in: ZRI-Bericht 5/92, University Kaiserslautern, 1992

Sicherung der Referentiellen Integrität

-

Optimierung durch Schemaanalyse

Joachim Reinert

5/92

Univ. Kaiserslautern
Fachbereich Informatik
Postfach 3049
6750 Kaiserslautern

Juli 1992

Sicherung der Referentiellen Integrität

-

Optimierung durch Schemaanalyse

Joachim Reinert
Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049, W-6750 Kaiserslautern
e-mail: jreinert@informatik.uni-kl.de
Telefon: 0631 2053282

Abstract

In der letzten Zeit wird die Diskussion um die referentielle Integrität, deren Definition und Sicherung in konkreten Systemen, wieder heftiger geführt. Dabei spielen sowohl pragmatische als auch datenmodellthoretische Gründe eine wichtige Rolle. Um die referentielle Integrität in einer realen Anwendung umsetzen zu können, muß die Performance eines Systems erhalten bleiben. Deshalb ist die Optimierung von Überprüfung und Wartung der referentiellen Integrität eine wesentliche Forderung an neue Systeme. Im folgenden Aufsatz wird ein Ansatz zur Optimierung für die Wartung von referentieller Integrität durch eine intensive Schemaanalyse vorgestellt.

1. Einleitung und Motivation

Um Gegebenheiten und Abläufe der realen Welt durch Rechner unterstützen zu lassen, muß der relevante Weltausschnitt (Miniwelt) schließlich so modelliert werden, daß er auf einem Rechner "ablauffähig" wird. Um diese Vorgang der Abbildung immer besser zu unterstützen wurden die Modelle, die einer Verarbeitung durch Rechner zu grunde gelegt werden können (die ein Rechner "versteht") immer wieder erweitert. Ein wichtiger Aspekt bei der Modellierung einer Miniwelt ist es, zulässige "Situationen" zu beschreiben. Dadurch soll sichergestellt werden, daß alle "Situationen", die mit dem Modell der Miniwelt beschreibbar sind, wenigstens in sich *konsistent* sind.

Im Rückblick stellt die Entwicklung des relationalen Datenmodell [Co70] einen wesentlich Entwicklungsschritt solcher Modelle im Bereich der Datenbanktechnologie dar. Es stellt dem Benutzer einfache und doch sehr mächtige Modellierungsmöglichkeiten zur Verfügung. Bei der Realisierung auf Rechnern wurden jedoch bisher in fast allen verfügbaren Systemen bestimmte Elemente des Modells außer acht gelassen. Ein solches Element stelle die referentielle Integrität dar. Jedoch wird im nun vorliegenden Vorschlag zur Standardisierung von SQL [SQL2] auch dieses Element mit einer konkreten Semantik versehen, so daß in den nächsten Jahren viele Hersteller von Datenbanksoftware auch die referentielle Integrität mit dieser Semantik realisieren werden. Jedoch birgt der Versuch, diese Semantik der referentiellen Integrität direkt aus dem Standardisierungsdokument zu implementieren, die Gefahr eines sehr hohen Laufzeitaufwandes. Die Optimierung dieses Laufzeitfaktors ist Gegenstand die Aufsatzes.

Der Aufsatz gliedert sich wie folgt: Zunächst werden die Begriffe Integrität und Konsistenz gegeneinander abgegrenzt. Danach wird grundlegend die Syntax und Semantik der referentiellen Integrität eingeführt. Es schließt sich die Diskussion der Probleme mit diesen Definitionen und die Beschreibung der Semantik von SQL2 an. Um referentielle Integritätsbedingungen mit dieser Semantik effizient zusichern zu können, wird eine intensive Schemaanalyse vorgeschlagen. Abschließend erfolgt eine kritische Betrachtung unseres Ansatzes im Vergleich zu anderen Ansätzen in der Literatur sowie der referentiellen Integrität insgesamt.

