierung in relationalen Datenbanken

Manfred Sommer

Stand: Oktober 2002

Inhaltsverzeichnis

1 Exemplarische Einführung in rekursive Strukturen

In Entity-Relationship-Modellen sind die meisten *relationships* Beziehungen zwischen zwei verschiedenen Entitätstypen A und B. Diese Beziehungen werden sehr anschaulich als *binär* bezeichnet, Beziehungen zwischen mehr als zwei Klassen hingegen als mehrseitig oder *n-är*. In selteneren, aber nichts desto trotz praktisch bedeutsamen Fällen kann es auch eine Beziehung eines Entitätstyps zu sich selbst geben, die dann als *rekursiv* bezeichnet wird [Bruc92: 143].

In der Objektmodellierung werden dieselben Sachverhalte in etwas anderer Terminologie ähnlich formuliert. „Gewöhnlich ist eine Assoziation eine Beziehung zwischen zwei verschiedenen Klassen. Grundsätzlich kann eine Assoziation aber auch rekursiver Natur sein; eine Klasse hat in diesem Fall eine Beziehung zu sich selbst, wobei gewöhnlich unterstellt wird, dass jeweils zwei unterschiedliche Objekte dieser Klasse verbunden werden. An einer Assoziation können aber auch drei oder mehr verschieden Klassen beteiligt sein.“ [Oest98: 268]

1.1 Abgrenzung rekursiver Beziehungen zu Assoziationen

Bei den binären Beziehungen zwischen zwei verschiedenen Entitätstypen bzw. Klassen kann es sich um 1:N- oder N:M-Beziehungen handeln, je nachdem, wie viele Entitäten bzw. Instanzen auf beiden Seiten der Beziehung beteiligt sind.

- Beispiel für eine 1:N-Beziehung: ein Mitarbeiter kann (keine, eine oder) mehrere Bestellungen bearbeiten; eine Bestellung darf höchstens von einem Mitarbeiter bearbeitet werden.
- Beispiel für eine N:M-Beziehung: ein Mitarbeiter kann (keine, eine oder) mehrere Bestellungen bearbeiten; eine Bestellung kann von mehreren Mitarbeitern bearbeitet werden.

Datenmodelle werden hier als Entity-Relationship-Modelle in der IE-Notation mit den Tools ERwin 4.0 und Visio 2002 dargestellt; Objektmodelle als Klassendiagramme in der UML-Notation mit dem Tool Rational Rose. Abbildung 1 zeigt die beiden soeben genannten Beispiele als Klassendiagramm, Abbildung 2 als Entity-Relationship-Modell.

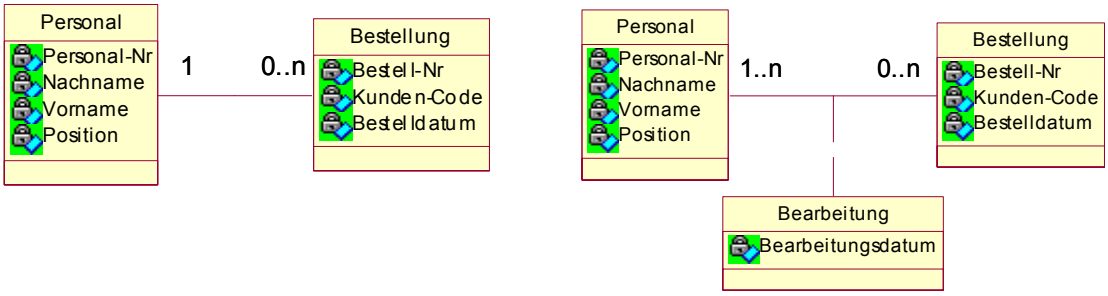

Abbildung 1: 1:N- bzw. N:M-Beziehung zwischen zwei Klassen

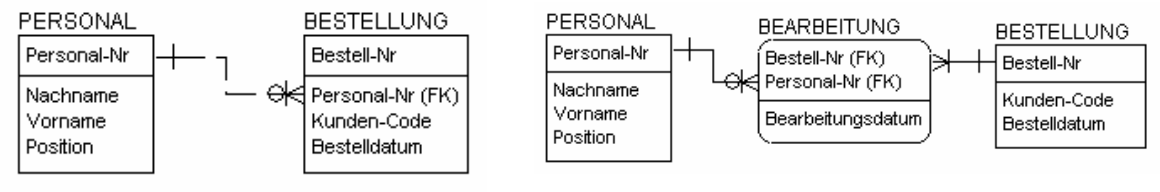

Abbildung 2: 1:N- bzw. N:M-Beziehung zwischen zwei Entitätstypen

Während im Allgemeinen Objekte (Entitäten) verschiedener Klassen (Entitätstypen) miteinander in Beziehung stehen, handelt es sich im speziellen Fall rekursiver Strukturen um Objekte derselben Klasse bzw. Entitäten des desselben Entitätstyps. Anknüpfend an das obige Beispiel könnte man fragen:

- Welcher Mitarbeiter X ist der fachliche Vorgesetzte von Mitarbeiter Y. Die Semantik der Beziehung könnte auch – wie z.B. in ARIS – lauten: „ist disziplinarisch vorgesetzt“ oder „ist fachlich vorgesetzt“. [IDS99: 4-72]
- Welche Bestellung X ist eine Nachbestellung zu einer Bestellung Y.

Grundsätzlich kann jede Klasse als selbstreferenziell aufgefasst werden. Ob es für die so entstehenden Beziehungen zwischen den Objekten derselben Klasse eine semantisch sinnvolle Interpretation gibt, ist eine andere Frage. Für die Modellierung eines konkreten Informationssystems kommt es – noch einschränkender – darauf an, ob gemäß Anforderungskatalog ein Bedarf für ihre Berücksichtigung besteht. Bei einer Klasse wie Personal ist man häufig daran interessiert zu wissen, wer der Vorgesetzte von wem ist.

Auch bei rekursiven Beziehungen ist es für das relationale Datenmodell von großer Bedeutung, ob es sich um Einfach-zu-Mehrfach- oder um Mehrfach-zu-Mehrfach-Beziehungen handelt.

- Beispiel für eine rekursive 1:N-Beziehung: ein Vorgesetzter kann (keinen, einen oder) mehrere Mitarbeiter haben; ein Mitarbeiter kann (keinen oder) einen Vorgesetzten haben.
- Beispiel für eine rekursive N:M-Beziehung: ein Vorgesetzter kann (keinen, einen oder) mehrere Mitarbeiter haben; ein Mitarbeiter kann (keinen, einen oder) mehrere Vorgesetzte haben; mehrere Vorgesetzte z.B. dann, wenn er unterschiedlichen Projekten mit verschiedenen Projektleitern zugeordnet ist.

Man könnte zwar auf eine rekursive Modellierung dieser Sachverhalte verzichten, indem man Mitarbeiter und deren Vorgesetzte als zwei unterschiedliche Klassen auffasst. Abbildung 3 zeigt eine 1:N-Beziehung zwischen diesen beiden Klassen (Entitätstypen).

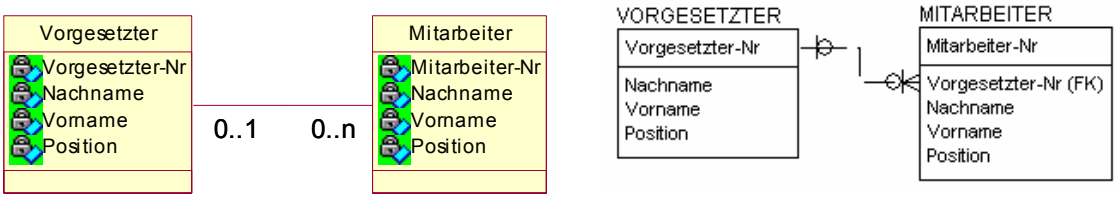

Abbildung 3: 1:N-Beziehung zwischen den Klassen bzw. Entitätstypen Vorgesetzter und Mitarbeiter

Beide Klassen weisen jedoch dieselben Eigenschaften auf. Noch nachteiliger: alle Mitarbeiter, die auch Vorgesetzte sind, müssen zusätzlich auch als Objekte dieser Klasse gespeichert werden. Dasselbe gilt analog für fast alle Vorgesetzten, da es nur relativ wenige geben dürfte, die nicht zugleich Mitarbeiter eines Anderen sind. Dieses Datenmodell produziert also eine erhebliche Redundanz und ist deshalb keine geeignete Lösung für eine eigentlich rekursive Struktur.

Sinnvoller ist es, auf die weitgehend redundante Klasse Vorgesetzter zu verzichten und nur mit der Klasse Mitarbeiter zu arbeiten, die einen allgemeineren Klassennamen wie z.B. Personal tragen sollte und alle Objekte als Unikate speichert. Mit dem Wegfall der überflüssigen Klasse entfällt auch die Beziehung zwischen den beiden Klassen, was aber natürlich nicht dazu führen darf, dass die Zusammenhänge zwischen den Objekten verloren gehen. Schließlich will man immer noch wissen, wer wessen Vorgesetzter ist.

Eine rekursive Beziehung führt somit zu der Klasse zurück, von der sie ausgeht (Abbildung 4). In der Datenmodellierungsnotation IDEF1X wird sie sehr anschaulich als „fish hook“ [Bruc92: 145] bezeichnet, weil sie wie ein Angelhaken auf denselben Entitätstyp zurückzeigt. Um eine 1:N-Beziehung handelt es sich deshalb, weil ein Mitarbeiter keinen oder höchstens einen Vorgesetzten haben kann, dieser Vorgesetzte aber bei mehreren Mitarbeitern als solcher in Erscheinung treten kann. Natürlich kann es auch Mitarbeiter geben, die selber keine Vorgesetzten sind.

Abbildung 4: Rekursive 1:N-Beziehung zwischen Objekten derselben Klasse bzw. Entitäten desselben Entitätstyps

Rekursive 1:N-Beziehungen sind in der Datenmodellierung etabliert. Ettlinger’s These, die Objektmodellierung ignoriere sie weitestgehend, kann mit dem Hinweis darauf, dass der Begriff „rekursiv“ im UML-Benutzerhandbuch nicht auftauche, nicht überzeugend belegt werden [Ettl99: 51]. In der englischen Originalausgabe wird der Sachverhalt – anders als Ettlinger behauptet – nicht nur als Schnittstellenspezifikation abgehandelt [BoRu99a: 146]. Vielmehr sprechen die Autoren ausdrücklich von reflexiven Assoziationen [BoRu99a: 65], wenn auch nur in der Randspalte, was in der deutschen Ausgabe obendrein leider weggefallen ist [BoRu99b: 70]. Offensichtlich ist mit „reflexiv“ aber dasselbe wie mit „rekursiv“ gemeint. Bei deutschsprachigen OO-Autoren finden sich beide Termini, z.B. „reflexiv“ bei Heide Balzert [Balz99: 42] und „rekursiv“ bei Oestereich [Oest99: 271].

1.2 Rekursive N:M-Beziehungen: Stammdaten- und Strukturentitäten

Eine N:M-Beziehung zwischen den Objekten ein und derselben Klasse lässt sich nicht mehr durch eine einzige Tabelle wie in Abbildung 4 relational abbilden. Bezogen auf unser Personalbeispiel hieße dies, dass jemand z.B. in verschiedenen Projekten einen anderen, also insgesamt mehrere Vorgesetzte haben kann. Dies ist in einem „fish hook"-Datemodell nicht unterzubringen, da im Referenzattribut Vorgesetzter.Personal-Nr nur ein Attributwert gespeichert werden kann. Man braucht also eine zweite Tabelle Personalstruktur mit zwei parallelen Beziehungen zur Mastertabelle Personal (Abbildung 5). Die beiden Beziehungen, die durch Rollennamen unterschieden werden, stellen die strukturellen Abhängigkeiten zwischen den klassengleichen Objekten dar.

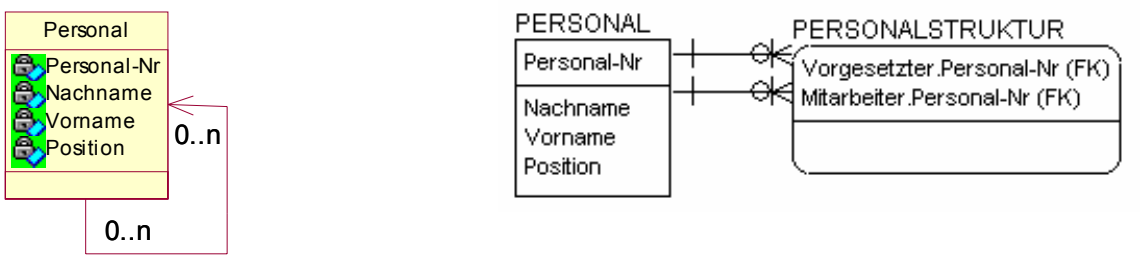

Abbildung 5: Rekursive N:M-Beziehung zwischen Objekten derselben Klasse bzw. Entitäten desselben Entitätstyps

In Abbildung 5 wird rechts an zwei optischen Auffälligkeiten sofort ersichtlich, dass unser Datenmodell eine rekursive N:M-Beziehung abbildet.

- Die Stammdatentabelle Personal und die Strukturtabelle Personalstruktur sind durch zwei parallele Beziehungen – und nicht wie bisher nur durch eine – miteinander verbunden.
- Der Primärschlüssel der Strukturtabelle besteht aus dem „gedoppelten" Primärschlüssel der Stammdatentabelle. Diese „Doppelung" ist natürlich nur scheinbar. Tatsächlich migriert der Primärschlüssel jeweils in unterschiedlicher Rolle: einmal als Personalnummer des Vorgesetzten, das andere Mal als Personalnummer des Mitarbeiters.

Anders als im Datenmodell führt der Übergang von einer 1:N- zu einer N:M-Beziehung im Klassenmodell nicht zur Einführung einer „Strukturklasse". Der Unterschied wird lediglich daraus ersichtlich, dass die Multiplizität jetzt an beiden Beziehungsenden 0..n ist (Abbildung 5 links).

Häufig wird die Strukturtabelle zusätzlich Nichtschlüsselattribute aufweisen, welche die rekursive Beziehung zwischen den Objekten näher charakterisieren. Im Datenmodell ändert sich nichts Grundsätzliches (Abbildung 6 rechts). Das Objektmodell benötigt hingegen eine zusätzliche assoziative Klasse, um die Attribute der rekursiven Beziehung aufnehmen zu können (Abbildung 6 links).

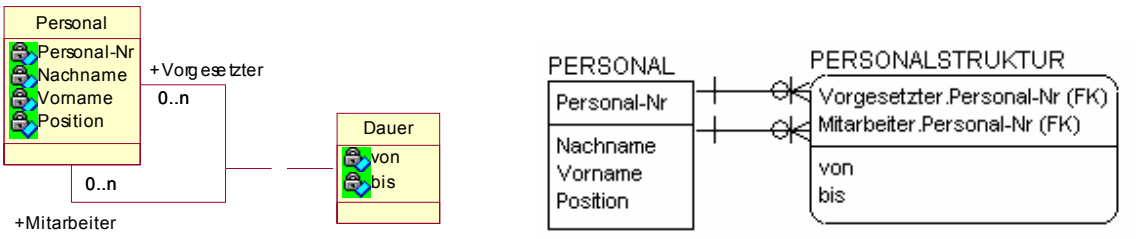

Abbildung 6: Rekursive N:M-Beziehung mit assoziativer Klasse bzw. Strukturentitätstyp

Wenn man denselben Sachverhalt im Objektmodell durch eine Strukturklasse statt durch eine assoziative Klasse abbildet, sieht das Klassendiagramm wie in Abbildung 7 links aus. Das korrespondierende Datenmodell bleibt gleich (Abbildung 7 rechts).

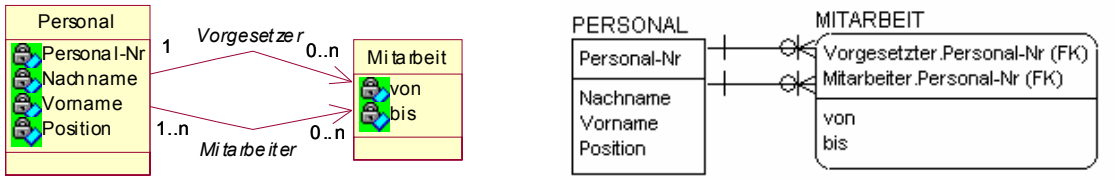

Abbildung 7: Rekursive N:M-Beziehung mit Strukturklasse bzw. Strukturentitätstyp

1.3 Abgrenzung rekursiver zu Parallelstrukturen

Obwohl auch im Klassendiagramm der über die Klasse Mitarbeit laufende Verweis der Klasse Personal auf sich selbst im Vordergrund steht, kommen Zweifel auf, ob es sich noch um eine reflexives Beziehungsmuster im objektorientierten Sinn handelt. „Es ist absolut zulässig, dass beide Enden einer Assoziation zirkulär auf dieselbe Klasse zurückkommen. Das bedeutet, dass ein gegebenes Objekt einer Klasse mit anderen Objekten derselben verknüpft werden kann“ [BoRu99a: 70]. Die Frage ist, ob diese „Zirkularität“ direkt erfolgen muss oder indirekt über eine andere Klasse vermittelt sein darf.

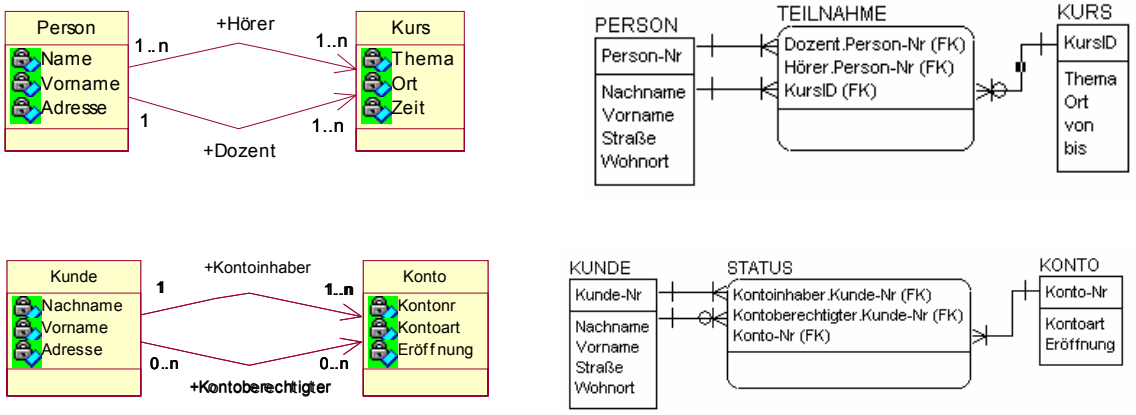

Abbildung 8: Weitere Beispiele rekursiver Beziehung mit Strukturklassen bzw. Strukturentitätstypen

Der Übergang von einem rekursiven N:M-Beziehungsmuster zwischen Objekten einer Klasse zu einer „echten Parallelstruktur“ [Kudl88: 124] zwischen selbstständigen und verschiedenartigen Klassen ist fließend. In Abbildung 8 oben links geht es um die Mitwirkung von Dozenten und die Teilnahme von Hörern an bestimmten Kursen, jedoch nicht primär um die Beziehungen zwischen Dozenten und Hörern als verschiedenen Instanzen der Klasse Person. Zudem bildet die Klasse Kurs auch nicht diese Strukturen ab, was aus dem Datenmodell (Abbildung 8 rechts) deutlicher hervorgeht. Die Strukturentität ist nämlich Teilnahme, die ihrerseits die Masterentität Kurs referenziert. Analog zur Strukturentität Personalstruktur kann man aus der Strukturentität Teilnahme ablesen, welche Person in einem bestimmtem Kurs Hörer ist und – in der Organigrammmetapher gesprochen – welche Dozenten-Person diese Hörer-Person hierarchisch „über sich“ hat. In einem anderen Kurs kann diese Person selbst Dozent sein und somit ihrerseits andere Personen als Hörer „unter sich“ haben.

Ähnlich verhält es sich bei der Beziehung zwischen Kunde und Konto (Abbildung 8 unten). Wenn derselbe Kunde in Bezug auf unterschiedliche Konten einmal als Kontoinhaber und das andere Mal als Mitverfügungsberechtigter in Erscheinung treten kann, macht die Parallelstruktur zwischen beiden Klassen mit unterschiedlichen Rollennamen Sinn [Balz99: 42]. Obwohl primär interessieren dürfte, wer bei den einzelnen Konten welche Rechte besitzt, lässt sich auch das Beziehungsgeflecht zwischen den Kunden selbst eruieren, weil die beiden Fremdschlüssel Kontoinhaber.Kunde-Nr und Kontoberechtigter.Kunde-Nr derselben Domäne unterliegen.

Aus der Perspektive der relationalen Datenmodellierung sind die Beziehungen zwischen den Entitätstypen

- Person und Teilnahme
- Kunde und Status

in Abbildung 8 genauso rekursiv wie die zwischen Personal und Personalstruktur in Abbildung 6. Aus dem Blickwinkel der Objektmodellierung muss man den Begriff der reflexiven Assoziation dehnen, um die zuletzt beschriebenen Sachverhalte darunter subsumieren zu können. Zwei Gründe sprechen aber dafür, sie nicht puristisch aus der Behandlung rekursiver Strukturen zu verbannen:

- Erstens schlagen sich die objektorientierten Vorbehalte im korrespondierenden Entity-Relationship-Modell nicht nieder. Da wir hier stets von einer Speicherung in einer relationalen Datenbank ausgehen, ist es ein „Streit um des Kaisers Bart“.
- Zweitens wird auf den Begriff „rekursiv“ in der Objektliteratur von einem Teil der Autoren verzichtet (dann kann er auch nicht in Widerspruch zum restriktiveren Konzept der reflexiven Assoziation geraten), während andere ihn ähnlich weitgefasst verwenden wie hier vorgeschlagen (z.B. [Oest99: 271]).

2 Typisierung rekursiver Strukturen

Zur näheren Betrachtung einzelner rekursiver Strukturen bietet es sich an, diese nach den Topologien der ihnen korrespondierenden Graphen zu klassifizieren:

- Baumstruktur
- Netzwerkstruktur
- Ringstruktur
- Listenstruktur (verkette Liste)
- Paarstruktur
- Hybridstruktur

Intuitiv lassen sie sich diese rekursiven Strukturen mit Hilfe von Hierarchiediagrammen voneinander abgrenzen, in denen die Elemente verschiedenen Ebenen zugeordnet werden (Abbildung 9).

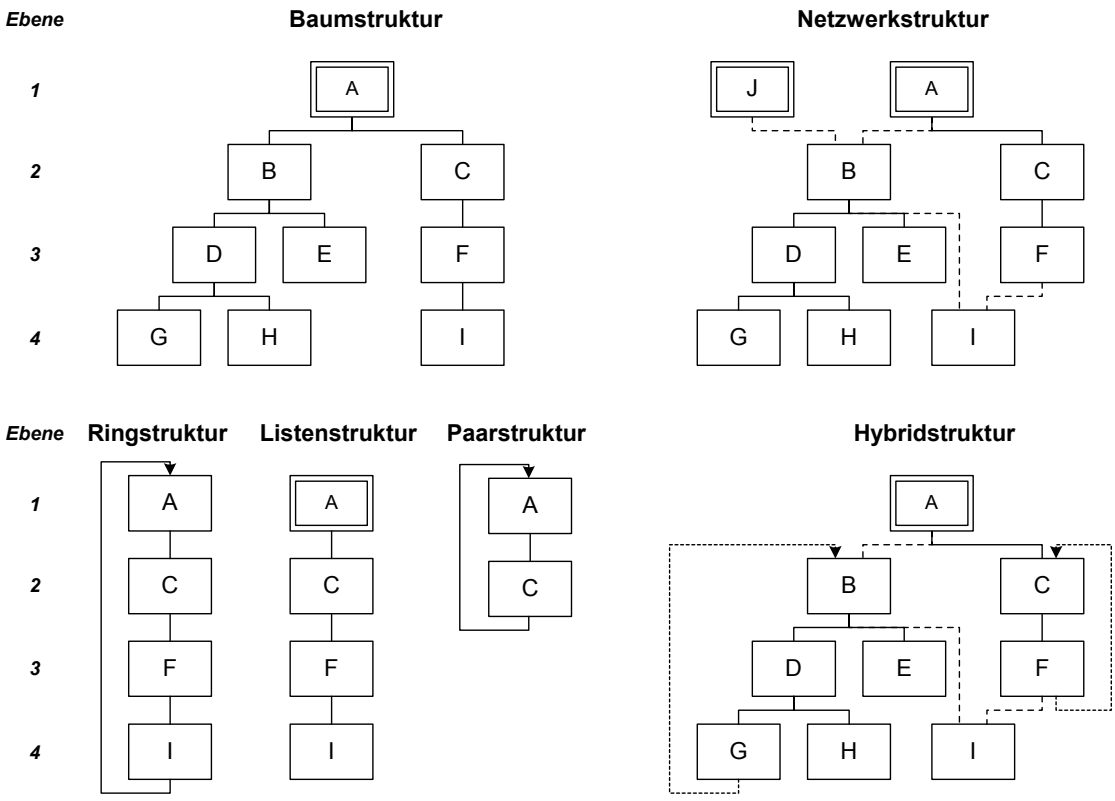

Abbildung 9: Hierarchiediagramme rekursiver Strukturen

Eine *Baumstruktur* gibt eine klassische Hierarchie wieder, wie sie aus Organigrammen der Aufbauorganisation oder aus Funktionsbäumen in der Softwareentwicklung bekannt ist. Jedes Element darf auf einer tieferen Ebene kein, ein oder mehrere Kindelemente haben; auf der übergeordneten Ebene darf es jedoch nur ein oder überhaupt kein Elternelement haben. Ferner muss es genau ein Element auf der obersten Ebene geben. Daraus folgt zugleich, dass eine Baumstruktur weder mit einer Netzwerkstruktur noch mit einer

Ringstruktur kompatibel ist, während eine Listenstruktur (Elemente C-F-I) als degenerierter Teilbaum auftreten kann.