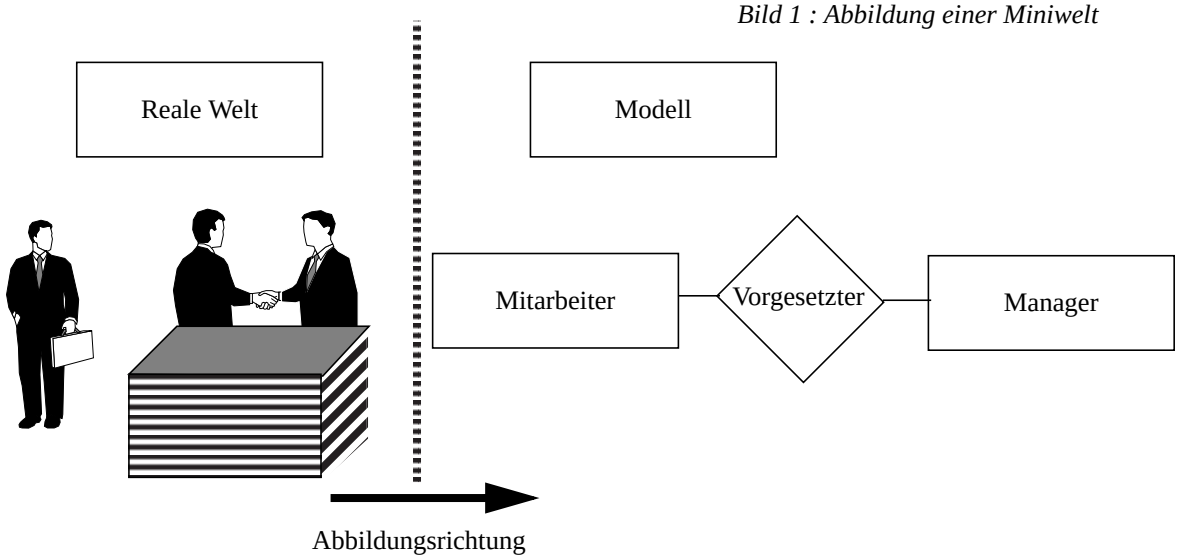
2. Integrität und Konsistenz

Die Begriffe “Integrität” und “Konsistenz” sind in der Datenbankfachwelt durchaus bekannt, jedoch werden sie oft sehr unterschiedlich gebraucht. Im folgenden soll kurz auf die Unterschiedlichkeit beider Begriffe eingegangen und die Sprachregelung für dieses Papier festgelegt werden.

2.1 Modellierung der realen Welt

Die Abbildung einer Miniwelt in ein Modell kann in folgendem Bild (Bild 1 : Abbildung einer Miniwelt) veranschaulicht werden:

Bild 1 : Abbildung einer Miniwelt



Die vielfältigen Facetten der realen Welt können bei der Abbildung unmöglich vollständig im Modell erfaßt werden. Um allerdings eine sinnvolle Unterstützung für die Vorgänge der realen Welt darzustellen, muß das Modell der Realität ermöglicht gut entsprechen, d.h., alle *wesentlichen* Elemente und Vorgänge werden durch das Modell erfaßt. In der Datenbankliteratur unterscheidet man nun zwei Qualitäts-Eigenschaften des Modells. Zum einen bezeichnet der Begriff der *Konsistenz* die Eigenschaft des Modells (bzw. einer konkreten Ausprägung davon) in sich selbst widerspruchsfrei zu sein. Zum anderen beschreibt der Begriff der *Integrität* die Übereinstimmung des Modells (bzw. einer Ausprägung davon) mit der realen Welt. Diese Begriffe beschreiben sehr unterschiedliche Sachverhalte. Wenn beispielsweise die Abbildung der realen Welt falsch ist, so kann das dadurch gewonnene Modell durchaus in sich konsistent sein. Wenn z.B. in der oben dargestellten Welt (Bild 1 : Abbildung einer Miniwelt) einem Manager ein neuer Mitarbeiter zugewiesen wird, ohne daß sich das im Modell, d.h. in der Ausprägung davon, widerspiegelt, so bleibt das Modell zwar widerspruchsfrei, jedoch verletzt es die Forderung nach der Integrität. Es kann jedoch auch vorkommen, daß in der realen Welt widersprüchliche Aussagen bestehen. Werden diese dann trotzdem in das Modell abgebildet, so bleibt die Integrität zwar erhalten, jedoch geht die Konsistenz des Modell verloren.
Damit ergibt sich für Systeme, die Modelle der Realität realisieren sollen, bestenfalls daß, so lange diese Systeme noch keine eigene “Wahrnehmungsfähigkeit” der Realität besitzen, nur die Konsistenz durch das System gewährleistet werden kann, nicht jedoch die Integrität. Leider sind in vielen Modellen für bestimmte Konstrukte Begriffe in Verbindung mit dem Wort “Integrität” eingeführt. Beispielsweise wird im relationalen Modell [Co70] von referentieller Integrität und von Entity-Integrität gesprochen. Der korrekte Ausdruck müßte in diesem Fall eigentlich referentielle Konsistenz respektive Entity-Konsistenz heißen.