Eine *Netzwerkstruktur* ist allgemeiner als eine Baumstruktur, da sie mehr als ein Elternelement auf höheren Ebenen besitzen kann (gestrichelte Beziehungskanten). Deshalb können die beiden Elemente A und J als Eltern von Element B und die beiden Elemente B und F als Eltern von Element I auftreten. Wenn sich alle Elternelemente jeweils auf der nächst höheren Ebene anordnen lassen, kann man die Netzwerkstruktur als ebenenhomogen bezeichnen, andernfalls als ebenenheterogen. Die Netzwerkstruktur in Abbildung 9 ist ebenenheteroregen, weil die dem Element I übergeordneten Elemente B und F auf unterschiedlichen Ebenen liegen. Netzwerkstrukturen beinhalten sehr häufig Baumstrukturen als Teilstrukturen, weil nicht sämtliche Elemente mehr als ein Elternelement haben müssen (in Abbildung 9 die Elemente D, G und H). Ringstrukturen sind hier jedoch nicht erlaubt.

Ring-, Listen- und Paarstrukturen sind spezielle rekursive Strukturen, die eher selten in Reinkultur vorkommen. Eine *Ringstruktur* ist dadurch gekennzeichnet, dass genau ein Element einen Vorgänger auf tieferer Ebene besitzt; in Abbildung 9 hat das Element A auf Ebene 1 das Element I auf Ebene 4 zum Vorgänger.

Eine *Paarstruktur* kann als spezielle Ringstruktur betrachtet werden, nämlich als Ringstruktur mit nur zwei Elementen.

Eine *Listenstruktur* hat toplogisch auf den ersten Blick eine große Ähnlichkeit mit einer Ringstruktur („offener Ring“). Tatsächlich ist jedoch keine spezielle Ringsstruktur, da sie gegen deren konstituierendes Merkmal verstößt, dass jedes Element genau ein Elternelement haben muss. Vielmehr ist sie eine spezielle Baumstruktur, in der jedes Element höchstens ein Kindelement besitzt. In betriebswirtschaftlichen Informationssystemen dürften echte Ring-, Listen- und Paarstrukturen nur sehr selten vorkommen.

Wenn in einer Netzwerk- oder Baumstruktur mindestens ein Element einen Vorgänger auf gleicher oder tieferer Ebene besitzt und diese Verletzung einer echten Netzwerk- oder Baumstruktur durch keine andere Ebenenzuordnung von Elementen beseitigt werden kann, dann enthält sie einen u.U. nicht sofort erkennbaren Ring als Teilstruktur. Eine solche Verquickung von Netzwerk- und Ringstrukturen sei als *Hybridstruktur* bezeichnet. Sie ist die allgemeinste rekursive Struktur, da sie keine der zuvor besprochenen Strukturen ausschließt. Deshalb können in ihr auch Baum-, Listen- und Paarstrukturen als Teilstrukturen auftreten.

Graphische Darstellungen rekursiver Strukturen werden häufig allgemein als Hierarchiediagramme oder spezieller als *Organigramme* bezeichnet, auch dann, wenn es sich nicht um streng hierarchische Baumstrukturen handelt. Während im Personal- und Organisationswesen streng hierarchische Strukturen sehr verbreitet sind, weisen z.B. Beteiligungsstrukturen zwischen Unternehmen eine größere topologische Vielfalt auf, weil Unternehmen häufig mehrere Mütter haben und Töchter auch indirekt wieder an einer ihrer Mütter beteiligt sein können [Somm01]. Auch bei der graphischen Darstellung solcher nicht-hierarchischen Strukturen hat sich der Begriff Organigramm eingebürgert.

Rekursive Strukturen kann man auch gut mit quadratischen *Verflechtungsmatrizen* darstellen. Insbesondere bei sehr komplexen Hybridstrukturen werden die direkten Abhängigkeiten von Vorgängerelementen bzw. zu Nachfolgerelementen leichter erkennbar als

in Hierarchiediagrammen. Die Stärke von Hierarchiediagrammen besteht hingegen darin, sämtliche Abhängigkeiten ebenenübergreifend zu visualisieren.

In den folgenden Abschnitten werden die einzelnen rekursiven Strukturen sowohl mit Hierarchiediagrammen als auch mit Verflechtungsmatrizen veranschaulicht. Als Beispiel verwenden wir Beteiligungsstrukturen in einem fiktiven Unternehmensverbund.

In den folgenden Abschnitten werden diese rekursiven Strukturen exakt definiert sowie graphisch durch Hierarchiediagramme und tabellarisch durch Verflechtungsmatrizen dargestellt. Jeweils anschließend wird gezeigt, wie sie in Daten- und Objektmodellen abzubilden sind und in relationalen Datenbanken prototypisch implementiert werden können. Besonderes Augenmerk wird dabei darauf gelegt, wie mit gespeicherten Prozeduren unterstützt werden kann, dass bei der Datenpflege möglichst keine Verstöße gegen die jeweils behandelte rekursive Struktur auftreten.

3 Baumstruktur

Eine streng hierarchische Unternehmensstruktur sieht wie in Abbildung 10 aus. Die Prozentzahlen an den Kanten geben an, wie stark die jeweilige Mutter an diesem Unternehmen beteiligt ist. Werte unter 100% bedeuten, dass Dritte an dem Unternehmen beteiligt sind, die nicht zum Unternehmensverbund gehören. Inter Bike ist das einzige Wurzelelement dieser Unternehmensgruppe der Zweiradindustrie. Australian Cycles, Zuse Zweirad, Bike & Fun, US Racing, Meyer's Radreisen, Bayern Bike und Velorapid sind Blattelemente, weil sie keine Kindelemente haben.

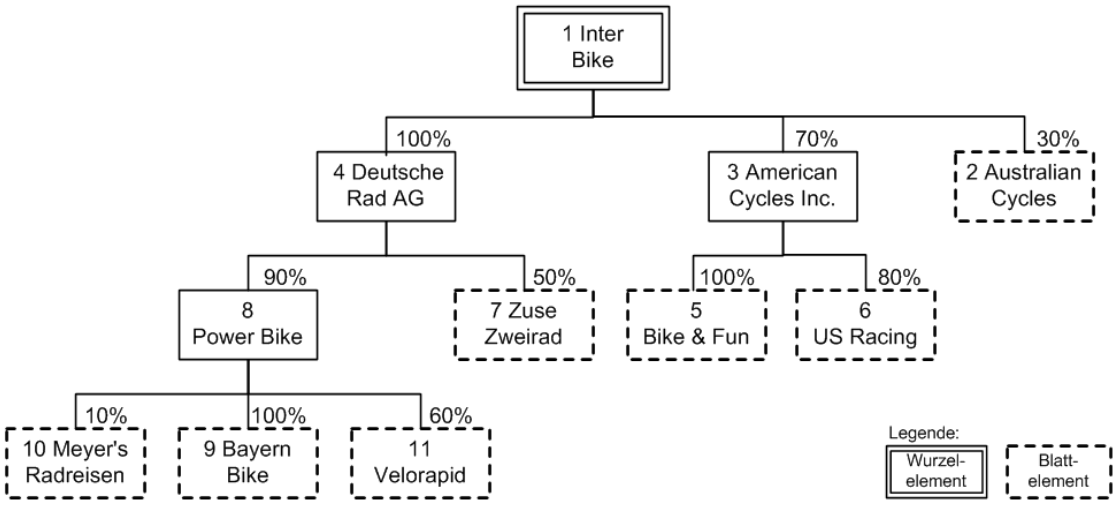

Abbildung 10: Graph einer Baumstruktur

Allgemein gilt für eine Baumstruktur:

- Jedes Element kann beliebig viele Kindelemente haben. Blattelemente haben keine Kindelemente.
- Alle Elemente dürfen höchstens ein Elternelement haben. Ein Element, das sog. Wurzelelement, darf kein Elternelement haben. Gibt es mehr als ein Wurzelelement, liegt eine multiple Baumstruktur aus vollständig unabhängigen Teilbäumen vor [DoHu99: 260]. Sind die Teilbäume untereinander verbunden, dann kann es mehrere Wurzelelemente nur in einer Netzwerkstruktur geben (→ Netzwerkstruktur).

Die rekursiven Beziehungen der Unternehmen untereinander können nicht nur graphisch, sondern auch tabellarisch in einer quadratischen Verflechtungsmatrix dargestellt werden (Abbildung 11). Für eine Baumstruktur ist sie immer triangulierbar, weil sie durch geeignetes symmetrisches Vertauschen von Zeilen und Spalten so angeordnet werden kann, dass sämtliche Beziehungsattributwerte unterhalb der Hauptdiagonalen zu stehen kommen.

	Elternelemente										
Kindelemente	1 Inter Bike	2 Australian Cycles	3 American Cycles Inc.	4 Deutsche Rad AG	5 Bike & Fun	6 US Racing	7 Zuse Zweirad	8 Power Bike	9 Bayern Bike	10 Meyer's Radreisen	11 Velorapid
1 Inter Bike											
2 Australian Cycles	**B**										
3 American Cycles Inc.	**B**										
4 Deutsche Rad AG	**B**										
5 Bike & Fun			**B**								
6 US Racing			**B**								
7 Zuse Zweirad				**B**							
8 Power Bike				**B**							
9 Bayern Bike								**B**			
10 Meyer's Radreisen								**B**			
11 Velorapid								**B**			

Wurzelelement ←

Blattelemente ↑ ↑ ↑ ↑ ↑ ↑ ↑

B Baumstruktur

Abbildung 11: Verflechtungsmatrix einer Baumstruktur

Die Zeile des Wurzelements enthält keine Matrixelemente. Die übrigen Zeilen enthalten genau ein Element, das hier allgemein mit B (für: Baum), gekennzeichnet wird. Mehr als ein B pro Zeile ist in einer Baumstruktur nicht möglich (→ Netzwerkstruktur). Die Blattelemente weisen in den Spalten der Verflechtungsmatrix keine Elemente auf. Spalten mit mehreren B's sind für Baumstrukturen typisch, weil ein Elternelement üblicherweise mehrere Kindelemente hat. In einer Spalte mit nur einem B hätte das Elternelement nur ein Kindelement. Dann gäbe es in diesem Baum eine Listenstruktur als Teilstruktur.

3.1 Datenmodell und Klassenmodell

Eine Baumstruktur kann im Daten- und Klassenmodell immer als Angelhakenbeziehung (fish hook) der Entität bzw. Klasse zu sich selbst dargestellt werden. Die mit einer Baumstruktur vereinbaren Kardinalitäten sind aus Abbildung 12 ersichtlich:

- oben: (0,1) : (0,n)
- unten: (1,1) : (0,n)

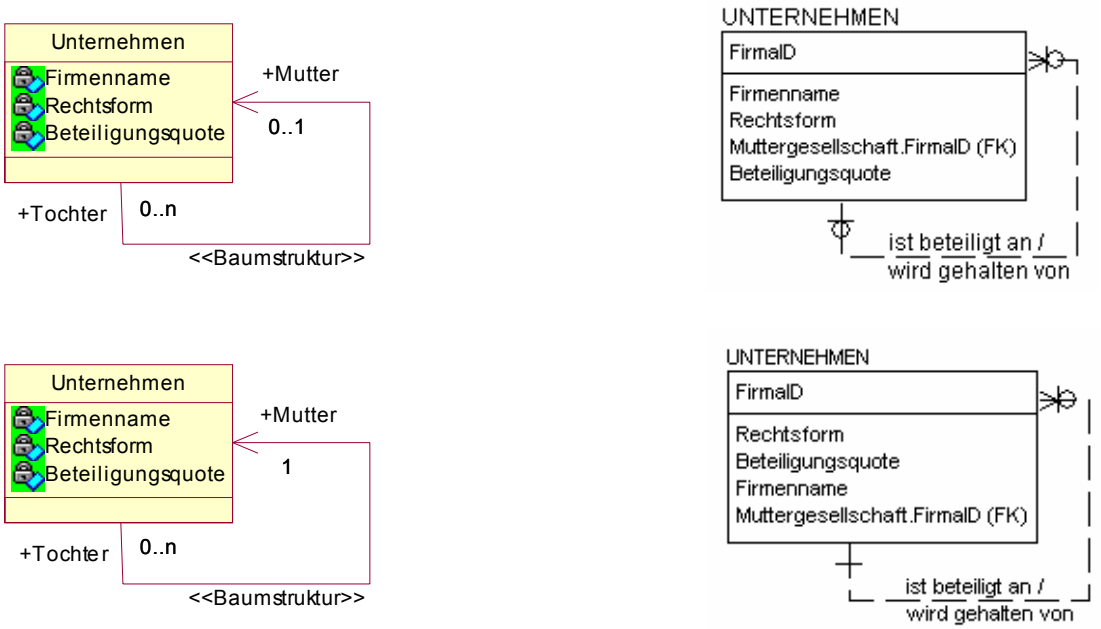

Abbildung 12: Klassenmodell und Datenmodell einer Baumstruktur

- Detailseitige Kardinalität (0,n): Der Primärschlüssel FirmaID kann keinmal, einmal oder mehrfach als Fremdschlüssel Muttergesellschaft.FirmaID auftauchen.
  - Dabei bedeutet die detailseitige Minimalkardinalität 0, dass es Primärschlüsselwerte FirmaID geben kann, die nicht in der Domäne des Fremdschlüssels Muttergesellschaft.FirmaID auftauchen. Sie repräsentieren die Blattelemente. Eine detailseitige Minimalkardinalität 1 würde demzufolge implizieren, dass es keine Blattelemente gäbe, was mit einer Baumstruktur unvereinbar ist (→ Ringstruktur).

  - Die detailseitige Maximalkardinalität n bedeutet, dass ein Primärschlüsselwert FirmaID in mehr als einem Datensatz als Fremdschlüssel Muttergesellschaft.FirmaID in Erscheinung treten kann; dadurch entsteht die Baumstruktur. Eine detailseitige Maximalkardinalität 1 würde demzufolge implizieren, dass kein Elternelement mehr als ein Kindelement besäße, was ebenfalls mit einer Baumstruktur unvereinbar ist (→ Listenstruktur).

- Masterseitige Kardinalität (0,1): Der Fremdschlüssel Muttergesellschaft.FirmaID kann keinmal oder einmal den Primärschlüsselwert FirmaID referenzieren: alle Elemente dürfen höchstens ein Elternelement haben.
  - Dabei bedeutet die masterseitige Minimalkardinalität 0, dass es Elemente mit Nullwerten für den Fremdschlüssel Muttergesellschaft.FirmaID geben kann; sie repräsentieren Wurzelelemente.
  - Die masterseitige Minimalkardinalität könnte auch 1 sein (1,1). Da der Fremdschlüssel dann keine Nullwerte mehr annehmen kann, müssten Wurzelelemente anderweitig erkannt werden, z.B. daran, dass ihr Fremdschlüssel mit ihrem eigenen Primärschlüssel übereinstimmt oder einen hierfür reservierten Wert wie z.B. 0 annimmt.
  - Die masterseitige Maximalkardinalität darf nicht größer als 1 sein.

3.2 Implementierung in einer relationalen Datenbank

Die SQL Server-Datenbank Baumstruktur kann im Query Analyzer mit einem SQL-Skript erzeugt werden (Datei BaumstrukturDatenbankAnlegen.sql). Wenn eine Datenbank Baumstruktur auf diesem SQL-Server bereits existiert, sollte sie zuvor gelöscht werden.

```
IF EXISTS (SELECT * FROM master..sysdatabases WHERE name = 'Baumstruktur')
   DROP DATABASE Baumstruktur
CREATE DATABASE Baumstruktur
```

Anschließend wir diese neu angelegte Datenbank verwendet.

```
USE Baumstruktur
```

Die Tabelle Unternehmen wird mit den aus Abbildung 12 ersichtlichen Spalten angelegt.

```
CREATE TABLE Unternehmen (
   FirmaID              INT NOT NULL,
   Firmenname           VARCHAR(80) NOT NULL,
   Rechtsform           VARCHAR(50) NULL,
   Beteiligungsquote    DECIMAL(5,4) NULL,
   Muttergesellschaft   INT NULL,
   CONSTRAINT PK_Unternehmen_FirmaID PRIMARY KEY (FirmaID),
   CONSTRAINT FK_Unternehmen_Muttergesellschaft_FirmaID
        FOREIGN KEY (Muttergesellschaft)
        REFERENCES Unternehmen (FirmaID)
        ON DELETE NO ACTION),
   CONSTRAINT CK_Muttergesellschaft CHECK (Muttergesellschaft <> FirmaID))
```

Die beiden ersten Constraints erfüllen dabei folgende Aufgaben:

- Der erste Constraint definiert den Primärschlüssel.
- Der zweite Constraint sorgt für die rekursive Beziehung vom Fremdschlüssel Muttergesellschaft zum Primärschlüssel FirmaID derselben Tabelle.
- Der dritte Constraint wird im Abschnitt 3.2.2.1 erläutert.

Mit den im SQL-Skript abschließend enthaltenen INSERT-Befehlen werden die Beispieldaten der Abbildung 10 eingefügt. Die Access-Datenbank Baumstruktur.mdb entspricht dieser SQL Server-Datenbank.

3.2.1 Referenzielle Integritätsaktionen

Im Gegensatz zu früheren SQL Server-Versionen beherrscht der SQL Server 2000 zwar grundsätzlich die deklarative Löschweitergabe (ON DELETE CASCADE) sowie die deklarative Aktualisierungsweitergabe (ON UPDATE CASCADE), nicht jedoch in rekursiven Beziehungen. Den Versuch, eine deklarative Löschweitergabe anzulegen, weist der SQL Server 2000 mit folgender Fehlermeldung zurück:

```
„Das Einführen der FOREIGN KEY-Einschränkung 'FK_Unternehmen_Muttergesell-
schaft_FirmaID' für die Unternehmen-Tabelle kann Schleifen oder mehrere kaska-
dierende Pfade verursachen. Geben SIE ON DELETE NO ACTION oder ON UPDATE NO
ACTION an, oder ändern Sie andere FOREIGN KEY-Einschränkungen".
```

Das ist auch gut so, weil es sonst beim Löschen von Muttergesellschaften zu einem sich unkontrolliert fortpflanzenden Löschen aller direkt und indirekt abhängigen Tochterunternehmen käme. Da die Löschweitergabe nicht zur Verfügung steht, gibt es nur zwei Auswege.

- Entweder verzichtet man gänzlich auf die referenzielle Integrität. Dann kann serverseitig nicht mehr garantiert werden, dass nur existierende Unternehmen als Mütter fungieren.
- Oder man gewährleistet die referenzielle Integrität mit einer Löschverhinderung (ON DELETE NO ACTION). Dann können Nicht-Blattelemente nur mit einem die Löschverhinderung substituierenden INSTEAD OF DELETE-Trigger gelöscht werden. Implementiert man einen solchen Trigger (→ 3.2.1.1), dann greift er allerdings auch für Blattelemente.

3.2.1.1 Löschen von Baumelementen

Sinnvollerweise hängt man die Kindelemente eines zu löschenden Baumelements unter dessen Elternelement. Beim Löschen der Deutschen Rad AG (Abbildung 10) erhalten dann die Firmen Power Bike und Zuse Zweirad die Firma Inter Bike als neue Mutter.

Bei einem zur Löschung anstehenden Element müssen deshalb zuerst dessen Kindelemente einem anderen Elternelement unterstellt werden. Dadurch wird es selbst zu einem löschbaren Blattelement. Natürlich kann der Anwender diese Datenmanipulationen schrittweise von Hand erledigen. Sie lassen sich aber mit einem INSTEAD OF DELETE-Trigger automatisieren. Ein AFTER DELETE-Trigger kommt hier nicht in Frage, da er an der deklarierten Löschverhinderung scheitern würde. Ein INSTEAD OF DELETE-Trigger ersetzt hingegen die deklarative Löschverhinderung.

```
/* Datei: BaumstrukturElementeLöschen.sql */
```

```
CREATE TRIGGER Unternehmen_I_D
ON Unternehmen
INSTEAD OF DELETE
AS
DECLARE @Löschen INT    /* zu löschendes Elements */
DECLARE @Mutter INT     /* Mutter des zu löschenden Elements */
SELECT @Löschen = FirmaID FROM deleted
SELECT @Mutter = Muttergesellschaft FROM deleted
IF EXISTS(SELECT * FROM Unternehmen WHERE FirmaID = @Mutter)
      /* Löschen von anderen als Wurzelelementen */
      BEGIN
      UPDATE Unternehmen
         SET Muttergesellschaft = @Mutter WHERE FirmaID IN
         (SELECT T.FirmaID
         FROM Unternehmen M INNER JOIN Unternehmen T
         ON M.FirmaID = T.Muttergesellschaft
         WHERE M.FirmaID = @Löschen)
      DELETE FROM Unternehmen WHERE FirmaID = @Löschen
      END
/* Trigger hier beenden, wenn Wurzelelement nicht gelöscht werden soll */
```

In der internen Tabelle deleted steht der zu löschende Datensatz, dessen FirmaID der lokalen Variablen @Löschen und dessen Muttergesellschaft der lokalen Variablen @Mutter zugewiesen wird. Mit der EXISTS-Abfrage wird dann geprüft, ob das zulöschende Element ein Elternelement besitzt, also kein Wurzelelement ist. Ist das der Fall (IF-Zweig), dann wird der Primärschlüssel dieses Elternelements den Fremdschlüsseln aller Kindelemente des zu löschenden Elements mit einem UPDATE-Befehl zugewiesen. Das zu löschende Element ist jetzt zu einem Blattelement geworden und kann mit DELETE gelöscht werden. Besitzt das zu löschende Element kein Elternelement, ist also ein Wurzelelement, dann tut der Trigger gar nichts. Ein Wurzelelement wird also nicht gelöscht und damit ein Auseinanderfallen der Baumstruktur verhindert.

Man beachte, dass dieser Trigger die Beteiligungsquoten nicht automatisch neu berechnet. Die Beteiligungsquoten müssen also händisch nachgepflegt werden. Ein aufwendigerer Trigger, der mit Hilfe von Cursorn auch die Beteiligungsquoten pflegt, wird bei der Netzwerkstruktur vorgestellt (→ Abschnitt 4.2.2.2 und Datei NetzwerkstrukturBeliebigenRingVerhindern.sql)

Will man ein Löschen auch des Wurzelelements zulassen, dann müssen die Fremdschlüssel seiner Kindelemente Null-Werte annehmen. Programmierungstechnisch ist dies kein Problem, da der INSTEAD OF DELETE-Trigger lediglich um einen weitgehend identischen ELSE-Zweig ergänzt werden muss. Fachlich muss man sich aber darüber im Klaren sein, dass hierdurch ein multipler Baum entsteht, wenn das zu löschende Wurzelelement mehr als ein Kindelement besitzt.

```
. . .
/* Trigger hier beenden, wenn Wurzelelement nicht gelöscht werden soll */
ELSE
      /* Löschen eines Wurzelelements */
      BEGIN
      /* Alle Kindelemente des zu löschenden Wurzelelements werden
      Wurzelelemete (multipler Baum) */
      UPDATE Unternehmen
         SET Muttergesellschaft = NULL
         WHERE FirmaID IN
         (SELECT T.FirmaID
         FROM Unternehmen M INNER JOIN Unternehmen T
```

```
      ON M.FirmaID = T.Muttergesellschaft
      WHERE M.FirmaID = @Löschen)
   DELETE FROM Unternehmen WHERE FirmaID = @Löschen
   END
```

Die hier vorgeschlagene Lösung löscht nur ein einziges Element an einer beliebigen Stelle einer Baumstruktur. Soll hingegen ein ganzer Teilbaum gelöscht werden, müssen dessen Elemente wegen des Verzichts auf das kaskadierende Löschen einzeln von unten nach oben gelöscht werden. Das ist zwar etwas umständlich, zwingt aber zu einem sorgfältigen Vorgehen.

3.2.1.2 Aktualisieren des Primärschlüssels

Änderungen des Primärschlüssels können bei rekursiven Beziehungen weder im SQL Server 2000 mit ON UPDATE CASCADE [BeMo00: 515] noch in Access mit einer Aktualisierungsweitergabe weitergereicht werden, weshalb es bei der Defaulteinstellung ON UPDATE NO ACTION bleiben muss. Dann lassen sich aber nur Primärschlüssel von Blattelementen aktualisieren. Die Aktualisierung der Primärschlüssel von Nicht-Blattelementen kann wiederum nur mit einem INSTEAD OF UPDATE -Trigger herbeigeführt werden.

```
/* Datei: BaumstrukturKaskadierendesUpdate.sql */
CREATE TRIGGER Unternehmen_I_U
ON Unternehmen INSTEAD OF UPDATE
AS
DECLARE @FirmaID_alt INT     /* alte Firmennummer */
DECLARE @FirmaID_neu INT     /* neue Firmennummer */
SELECT @FirmaID_alt = FirmaID FROM deleted
SELECT @FirmaID_neu = FirmaID FROM inserted
IF UPDATE(FirmaID)
      BEGIN
      INSERT INTO Unternehmen SELECT * FROM inserted
      UPDATE Unternehmen
         SET Muttergesellschaft = @FirmaID_neu
         WHERE Muttergesellschaft = @FirmaID_alt
      DELETE FROM Unternehmen WHERE FirmaID = @FirmaID_alt
      END
```

Dies geschieht in drei Schritten. Zuerst wird der Datensatz mit der neuen FirmaID aus der internen Tabelle inserted eingefügt, was wegen ihrer Primärschlüsseleigenschaft natürlich nur dann gelingt, wenn diese nicht bereits in der Tabelle Unternehmen vorhanden ist. Anschließend werden alle Fremdschlüssel auf den neuen Primärschlüsselwert aktualisiert. Abschließend wird der Datensatz mit der alten FirmaID aus der Tabelle Unternehmen gelöscht.

Von einer Aktualisierung eines Primärschlüssels wird jedoch abgeraten, da Primärschlüssel auch in relationalen Datenbanken wie eine lebenslängliche Objekt-ID behandelt werden sollten.

3.2.2 Gewährleistung einer streng hierarchischen Baumstruktur

Die beiden Datenmodell-Varianten in Abbildung 12 schließen zwar eine Netzwerkstruktur definitiv aus, implizieren jedoch nicht zwingend eine streng hierarchische Baumstruktur, weil gegen diese mehrfach verstoßen werden kann. Dorsey und Hudicka sehen drei Problembereiche [DoHu99: 266]:

- Ein Unternehmen könnte seine eigene Mutter sein.
- Der Unternehmensverbund könnte eine Ringstruktur beinhalten.
- Bei mehr als einem Wurzelelement zerfällt die Baumstruktur in mehrere unabhängige Teilbäume (multipler Baum).

In den folgenden Abschnitten werden Lösungen vorgeschlagen, um diese Verletzungen einer Baumstruktur zu unterbinden.

3.2.2.1 Verhinderung von Selbstreferenzialität

Ein Unternehmen ist seine eigene Mutter, wenn sein Fremdschlüssel Muttergesellschaft.FirmaID denselben Wert wie sein Primärschlüssel FirmaID annimmt. Das in Abbildung 12 dargestellte logische Datenmodell kann dies nicht verhindern, weil hierdurch nicht gegen die referenzielle Integrität verstoßen wird. Abhilfe schafft jedoch die Domänenintegritätsbedingung, die im CREATE TABLE-Befehl (Abschnitt 3.2) bereits als dritter Constraint berücksichtigt wurde.

```
CONSTRAINT CK_Muttergesellschaft CHECK (Muttergesellschaft <> FirmaID))
```

In einer Access-Datenbank gewährleistet man dies durch eine Gültigkeitsregel auf Tabellenebene.

Vorkehrungen gegen diese Verletzung einer Baumstruktur sind nicht nur serverseitig (auf Tabellenebene), sondern auch clientseitig (auf Formularebene) möglich. Abbildung 13 zeigt ein Access-Formular, das die aktuell bearbeitete Gesellschaft in der Dropdown-Liste des Kombinationsfeldes Muttergesellschaft über dessen Datensatzherkunft gar nicht erst zur Auswahl anbietet. Wie immer ist eine derartige Lösung in der Präsentationsschicht benutzerfreundlicher, aber im Gegensatz zur serverbasierten Lösung nicht unterwanderungssicher. Optimal ist die Kombination server- und clientseitiger Vorkehrungen.

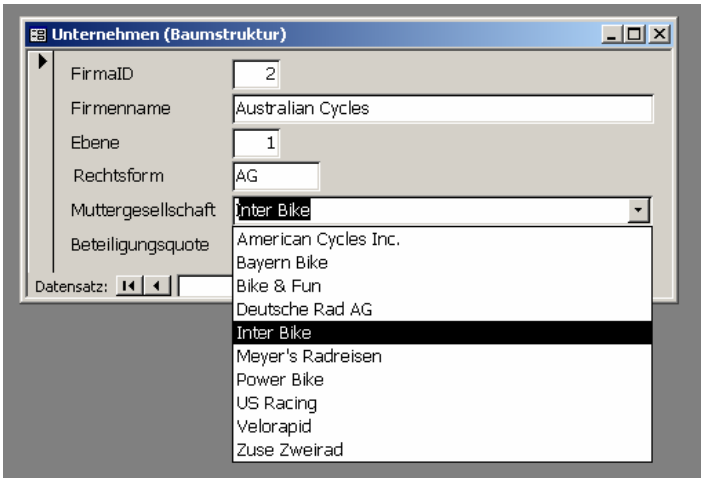

Abbildung 13: Ausschluss eines Unternehmens von der Liste potentieller Muttergesellschaften

3.2.2.2 Verhinderung eines Rings in einer Baumstruktur

Dorsey und Hudicka behaupten, Daten- und Objektmodelle wie in Abbildung 12 ließen Ringstrukturen zu. „You may have a closed loop in your data“ [DoHu99: 266]. Diese These ist aber nur für Ringstrukturen aufrecht zu erhalten, die entweder

(a) das „Wurzelelement“ einschließen oder

(b) vom übrigen Baum vollständig separiert sind.

(c) Sonst ergibt sich eine mit Abbildung 12 unvereinbare Netzwerkstruktur, die keiner besonderen Behandlung bedarf.

(a) Verhinderung eines Rings durch das „Wurzelelement“

Wenn der Ring das „Wurzelelement“ einschließt, dann ist der „Baum“ per definitionem kein Baum mehr. Das auf unsere Abbildung 10 übertragene Beispiel der beiden Autoren belegt dies: Inter Bike ist Mutter der Deutschen Rad AG und die Deutsche Rad AG ist Mutter von Power Bike; wäre nun Power Bike zusätzlich wiederum Mutter von Inter Bike, so wäre Inter Bike kein Wurzelelement und damit das Ganze kein Baum mehr. Hiergegen ist auf der Ebene des Daten- und Objektmodells bzw. des CREATE TABLE-Befehls kein Kraut gewachsen.

Wenn es nur ein einziges Wurzelelement gibt, dann darf dessen Fremdschlüssel seinen Null-Wert nicht verlieren, weil es in einer Baumstruktur mindestens ein Wurzelelement geben muss. Dies lässt sich ebenfalls mit einem INSTEAD OF UPDATE-Triggers erreichen:

```
/* Datei: BaumstrukturRingDurchWurzelelementVerhindern.sql */
CREATE TRIGGER Unternehmen_I_U
ON Unternehmen INSTEAD OF UPDATE
AS
DECLARE @FirmaID INT                          /* Firmennummer */
DECLARE @Firmenname_neu VARCHAR(80)           /* neuer Firmenname */
DECLARE @Rechtsform_neu VARCHAR(80)           /* neue Rechtsform */
DECLARE @Beteiligungsquote_neu DECIMAL(5,4) /* neue Beteiligungsquote */
DECLARE @Muttergesellschaft_alt INT           /* alte Muttergesellschaft */
DECLARE @Muttergesellschaft_neu INT           /* neue Muttergesellschaft */
SELECT @FirmaID = FirmaID FROM deleted
SELECT @Firmenname_neu = Firmenname FROM inserted
SELECT @Rechtsform_neu = Rechtsform FROM inserted
SELECT @Beteiligungsquote_neu = Beteiligungsquote FROM inserted
SELECT @Muttergesellschaft_alt = Muttergesellschaft FROM deleted
SELECT @Muttergesellschaft_neu = Muttergesellschaft FROM inserted
/* Verhinderung eines Rings durch das bisherige Wurzelement */
IF UPDATE(Muttergesellschaft) AND @Muttergesellschaft_alt IS NOT NULL
     UPDATE Unternehmen
        SET Muttergesellschaft = @Muttergesellschaft_neu
        WHERE FirmaID = @FirmaID
/* Zugelassene Aktualisierungen */
IF UPDATE(Firmenname) OR UPDATE(Rechtsform) OR
(UPDATE(Beteiligungsquote) AND @Muttergesellschaft_alt IS NOT NULL)
     UPDATE Unternehmen
        SET Firmenname = @Firmenname_neu, Rechtsform = @Rechtsform_neu,
        Beteiligungsquote = @Beteiligungsquote_neu
        WHERE FirmaID = @FirmaID
```

Der Fremdschlüssel Muttergesellschaft kann nur aktualisiert werden, wenn er bisher keinen Nullwert hatte, was nur für Nichtwurzelelemente zutreffen kann. Nur am Rande

sei angemerkt, dass diese Lösung umgekehrt nicht ausschließt, dass ein bisheriges Nichtwurzelelement durch Zuweisung eines Nullwerts für seine Muttergesellschaft zu einem weitern Wurzelelement wird und dadurch ein multipler Baum entsteht. Hierauf wird im Abschnitt 3.2.2.3 eingegangen. Noch schlimmer: ein auf diese Weise entstandener multipler Baum lässt sich nicht wieder rückgängig machen, da Nullwerte für den Fremdschlüssel Muttergesellschaft nicht geändert werden dürfen. Das ganze Dilemma entsteht dadurch, dass hier Nullwerte bzw. Nichtnullwerte im Fremdschlüssel als Aufhänger für die Verhinderung einer Ringstruktur genommen werden. Im folgenden Unterabschnitt (b) wird eine elegantere, aber programmierungstechnisch etwas anspruchsvollere Lösung vorgestellt, die beliebige Ringe mittels Rekursion unterdrückt.

Aus den im Abschnitt 3.2.1.2 erörterten Gründen lässt der obige INSTEAD OF UPDATE-Trigger keine Aktualisierung des Primärschlüssel FirmaID zu, wozu es ausreicht, den Fall

```
IF UPDATE(FirmaID)
```

überhaupt nicht zu behandeln. Hingegen können und sollten Firmenname, Rechtsform und Beteiligungsquote immer aktualisiert werden, also auch dann, wenn der Anwender zugleich vergeblich versucht hat, den Primärschlüssel zu ändern. Die Beteiligungsquote soll allerdings nur dann aktualisiert werden, wenn die Muttergesellschaft keinen Nullwert aufwies. Dadurch wird verhindert, dass der Benutzer dem Wurzelelement eine Beteiligungsquote verpasst, was keinen Sinn machen würde.

In einer Access-Datenbank kann man zusätzlich auf Formularebene mit einer Ereignisprozedur das Steuerelement für die Muttergesellschaft ausblenden, wenn es sich um ein Wurzelelement handelt (*frmUnternehmen: Form_Current* in *Baumstruktur.mdb*). In der dortigen Ereignisprozedur wird die Wurzelelementeigenschaft an einem zusätzlichen Attribut „Ebene“ festgemacht, wobei dieses für Wurzelelemente 0 sein soll. Abbildung 14 zeigt den Effekt.

Abbildung 14: Unterstützung der Wurzelelementeigenschaft in einem Formular

(b) Verhinderung eines beliebigen Rings

Handelt es sich um einen Ring, der das Wurzelelement (Ebene 0) nicht einschließt und auch nicht zu einer Netzwerkstruktur führt (→ c), dann muss dieser Ring vom übrigen Baum vollständig unabhängig sein. Nehmen wir in Abbildung 10 an, die Deutsche Rad AG sei zusätzlich noch Kind von Meyer's Radreisen, dann entstünde ein Ring:

Deutsche Rad AG – Power Bike – Meyer's Radreisen – Deutsche Rad AG

Da die Deutsche Rad AG aber in einer dem Datenmodell der Abbildung 12 gehorchenden Datenbank nicht zwei Mütter haben kann, müsste die Beteiligung von Inter Bike an der Deutschen Rad AG gekappt werden, wodurch zwei separate Teilstrukturen entstehen. Auch diese Ringbildung ist im SQL Server 2000 durch eine – wenn auch etwas kompliziertere – Konstruktion aus Trigger und gespeicherter Prozedur zu unterbinden. Vom neuen INSTEAD OF UPDATE-Trigger werden hier nur die Anweisungen wiedergegeben, welche die neue Funktionalität ausmachen. Der vollständige Trigger findet sich in der Datei BaumstrukturBeliebigenRingVerhindern.sql

```
/* UPDATE-Trigger: Beliebigen Ring verhindern */
CREATE TRIGGER Unternehmen_I_U
ON Unternehmen INSTEAD OF UPDATE
AS
...
DECLARE @Topologietyp VARCHAR(15)           /* Topologietyp */
...
/* Verhinderung eines beliebigen Rings */
IF UPDATE(Muttergesellschaft)
     BEGIN
     EXEC usp_Topologie @FirmaID, @Muttergesellschaft_neu, @Topologietyp
        OUTPUT
     IF @Topologietyp = 'Baum'
          UPDATE Unternehmen
             SET Muttergesellschaft = @Muttergesellschaft_neu
             WHERE FirmaID = @FirmaID
     END
/* Zugelassene Aktualisierungen */
...
```

Der wesentliche Unterschied zum vorherigen Trigger unter (a) besteht darin, dass die Aktualisierung der Muttergesellschaft jetzt nicht mehr nur dann abgewiesen wird, wenn dadurch mehr als ein Wurzelelement entstehen würde, sondern dass jeder beliebige Ring, der bei einer Aktualisierung der Muttergesellschaft entstünde, durch eine gespeicherte Prozedur usp_Topologie aufgespürt wird. Wenn sie einen Ring findet, gibt sie dieses als Topologietyp zurück.

```
/* Datei: BaumstrukturTopologieErmitteln.sql */
CREATE PROC usp_Topologie @Firma INT, @Mutter_neu INT, @Topologietyp
VARCHAR(15) OUTPUT
AS
DECLARE @Mutter_von_Mutter INT
SELECT @Mutter_von_Mutter = Muttergesellschaft
   FROM Unternehmen
   WHERE FirmaID = @Mutter_neu
IF @Mutter_von_Mutter = @Firma
     SET @Topologietyp = 'Ring'       /* Ringstruktur */
ELSE
     IF @Mutter_von_Mutter IS NULL
          SET @Topologietyp = 'Baum'   /* (Multiple) Baumstruktur */
     ELSE
          EXEC usp_Topologie @Firma, @Mutter_von_Mutter, @Topologietyp OUTPUT
```

Der Prozedur werden der Primärschlüssel (@Firma) und der neue Fremdschlüssel (@Mutter_neu) des bearbeiteten Elements sowie @Topologietyp als anfangs leerer Output-Parameter übergeben. Das Elternelement dieses Elements wird in der lokalen Variablen @Mutter_von_Mutter abgelegt. Die Prozedur ruft sich rekursiv selbst auf, um immer wieder erneut die Mutter zur Mutter zu suchen. Findet sie auf diesem Weg

als Mutter das bearbeitete Element selbst (@Mutter_von_Mutter = @Firma), dann muss es sich um einen Ring handeln, und die Prozedur liefert dem Trigger den Wert „Ring“ als Output-Parameter zurück. Der Trigger belässt es daraufhin bei der alten Muttergesellschaft. Findet die gespeicherte Prozedur hingegen eine Mutter mit leerem Fremdschlüssel (@Mutter_von_Mutter IS NULL), dann handelt es sich um ein Wurzelelement. Es ist kein Ring entstanden und dem Trigger wird „Baum“ zurückgemeldet. Dann und nur dann ändert der Trigger den Fremdschlüssel Muttergesellschaft.

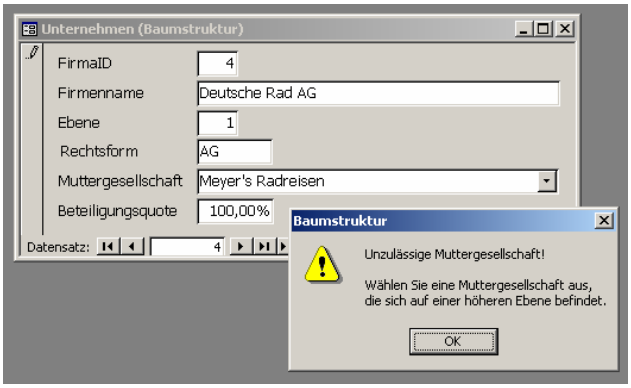

Abbildung 15: Unterbindung einer Ringstrukturen in einem Formular

Eine clientseitige Lösung ist dann besonders einfach, wenn in der Unternehmenstabelle eine Spalte für die Ebene mitgeführt wird, auf der sich das jeweilige Unternehmen in der Hierarchie befindet. Mit der Ereignisprozedur *frmUnternehmen: Form_BeforeUpdate* der Datenbank Baumstruktur.mdb werden nur Muttergesellschaften akzeptiert, die sich auf einer hierarchisch höheren Ebene oder zumindest auf derselben Ebene befinden.

(c) Netzwerkstruktur

Schließt der Ring das Wurzelelement nicht ein (a) und bildet er auch keinen eigenständigen Teilgraphen (b), dann muss er Bestandteil einer mit Abb. 9.12 unvereinbaren Netzwerkstruktur sein. Wäre in Abbildung 10 die Deutsche Rad AG nicht nur von Inter Bike, sondern zusätzlich von Meyer’s Radreisen abhängig, dann gäbe es zwar einen mit dem „Rest“ verbundenen Ring. Wir hätten jetzt aber eine Netzwerkstruktur (→ 4.) und keinen Baum mehr, da die Deutsche Rad AG zwei Mütter hätte. Das ist insofern unproblematisch, als diese Datenkonstellation in einer mit Abbildung 12 kompatiblen relationalen Datenbank überhaupt nicht zu speichern ist. Hier wird die Baumstruktur also vom Datenmodell durchaus gesichert. In diesem Fall ist der eingangs dieses Abschnitts zitierte Einwand von Dorsey und Hudicka also nicht stichhaltig.

3.2.2.3 Verhinderung einer multiplen Baumstruktur

Dorsey und Hudicka stellen zu Recht fest, dass eine Baumstruktur bei mehr als einem Wurzelelement in mehrere unabhängige Teilbäume zerfällt. Wenn die Deutsche Rad AG z.B. keine Mutter mehr hätte, dann gäbe es zwei unverbundene Teilbäume. Obwohl eine solche multiple Baumstruktur gegen kein Strukturprinzip für strenge Hierarchien

verstößt, könnte man im Objektmodell mit einem gesonderten Stereotyp wie <<multiple Baumstruktur>> auf sie aufmerksam machen.

In einem SQL-Server wäre eine multiple Baumstruktur dadurch zu unterbinden, dass die Bedingung für die Aktualisierbarkeit der Muttergesellschaft im INSTEAD OF UPDATE-Trigger verschärft würde:

```
/* Datei: BaumstrukturBeliebigenRingVerhindern.sql */
IF UPDATE(Muttergesellschaft) AND @Muttergesellschaft_neu IS NOT NULL
```

Hiergegen spricht jedoch nicht nur, dass es keine zwingenden topologischen Gründe für eine Verdikt gegen multiple Baumstrukturen gibt, sondern auch ein ganz pragmatisches datenpflegerisches Argument. Wenn irgendein anderes Element zum Wurzelelement werden soll, dann muss es zumindest kurzeitig zwei Wurzelelemente (und damit eine multiple Baumstruktur) geben – das neue und das alte – bevor das alte Wurzelelement unter ein anderes Element umgehängt wird. Mit der folgenden Versuchsanordnung kann man sich hiervon überzeugen:

- Die Deutsche Rad AG soll zum neuen Wurzelelement werden
- Der zwischenzeitlich dadurch entstehende Teilbaum unter Inter Bike soll anschließend z.B. unter Zuse Zweirad gehängt werden.

Dieses Manöver gelingt nur mit dem im Abschnitt 3.2.2.2 (b) beschriebenen INSTEAD OF UPDATE-Trigger.

Von größerer praktischer Bedeutung dürfte wiederum sein, den Benutzer bei der Datenpflege vor einer versehentlichen Eingabe eines weiteren Wurzelelements zu warnen. Dies geschieht am leichtesten clientseitig wie in *Baumstuktur.mdb – frmUnternehmen: Form_BeforeUpdate*.

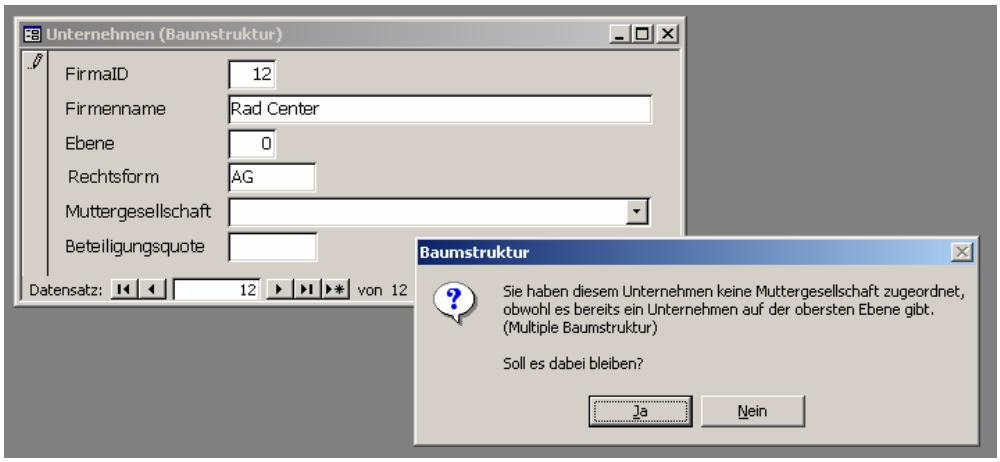

Abbildung 16: Warnung vor einer multiplen Baumstruktur

3.2.2.4 Fazit

Das logische Datenmodell allein schließt nicht sämtliche denkbaren Verletzungen einer Baumstruktur aus. Der Gefahr, dass ein Unternehmen seine eigene Mutter ist, kann mit einem Constraint bzw. einer Gültigkeitsregel einfach vorgebeugt werden. Ebenso kann

mit server- und clientseitiger Programmierung verhindert werden, dass sich ein Ring in eine Baumstruktur einschleicht. Da multiple Baumstrukturen nicht grundsätzlich gegen das Hierarchieprinzip verstoßen, reicht es aus, den Anwender mit einem diesbezüglichen Hinweis zu warnen. Unterbunden werden sollten sie nicht.

4 Netzwerkstruktur

Wie bereits erwähnt, sind Netzwerkstrukturen allgemeiner als Baumstrukturen, weil sie die Beschränkung auf ein Elternelement aufheben.

- Jedes Element kann nicht nur mehrere Kindelemente, sondern auch mehrere Elternelemente haben. Mindestens ein Element muss mehr als ein Elternelement haben, sonst handelt es sich um eine Baumstruktur. Mindestens ein Element darf kein Elternelement haben – es muss also ein Wurzelelement geben – sonst handelt es sich um eine Ringstruktur. Ferner darf mindestens ein Element kein Kindelement haben – es muss also ein Blattelement geben – sonst handelt es sich ebenfalls um eine Ringstruktur.
- Gibt es mehr als ein Wurzelelement und sind die von ihnen abhängigen Elemente voneinander völlig separiert, liegt eine multiple Netzwerkstruktur vor [DoHu99: 260].