2.2 “Integritätsbedingungen” im relationalen Datenmodell

Wie oben schon bemerkt wurde, müßte die Bedingungen, die von innerhalb des relationalen Modells an Daten gestellt werden, eigentlich Konsistenzbedingungen genannt werden. Aus historischen Gründen unterbleibt in diesem Ansatz diese konzeptionelle Unterscheidung.
Im schon erwähnten relationalen Datenmodell [Co70] kann zwischen modellinhärenten und explizit definierten Integritätsbedingungen (eigentlich Konsistenzbedingungen) unterschieden werden. Modellinhärent bedeutet, daß jede Ausprägung dieses Modells diese Integritätsbedingungen implizit erfüllen muß. Dem relationalen Modells sind die Mengeneigenschaft der einzelnen Tabellen (Garantierung des voll definierten Primärschlüssels) sowie die Wahrung der referentiellen Integrität inhärent. Für explizite Integritätsbedingungen (z.B. Assertions, Domain-Checks,...) wurden in der Literatur viele Vorschläge gemacht [EC75, HM75, Da81].
Während andere Vorschläge zur Wahrung von Integrität, z.B. Trigger, schon sehr früh in bestehende relationale Systeme integriert wurden, war die referentielle Integrität, obwohl im ersten Modell-Entwurf von Codd [Co70] enthalten und von Codd [Co84] und Date [Da81, Da86, Da90] weiterentwickelt, bis vor kurzem von keinem kommerziell verfügbaren System realisiert. Experimentelle Prototypen, die vor allem auf die Weiterentwicklung des Datenmodells Wert legten, realisierten meist nur spezielle Teile des Konzeptes der referentiellen Integrität [Mi88]. Einen wesentlichen Impuls erhielt die Realisierung der referentiellen Integrität in vollem Umfang durch die erkennbare Forderung von Anwendern des relationalen Modells. Diese wollten zunächst das als hilfreich erkannte relationale Modell vollständig realisiert sehen, bevor neue Modelle betrachtet werden sollten. Diese Entwicklung schlug sich auch in der Standardisierung von SQL als Manipulationssprache nieder [SQL, SQL2]. Wie die folgende Diskussion deutlich machen wird, liegt ein Hauptgrund der zögernden Realisierung der referentiellen Integrität in den semantischen Problemen, die sich bei näherer Betrachtung des Konzeptes ergeben. Dabei soll auch untersucht werden, wie sich Triggerkonzepte (z.B. [Es76]) für die Realisierung von referentieller Integrität eignen.