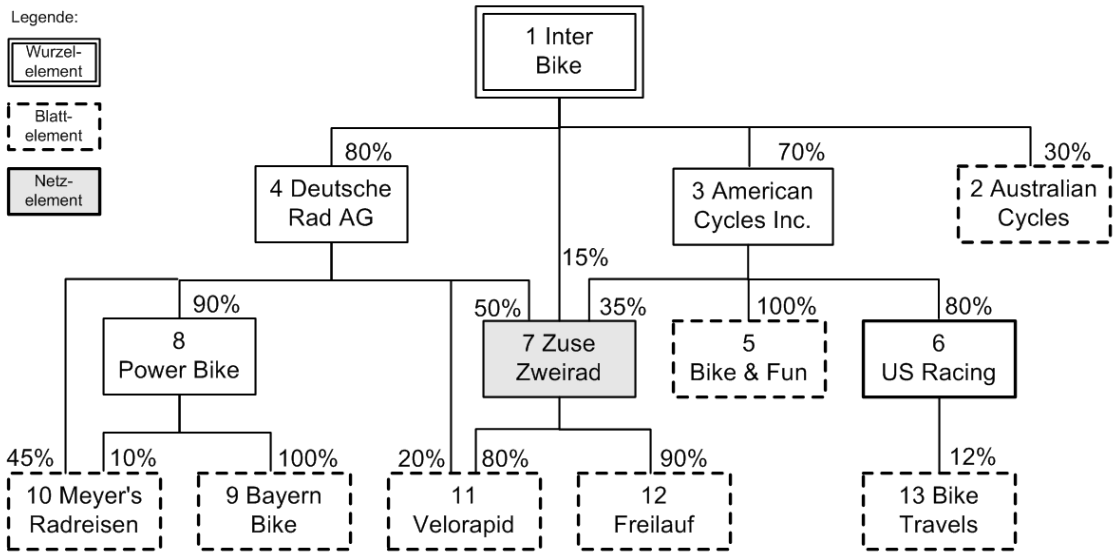

Abbildung 17: Graph einer Netzwerkstruktur

Anders als in Abbildung 10 hat Zuse Zweirad in Abbildung 17 nicht nur die Deutsche Rad AG, sondern auch American Cycles Inc. und Inter Bike als Mütter. Man beachte ferner, dass die Mutter Inter Bike auf einer höheren Ebene angesiedelt ist als die beiden anderen Mütter. Velorapid hängt jetzt von Zuse Zweirad und der Deutschen Rad AG statt von Power Bike ab. An Meyer's Radreisen ist außer Power Bike auch noch die Deutsche Rad AG beteiligt. Hinzugekommen ist noch Freilauf als Tochter von Zuse Zweirad. Aus Gründen der sprachlichen Vereinfachung bezeichnen wir im Folgenden

solche Element, die mehr als eine Mutter haben, als Netzelemente, die übrigen als Baumelemente. Netz- und Baumelement können zugleich auch Blattelemente sein. Netzelemente können jedoch niemals zugleich Wurzelelement sein.

Mit diesen gegenüber der Baumstruktur erweiterten Abhängigkeiten verfügen wir geeignete Beispiele zum Test der zwangsläufig komplexer werdenden Trigger.

Die Verflechtungsmatrix einer Netzwerkstruktur unterscheidet sich von der Baumstruktur dadurch, dass in mindestens einer Zeile mehrere Elemente auftreten. In Abbildung 18 sind diese Elemente durch ein N hervorgehoben, um zu verdeutlichen, welche Beziehungen für den Netzwerkcharakter verantwortlich sind. Auch hier wird deutlich, dass die Elemente 7, 10 und 11 Netzelemente sind.

	Elternelemente												
Kindelemente	1 Inter Bike	2 Australian Cycles	3 American Cycles Inc.	4 Deutsche Rad AG	5 Bike & Fun	6 US Racing	7 Zuse Zweirad	8 Power Bike	9 Bayern Bike	10 Meyer's Radreisen	11 Velorapid	12 Freilauf	13 Bike Travel
1 Inter Bike													
2 Australian Cycles	**B**												
3 American Cycles Inc.	**B**												
4 Deutsche Rad AG	**B**												
5 Bike & Fun			**B**										
6 US Racing			**B**										
7 Zuse Zweirad	**N**		**N**	**N**									
8 Power Bike				**B**									
9 Bayern Bike								**B**					
10 Meyer's Radreisen				**N**				**N**					
11 Velorapid				**N**			**N**						
12 Freilauf							**B**						
13 Bike Travel						**B**							

B Baumelement **N** Netzelement

Abbildung 18: Verflechtungsmatrix einer Netzwerkstruktur

Eine Netzwerkstruktur, in der alle Mütter von Netzelementen jeweils auf der nächst höheren Ebene liegen, kann man als ebenenhomogenes Netzwerk bezeichnen. Gilt diese Einschränkung nicht – was häufig vorkommen dürfte – kann man von einem ebenenheterogenen Netzwerk sprechen. Für die Daten- und Objektmodellierung ist dieser Unterschied irrelevant. Ein ebenenheterogenes Netzwerk stellt aber höhere Anforderungen an die graphische Aufbereitung in einem Organigramm, da die Positionierung der Knoten und die möglichst überschneidungsfreie Linienführung der Kanten über mehrere Ebenen hinweg optimiert werden muss.

Das Stichwort graphische Aufbereitung sei zu dem Hinweis genutzt, dass Abbildung 17 händisch mit Microsoft Visio 2002 erstellt wurde. Der Organigramm-Assistent von Visio kann nämlich aus einer Datenquelle nur Baumstrukturen korrekt auslesen. Sobald ein einziges Element mehr als ein Elternelement aufweist, bricht der Organigramm-Assistent die vollständige Aufbereitung des Organigramms ab. Der Organigramm-Assistent von Visio ist also mit Netzwerkstrukturen überfordert. Dasselbe gilt für Ringstrukturen (→ Abschnitt 5) und erst recht für hybride rekursive Strukturen (→ Abschnitt 8), die sich aus Netzwerk-, Ring- und Baumstrukturen zusammensetzen. Für die graphische Aufbereitung derart komplexer Strukturen benötigt man spezielle Softwarelösungen wie z.B. Visual OrgChart [Somm01].

4.1 Datenmodell und Klassenmodell

Das Klassenmodell der Netzwerkstruktur in Abbildung 19 links ähnelt auf den ersten Blick dem der Baumstruktur (Abbildung 12). Die Multiplizität der Beziehung ist bei der Netzwerkstruktur jedoch auf beiden Seiten (0,n), während sie bei der Baumstruktur auf Seiten des Mutterunternehmens (0,1) oder (1,1) ist. Allerdings haben wir in Abbildung 19 links die Beteiligungsquote unterschlagen. Sobald die die Netzwerkstruktur konstituierende rekursive Beziehung jedoch über eigene Attribute wie z.B. die Beteiligungsquote verfügt, müssen diese in einer assoziativen Klasse untergebracht werden (Abb. 9.19 rechts).

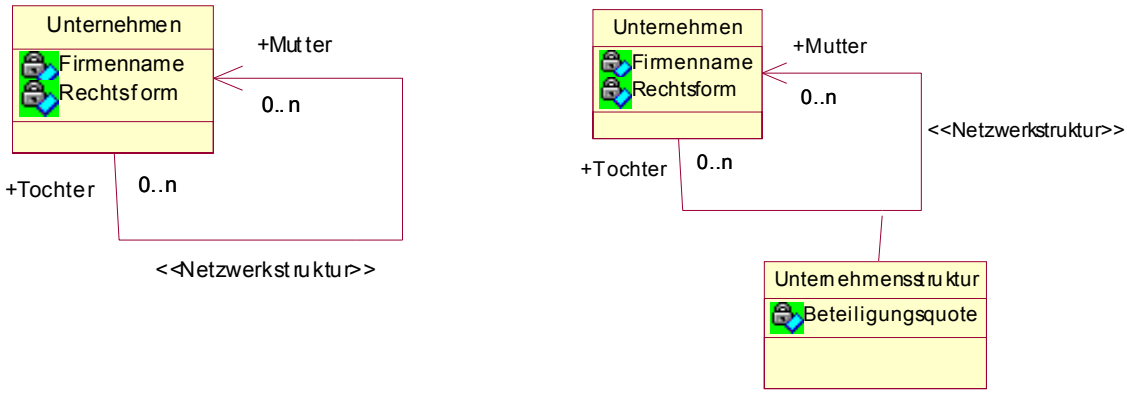

Abbildung 19: Klassenmodell einer Netzwerkstruktur

Das Datenmodell einer Netzwerkstruktur lässt sich auch dann nur scheinbar mit einer Angelhakenbeziehung wie in Abbildung 20 links darstellen, wenn es keine weiteren, die Struktur beschreibenden Attribute gibt. Da mindestens ein Unternehmen mehrfach als Mutter auftritt, kann in einer einzigen Entität nicht mehr nur FirmaID als Primärschlüssel fungieren. Konsequenterweise versucht ERwin auch gar nicht erst, die FirmaID zusätzlich noch als Fremdschlüssel in diese Entität einzutragen; die Kardinalitäten dieser N:M-Angelhakenbeziehung sind zudem – ebenfalls zurecht – nicht einstellbar.

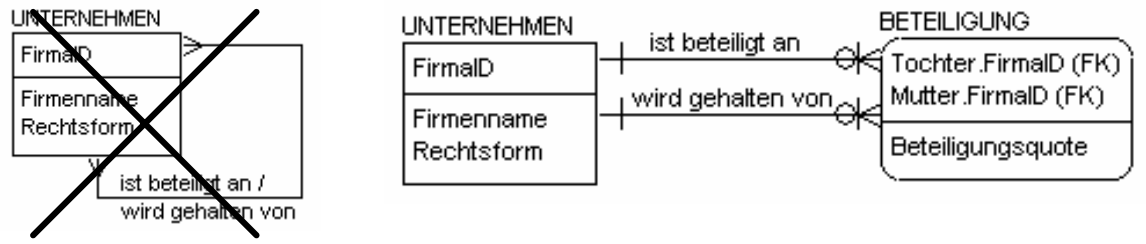

Abbildung 20: Datenmodell einer Netzwerkstruktur

Eine Netzwerkstruktur wird deshalb im Datenmodell auch auf logischer Ebene durch Hinzunahme einer Strukturentität modelliert (BETEILIGUNG in Abbildung 20 rechts). Auf physikalischer Ebene ist eine entsprechende Tabelle ohnehin unvermeidlich. Die Stammdatenentität UNTERNEHMEN hat zwei parallele Beziehungen zur Strukturentität. Die Aussagen zu den Kardinalitäten müssen sich auf beide Beziehungen erstrecken.

- Masterseitige Kardinalitäten (1,1): Im Gegensatz zur Baumstruktur können die masterseitigen Minimalkardinalitäten nicht 0, sondern müssen 1 sein, weil die Fremdschlüssel Tochter.FirmaID und Mutter.FirmaID den zusammengesetzten Primärschlüssel der Strukturtabelle bilden und deshalb keine Nullwerte annehmen können.
  - Will man die auf der höchsten Ebene angesiedelten Unternehmen auch in der Beteiligungstabelle, dann müssen die Fremdschlüssel Mutter.FirmaID und Tochter.FirmaID übereinstimmen oder wieder einen hierfür reservierten fiktiven Wert wie z.B. 0 annehmen (→ *Netzwerkstuktur.mdb – tblBeteiligung und qryBeteiligung*).
  - Verzichtet man hierauf, erhält man die Wurzelemente nur bei einem Outer Join über die Beziehung „ist beteiligt an".
- Detailseitige Kardinalitäten (0,n): Die detailseitigen Minimalkardinalitäten müssen bei beiden Beziehungen 0 sein, weil sonst Ringstrukturen entstünden [DoHu99: 263].
  - Eine detailseitige Minimalkardinalität von 1 in der Beziehung „ist beteiligt an" würde bedeuten, dass alle Primärschlüsselwerte FirmaID als Fremdschlüssel Tochter.FirmaID auftauchen, also sämtliche Unternehmen auch Tochtergesellschaften sind. Dann gäbe es kein Wurzelelement, was zu Ringstrukturen führen würde.
  - Eine detailseitige Minimalkardinalität von 1 in der Beziehung „wird gehalten von" würde bedeuten, dass alle Primärschlüsselwerte FirmaID als Fremdschlüssel Mutter.FirmaID auftauchen, also sämtliche Unternehmen auch Muttergesellschaften sind. Dann gäbe es kein Blattelement, was ebenfalls zu Ringstrukturen führen würde.
  - Umgekehrt garantieren detailseitige Minimalkardinalität von 0 nicht, dass es überhaupt keinen Ring in einer Netzwerkstruktur gibt (→ Abschnitt 4.2.2.2), son-

dern nur, dass nicht sämtliche Unternehmen der obersten bzw. der untersten Ebene an Ringstrukturen beteiligt sind.

Ansonsten gelten die Aussagen zu den Kardinalitäten bei Baumstrukturen analog.

4.2 Implementierung in einer relationalen Datenbank

Die Access-Datenbank Netzwerkstruktur.mdb enthält die Beispieldaten der Abbildung 17. Die entsprechende SQL Server-Datenbank samt Datensätzen kann im Query Analyzer wieder mit einem SQL-Skript der erzeugt werden (Datei NetzwerkstrukturDatenbankAnlegen.sql).

```
/* Datei: NetzwerkstrukturDatenbankAnlegen.sql */
CREATE TABLE Unternehmen (
  FirmaID       INT NOT NULL,
  Firmenname    VARCHAR(80) NOT NULL,
  Rechtsform    VARCHAR(50) NULL,
    CONSTRAINT PK_Unternehmen PRIMARY KEY (FirmaID))

CREATE TABLE Beteiligung (
   TochterID            INT NOT NULL,
   MutterID             INT NOT NULL,
   Beteiligungsquote    DECIMAL(5,4) NULL,
   CONSTRAINT PK_BETEILIGUNG
      PRIMARY KEY (TochterID, MutterID),
   CONSTRAINT FK_Beteiligung_Unternehmen_Tochtergesellschaft
      FOREIGN KEY (TochterID)
      REFERENCES Unternehmen,
   CONSTRAINT FK_Beteiligung_Unternehmen_Muttergesellschaft
      FOREIGN KEY (MutterID)
      REFERENCES Unternehmen,
   CONSTRAINT CK_Muttergesellschaft
      CHECK (MutterID <> TochterID))
```

Die Tabelle Unternehmen ist selbstredend. Die Tabelle Beteiligung wird mit den aus Abbildung 20 ersichtlichen Spalten angelegt. Die Constraints erfüllen dabei folgende Aufgaben.

- Der erste Constraint definiert den zusammengesetzten Primärschlüssel.
- Der zweite und dritte Constraint sorgen für die rekursive Beziehung, wobei die Fremdschlüssel Tochtergesellschaft und Muttergesellschaft beide jeweils den Primärschlüssel FirmaID der Tabelle Unternehmen referenzieren.
- Der vierte Constraint wird im Abschnitt 4.2.2.1 erläutert.

4.2.1 Referenzielle Integritätsaktionen

In einer Netzwerkstruktur schlägt sich die Rekursivität nicht in der Stammdatentabelle, sondern in der Strukturtabelle nieder. Deshalb sind auch beide Tabellen von Löschoperationen und Primärschlüsselaktualisierungen betroffen. Während bei der Baumstruktur, die durch einen selbstreferenziellen Fremdschlüssel in der Stammdatentabelle realisiert wird, die referentielle Integrität deklarativ überhaupt nicht durchsetzbar ist (→ 3.2.1), ist dieses hier bei Wurzelelementen (→ 4.2.1.1) und Blattelementen (→ 4.2.1.2) möglich. Bei Baum- und Netzelementen muss jedoch wiederum mit einem Trigger gearbeitet werden.

4.2.1.1 Löschen von Wurzelelementen

Wenn das Wurzelelement Inter Bike in Abbildung 17 gelöscht wird, sollte dieses erstens aus der Stammdatentabelle Unternehmen verschwinden und zweitens in der Strukturtabelle Beteiligung die vier Datensätze mitlöschen, in denen sein Primärschlüssel als Fremdschlüssel MutterID auftritt. Danach sind die Deutsche Rad AG und American Cycles die beiden Wurzelemente der durch die übrigen Töchter gebildeten Netzwerkstruktur, mit Ausnahme von Australian Cycles, das völlig unverbunden daneben steht. Dieses kaskadierende Löschen kann mit deklarativer referentieller Integrität (DRI) erzwungen werden.

```
/* Datei: NetzwerkstrukturDRIWurzelelemente.sql */
...
/* Tabelle Beteiligung erzeugen */
CREATE TABLE Beteiligung (
...
   CONSTRAINT FK_Beteiligung_Unternehmen_Tochtergesellschaft
      FOREIGN KEY (TochterID)
      REFERENCES Unternehmen
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
   CONSTRAINT FK_Beteiligung_Unternehmen_Muttergesellschaft
      FOREIGN KEY (MutterID)
      REFERENCES Unternehmen
      /* Beziehungen von Wurzelelementen werden gelöscht */
      ON DELETE CASCADE
      /* Primärschlüssel von Wurzelelementen werden aktualisiert */
      ON UPDATE CASCADE,
...
```

Allerdings ist diese DRI nicht sehr mächtig, da sie für sämtliche Nicht-Wurzelelemente – also die überwältigende Mehrheit aller Elemente – versagen muss, weil Nicht-Wurzelelemente gerade dadurch charakterisiert sind, dass ihr Primärschlüssel als Fremdschlüssel TochterID in der Strukturtabelle vorkommt, für den jedoch die die Löschverhinderung definiert wurde.

4.2.1.2 Löschen von Blattelementen

Genau umgekehrt verhält es sich, wenn das kaskadierende Löschen für den Fremdschlüssel TochterID deklariert wird, während für den Fremdschlüssel MutterID NO ACTION gilt. Dann sind nur Blattelemente löschbar. Das Löschen der Fa. Freilauf (Abbildung 17) würde diese nicht nur aus der Tabelle Unternehmen entfernen, sondern auch den ihre Abhängigkeit von Zuse Zweirad repräsentierenden Datensatz aus der Tabelle Beteiligung.

```
/* Datei: NetzwerkstrukturDRIBlattelemente.sql */
...
/* Tabelle Beteiligung erzeugen */
CREATE TABLE Beteiligung (
...
   CONSTRAINT FK_Beteiligung_Unternehmen_Tochtergesellschaft
      FOREIGN KEY (TochterID)
      REFERENCES Unternehmen
      /* Beziehungen von Blattelementen werden gelöscht */
      ON DELETE CASCADE
      /* Primärschlüssel von Blattelementen werden aktualisiert */
      ON UPDATE CASCADE,
```

```
   CONSTRAINT FK_Beteiligung_Unternehmen_Muttergesellschaft
      FOREIGN KEY (MutterID)
      REFERENCES Unternehmen
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
...
```

Auf diese Art und Weise, sind auch Blattelemente mit mehreren Müttern wir z.B. Meyer's Radreisen in Abbildung 17 zu löschen. Nicht zu löschen sind hingegen Unternehmen, deren Primärschlüssel als Fremdschlüssel MutterID in Erscheinung tritt, m.a.W. sämtliche Nicht-Blattelemente.

4.2.1.3 Löschen anderer Elemente als Wurzel- und Blattelemente

Nahe liegend wäre die Idee, die beiden soeben vorgestellten DRI-Lösungen für Wurzelelemente und Blattelemente zu kombinieren. Dieser Versuch wird jedoch vom SQL Server wieder mit der bereits bekannten Fehlermeldung zurückgewiesen:

```
„Das Einführen der FOREIGN KEY-Einschränkung 'FK_Unternehmen_Muttergesell-
schaft_FirmaID' für die Unternehmen-Tabelle kann Schleifen oder mehrere kaska-
dierende Pfade verursachen. Geben SIE ON DELETE NO ACTION oder ON UPDATE NO
ACTION an, oder ändern Sie andere FOREIGN KEY-Einschränkungen".
```

Da es in einer Netzwerkstruktur normalerweise mehr Blatt- als Wurzelelemente geben wird, wird man sich eher für das deklarative kaskadierende Löschen von Blattelementen entscheiden (→ 4.2.1.2). Damit ist jedoch das Löschen der meisten Elemente einer Netzwerkstruktur keine Lösung gefunden. Hier hilft wiederum nur ein INSTEAD OF DELETE-Trigger.

Um besser zu verstehen, was der Trigger leisten soll, nehmen wir an, dass US Racing in Abbildung 17 gelöscht werden soll. Dann müsste nicht nur der Datensatz für dieses Unternehmen aus der Tabelle Unternehmen entfernt werden

```
FirmaID    Firmenname    Rechtsform
   6       US Racing        AG
```

sondern auch der Datensatz in der Tabelle Beteiligung, der die Beziehung zur Mutter American Cycles abbildet:

```
TochterID  MutterID   Beteiligungsquote
   6          3            0,8
```

Schließlich muss der Fremdschlüssel MutterID im Datensatz

```
TochterID  MutterID   Beteiligungsquote
   13         6            0,12
```

von 6 auf 3 geändert werden, damit Bike Travels nach dem Wegfall von US Racing direkt unter American Cycles gehängt wird. Nachdem wir diese Aufgabenstellung in einen Trigger umgesetzt haben, soll dann abschließend auch die Beteiligungsquote von American Cycles an Bike Travels automatisch neu berechnet werden.

```
/* Datei: NetzwerkstrukturElementeLöschen_1.sql */

/* DELETE-Trigger anlegen */
CREATE TRIGGER Unternehmen_I_D
ON Unternehmen
INSTEAD OF DELETE
AS
DECLARE @Löschen INT         /* zu löschendes Elements */
DECLARE @Mutter INT          /* Mutter des zu löschenden Elements */
```

```
SELECT @Löschen = FirmaID FROM deleted
SELECT @Mutter = MutterID FROM Beteiligung WHERE TochterID = @Löschen
IF @Mutter IS NOT NULL
      /* Löschen von anderen als Wurzelelementen */
      BEGIN
      /* Umhängen der Kindelemente des zu löschenden Elements unter die Großmutter */
      UPDATE Beteiligung SET MutterID = @Mutter WHERE MutterID = @Löschen
      /* Löschen des Elements in der Strukturtabelle*/
      DELETE FROM Beteiligung WHERE TochterID = @Löschen
      /* Löschen des Elements in der Stammdatentabelle*/
      DELETE FROM Unternehmen WHERE FirmaID = @Löschen
      END
ELSE
      /* Löschen von Wurzelelementen */
      BEGIN
      /* Löschen des Wurzelelements in der Strukturtabelle */
      DELETE FROM Beteiligung WHERE MutterID = @Löschen
      /* Löschen des Wurzelelements in der Stammdatentabelle */
      DELETE FROM Unternehmen WHERE FirmaID = @Löschen
      END
```

Das Löschen von Wurzelelementen (ELSE-Zweig) sollte hier schon deshalb zugelassen werden, weil eine Netzwerkstruktur erstens mehr als ein Wurzelelement haben kann und zweitens beim Löschen eines solchen nicht zwangläufig eine multiple Netzwerkstruktur entstehen muss. Selbst wenn dies der Fall wäre, wäre darin kein Verstoß gegen die Netztopologie zu sehen.

Das Löschen von anderen als Wurzelelementen erfolgt dies in drei Schritten. Erstens werden in der Beteiligungstabelle alle das Elternelement repräsentierenden Fremdschlüssel (MutterID), die auf das zu löschende Element verweisen mit dem Primärschlüssel des Elternelements des löschenden Elements aktualisiert.

```
UPDATE Beteiligung SET MutterID = @Mutter WHERE MutterID = @Löschen
```

Zweitens werden in der Beteiligungstabelle alle Datensätze gelöscht, in denen das zu löschende Element als Kindelement auftaucht.

```
DELETE FROM Beteiligung WHERE TochterID = @Löschen
```

Und drittens wird das zu löschende Element aus der Unternehmenstabelle entfernt.

```
DELETE FROM Unternehmen WHERE FirmaID = @Löschen
```

Diese Lösung funktioniert für

- Baumelemente wie US Racing, aber auch für
- Wurzelelemente wie Inter Bike und
- Blattelemente mit nur einer Mutter wie Bayern Bike oder mit mehreren Müttern wie Meyer's Radreisen.

Sie funktioniert nicht für

- Baumelemente wie Power Bike, die eine Tochter (hier: Meyer's Radreisen) mit einer direkten Beziehung zur eigenen Mutter (hier: Deutsche Rad AG) haben
- Netzelemente mit Töchtern wie Zuse Zweirad.

Der erste Defekt ist einfach zu erklären und zu beheben. Beim Löschen von Power Bike würde der UPDATE-Befehl versuchen, Meyer's Radreisen unter die Deutsche Rad AG zu hängten. Da zwischen diesen beiden Unternehmen bereits eine direkte Beziehung besteht (Abbildung 21), scheitert der UPDATE-Befehl an dieser Schlüsselverletzung.

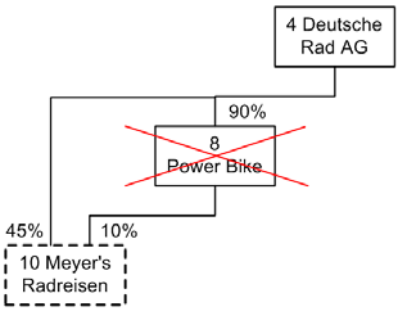

Abbildung 21: Löschen einer zuvor bestehenden Beziehung einer Tochter zur Großmutter

Das Löschen dieser Beziehung vor dem UPDATE-Befehl verhindert die Schlüsselverletzung. Das Löschen ist unschädlich, weil die Beziehung durch das Umhängen ohnehin wiederhergestellt wird.

```
/* Datei: NetzwerkstrukturElementeLöschen_2.sql */
...
      /* Löschen einer zuvor bestehenden Beziehung einer Tochter zu ihrer Großmutter */
      DELETE FROM Beteiligung WHERE MutterID = @Mutter AND TochterID IN
         (SELECT TochterID FROM Beteiligung WHERE MutterID = @Löschen)
      /* Umhängen der Kindelemente des zu löschenden Elements unter die Großmutter */
      UPDATE Beteiligung SET MutterID = @Mutter WHERE MutterID = @Löschen
...
```

Mit einer so einfachen Modifikation ist der zweite Defekt leider nicht zu beheben. Das Löschen von Netzelementen mit diesem Trigger führt zu einer unvollständigen Beziehungsstruktur, wie am Beispiel von Zuse Zweirad gezeigt werden kann.

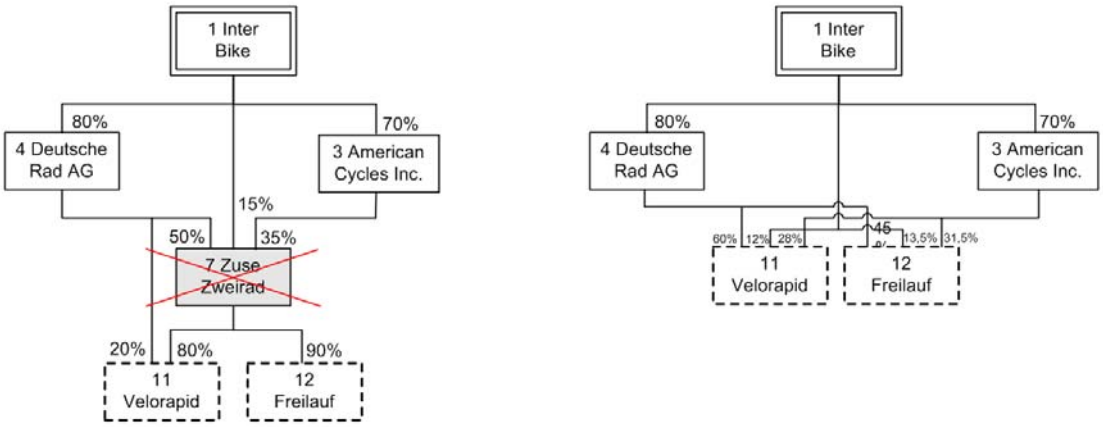

Abbildung 22: Löschen von Netzelementen, die keine Blattelemente sind

Mit dem bisher beschriebenen Trigger sind Velorapid und Freilauf nach dem Löschen von Zuse Zweirad nur von der Deutschen Rad AG abhängig, nicht jedoch von Inter Bike und American Cycles, wie es erforderlich wäre (Abbildung 22 rechts). Das liegt daran, dass der UPDATE-Befehl nur für die erste gefundene Mutter des zu löschenden Elements durchgeführt wird, nicht jedoch für die anderen beiden. Damit Velorapid (und Freilauf) nach dem Löschen von Zuse Zweirad jeweils Töchter der Deutschen Rad AG,

von Inter Bike und von American Cycles werden, muss ein ganz anderer Ansatz gewählt werden.

Ursache der fehlenden Beziehungen von Velorapid zu Inter Bike und American Cycles ist nämlich, dass die Variable @Mutter ein Skalar ist und somit nur ein Elternelement von Zuse Zweirad aufnehmen kann. Da Transact-SQL keine Arrays kennt, verwenden wir Cursor.

```
/* Datei: NetzwerkstrukturElementeLöschen_3.sql */
CREATE TRIGGER Unternehmen_I_D
ON Unternehmen
INSTEAD OF DELETE
AS
DECLARE @Löschen INT         /* zu löschendes Elements */
DECLARE @Mutter INT          /* Mutter des zu löschenden Elements */
DECLARE @Tochter INT         /* Tochter des zu löschenden Elements */
SELECT @Löschen = FirmaID FROM deleted
SELECT @Mutter = MutterID FROM Beteiligung WHERE TochterID = @Löschen
IF @Mutter IS NOT NULL
     /* Löschen von anderen als Wurzelelementen */
     BEGIN
     DECLARE Cursor_Mutter INSENSITIVE CURSOR
        FOR SELECT MutterID FROM Beteiligung WHERE TochterID = @Löschen
     DECLARE Cursor_Tochter INSENSITIVE CURSOR
        FOR SELECT TochterID FROM Beteiligung WHERE MutterID = @Löschen
     /* Cursor-Schleife für alle Mütter */
     OPEN Cursor_Mutter
     FETCH NEXT FROM Cursor_Mutter INTO @Mutter
     WHILE @@fetch_status = 0
          BEGIN
          /* Cursor-Schleife für alle Töchter */
          OPEN Cursor_Tochter
          FETCH NEXT FROM Cursor_Tochter INTO @Tochter
          WHILE @@fetch_status = 0
               BEGIN
               /* Löschen einer zuvor bestehenden Beziehung der Tochter
               zu ihrer Großmutter */
               DELETE FROM Beteiligung WHERE MutterID = @Mutter
                  AND TochterID = @Tochter
               /* Neue Beziehung der Tochter zu ihrer Großmutter */
               INSERT INTO Beteiligung(TochterID, MutterID,
                  Beteiligungsquote) VALUES(@Tochter, @Mutter, NULL)
          FETCH NEXT FROM Cursor_Tochter INTO @Tochter
          END
     CLOSE Cursor_Tochter
     FETCH NEXT FROM Cursor_Mutter INTO @Mutter
     END
     CLOSE Cursor_Mutter
     DEALLOCATE Cursor_Tochter
     DEALLOCATE Cursor_Mutter
     /* Löschen des Elements in der Strukturtabelle */
     DELETE FROM Beteiligung WHERE TochterID = @Löschen
     DELETE FROM Beteiligung WHERE MutterID = @Löschen
     /* Löschen des Elements in der Stammdatentabelle */
     DELETE FROM Unternehmen WHERE FirmaID = @Löschen
     END
ELSE
     /* Löschen von Wurzelelementen */
...
```

Im ELSE-Zweig werden wie bisher Wurzelelemente gelöscht. Im IF-Zweig werden alle Nicht-Wurzelelemente jetzt einheitlich über zwei geschachtelte Cursor gelöscht, obwohl es bei Baumelementen auch ohne Cursor ginge. Mit Cursor_Mutter werden alle Mütter des zu löschenden Elements durchlaufen. Mit Cursor_Tochter werden alle Töchter der Reihe nach unter die jeweilige Mutter des zu löschenden Elements gehängt. Das Umhängen muss hier mit INSERT statt mit UPDATE vorgenommen werden, damit für jede Kombination von Mutter und Tochter des zu löschenden Elements eine neue direkte Beziehung aufgebaut wird. Da wir an dieser Stelle noch nicht die neuen Beteiligungsquoten berechnen wollen, erhalten sie Nullwerte. Nachdem die beiden verschalteten Cursor-Schleifen durchlaufen wurden, werden sämtliche Datensätze für das zu löschende Element aus der Strukturtabelle und aus der Stammdatentabelle entfernt.

Ein weiterer Vorteil der Cursor-Lösung besteht darin, dass jetzt auch die Beteiligungsquoten automatisch neu berechnet werden können. Die neue Beteiligungsquote der Mutter an der Tochter @BQ_M_T_neu ergibt sich aus dem Produkt der Beteiligungsquoten der Mutter am zu löschenden Element @BQ_Mutter und des zu löschenden Elements an der Tochter @BQ_Tochter, erhöht um eine eventuell bereits zuvor bestehende direkte Beteiligung der Mutter des zu löschenden Elements an dessen Tochter @BQ_M_T_alt:

```
SET @BQ_M_T_neu = @BQ_Mutter * @BQ_Tochter + ISNULL(@BQ_M_T_alt, 0)
```

Die Beteiligungsquoten @BQ_Mutter und @BQ_Tochter werden in die beiden Cursor mit aufgenommen, während die Beteiligungsquote @BQ_M_T_alt jeweils mit einem SELECT-Befehl ausgelesen wird.

```
SELECT @BQ_M_T_alt = Beteiligungsquote FROM Beteiligung
     WHERE MutterID = @Mutter AND TochterID = @Tochter
```

Damit liegt jetzt eine vollständige Lösung zur Wahrung der referentiellen Integrität beim Löschen beliebiger Elemente in einer Netzwerksstruktur vor, die zusätzlich auch ein Attribut der Strukturtabelle (hier die Beteiligungsquote) entsprechend neu berechnet. Der Befehl zur Erzeugung dieses Triggers sieht wie folgt aus.

```
/* Datei: NetzwerkstrukturElementeLöschen_4.sql */
CREATE TRIGGER Unternehmen_I_D
ON Unternehmen
INSTEAD OF DELETE
AS
DECLARE @Löschen INT      /* zu löschendes Elements */
DECLARE @Mutter INT       /* Mutter des zu löschenden Elements */
DECLARE @Tochter INT      /* Tochter des zu löschenden Elements */
/* Beteiligungsquote der Mutter am zu löschenden Element */
DECLARE @BQ_Mutter DECIMAL(5,4)
/* Beteiligungsquote des zu löschenden Elements an der Tochter */
DECLARE @BQ_Tochter DECIMAL(5,4)
/* BQ der Mutter an der Tochter des zu löschenden Elements vor dem Löschen */
DECLARE @BQ_M_T_alt DECIMAL(5,4)
/* BQ der Mutter an der Tochter des zu löschenden Elements nach dem Löschen */
DECLARE @BQ_M_T_neu DECIMAL(5,4)
SELECT @Löschen = FirmaID FROM deleted
SELECT @Mutter = MutterID FROM Beteiligung WHERE TochterID = @Löschen
IF @Mutter IS NOT NULL       /* Löschen von anderen als Wurzelelementen */
     BEGIN
     DECLARE Cursor_Mutter INSENSITIVE CURSOR
       FOR SELECT MutterID, Beteiligungsquote FROM Beteiligung WHERE Tochte-
rID = @Löschen
```

```
      DECLARE Cursor_Tochter INSENSITIVE CURSOR
         FOR SELECT TochterID, Beteiligungsquote FROM Beteiligung
            WHERE MutterID = @Löschen
      /* Cursor-Schleife für alle Mütter */
      OPEN Cursor_Mutter
      FETCH NEXT FROM Cursor_Mutter INTO @Mutter, @BQ_Mutter
      WHILE @@fetch_status = 0
            BEGIN
            /* Cursor-Schleife für alle Töchter */
            OPEN Cursor_Tochter
            FETCH NEXT FROM Cursor_Tochter INTO @Tochter, @BQ_Tochter
            WHILE @@fetch_status = 0
                  BEGIN
                  SELECT @BQ_M_T_alt = Beteiligungsquote FROM Beteiligung
                     WHERE MutterID = @Mutter AND TochterID = @Tochter
                  SET @BQ_M_T_neu = @BQ_Mutter * @BQ_Tochter +
                     ISNULL(@BQ_M_T_alt, 0)
                  DELETE FROM Beteiligung WHERE MutterID = @Mutter
                     AND TochterID = @Tochter
                  INSERT INTO Beteiligung(TochterID, MutterID,
                     Beteiligungsquote) VALUES(@Tochter, @Mutter, @BQ_M_T_neu)
                  SELECT @BQ_M_T_alt = 0
                  FETCH NEXT FROM Cursor_Tochter INTO @Tochter, @BQ_Tochter
                  END
            CLOSE Cursor_Tochter
            FETCH NEXT FROM Cursor_Mutter INTO @Mutter, @BQ_Mutter
            END
      CLOSE Cursor_Mutter
      DEALLOCATE Cursor_Tochter
      DEALLOCATE Cursor_Mutter
      DELETE FROM Beteiligung WHERE TochterID = @Löschen
      DELETE FROM Beteiligung WHERE MutterID = @Löschen
      DELETE FROM Unternehmen WHERE FirmaID = @Löschen
      END
ELSE                        /* Löschen von Wurzelelementen */
      BEGIN
      DELETE FROM Beteiligung WHERE MutterID = @Löschen
      DELETE FROM Unternehmen WHERE FirmaID = @Löschen
      END
```

4.2.1.4 Aktualisierung des Primärschlüssels

Wie bei der Baumstruktur wird auch hier von der Aktualisierung des Primärschlüssels grundsätzlich abgeraten (→ 3.2.1.2). Deshalb wird die Implementierung hier nicht als Empfehlung, sondern nur der Vollständigkeit halber angesprochen.

Wie bereits angedeutet, kann die Aktualisierungsweitergabe deklarativ mit ON UPDATE CASCADE nur für Wurzelelemente (→ 4.2.1.1) oder für Blattelemente (→ 4.2.1.2) durchgesetzt werden, nicht jedoch für beide gleichzeitig und erst recht nicht für alle übrigen Elemente einer Netzwerkstruktur. Hierfür bedürfte es ebenfalls eines Triggers (Datei NetzwerkstrukturKaskadierendesUpdate.sql).

4.2.2 Gewährleistung einer Netzwerkstruktur

Auch beim Datenmodell zur Netzwerkstruktur (Abbildung 20) kann es Verstöße gegen dieselbe geben [DoHu99: 262 f.]:

- Ein Unternehmen könnte seine eigene Mutter sein.
- Der Unternehmensverbund könnte eine Ringstruktur beinhalten.

Die dritte, bei der Baumstruktur erörterte Gefahr mehrerer Wurzelelemente spielt hingegen bei Netzwerkstrukturen keine Rolle.

In den folgenden Abschnitten werden wiederum analoge Lösungen vorgestellt, um diese Verletzungen einer Netzwerkstruktur zu unterbinden.

4.2.2.1 Verhinderung von Selbstreferenzialität

Wenn in einem Datensatz der Tabelle Beteiligung die beiden Fremdschlüssel TochterID und MutterID denselben Wert annehmen würden, wäre dieses Unternehmen seine eigene Mutter, was aber durch den vierten CHECK-CONSTRAINT unterbunden wird.

```
CONSTRAINT CK_Muttergesellschaft CHECK (MutterID <> TochterID)
```

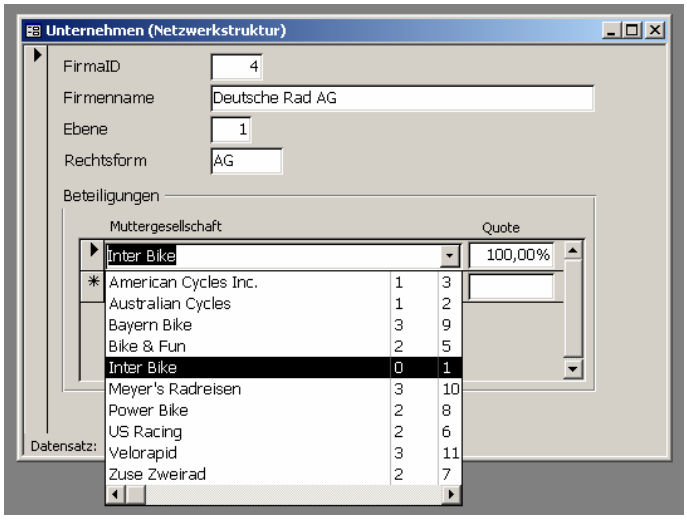

Abbildung 23: Formular für die Datenpflege einer Netzwerkstruktur

In der Access-Datenbank Netzwerkstruktur.mdb findet sich diese Einschränkung in der Tabelleneigenschaft Gültigkeitsregel.

Auch hier stehen nicht nur auf Tabellenebene serverseitige, sondern auch auf Formularebene clientseitige Gegenmittel zur Verfügung. In Abbildung 23 wird die im Hauptformular bearbeitete Gesellschaft aus der Datensatzherkunft der Dropdown-Liste für die Auswahl der Muttergesellschaften im Unterformular ausgeschlossen. Im Gegensatz zur Baumstruktur (Abbildung 13) ist hier ein Unterformular notwendig, da es mehrere Muttergesellschaften geben kann.

4.2.2.2 Verhinderung eines Rings in einer Netzwerkstruktur

Auch das Netzwerk-Datenmodell (Abbildung 20) schließt das Auftreten von Ringstrukturen nicht aus. Der bei Baumstrukturen erörterte Fall (c) ist hier irrelevant, da wir hier eine Netzwerkstruktur nicht ausschließen müssen. Wir können uns also auf die Fallunterscheidung beschränken, ob der Ring ein „Wurzelelement“ einschließt oder nicht.

(a) Verhinderung eines Rings durch das „Wurzelelement“

Wenn es nur ein Wurzelelement gibt und der Ring dieses einschließt, dann ist es per definitionem keine Netzwerkstruktur mehr. In Abbildung 17 wäre dies z.B. bei folgendem Ring der Fall: Inter Bike → Deutsche Rad AG → Power Bike → Inter Bike).

Wenn es zwei oder mehr Wurzelelemente gibt, von denen mindestens eines nicht in eine Ringsstruktur eingebunden ist, dann ist es keine echte Netzwerkstruktur mehr, sondern eine hybrid-rekursive Struktur (→ Abschnitt 8). In Abbildung 24 ist die Deutsche Rad AG außer von Inter Bike noch von Euro Bike abhängig, das aber über einen Ring (Meyer’s Radreisen und Power Bike) seinerseits wieder von der Deutschen Rad AG abhängig ist.

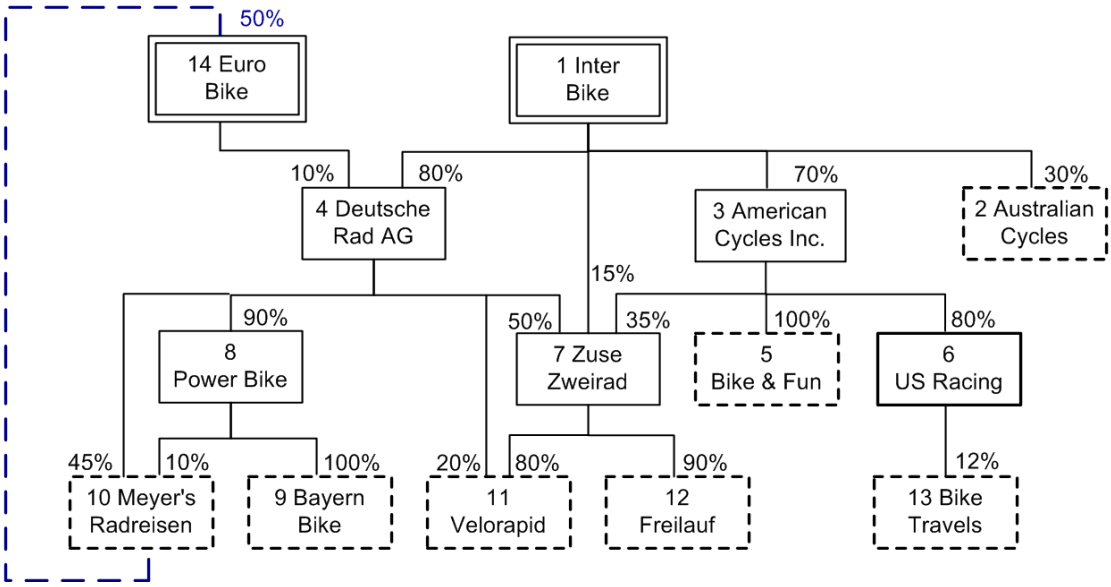

Abbildung 24: Ring durch ein Wurzelelement einer Netzwerkstruktur

Abbildung 24 ist gegenüber Abbildung 17 um zwei Datensätze erweitert, wenn die den Ring verursachende Beziehung vom Meyer’s Radreisen zu Euro Bike ignoriert.

```
INSERT INTO Unternehmen VALUES('14','Euro Bike','AG')
INSERT INTO Beteiligung VALUES('4','14','0.1')
```

Da eine Topologie wie in Abbildung 24 vom Netzwerk-Datenmodell (Abbildung 20) nicht ausgeschlossen wird, muss sie wieder server- oder clientseitig prozedural verhindert werden. Eine clientseitige Lösung findet sich in der Beim Anzeigen-

Ereignisprozedur des Formulars frmUnternehmen der Access-Datenbank Netzwerkstuktur.mdb.

Serverseitig braucht man hier zwei Trigger, da ein Ring durch das Wurzelelement sowohl durch das Einfügen eines neuen Datensatzes in die Tabelle Beteiligung wie auch durch eine Änderung der TochterID eines dort vorhandenen Datensatzes entstehen kann.

Während ein Wurzelelement in einer Baumstruktur sehr leicht am Nullwert des Fremdschlüssels Muttergesellschaft zu erkennen ist, gestaltet sich dies in einer Netzwerkstruktur etwas komplizierter. Ein Wurzelelement erkennt man hier nämlich daran, dass es in der Beteiligungstabelle keinen Datensatz gibt, in dem sein Primärschlüssel als Fremdschlüssel TochterID fungiert. Welche Elemente das sind, lässt sich mit einer OUTER JOIN-Abfrage ermitteln.

```
SELECT Unternehmen.FirmaID
   FROM Beteiligung RIGHT OUTER JOIN Unternehmen
   ON Beteiligung.TochterID = Unternehmen.FirmaID
   WHERE Beteiligung.TochterID IS NULL
```

Das Einfügen eines neuen Datensatzes in die Beteiligungstabelle muss in einem INSTEAD OF INSERT-Trigger also genau dann verhindert werden, wenn dessen TochterID zu einem dieser Wurzelelementprimärschlüssel gehört.

```
/* Datei: NetzwerktrukturRingDurchWurzelelementVerhindern.sql */
/* INSERT-Trigger: Ring durch Wurzelelement verhindern */
CREATE TRIGGER Beteiligung_I_I
ON Beteiligung INSTEAD OF INSERT
AS
DECLARE @TochterID INT
DECLARE @MutterID INT
DECLARE @Beteiligungsquote DECIMAL(5,4)
SELECT @TochterID = TochterID FROM inserted
SELECT @MutterID = MutterID FROM inserted
SELECT @Beteiligungsquote = Beteiligungsquote FROM inserted
/* Verhinderung eines Rings durch das bisherige Wurzelement */
IF @TochterID IN (SELECT Unternehmen.FirmaID
FROM Beteiligung RIGHT OUTER JOIN Unternehmen
ON Beteiligung.TochterID = Unternehmen.FirmaID
WHERE Beteiligung.TochterID IS NULL)
      ROLLBACK TRANSACTION
ELSE
      INSERT INTO Beteiligung VALUES(@TochterID,@MutterID,@Beteiligungsquote)
```

Der Versuch, die den Ring in Abbildung 24 verursachende Beziehung einzufügen, wird jetzt abgewiesen.

```
INSERT INTO Beteiligung VALUES('14','10','0.5')
```

Ein Ring durch ein bisheriges Wurzelelement kann aber nicht nur durch eine INSERT-, sondern auch durch eine UPDATE-Operation entstehen, in Abbildung 24 z.B. dadurch, dass US Racing nicht Mutter von Bike Travels, sondern von Inter Bike wird, was durch einen INSTEAD OF UPDATE-Trigger zu verhindern ist. Ein solcher Trigger ist ebenfalls in der Datei NetzwerktrukturRingDurchWurzelelementVerhindern.sql enthalten. Er ist dem im Abschnitt 3.2.2.2 vorgestellten, analogen Trigger für die Baumstruktur sehr ähnlich. Wir verzichten aber vor allem deshalb auf eine nähere Erläuterung, da der ganze Ansatz, Ringe durch Wurzelelemente zu unterdrücken, zu speziell ist. Vielmehr

müssen wir auch hier wieder eine allgemeine Lösung zur Vermeidung beliebiger Ringe finden.

(b) Verhinderung eines beliebigen Rings

Ein Ansatzpunkt für eine allgemeine Lösung zum Aufspüren von Ringen in Netzwerkstrukturen könnte die Einbeziehung der Ebenen sein, auf denen sich die Elemente befinden. In einer Netzwerkstruktur sollten alle Elemente so platziert sein, dass sie sich auf der höchstmöglichen und tiefstnötigen Ebene befinden. Würde durch eine hinzukommende Beziehung ein Element von einem anderen Element derselben oder einer tieferen Ebene abhängig, dann wäre ein Ring entstanden. (Zur Behandlung rekursiver Strukturen mit Ebenen vgl. [BeMo00: 579-627]).

Die Crux liegt nun darin, dass ein auf diese Weise visuell identifizierter Ring ein Artefakt sein kann. Ein vorgreifender Blick auf Abbildung 26 illustriert diesen Irrtum. Australian Cycles ist jetzt zusätzlich von Power Bike abhängig, das eine Ebene tiefer platziert ist. Bei genauem Hinsehen wird jedoch deutlich, dass gar kein Ring entstanden sein kann, weil Australian Cycles nach wie vor ein Blattelement ist. Würde man Australian Cycles im Graph auf die unterste Ebene ziehen, würde schnell deutlich, dass die Netzwerkstruktur erhalten geblieben ist. Es dürfte sofort einleuchten, dass in komplexeren Graphen ein solches „visuelles Trial-and-Error"-Verfahren scheitern müsste. Auch für eine algorithmische Lösung ist die Ebenennummerierung kein geeigneter Ansatzpunkt.