3. Referentielle Integrität

3.1 Syntaktische und semantische Definition

Entwicklung der Referentiellen Integrität

Im ursprünglichen Modell von Codd [Co70] wurden die Begriffe Primär- und Fremdschlüssel für Tabellen geprägt. Der *Primärschlüssel* ist ein Attribut oder eine Attributgruppe die jedes Datenbankobjekt (Tupel) eindeutig identifiziert. Neben der Eigenschaft der Eindeutigkeit wird vom Primärschlüssel einer Relation zusätzlich gefordert, daß er für jedes Tupel vollständig definiert ist, d.h., in einem Tupel darf kein Attribut des Primärschlüssel undefiniert sein (den NULL-Wert besitzen). Außerdem muß der Primärschlüssel minimal sein. Das bedeutet in diesem Fall, daß keine echte Teilmenge der Attribute, die den Primärschlüssel darstellen, wiederum einen Schlüssel für die Relation darstellen. Durch ein Datenbanksystem kann im wesentlichen die Eindeutigkeit und die vollständige Definition in jedem Tupel geprüft werden (Konsistenzbedingung). Die Prüfung der Minimalität ist nicht möglich, da diese Eigenschaft in der Abbildung der Miniwelt auf das System enthalten ist (Integritätsbedingung).
In den ersten Ansätzen zur referentiellen Integrität wurden die *Fremdschlüssel* einer Relation dadurch automatisch abgeleitet, daß diese auf der Domäne, d.h. auf dem Wertebereich, eines Primärschlüssels definiert waren. Die Eigenschaft, die mit einem Fremdschlüssel verbunden ist, ist die Mengeninklusion, d.h., jeder Wert eines Fremdschlüssels muß auch als Wert eines entsprechenden Primärschlüssels vorhanden sein. Ausnahmen dieser Regel bilden diejenigen Fremdschlüssel, die einen NULL-Wert enthalten. Da die Mengeninklusionsbeziehung zwischen Primär- und Fremdschlüsseln definiert ist, der Primärschlüssel per Definition jedoch eindeutig ist, entsteht damit automatisch eine 1:n-

Beziehung.
Die Diskussion in der Literatur [Da90] hat jedoch gezeigt, daß es sinnvoller ist, die Definition von Fremdschlüsseln nicht implizit an die Domänen zu knüpfen, sondern eine explizite Definition im konzeptionellen Datenbankschema vorzunehmen. Gleichzeitig wurden für diese dem Datenmodell inhärente Integritätsbedingung Vorschläge für eine erweiterte automatische Wartung durch ein Datenbanksystem gemacht. Während die Einhaltung von Integritätsbedingungen vom Datenbanksystem grundsätzlich zugesichert wird, wurden Definitionsmöglichkeiten vorgeschlagen, auf Verletzungen dieser Integritätsbeziehung in unterschiedlicher Weise zu reagieren. Dabei haben sich folgende mögliche Reaktionen herauskristallisiert:

1. Wenn ein Tupel t gelöscht oder sein Primärschlüssel geändert wird, so wird diese Aktion auf die Tupel t' "kaskadiert", in denen vor der Löschung/Änderung von t ein Fremdschlüssel auf t verwiesen hat.
2. Wenn ein Tupel t gelöscht oder sein Primärschlüssel geändert wird, so wird in den Tupeln t', in denen vor der Löschung/Änderung von t ein Fremdschlüssel fk auf t verwiesen hat, der Fremdschlüssel auf den NULL-Wert gesetzt.
3. Als letzte Variante bleibt die Rücksetzung der Operation, die die Verletzung der Integrität ausgelöst hat.

Es wurden noch weitere Möglichkeiten vorgeschlagen, die jedoch im folgenden außerhalb der Betrachtung bleiben sollen. Eine Diskussion solcher Optionen findet in [MR91] statt.
Im Bereich der relationalen Datenbanksysteme hat sich in den letzten Jahren SQL als umfassende Sprachgrundlage entwickelt, die auch in mehreren internationalen Normen bzw. Normungsvorschlägen verbindlich niedergelegt wurde [SQL, SQL2, SQL3]. Daher soll SQL als syntaktische Definitionsgrundlage für die referentiellen Integritätsbedingungen benutzt werden. Eine referentielle Integritätsbedingung wird, in Anlehnung an SQL2 [SQL2], innerhalb eines CREATE TABLE- oder einem ALTER TABLE-Statement mit der Sprachklausel

```
CREATE TABLE <Tabellenname1>
   ...
   [CONSTRAINT <Constraintname>]
     FOREIGN KEY (<Attributliste1>)
     REFERENCES <Tabellenname2> (<Attributliste2>)
   [ON UPDATE {SET NULL | CASCADE | RESTRICTED}]
   [ON DELETE {SET NULL | CASCADE | RESTRICTED}]
```

definiert.
Dabei ist <Attributliste$_1$> eine Liste von Attributen der Tabelle <Tabellenname$_1$> (beim CREATE TABLE-Statement im gleichen Statement definiert) und <Attributliste$_2$> eine Liste von Attributen der Tabelle <Tabellenname$_2$>. Die <Attributliste$_1$> stellt den Fremdschlüssel dar und die <Attributliste$_2$>, die genausoviele Elemente enthält wie die <Attributliste$_1$> und deren Attribute jeweils auf der gleichen Domäne definiert sind wie die entsprechenden Elemente in der <Attributliste1>, beschreibt eine Attributgruppe der Tabelle <Tabellenname$_2$>, die dort einen Schlüsselkandidaten darstellt. Schlüsselkandidaten sind Attributgruppen, für die die Eigenschaften Minimalität und Eindeutigkeit für Primärschlüssel gelten, jedoch nicht notwendigerweise die Eigenschaft der vollen Definiertheit, d.h., der Primärschlüssel ist ein Schlüsselkandidat, die Umkehrung gilt jedoch im allgemeinen nicht.
In den folgenden Betrachtungen soll nur dann von einer Beziehung zwischen einem Tupel t der Tabelle T mit dem Schlüsselkandidaten sk und einem Tupel t' der Tabelle T' mit dem Fremdschlüssel fk gesprochen werden, wenn in im Schema von T' eine entsprechende Integritätsbedingung definiert ist (s.o.), sk in t und fk in t' jeweils keinen NULL-Wert enthalten und t.sk = t'.fk gilt. Im Normungsvorschlag zu SQL2 kann diese strikte Regel durch verschiedene Optionen unterbrochen werden [MR91].

Verarbeitungsmodell

Ziel ist es Referentielle Integrität in heutigen relationalen Systemen zu unterstützen. Diese Systeme arbeiten wesentlich mit einer ein-Tupel-Schnittstelle. Denn sowohl die meisten der Host Programmiersprachen bieten keine adequaten Möglichkeiten zur Verarbeitung von Mengen als auch der Zugriff auf den Sekundärspeicher (über den DB-Puffer) das Tupel als logische Einheit besitzt. Das unseren Betrachtungen zugrundeliegende Verarbeitungsmodell geht nun von

einer Verarbeitungseinheit aus, der nach dem Modell “a tuple at a time” arbeitet. Die zeitlich geordnete Folge der Tuple die diese Verarbeitungseinheit bearbeitet wird als *Zugriffsreihenfolge* bezeichnet. Die Kontrolle der Verarbeitungseinheit geschieht durch eine Abarbeitungsstrategie, die wiederum in das Datenbanksystem eingebettet ist, das die Semantik der Benutzeroperationen realisiert. Das zentral Problem dieses Papiers ist nun: Gegeben ein relationales Schema mit Referentiellen Integritätsbedingungen und eine konkrete Instanz (konkrete Tupel) desselben. Wie kann garantiert werden, daß jede Operation die der Benutzer ausführt ein eindeutiges Ergebnis besitzt, wenn die obige Semantik für die Referentielle Integrität (bzw. der Folgeoperationen) zugelassen wird. Nur wenn dies gewährleistet ist kann von einer eindeutigen Semantik sprechen.

3.2 Basisbeispiel

Zur Veranschaulichung der auftretenden Probleme wird ein Basisbeispiel eingeführt, das sukzessive verfeinert wird, um bestimmte Effekte der referentiellen Integrität aufzuzeigen.