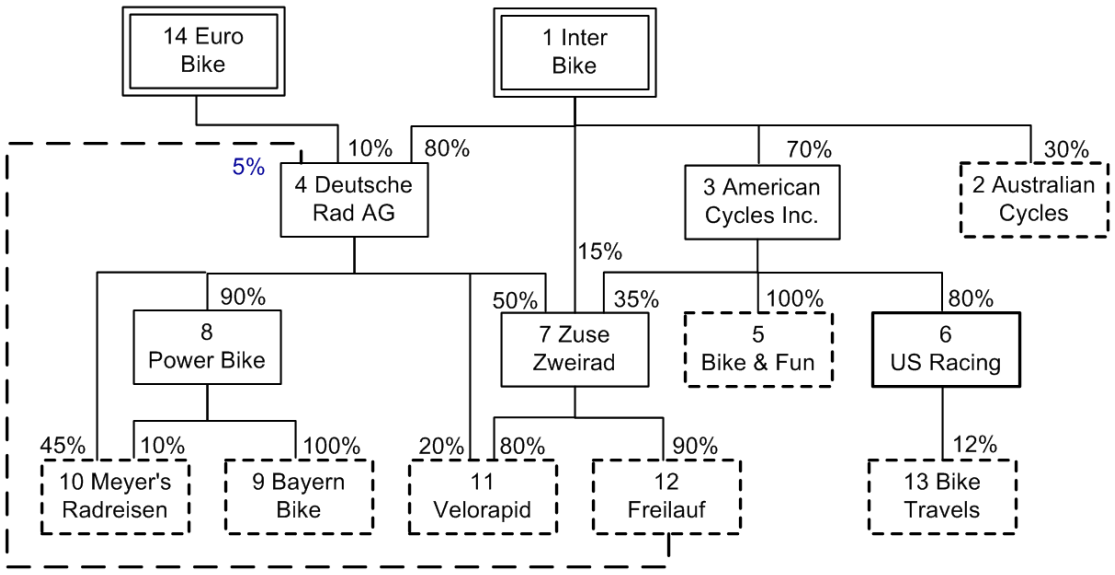

Abbildung 25: Ring durch ein beliebiges Element einer Netzwerkstruktur

Wir verwenden stattdessen wieder eine gespeicherte Prozedur zur Ringsuche, die derjenigen aus der Datei BaumstrukturTopologieErmitteln.sql (→3.2.2.2) entspricht, und wenden sie auf das Beispiel in Abbildung 25 an.

Die Deutsche Rad AG ist jetzt zusätzlich noch von der Fa. Freilauf abhängig, die ihrerseits über Zuse Zweirad wieder von der Deutschen Rad abhängt. Es entsteht also eine

Ring. Um dieses zu testen, wird zunächst ein INSTEAD OF INSERT-Trigger benötigt, der diese Prozedur aufruft.

```
/* Datei: NetzwerkstrukturBeliebigenRingVerhindern.sql */
/* INSERT-Trigger: Beliebigen Ring verhindern*/
CREATE TRIGGER Beteiligung_I_I
ON Beteiligung INSTEAD OF INSERT
AS
DECLARE @TochterID INT
DECLARE @MutterID INT
DECLARE @Beteiligungsquote DECIMAL(5,4)
DECLARE @Topologietyp INT
SELECT @TochterID = TochterID FROM inserted
SELECT @MutterID = MutterID FROM inserted
SELECT @Beteiligungsquote = Beteiligungsquote FROM inserted
/* Ringbildung durch Einfügen einer neuen Beziehung verhindern */
EXEC usp_Topologie @TochterID, @MutterID, @Topologietyp OUTPUT
IF @Topologietyp = '3'
   ROLLBACK TRANSACTION
ELSE
   INSERT INTO Beteiligung VALUES(@TochterID,@MutterID,@Beteiligungsquote)
```

Die weitgehend unveränderte gespeicherte Prozedur usp_Topologie sieht so aus:

```
/* Datei: NetzwerkstrukturTopologieErmitteln.sql */
/* Unzureichende Ringsuche */
CREATE PROC usp_Topologie @Tochter INT, @Mutter INT, @Topologietyp VARCHAR(15)
OUTPUT
AS
DECLARE @Mutter_von_Mutter INT
SELECT @Mutter_von_Mutter = MutterID FROM Beteiligung WHERE TochterID =
@Mutter
IF @Mutter_von_Mutter = @Tochter
   SET @Topologietyp = 'Ring'
ELSE
   IF @Mutter_von_Mutter IS NULL
      SET @Topologietyp = 'Kein Ring'
   ELSE
      EXEC usp_Topologie @Tochter, @Mutter_von_Mutter, @Topologietyp OUTPUT
```

Als Mutter von Freilauf findet sie Zuse Zweirad, aber als Mutter von Zuse Zweirad nicht unbedingt die Deutsche Rad AG, sondern eventuell Inter Bike oder American Cycles, die jedoch beide in keinen Ring involviert sind. Das liegt daran, dass die gespeicherte Prozedur jeweils nur eine Mutter aufspürt, was in einer Baumstruktur definitionsgemäß auch völlig ausreicht, einer Netzwerkstruktur aber gerade nicht gerecht wird.

Benötigt wird also ein Programmierkonstrukt, das jeweils sämtliche gefundenen Mütter auf der jeweils nächst höheren Ebene durchläuft. Transact-SQL kennt zwar keine Arrays, dafür aber das über den SQL-92-Standard hinausgehende Konzept des lokalen Cursors ([SQL2000]: Transact-SQL-Referenz – DECLARE CURSOR), das eine sehr elegante Lösung für die Tiefensuche ermöglicht.

```
/* Datei: NetzwerkstrukturTopologieErmitteln.sql */
/* Erfolgreiche Ringsuche */
CREATE PROC usp_Topologie @Tochter INT, @Mutter INT, @Topologietyp VARCHAR(15)
OUTPUT
AS
DECLARE @Mutter_von_Mutter INT
DECLARE Cursor_Mutter CURSOR LOCAL
```

```
   FOR SELECT MutterID FROM Beteiligung WHERE TochterID = @Mutter
OPEN Cursor_Mutter
FETCH NEXT FROM Cursor_Mutter INTO @Mutter_von_Mutter
/* Cursor-Schleife für alle Mütter */
WHILE @@fetch_status = 0
   BEGIN
   IF @Mutter_von_Mutter = @Tochter
      BEGIN
      SET @Topologietyp = 'Ring'
      RETURN
      END
   ELSE
      BEGIN
      EXEC usp_Topologie @Tochter, @Mutter_von_Mutter, @Topologietyp OUTPUT
      END
   IF @Topologietyp = 'Ring'
      RETURN
   ELSE
      FETCH NEXT FROM Cursor_Mutter INTO @Mutter_von_Mutter
   END
```

Innerhalb des ersten lokalen Cursors wird Zuse Zweirad als Mutter von Freilauf identifiziert. Da die FETCH-Anweisung erfolgreich ist (@@fetch_status = 0) und @Mutter_von_Mutter nicht mit @Tochter übereinstimmt, ruft sich die gespeicherte Prozedur erneut auf und öffnet dabei einen zweiten lokalen Cursor. Dieser findet zuerst Inter Bike als Mutter von Zuse Zweirad. Die gespeicherte Prozedur ruft sich ein zweites Mal selbst auf und öffnet einen dritten lokalen Cursor. Da dieser jedoch keinen Datensatz enthält, kehrt sie zum zweiten lokalen Cursor zurück und ruft via FETCH NEXT American Cycles als weitere Mutter von Zuse Zweirad ab. Es wird ein weiterer lokaler Cursor geöffnet, der aber wie schon zuvor feststellt, dass Inter Bike keine Mutter besitzt.

Als letzte Mutter in dem von Zuse Zweirad aus geöffneten, zweiten lokalen Cursor wird die Deutsche Rad AG gefunden. Da sie zugleich die neue Tochter von Freilauf ist, stimmt jetzt @Mutter_von_Mutter mit @Tochter überein. Der Topologietyp wird auf „Ring“ gesetzt und als Outputparameter an den Trigger zurückgegeben, der daraufhin die ganze Aktualisierungstransaktion zurückrollt, also einschließlich einer eventuellen Änderung der Beteiligungsquote.

In Abbildung 26 wird – wie bereits erwähnt – Power Bike als zusätzliche Mutter von Australian Cycles etabliert. Überzeugen Sie sich selbst davon, dass die gespeicherte Prozedur in diesem Fall nicht „Ring“ als Topologietyp zurückgibt und der Trigger somit den neuen Datensatz tatsächlich in die Beteiligungstabelle einfügt. Ausgehend von Power Bike findet die gespeicherte Prozedur nämlich folgende Äste, die beide zu keinem Ring führen:

Deutsche Rad AG → Inter Bike → [Ende]
→ Euro Bike → [Ende]

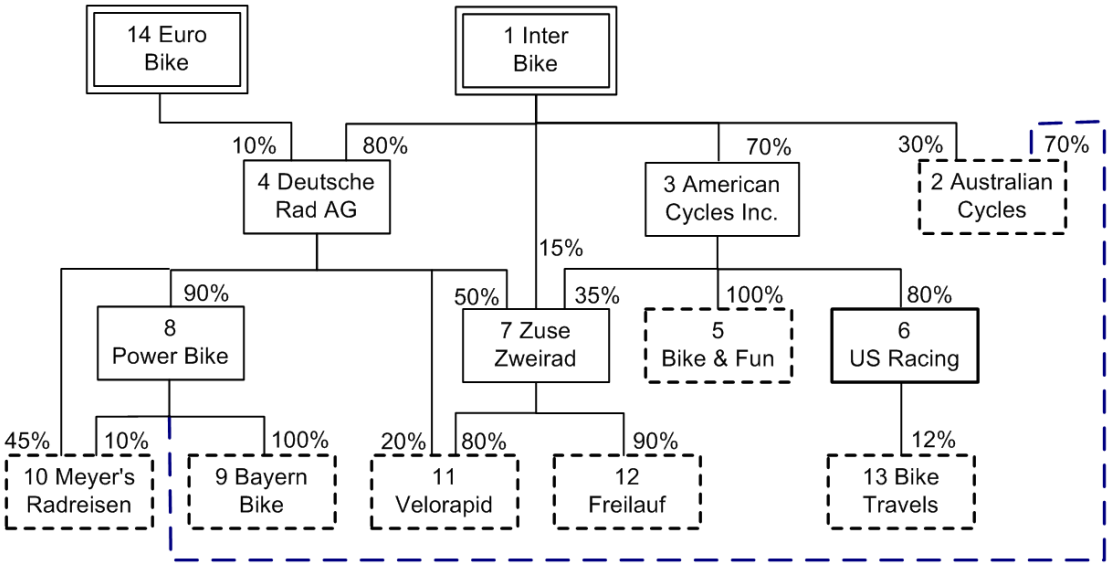

Abbildung 26: Neue Beziehung verursacht keinen Ring

Natürlich muss die Ringsuche auch für Aktualisierungsoperationen in der Beteiligungstabelle funktionieren. Die gespeicherte Prozedur bleibt völlig unverändert, aufgerufen wird sie in diesem Fall von einem INSTEAD OF UPDATE-Trigger, der analog zu demjenigen in der Baumstruktur aufgebaut ist. Der Hauptunterschied besteht darin, dass Änderungen beider Fremdschlüssel überwacht werden müssen, weil ein Ring sowohl durch Umhängen einer Tochter unter eine andere Mutter als auch durch Umhängen einer Mutter über eine andere Tochter entstehen kann.

```
/* Datei: NetzwerkstrukturBeliebigenRingVerhindern.sql */g
/* UPDATE-Trigger: Beliebigen Ring verhindern */
CREATE TRIGGER Beteiligung_I_U
ON Beteiligung INSTEAD OF UPDATE AS
DECLARE @TochterID_alt INT            /* alte Tochtergesellschaft */
DECLARE @TochterID_neu INT               /* neue Tochtergesellschaft */
DECLARE @MutterID_alt INT             /* alte Muttergesellschaft */
DECLARE @MutterID_neu INT             /* neue Muttergesellschaft */
DECLARE @Beteiligungsquote_neu DECIMAL(5,4)    /* neue Beteiligungsquote */
DECLARE @Topologietyp VARCHAR(15)     /* Topologietyp */
SELECT @TochterID_alt = TochterID FROM deleted
SELECT @TochterID_neu = TochterID FROM inserted
SELECT @MutterID_alt = MutterID FROM deleted
SELECT @MutterID_neu = MutterID FROM inserted
SELECT @Beteiligungsquote_neu = Beteiligungsquote FROM inserted
/* Update der übrigen Spalten */
IF UPDATE(Beteiligungsquote)
     UPDATE Beteiligung SET Beteiligungsquote = @Beteiligungsquote_neu
     WHERE TochterID = @TochterID_alt AND MutterID = @MutterID_alt
/* Ringbildung durch Umhängen der Tochter unter andere Mutter verhindern */
IF UPDATE(MutterID)
     BEGIN
     EXEC usp_Topologie @TochterID_neu, @MutterID_neu, @Topologietyp OUTPUT
     IF @Topologietyp = 'Ring'
          ROLLBACK TRANSACTION
     ELSE
          UPDATE Beteiligung SET MutterID = @MutterID_neu
          WHERE TochterID = @TochterID_alt AND MutterID = @MutterID_alt
```

```
     END
/* Ringbildung durch Umhängen einer anderen Tochter unter Mutter verhindern */
IF UPDATE(TochterID)
     BEGIN
     EXEC usp_Topologie @TochterID_neu, @MutterID_neu, @Topologietyp OUTPUT
     IF @Topologietyp = 'Ring'
          ROLLBACK TRANSACTION
     ELSE
          UPDATE Beteiligung SET TochterID = @TochterID_neu
          WHERE TochterID = @TochterID_alt AND MutterID = @MutterID_alt
     END
```

Die Ringbildung durch Umhängen der Tochter unter eine andere Mutter – IF UPDATE(MutterID) – kann damit getestet werden, dass nicht mehr Euro Bike, sondern Freilauf Mutter von Power Bike wird.
TochterID: 8
MutterID: 14→ 12

Der Trigger funktioniert auch beim Umhängen einer anderen Tochter unter dieselbe Mutter – IF UPDATE(TochterID). Beide Fälle müssen jedoch getrennt behandelt werden, weil sich die UPDATE-Befehle für den Fall, dass kein Ring gefunden wird, unterscheiden. Ein Test wäre z.B.:
TochterID: 11 3
MutterID: 7

Kein Ring entsteht hingegen, wenn Australian Cycles statt Velorapid zur Tochter von Zuse Zweirad wird.
TochterID: 11 2
MutterID: 7

Auch davon sollten Sie sich abschließend überzeugen.

4.2.2.3 Fazit

Auch eine Netzwerkstruktur kann nicht alleine durch das logische Datenmodell gesichert werden. Wie bei der Baumstruktur kann mit einem Constraint bzw. einer Gültigkeitsregel einfach verhindert werden, dass ein Unternehmen seine eigene Mutter wird. Komplizierter als bei der Baumstruktur, aber durch lokale Cursor letztlich elegant lösbar, gestaltet sich das Unterfangen, das Einschleichen eines Rings zu unterbinden.

5 Ringstruktur

Allgemein gilt für eine Ringstruktur:

- Jedes Element muss genau ein Kindelement haben. Blattelemente gibt es nicht.
- Jedes Element muss genau ein Elternelement haben. Wurzelelemente gibt es nicht.

Ringstrukturen dürften in der Realwelt selten vorkommen. Von Baum- und Netzwerkstrukturen unterscheiden sie sich dadurch, dass sie weder Wurzel- noch Blattelemente besitzen und nicht mehr als ein Kindelement aufweisen dürfen. Dass sie zudem höchstens ein Elternelement haben können, haben sie mit Baumstrukturen, nicht jedoch mit Netzwerkstrukturen gemein. Abbildung 27 zeigt einen Graphen einer Ringstruktur.

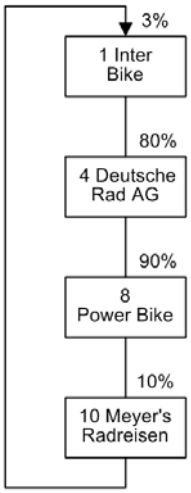

Abbildung 27: Graph einer Ringstruktur

	Eltern-E.			
Kindelemente	1 Inter Bike	4 Deutsche Rad AG	8 Power Bike	10 Meyer's Radreisen
1 Inter Bike				**R**
4 Deutsche Rad AG	**R**			
8 Power Bike		**R**		
10 Meyer's Radreisen			**R**	

R Ringelement

Abbildung 28: Verflechtungsmatrix einer Ringstruktur

Die Verflechtungsmatrix einer Ringstruktur ist nicht triangulierbar (Abbildung 28). Jede Zeile enthält genau ein hier allgemein mit R (für: Ring) gekennzeichnetes Element. Ebenso enthält jede Spalte genau ein R. Durch Vertauschen der Zeilen und Spalten lässt sich stets eine Anordnung der Verflechtungsmatrix erreichen, bei der genau ein R in der Nord-Ost-Zelle oberhalb der Hauptdiagonalen und alle übrigen R's direkt unterhalb der Hauptdiagonalen zu stehen kommen.

5.1 Datenmodell und Klassenmodell

Die Daten- und Klassenmodelle einer Ringstruktur (Abbildung 29) haben große Ähnlichkeit zur Baumstruktur (Abbildung 12). Insbesondere braucht man hier wiederum – anders in einer Netzwerkstruktur – keine assoziative Klasse bzw. Entität.

Der wesentliche Unterschied zur Baumstruktur liegt bei den Kardinalitäten. Die master- und detailseitigen Minimal- und Maximalkardinalitäten müssen sämtlich 1 sein, weil

jedes Element genau ein Elternelement und genau ein Kindelement hat. Diese Integritätsbedingungen sind nur mit Triggern einzuhalten.

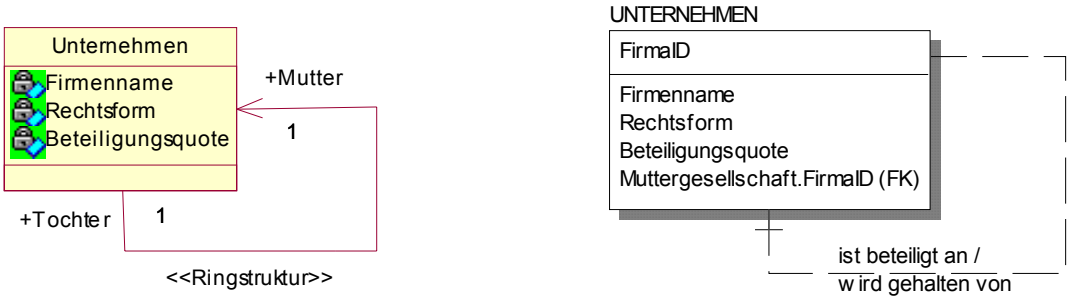

Abbildung 29: Klassen- und Datenmodell einer Ringstruktur

5.2 Implementierung in einer relationalen Datenbank

Die Access-Datenbank Ringstruktur.mdb enthält die Beispieldaten der Abbildung 27. Die entsprechende SQL Server-Datenbank samt Datensätzen kann im Query Analyzer wieder mit einem SQL-Skript der erzeugt werden (Datei RingstrukturDatenbankAnlegen.sql).

```
/* Datei: RingstrukturDatenbankAnlegen.sql */
CREATE TABLE Unternehmen (
   FirmaID              INT NOT NULL,
   Firmenname           VARCHAR(80) NOT NULL,
   Rechtsform           VARCHAR(50) NULL,
   Beteiligungsquote    DECIMAL(5,4) NULL,
   Muttergesellschaft   INT NULL,
      CONSTRAINT PK_Unternehmen_FirmaID PRIMARY KEY (FirmaID),
      CONSTRAINT FK_Unternehmen_Muttergesellschaft_FirmaID
         FOREIGN KEY (Muttergesellschaft)
         REFERENCES Unternehmen(FirmaID) ON DELETE NO ACTION,
      CONSTRAINT CK_Muttergesellschaft CHECK (Muttergesellschaft <> FirmaID))
```

Der Spaltenaufbau und die Constraints der Tabelle sind identisch zur Baumstruktur.

- Der erste Constraint definiert den Primärschlüssel.
- Der zweite Constraint sorgt für die rekursive Beziehung vom Fremdschlüssel Muttergesellschaft zum Primärschlüssel FirmaID derselben Tabelle.
- Der dritte Constraint garantiert, dass ein Unternehmen nicht seine eigene Mutter ist.

Mit den im SQL-Skript abschließend enthaltenen INSERT-Befehlen werden die Beispieldaten der Abbildung 27 eingefügt. Aus den INSERT-Befehlen wird ersichtlich, dass für das zuerst eingegebene Unternehmen der Fremdschlüssel zunächst einen Nullwert annehmen muss, der erst nach Eingabe seines Mutterunternehmens auf dessen Primärschlüsselwert gesetzt werden kann.

```
INSERT INTO Unternehmen VALUES('1','Inter Bike','AG','0.03',NULL)
INSERT INTO Unternehmen VALUES('4','Deutsche Rad AG','AG','0.8','1')
INSERT INTO Unternehmen VALUES('8','Power Bike','GmbH','0.9','4')
INSERT INTO Unternehmen VALUES('10','Meyers Radreisen','GbR','0.1','8')
UPDATE Unternehmen SET Muttergesellschaft = '10' WHERE FirmaID = '1'
```

Aus diesen datenpflegetechnischen Gründen darf die Spalte Muttergesellschaft auch nicht als NOT NULL spezifiziert werden, obwohl das Datenmodell in Abbildung 29 dies nahe legt.

Man steht hier allerdings vor einem „Henne-Ei-Problem“. Wenn man – wie hier vorgeschlagen – gewährleisten will, dass jedes Tochterunternehmen nur ein Unternehmen zur Mutter haben darf, das es in der Datenbank bereits gibt, dann muss in einer Ringstruktur genau ein Unternehmen zunächst ohne Mutter eingegeben und anschließend der Fremdschlüssel aktualisiert werden. Will man hingegen jedes Unternehmen jeweils zwingend mit einer Mutter anlegen, den Fremdschlüssel also auf NOT NULL einstellen, dann muss man auf den zweiten Constraint verzichten.

```
CREATE TABLE Unternehmen (
   FirmaID           INT NOT NULL,
   Firmenname        VARCHAR(80) NOT NULL,
   Rechtsform        VARCHAR(50) NULL,
   Beteiligungsquote    DECIMAL(5,4) NULL,
   Muttergesellschaft INT NOT NULL,
      CONSTRAINT PK_Unternehmen_FirmaID PRIMARY KEY (FirmaID),
      CONSTRAINT CK_Muttergesellschaft CHECK (Muttergesellschaft <> FirmaID))

INSERT INTO Unternehmen VALUES('1','Inter Bike','AG','0.03','10')
INSERT INTO Unternehmen VALUES('4','Deutsche Rad AG','AG','0.8','1')
INSERT INTO Unternehmen VALUES('8','Power Bike','GmbH','0.9','4')
INSERT INTO Unternehmen VALUES('10','Meyers Radreisen','GbR','0.1','8')
```

Der für die Einhaltung der Minimalkardinalität von 1 zu zahlende Preis ist allerdings, dass jetzt beliebige Fremdschlüssel auch nicht vorhandener Unternehmen verwendet werden können.

Um sowohl die referentielle Integrität als auch die Minimalkardinalität von 1 zu gewährleisten, könnte man ein fiktives Element mit dem Primärschlüssel 0 verwenden, das nur interimistisch verwendet wird.

```
CREATE TABLE Unternehmen (
   FirmaID              INT NOT NULL,
   Firmenname           VARCHAR(80) NOT NULL,
   Rechtsform           VARCHAR(50) NULL,
   Beteiligungsquote    DECIMAL(5,4) NULL,
   Muttergesellschaft   INT NOT NULL,
      CONSTRAINT PK_Unternehmen_FirmaID PRIMARY KEY (FirmaID))

INSERT INTO Unternehmen VALUES('0','Fiktives Unternehmen', NULL, NULL, '0')

ALTER TABLE Unternehmen WITH CHECK
   ADD CONSTRAINT FK_Unternehmen_Muttergesellschaft_FirmaID
      FOREIGN KEY (Muttergesellschaft)
      REFERENCES Unternehmen(FirmaID)
      ON DELETE NO ACTION
ALTER TABLE Unternehmen WITH NOCHECK
   ADD CONSTRAINT CK_Muttergesellschaft CHECK (Muttergesellschaft <> FirmaID)

INSERT INTO Unternehmen VALUES('1','Inter Bike','AG','0.03','0')
INSERT INTO Unternehmen VALUES('4','Deutsche Rad AG','AG','0.8','1')
INSERT INTO Unternehmen VALUES('8','Power Bike','GmbH','0.9','4')
INSERT INTO Unternehmen VALUES('10','Meyers Radreisen','GbR','0.1','8')
UPDATE Unternehmen SET Muttergesellschaft = '10' WHERE FirmaID = '1'
```

Die Tabelle wird zunächst ohne den zweiten und dritten Constraint angelegt und das fiktive Unternehmen eingegeben. Anschließend wird der zweite Constraint zur Sicherung der referentiellen Integrität mit Überprüfung der vorhandenen Daten und der dritte Constraint zur Vermeidung der Selbstreferenzierung ohne Überprüfung der vorhandenen Daten ergänzt. Danach können dann die eigentlichen Ringelemente eingegeben werden, wobei jedes Element ein existierendes Elternelement haben muss, zu denen auch das fiktive Element mit Primärschlüssel 0 zählt.

5.2.1 Referenzielle Integritätsaktionen

Abgesehen davon, dass im SQL Server 2000 bei einer selbstreferenziellen Beziehung weder eine deklarative Löschweitergabe (ON DELETE CASACADE) noch eine deklarative Aktualisierungsweitergabe (ON UPDATE CASACADE) technisch möglich sind (→ 3.2.1), ist sie in einer Ringstruktur auch fachlich falsch, weil es dann beim Löschen eines beliebigen Elements zu einem kreisförmigen Löschen aller übrigen Elemente käme. Wie bei der Baumstruktur empfiehlt sich ein INSTEAD OF DELETE-Trigger.

5.2.1.1 Löschen eines Ringelements

Wenn ein Ringelement gelöscht wird, muss der Fremdschlüssel Muttergesellschaft seines einzigen Kindelements direkt auf den Primärschlüssel seines Elternelements gesetzt werden, damit der Ring an der durch den Löschvorgang aufgetrennten Stelle gleich wieder geschlossen wird. Im Beispiel der Abbildung 27 soll nach dem Löschen von Power Bike die Deutsche Rad AG zur Muttergesellschaft von Meier's Radreisen werden.

Der dies bewerkstelligende INSTEAD OF DELETE-Trigger sieht dem für die Baumstruktur entwickelten mit zwei Unterschieden sehr ähnlich.

```
/* Datei: RingstrukturElementeLöschen.sql */
CREATE TRIGGER Unternehmen_I_D
ON Unternehmen
INSTEAD OF DELETE
AS
DECLARE @Löschen INT /* zu löschendes Elements */
DECLARE @Mutter INT  /* Mutter des zu löschenden Elements */
SELECT @Löschen = FirmaID FROM deleted
SELECT @Mutter = Muttergesellschaft FROM deleted
IF @Löschen <> '0'
      BEGIN
      UPDATE Unternehmen
         SET Muttergesellschaft = @Mutter WHERE FirmaID IN
         (SELECT T.FirmaID FROM Unternehmen M INNER JOIN Unternehmen T
         ON M.FirmaID = T.Muttergesellschaft WHERE M.FirmaID = @Löschen)
      /* Löschen des zu löschenden Elements */
      DELETE FROM Unternehmen WHERE FirmaID = @Löschen
      END
```

Erstens steht hier ein Löschen von Wurzelelementen nicht zur Diskussion, da es sie nicht gibt. Zweitens soll das Löschen des fiktiven Elements mit dem Primärschlüssel 0 ausgeschlossen werden, da es für das Einfügen neuer Ringelemente benötigt wird.

5.2.1.2 Aktualisieren des Primärschlüssels

Ein Primärschlüssel kann in einer Ringstruktur mit exakt demselben INSTEAD OF UPDATE-Trigger wie in einer Baumstruktur aktualisiert werden (→ 3.2.1.2), weshalb sich eine Wiederholung erübrigt. Wie schon bei der Netzstruktur sei jedoch auch hier nochmals betont, dass sich ein solcher Trigger unter dem Aspekt einer lebenslangen Objekt-ID nicht empfiehlt.

5.2.2 Gewährleistung einer Ringstruktur

Weil alle vier master- und detailseitigen Minimal- und Maximalkardinalitäten 1 sind, erlaubt das logische Datenmodell in Abbildung 29 nur Ringstrukturen und schließt Baum-, Netzwerk- und Listenstrukturen definitiv aus. Die Implementierung mit den bisher besprochenen DDL-Befehlen setzt aber nur die Anforderungen an die beiden detailseitigen Kardinalitäten um.

- Detailseitige Maximalkardinalität = 1: da die rekursive Beziehung nur mit einer Tabelle realisiert wird, kann jedes Element höchstens ein Elternelement haben, was eine Netzwerkstruktur ausschließt.
- Detailseitige Minimalkardinalität = 1: da der Fremdschlüssel Muttergesellschaft eingabepflichtig ist (NOT NULL), muss jedes Element ein Elternelement haben, was Wurzelelemente und damit Baum-, Netzwerk- und Listenstrukturen ausschließt.

Die Anforderungen an die masterseitigen Kardinalitäten können jedoch nicht mit Datendefinitionsbefehlen, sondern nur mit Triggern nicht eingehalten werden. Dabei geht es um folgendes.

- Masterseitige Maximalkardinalität = 1: wenn jedes Element höchstens ein Kindelement besitzt, sind Baumstrukturen unmöglich, und zwar sowohl als strenge Hierarchien (→ 3.) wie auch als Bäume in Netzwerkstrukturen (→ 4.). Das ist aber nur dann der Fall, wenn jeder Primärschlüssel nur einmal als Fremdschlüssel auftritt. Da dies durch Constraints in CREATE TABLE-Befehlen nicht gesichert werden kann, braucht man zusätzlich INSERT- und UPDATE-Trigger (→ 5.2.2.2).
- Masterseitige Minimalkardinalität = 1: wenn jedes Element ein Kindelement besitzen muss, gibt es keine Blattelemente wie in Baum-, Netzwerk- und Listenstrukturen. Das ist aber nur dann der Fall, wenn es keinen Primärschlüssel gibt, der überhaupt nicht als Fremdschlüssel in Erscheinung tritt. Dies kann erst recht nicht durch Constraints in CREATE TABLE-Befehlen gesichert werden. Hierauf wird im Abschnitt 5.2.2.3 näher eingegangen.

Als erstes sei jedoch – der guten Ordnung halber – wieder kurz darauf eingegangen (→ 5.2.2.1), dass auch in einer Ringstruktur kein Element seine eigene Mutter sein darf.

5.2.2.1 Verhinderung von Selbstreferenzialität

Wie bereits bei der Baum- und Netzwerkstruktur ausgeführt, kann dies im SQL Server 2000 mit einem zusätzlichen CHECK-CONSTRAINT verhindert werden.

```
CONSTRAINT CK_Muttergesellschaft CHECK (Muttergesellschaft <> FirmaID)
```

In einer Access-Datenbank wird diese Einschränkung wieder mit einer Gültigkeitsregel auf Tabellenebene realisiert.

5.2.2.2 Verhinderung mehrerer Kindelemente

In einer Ringstruktur darf weder beim Einfügen neuer Elemente noch beim Umhängen vorhandener Elemente ein Graph entstehen, in dem ein Elternelement mehr als ein Kindelement besitzt.

Nehmen wir zuerst an, dass der Ringstruktur aus Abbildung 27 Zuse Zweirad als Kindelement der Deutschen Rad AG hinzufügt wird, dann darf das Ergebnis keinesfalls wie in Abbildung 30 links aussehen. Sofern der Graph ansonsten unverändert bleiben soll, müsste Zuse Zweirad vielmehr zwischen der Deutschen Rad AG und Power Bike eingefügt werden (Abbildung 30 Mitte).

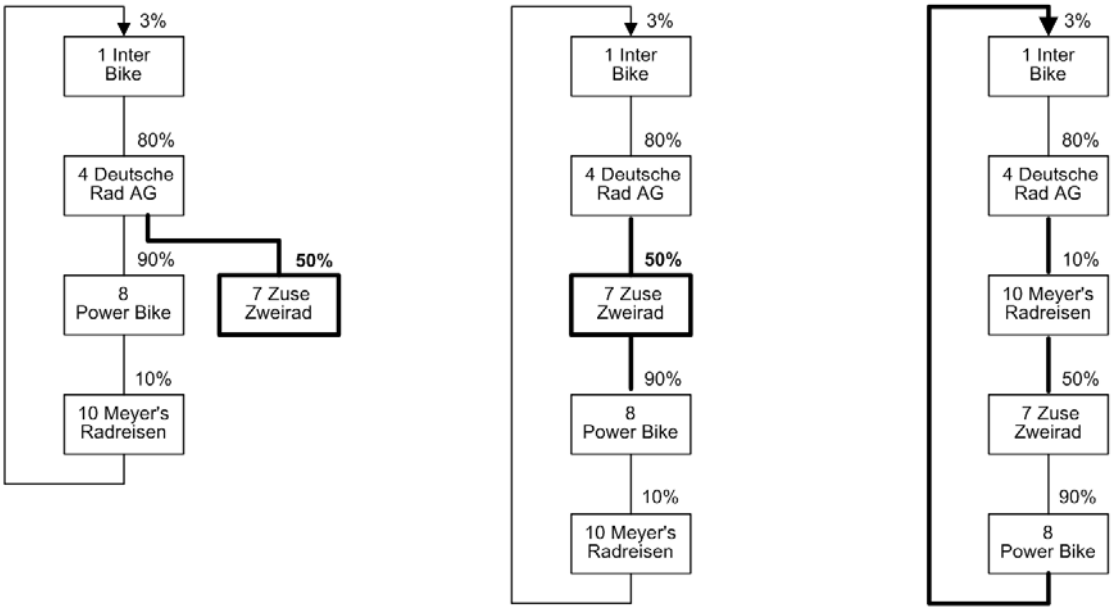

Abbildung 30: Verletzung (links) bzw. Erhaltung (Mitte und rechts) einer Ringstruktur

Der Einfügebefehl

```
INSERT INTO Unternehmen VALUES ('7','Zuse Zweirad','GmbH','0.5','4')
```

erzeugt jedoch den in Abbildung 30 links abgebildeten Graphen. Mit einem UPDATE-Befehl in einem AFTER INSERT-Trigger könnte anschließend Power Bike unter Zuse Zweirad umgehängt werden, um die in der Mitte abgebildete Struktur zu erhalten. Um den INSERT- und den UPDATE-Befehl entweder gemeinsam oder gar nicht auszuführen, ist es aber besser, beide in eine Transaktion innerhalb eines INSTEAD OF INSERT-Triggers zu packen.

```
/* Datei: RingstrukturMehrereKindelementeVerhindern.sql */
CREATE TRIGGER Unternehmen_I_I
ON Unternehmen
INSTEAD OF INSERT
AS
...
/* Bisherige Tochter der Mutter des einzufügenden Elements */
SELECT @Tochter = FirmaID FROM Unternehmen
   WHERE Muttergesellschaft = @Muttergesellschaft
BEGIN TRANSACTION
INSERT INTO Unternehmen VALUES(@FirmaID, @Firmenname, @Rechtsform,
   @Beteiligungsquote, @Muttergesellschaft)
/* Umhängen der bisherigen Tochter der Mutter des einzufügenden Elements */
UPDATE Unternehmen SET Muttergesellschaft = @FirmaID
   WHERE FirmaID = @Tochter
COMMIT TRANSACTION
```

Mit diesem Trigger kann ein neues Element strukturerhaltend in den Ring eingefügt werden. Falls die Beteiligungsquote von Zuse Zweirad an Power Bike eine andere sein soll als die der Deutschen Rad AG an Power Bike, muss sie händisch nachgepflegt werden.

Für die Erhaltung der Ringstruktur muss nicht nur beim Einfügen eines neuen Elements, sondern auch beim Umhängen eines vorhandenen Elements gesorgt werden. Nehmen wir jetzt also an, dass ausgehend vom mittleren Graphen in Abbildung 30 Meyer's Radreisen zukünftig Kind der Deutschen Rad AG statt von Power Bike sein soll. Die fett hervorgehobenen Kanten im rechten Graphen der Abbildung 30 machen deutlich, dass drei UPDATE-Befehle benötigt werden:

- Meyer's Radreisen wird Kind der Deutschen Rad AG.
- Zuse Zweirad wird Kind von Meyer's Radreisen.
- Inter Bike wird Kind der Power Bike – an Stelle von Meyer's Radreisen.

Alle drei UPDATE-Operationen werden in einer Transaktion eines INSTEAD OF UPDATE-Triggers zusammengefasst.

```
/* Datei: RingstrukturMehrereKindelementeVerhindern.sql */
CREATE TRIGGER Unternehmen_I_U
ON Unternehmen
INSTEAD OF UPDATE
AS
DECLARE @FirmaID INT       /* umzuhängendes Element */
DECLARE @Mutter_neu INT    /* neue Mutter des umzuhängendes Elements */
DECLARE @Mutter_alt INT    /* alte Mutter des umzuhängendes Elements */
/* Bisherige Tochter der Mutter des umzuhängenden Elements */
DECLARE @Tochter1 INT
DECLARE @Tochter2 INT      /* Bisherige Tochter des umzuhängenden Elements */
DECLARE @Firmenname VARCHAR(80)
DECLARE @Rechtsform VARCHAR(50)
DECLARE @Beteiligungsquote DECIMAL(5,4)
SELECT @FirmaID = FirmaID FROM inserted
SELECT @Mutter_neu = Muttergesellschaft FROM inserted
SELECT @Mutter_alt = Muttergesellschaft FROM deleted
SELECT @Tochter1 = FirmaID FROM Unternehmen
   WHERE Muttergesellschaft = @Mutter_neu
SELECT @Tochter2 = FirmaID FROM Unternehmen
   WHERE Muttergesellschaft = @FirmaID
SELECT @Firmenname = Firmenname FROM inserted
SELECT @Rechtsform = Rechtsform FROM inserted
SELECT @Beteiligungsquote = Beteiligungsquote FROM inserted
```

```
IF (@Mutter_neu <> @Mutter_alt) OR (UPDATE(Muttergesellschaft) AND
(@Mutter_neu IS NULL OR @Mutter_alt IS NULL))
      BEGIN
      BEGIN TRANSACTION
      /* Umhängen eines beliebigen Ringelements */
      UPDATE Unternehmen SET Muttergesellschaft = @Mutter_neu
         WHERE FirmaID = @FirmaID
      /* Schließen des Rings durch nachfolgendes Umhängen */
      UPDATE Unternehmen SET Muttergesellschaft = @FirmaID
         WHERE FirmaID = @Tochter1
      UPDATE Unternehmen SET Muttergesellschaft = @Mutter_alt
         WHERE FirmaID = @Tochter2
      COMMIT TRANSACTION
      END
IF UPDATE(Firmenname) OR UPDATE(Rechtsform) OR UPDATE(Beteiligungsquote)
      UPDATE Unternehmen SET Firmenname = @Firmenname,
         Rechtsform = @Rechtsform, Beteiligungsquote = @Beteiligungsquote
         WHERE FirmaID = @FirmaID
```

5.2.2.3 Verhinderung von Blattelementen

Der INSERT- und der UPDATE-Trigger (→ 5.2.2.2) verhindern nicht nur, dass ein Element unversehens mehr als ein Kindelemente aufweist. Zusammen mit dem DELETE-Trigger (→ 5.2.1.1) sorgen sie dafür, dass weder durch das Einfügen oder Ändern, noch durch das Löschen eines Elements ein Blattelement entstehen kann. Das liegt daran, dass die Trigger nach jedem Auftrennen eines Rings sofort dafür sorgen, dass er wieder geschlossen wird. Eine gesonderte Behandlung dieser Ringverletzung ist also nicht nötig. Mit der folgenden Abfrage kann man sich zudem davon überzeugen, dass es wirklich keine Blattelemente gibt.

```
/* Datei: RingstrukturMehrereKindelementeVerhindern.sql */
SELECT M.FirmaID AS Blattelement, M.Firmenname
   FROM Unternehmen M LEFT OUTER JOIN Unternehmen T
   ON M.FirmaID = T.Muttergesellschaft
   WHERE T.FirmaID IS NULL
```

Anders als bei der Beispieldatenbank zur Baumstruktur liefert diese Abfrage hier eine leere Ergebnismenge zurück.

5.2.2.4 Fazit

Das logische Datenmodell einer Ringstruktur ist nicht alleine durch die aus ihm via forward engineering ableitbaren DDL-Befehle einzuhalten. Damit ein Ring nicht zum Baum mutiert, muss gewährleistet sein, dass die masterseitige Kardinalitäten genau 1 sind. Die hierfür entwickelten Trigger sind aber relativ einfach. Jedenfalls wird man wohl behaupten können, dass es programmierungstechnisch einfacher ist, eine Ringstruktur zu erzwingen als die Ringbildung in einer Baum- oder Netzwerkstruktur zu unterbinden.

6 Listenstruktur

Allgemein gilt für eine Listenstruktur:

- Mit Ausnahme eines einzigen Blattelements muss jedes Element genau ein Kindelement haben.
- Mit Ausnahme eines einzigen Elternelements muss jedes Element genau ein Wurzelelement haben.

Diese Definition einer Listenstruktur hat eine gewisse Affinität zu derjenigen einer Ringstruktur. Tatsächlich sieht eine Liste graphisch aus wie ein an einer einzigen Stelle aufgetrennter Ring, weshalb sie gelegentlich auch als „offener Ring“ bezeichnet wird. Streng genommen ist ein „offener Ring“ jedoch ein Widerspruch in sich selbst. Logisch korrekter kann man eine Listenstruktur auch als spezielle, nicht-multiple Baumstruktur interpretieren, in der jedes Element höchstens ein Kindelement haben darf. Dies impliziert zwangsläufig, dass es genau ein Blattelement gibt. Abbildung 31 zeigt einen Graphen einer Listenstruktur.

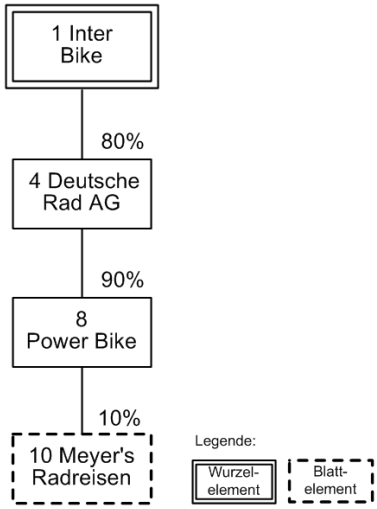

Abbildung 31: Graph einer Listenstruktur

Die Verflechtungsmatrix einer Listenstruktur ist immer triangulierbar (Abbildung 32). Bei n Elementen enthalten n-1 Zeilen genau ein hier allgemein mit L (für: Liste) gekennzeichnetes Element, eine Zeile – nämlich die des Wurzelelements – enthält kein L. Ebenso enthalten n-1 Spalten genau ein L, während eine Spalte – nämlich die des einzigen Blattelements – leer bleibt. Durch Vertauschen der Zeilen und Spalten lässt sich stets eine Anordnung der Verflechtungsmatrix erreichen, bei der alle L’s direkt unterhalb der Hauptdiagonalen zu stehen kommen.

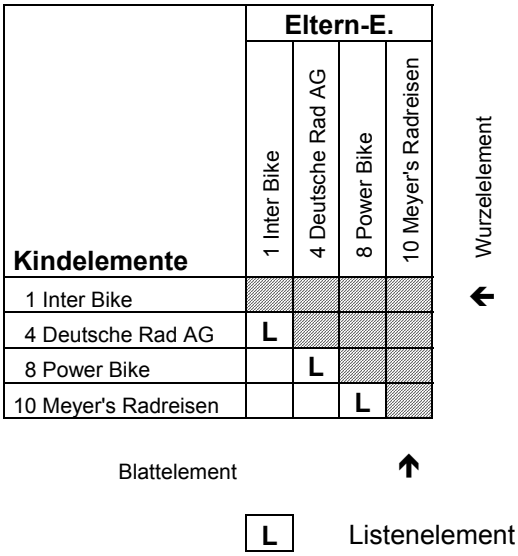

Abbildung 32: Verflechtungsmatrix einer Listenstruktur

6.1 Datenmodell und Klassenmodell

Die Daten- und Klassenmodelle einer Listenstruktur haben die größte Ähnlichkeit mit denen einer Ringstruktur. Von diesen unterscheiden sie sich nur hinsichtlich der Minimalkardinalitäten, wie aus Abbildung 33 ersichtlich ist.

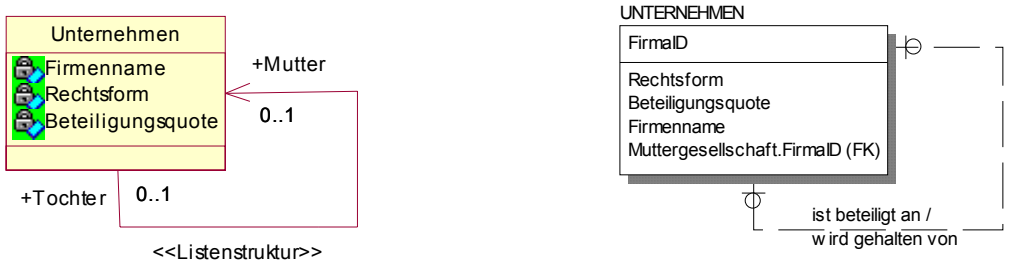

Abbildung 33: Klassen- und Datenmodelle einer Listenstruktur

Die master- und detailseitigen Maximalkardinalitäten müssen wie bei der Ringstruktur 1 sein, weil jedes Element höchstens ein Eltern- und ein Kindelement hat. Die master- und detailseitigen Minimalkardinalitäten müssen jedoch – anders als im Ring – 0 sein, weil es in einer Liste ein Element ohne Elternelement (Wurzelelement) und ein Element ohne Kindelement (Blattelement) gibt. Dass es nur genau ein Wurzelelement und genau ein Blattelement geben darf, lässt sich im Datenmodell nicht spezifizieren und ist nur mit Triggern einzuhalten. Abbildung 33 lässt mehrere Wurzel- und Blattelemente zu.

6.2 Implementierung in einer relationalen Datenbank

Die Access-Datenbank Listenstruktur.mdb enthält die Beispieldaten der Abbildung 31. Die entsprechende SQL Server-Datenbank samt Datensätzen kann im Query Analyzer

wieder mit einem SQL-Skript der erzeugt werden (Datei ListenstrukturDatenbankAnlegen.sql).

```
/* Datei: ListenstrukturDatenbankAnlegen.sql */
CREATE TABLE Unternehmen (
   FirmaID           INT NOT NULL,
   Firmenname        VARCHAR(80) NOT NULL,
   Rechtsform        VARCHAR(50) NULL,
   Beteiligungsquote    DECIMAL(5,4) NULL,
   Muttergesellschaft INT NULL,
      CONSTRAINT PK_Unternehmen_FirmaID PRIMARY KEY (FirmaID),
      CONSTRAINT FK_Unternehmen_Muttergesellschaft_FirmaID
         FOREIGN KEY (Muttergesellschaft)
         REFERENCES Unternehmen(FirmaID) ON DELETE NO ACTION,
      CONSTRAINT CK_Muttergesellschaft CHECK (Muttergesellschaft <> FirmaID))

INSERT INTO Unternehmen VALUES('1','Inter Bike','AG', NULL, NULL)
INSERT INTO Unternehmen VALUES('4','Deutsche Rad AG','AG','0.8','1')
INSERT INTO Unternehmen VALUES('8','Power Bike','GmbH','0.9','4')
INSERT INTO Unternehmen VALUES('10','Meyers Radreisen','GbR','0.1','8')
```

Der CREATE TABLE-Befehl ist absolut identisch mit dem zu Beginn des Abschnitts 5.2 für die Ringstruktur vorgestellten. Der Fremdschlüssel Muttergesellschaft lässt hier Nullwerte zu, um diesen dem Wurzelelement zuzuweisen.

6.2.1 Referenzielle Integritätsaktionen

Aus denselben technischen und fachlichen Gründen wie bei der Ringstruktur (→ 5.2.1) kommt ein ON DELETE CASCADE auch hier nicht in Frage. Die referentielle Integrität wird wiederum mit ON DELETE NO ACTION in Verbindung mit einem INSTEAD OF DELETE-Trigger gewahrt.

6.2.1.1 Löschen eines Listenelements

Das Löschen von Listenelementen kann mit fast demselben INSTEAD OF DELETE-Trigger wie bei der Ringstruktur erfolgen (→ 5.2.2.1), da dieser auch für das Wurzelelement und das Blattelement funktioniert. Zwei kleine Modifikationen sind sinnvoll.

Da hier kein fiktives Element mit Primärschlüssel 0 benötigt wird, kann erstens auf die Bedingung

```
IF @Löschen <> '0'
```

verzichtet werden. Zweitens sollte beim Löschen des Wurzelelements berücksichtigt werden, dass die Beteiligungsquote des neuen Wurzelelements einen Nullwert annimmt.

```
/* Datei: ListenstrukturElementeLöschen.sql */
CREATE TRIGGER Unternehmen_I_D
ON Unternehmen
INSTEAD OF DELETE
...
/* Das neue Wurzelement darf keine Beteiligungsquote haben */
UPDATE Unternehmen
   SET Beteiligungsquote = NULL WHERE Muttergesellschaft IS NULL
```

6.2.1.2 Aktualisieren des Primärschlüssels

Auch für das Aktualisieren des Primärschlüssels kann derselbe INSTEAD OF UPDATE-Trigger wie bei der Ringstruktur verwendet werden (→ Datei: ListenstrukturKaskadierendesUpdate.sql).

6.2.2 Gewährleistung einer Listenstruktur

Auch das Datenmodell in Abbildung 33 schließt eine Baum- und eine Netzwerkstruktur definitiv aus, impliziert jedoch nicht zwingend eine Listenstruktur, weil gegen diese mehrfach verstoßen werden kann.

- Ein Unternehmen könnte seine eigene Mutter sein [DoHu99: 268] (→ 6.2.2.1).
- Ein Unternehmen könnte mehr als eine Tochter haben (→ 6.2.2.2).
- Statt eines neuen Wurzelelements könnte ein fiktives Element entstehen (→ 6.2.2.3).
- Es könnte ein Ring entstehen [DoHu99: 268] (→ 6.2.2.4).

6.2.2.1 Verhinderung von Selbstreferenzialität

Wie bei allen bisher besprochenen rekursiven Strukturen erfolgt dies auch hier:

```
CONSTRAINT CK_Muttergesellschaft CHECK (Muttergesellschaft <> FirmaID)
```

6.2.2.2 Verhinderung mehrerer Kindelemente

Auch in einer Listenstruktur darf weder beim Einfügen neuer Elemente noch beim Umhängen vorhandener Elemente ein Graph entstehen, in dem ein Elternelement mehr als ein Kindelement besitzt. Die zu Abbildung 30 analogen Beispiele sind:

- Das Einfügen von Zuse Zweirad als Kindelement der Deutschen Rad AG darf nicht zum linken, sondern muss zum mittleren Graphen in Abbildung 34 führen.
- Wenn anschließend Meyer's Radreisen zum Kindelement der Deutschen Rad AG wird, muss der Graph wie in Abbildung 34 rechts aussehen.

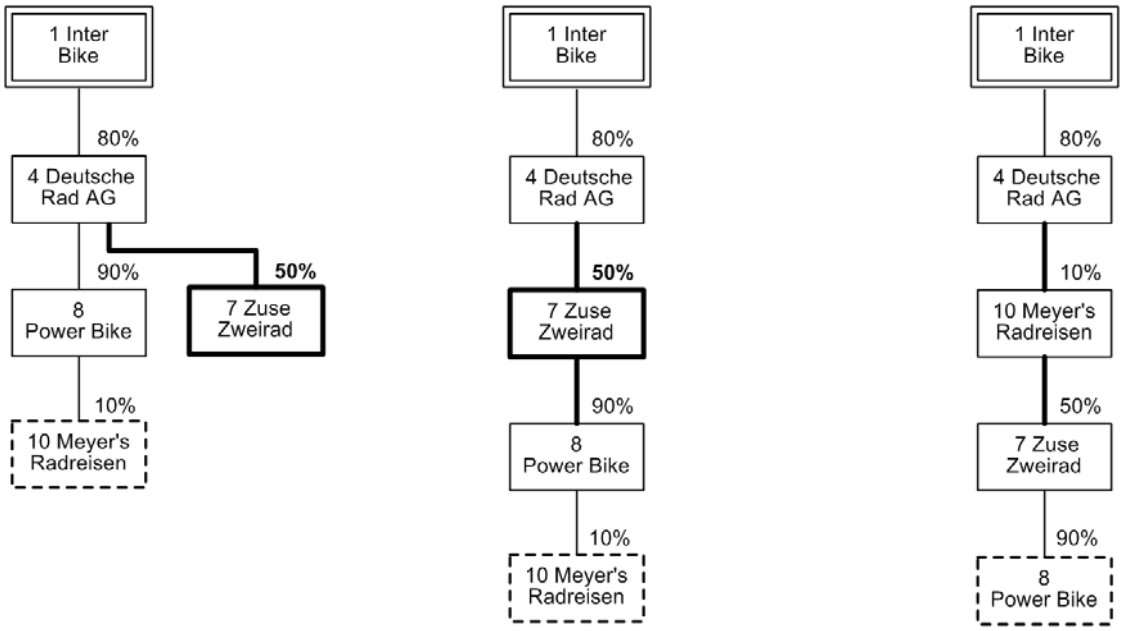

Abbildung 34: Verletzung (links) bzw. Erhaltung (Mitte und rechts) einer Listenstruktur

Beide Aufgaben erfüllen die INSTEAD OF INSERT- und INSTEAD OF UPDATE-Trigger, wie sie für die Ringstruktur entwickelt wurden (→ 5.2.2.2), auch für die Listenstruktur. Sie werden in der Datei ListenstrukturMehrereKindelementeVerhindern.sql nur der Vollständigkeit halber erneut dokumentiert. Sie funktionieren auch dann, wenn das neue oder das umgehängte Element zum Blattelement wird, nicht jedoch dann, wenn ein neues Wurzelelement entstehen soll. Darauf wird im folgenden Abschnitt eingegangen.

6.2.2.3 Verhinderung eines fiktiven Elements

Anders als bei der Ringsstruktur versagen die beiden im vorangegangenen Abschnitt erneut verwendeten Trigger dann, wenn ein neues Wurzelelement eingefügt oder ein vorhandenes Element zum neuen Wurzelelement gemacht werden soll. Es entstehen dann fehlerhafte Graphen wie links in den Abbildungen 35 und 36.

Beim Einfügen von Zuse Zweirad als neues Wurzelelement mit

```
INSERT INTO Unternehmen VALUES('7','Zuse Zweirad','GmbH', NULL, NULL)
```

wird das bisherige Wurzelelement Inter Bike nicht unter das neue Wurzelelement gehängt (Abbildung 35 rechts), sondern bleibt als solches bestehen (Abbildung 35 links), was ein Verstoß gegen die Listenstruktur ist.

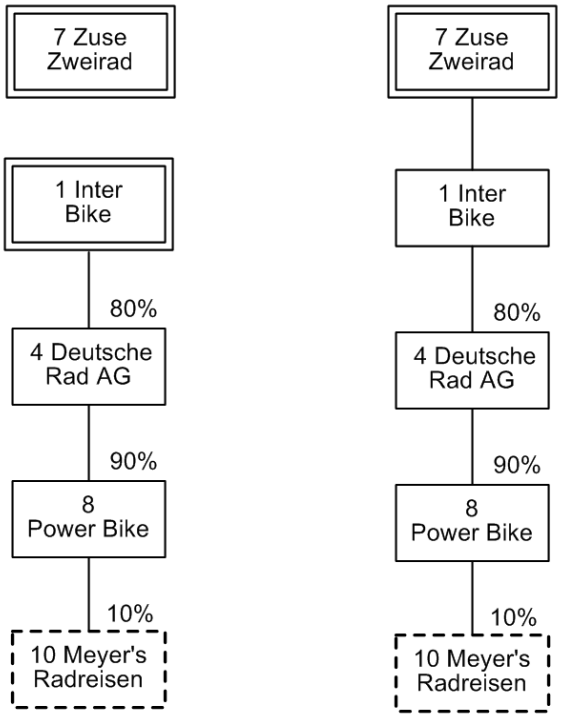

Abbildung 35: Neues Wurzelelement in einer Listenstruktur durch Einfügen

Dasselbe passiert, wenn Power Bike mit

```
UPDATE Unternehmen SET Muttergesellschaft = NULL WHERE FirmaID = '8'
```

zum neuen Wurzelelement gemacht wird (Abbildung 36).

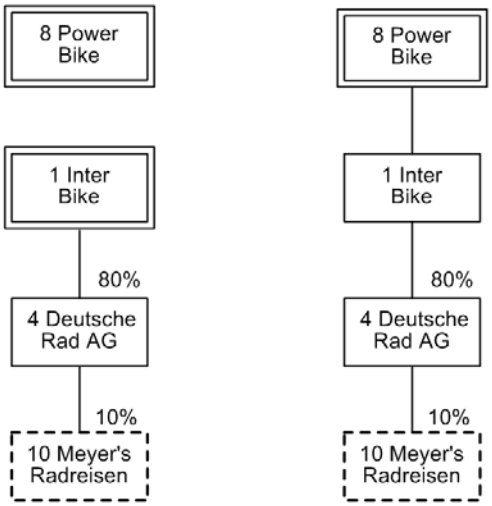

Abbildung 36: Neues Wurzelelement in einer Listenstruktur durch Umhängen

In beiden Fällen fehlt die Abhängigkeit des alten vom neuen Wurzelelement. Im INSTEAD OF INSERT-Trigger lässt sich dies mit den folgenden Erweiterungen beheben:

```
/* Datei: ListenstrukturFiktivesElementVerhindern.sql */
CREATE TRIGGER Unternehmen_I_I
ON Unternehmen
INSTEAD OF INSERT
...
DECLARE @Wurzel INT
...
/* Wurzelelement */
SELECT @Wurzel = FirmaID FROM Unternehmen WHERE Muttergesellschaft IS NULL
/* Bisherige Tochter der Mutter des einzufügenden Elements */
SELECT @Tochter = FirmaID FROM Unternehmen
   WHERE Muttergesellschaft = @Muttergesellschaft
IF @Muttergesellschaft IS NULL
      SET @Tochter = @Wurzel
...
```

Analog ist der INSTEAD OF UPDATE-Trigger zu erweitern.

6.2.2.4 Verhinderung einer Ringbildung

Durch das Einfügen eines Elements in eine Listenstruktur kann niemals ein Ring entstehen. Hierfür gibt es nur eine Möglichkeit: den Fremdschlüssel des Wurzelelements mit dem Primärschlüssel des Blattelements aktualisieren. Der in den Abschnitten 6.2.2.2 und 6.2.2.3 entwickelte INSTEAD OF UPDATE-Trigger verhindert diese Ringbildung jedoch bereits, weil er den Ring nicht schließt, sondern das alte Wurzelelement unter das bisherige Blattelement umhängt. Ein gesondertes Abfangen der Ringbildung durch eine gespeicherte Prozedur zum Aufspüren eines Rings, wie wir sie bei der Baum- und Netzwerkstruktur vorgeschlagen haben, ist hier also nicht erforderlich.

6.2.2.5 Fazit

Die Daten- und Klassenmodelle der Listenstruktur stimmen mit denen einer Ringstruktur überein. Auch zu ihrer Einhaltung ergänzend notwendigen Trigger sind nahezu identisch. Lediglich der Fall, dass das einzufügende zu aktualisierende Element das neue Wurzelelement ist, muss in den beiden Triggern zusätzlich berücksichtigt werden.

7 Paarstruktur

Paarstrukturen sind spezielle Ringstrukturen und deshalb noch seltener als diese. Sie verfügen nur über zwei Elemente, die wechselseitig von einander abhängen. Allgemein gilt für eine Paarstruktur:

- Jedes der beiden Elemente muss genau ein Kindelement haben.
- Jedes der beiden Elemente muss genau ein Elternelement haben.

Wurzel- und Blattelemente kann es wie in der Ringstruktur nicht geben. Abbildung 37 zeigt den Graphen einer Paarstruktur.

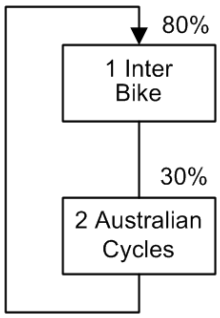

Abbildung 37: Graph einer Paarstruktur

Die Verflechtungsmatrix einer Paarstruktur ist nicht triangulierbar (Abbildung 38). Sie besitzt nur vier Felder mit zwei leeren Zellen auf der Hauptdiagonalen und zwei besetzten Zellen auf der Nebendiagonalen.

	Elt.	
Kindelemente	1 Inter Bike	2 Australian Cycles
1 Inter Bike		P
2 Australian Cycles	P	

P Paarstruktur

Abbildung 38: Verflechtungsmatrix einer Paarstruktur

7.1 Datenmodell und Klassenmodell

Da die Daten- und Klassenmodelle absolut identisch mit denen der Ringstruktur sind, bedarf es hier keiner Wiederholung (→ Abschnitt 5.1).

7.2 Implementierung in einer relationalen Datenbank

Da eine Paarstruktur immer aus zwei wechselseitig voneinander abhängigen Elementen besteht, vereinfacht sich die Datenpflege und damit die Einhaltung der Integritätsanforderungen erheblich. Es müssen weder neue Elemente hinzukommen noch vorhandene gelöscht werden. UPADTE-Operationen sind also völlig ausreichend. Aktualisierbar müssen zudem nur die Nichtschlüsselattribute sein, da sich der Primärschlüssel und der Fremdschlüssel nicht ändern können.

```
/* Datei: PaarstrukturDatenbankAnlegen.sql */
CREATE TABLE Unternehmen (
   FirmaID            INT NOT NULL,
   Firmenname         VARCHAR(80) NOT NULL,
   Rechtsform         VARCHAR(50) NULL,
   Beteiligungsquote    DECIMAL(5,4) NULL,
   Muttergesellschaft INT NOT NULL,
      CONSTRAINT PK_Unternehmen_FirmaID PRIMARY KEY (FirmaID))

INSERT INTO Unternehmen VALUES('1','Inter Bike','AG','0.8','2')
INSERT INTO Unternehmen VALUES('2','Australian Cycles','AG','0.3','1')

/* Paarstruktur gewährleisten */
CREATE TRIGGER Unternehmen_I_I
ON Unternehmen
INSTEAD OF INSERT
AS
ROLLBACK TRANSACTION

CREATE TRIGGER Unternehmen_I_D
ON Unternehmen
INSTEAD OF DELETE
AS
ROLLBACK TRANSACTION

CREATE TRIGGER Unternehmen_I_U
ON Unternehmen
INSTEAD OF UPDATE
AS
DECLARE @FirmaID INT
DECLARE @Firmenname VARCHAR(80)
DECLARE @Rechtsform VARCHAR(50)
DECLARE @Beteiligungsquote DECIMAL(5,4)
SELECT @FirmaID = FirmaID FROM inserted
SELECT @Firmenname = Firmenname FROM inserted
SELECT @Rechtsform = Rechtsform FROM inserted
SELECT @Beteiligungsquote = Beteiligungsquote FROM inserted
IF UPDATE(Firmenname) OR UPDATE(Rechtsform) OR UPDATE(Beteiligungsquote)
     UPDATE Unternehmen
        SET Firmenname = @Firmenname, Rechtsform = @Rechtsform,
        Beteiligungsquote = @Beteiligungsquote WHERE FirmaID = @FirmaID
```

Nachdem die einzig notwendige Tabelle ohne weitere Constraints angelegt und die beiden Ausgangsdatensätze eingegeben wurden, wird das Einfügen und Löschen von Da-

tensätzen mit einem INSTEAD OF INSERT-Trigger und einem INSTEAD OF DELETE-Trigger unterbunden. Abschließend beschränkt der INSTEAD OF UPDATE-Trigger Aktualisierungen der beiden Datensätze auf die Nichtschlüsselattribute. Man sich in einem Client wie dem Enterprise Manager sehr einfach davon überzeugen, dass keine Datenmanipulationen möglich sind, die die Paarstruktur gefährden würden.

8 Hybridstruktur

Wie im Abschnitt 2 erläutert, sei eine rekursive Struktur, die keine der zuvor besprochenen Strukturen ausschließt, als Hybridstruktur bezeichnet. Im Graph der Abbildung 39 erkennt man u.a. folgende Teilstrukturen:

- Die Deutsche Rad AG, Power Bike und Zuse Zweirad bilden einen Baum.
- Die Deutsche Rad AG ist ein Netzelement, da sie zwei Elternelemente hat.
- Die Deutsche Rad AG bildet mit Power Bike und Meyer's Radreisen einen Ring.
- American Cycles und US Racing bilden ein Paar.
- Australian Cycles Freilauf und Bike Travels bilden eine Liste.

Es sei nochmals betont, dass insbesondere die Ring-, die Paar- und die Listenstruktur – anders als in den ihnen zuvor gewidmeten Abschnitte 5 bis 7 – hier nicht in Reinkultur, sondern als Partialstrukturen eines hybriden Verbundes vorkommen.

Die Verflechtungsmatrix (Abbildung 40) macht dies ebenfalls gut sichtbar. Sie wurde soweit trianguliert, dass nur jeweils ein Element der Ring- bzw. Paarstruktur oberhalb der Hauptdiagonalen in einem möglichst geringen Abstand zu dieser stehen bleibt.

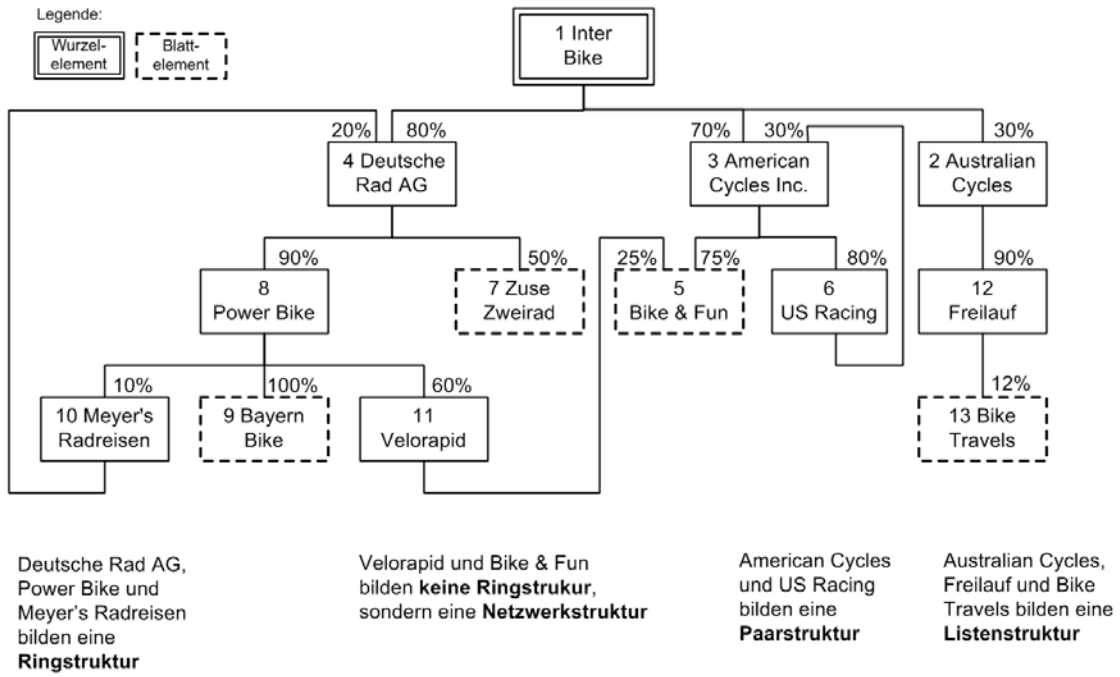

Abbildung 39: Graph einer Hybridstruktur

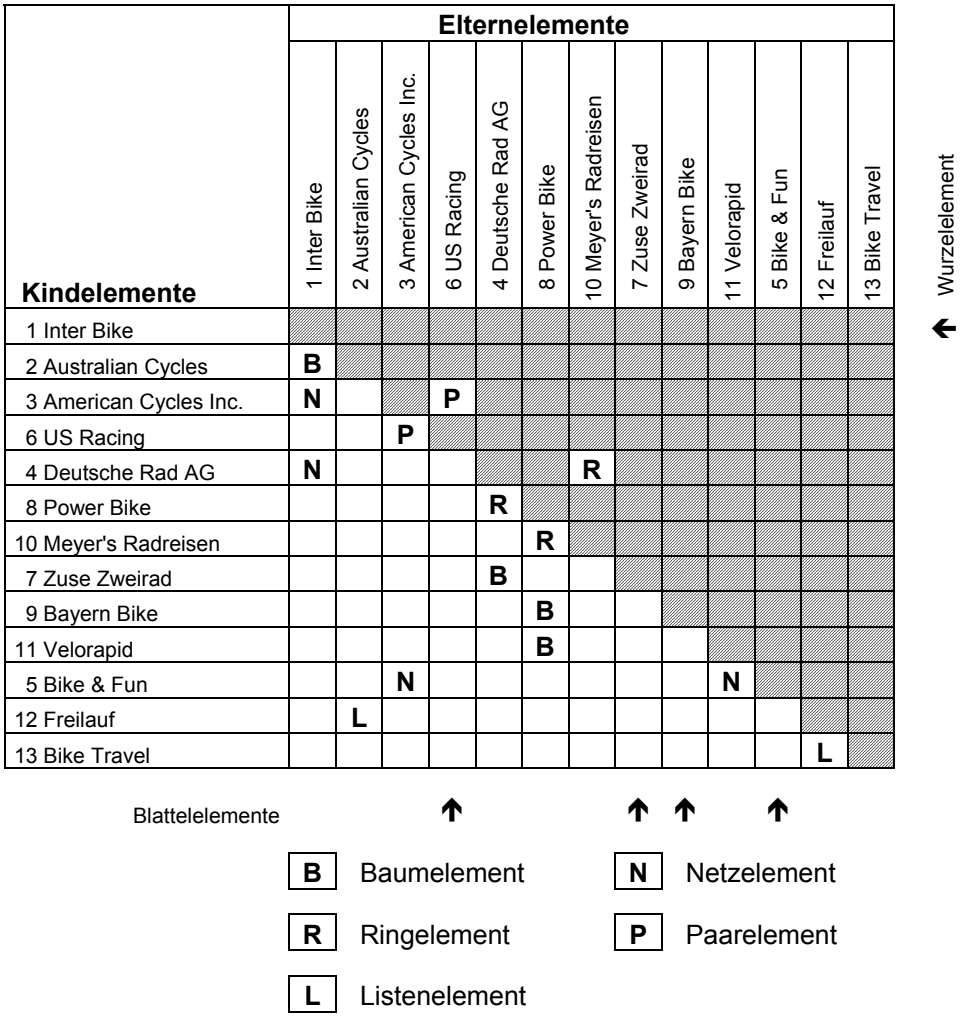

Abbildung 40: Verflechtungsmatrix einer Hybridstruktur

Daten- und Klassenmodelle einer Hybridstruktur entsprechen denjenigen der Netzwerkstruktur (Abbildungen 19 und 20). Bei der Implementierung in einer relationalen Datenbank kann man sich deshalb an den SQL-Skripten für die Netzwerkstruktur halten und muss lediglich auf die Prozeduren verzichten, die dort eine Ringbildung unterdrücken. Das vollständige Skript für die Beispieldatenbank findet sich in der Datei Hybridstruktur.sql.

Literaturverzeichnis

A. Bücher

[Balz99]
Balzert, H.: Lehrbuch der Objektmodellierung. Analyse und Entwurf. Heidelberg 1999. Spektrum Akademischer Verlag

[BeMo00]
Ben-Gan, I.; Moreau, T.: Advanced Transact-SQL for SQL Server 2000. Berkley 2000. Apress

[BoRu99a]
Booch, G.; Rumbaugh, J.; Jacobson, I.: The Unified Modeling Language User Guide. Reading 1999. Addision Wesley Longman

[BoRu99b]
Booch, G.; Rumbaugh, J.; Jacobson, I.: Das UML-Benutzerhandbuch. München 1999. Addision-Wesley

[Bruc92]
Bruce, Th.: Designing Quality Databases with IDEF1X Information Models. New York 1992. Dorset House Publishing

[DoHu99]
Dorsey, P.; Hudicka, J.R.: Oracle8 Design Using UML Object Modeling. Berkeley 1999. McGraw-Hill

[Oest99]
Oestereich, B.: Objektorientierte Softwareentwicklung. Analyse und Design mit der Unified Modeling Language. München 1999. R.Oldenburg Verlag. (4. Aufl.)

B. Zeitschriften

[Ettl99]
Ettlinger, B.: IDEF1X vs. UML. A Comparative Analysis.In: The ERwin Insider & BPwin Insider 1999, S. 19-53

[Behr01]
Behrens, Chr.: The Zen of Recursion. Use SQL Server stored procedures recursively to traverse hierarchies. In: SQL Server Magazine Online (MSDN) 2001, Heft 4

C. WWW-Publikationen

[Somm01]
Sommer, M.: Graphische Aufbereitung komplexer Konzernstrukturen mit Visual OrgChart. 2001. URL: http://www.nsl.de/standardsoftware/vogc/VOGC_Info2.htm. (Abrufdatum: 2002-07-27)

D. Online-Dokumente in Dateiform

[IDS99]
IDS Scheer: ARIS Methode Version 4.1. Saarbrücken 1999. (PDF-Datei)

[SQL2000]
Microsoft Corporation: SQL Server-Onlinedokumentation. 2000

Abkürzungsverzeichnis

ARIS	Architektur integrierter Informationssysteme
DDL	Data definition language
DRI	Deklarative referentielle Integrität
ERM	Entity-Relationship-Modell(ierung)
IE	Information Engineering
UML	Unified Modeling Language

Danksagung

Für die kritische Durchsicht des Manuskripts und wertvolle Hinweise danke ich Herrn Dipl.-Inf. Oliver Thöne.