In einem Unternehmen werden Mitarbeiter innerhalb eines Lohnzahlungszeitraumes in verschiedenen Abteilungen eingesetzt. Um eine genaue Kostenabrechnung durchzuführen, sollen die Anteile an der Arbeitszeit eines Mitarbeiters für eine Abteilung gesammelt werden. Daneben soll die hierarchische Führungsstruktur des Unternehmens erfaßt werden.
Wenn ein Mitarbeiter das Unternehmen verläßt, so werden diese Daten überflüssig. Die ihm gegebenenfalls zugeordneten Mitarbeiter werden zunächst keinem anderen Manager zugeordnet. Außerdem gelte, daß die Auflösung einer Abteilung nur möglich ist, wenn ihr keine Arbeitszeitanteile mehr zugeordnet sind. Modifikationen der Abteilungsnummer sollen zulässig sein, aber die Beziehung zu den entsprechenden Arbeitszeitanteilen muß erhalten bleiben. Demgegenüber soll eine Modifikation der Personalnummer nur dann zulässig sein, wenn keine Arbeitszeitanteile zur Verrechnung ausstehen. Eine vorhandene Beziehung zu den Mitarbeitern soll bei einer solchen Änderung erhalten bleiben.
Bei einer Informationsmodellierung nach dem Entity-Relationship-Modell (ER-Modell) [Ch76] ergibt sich das folgende Bild:

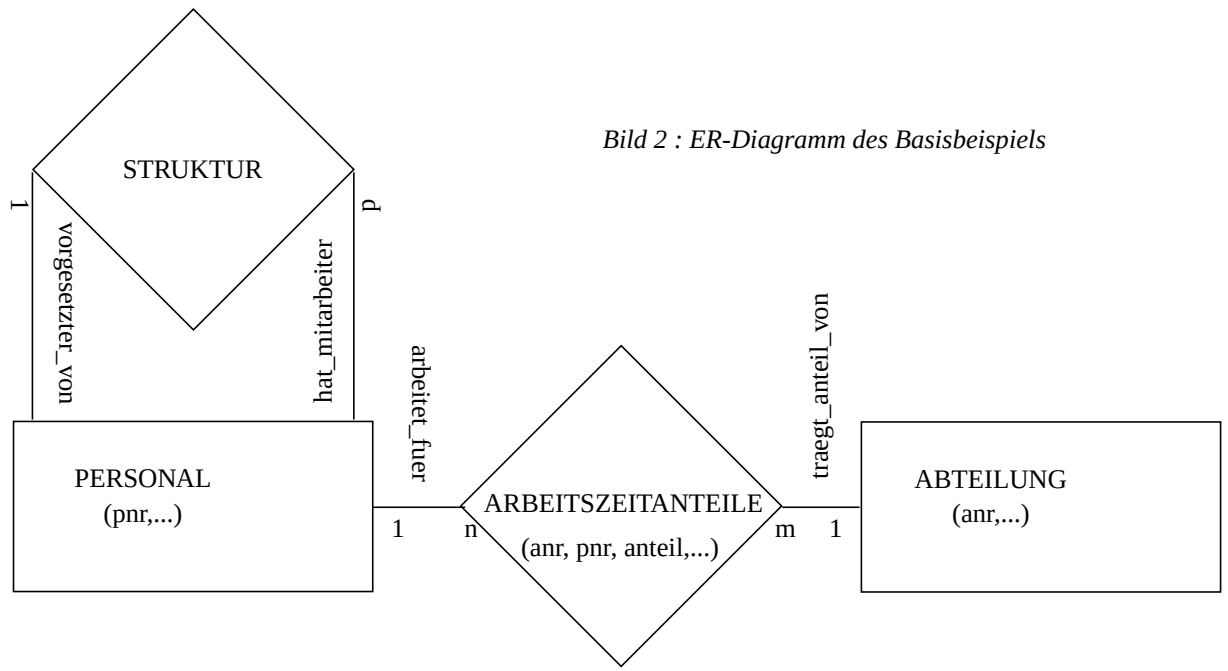


Bild 2 : ER-Diagramm des Basisbeispiels

Aus dem Bild (Bild 2 : ER-Diagramm des Basisbeispiels) wird deutlich, daß die klassischen Möglichkeiten der graphischen Darstellung des ER-Modells nicht ausreichen, um alle Informationen der Miniwelt zu erfassen. Dafür sind

weitaus reichhaltigere Modelle notwendig. Dies ist jedoch nicht Gegenstand dieses Papiers und würde den Rahmen der Diskussion bei weitem sprengen.

Die erweiterten Modellierungsmöglichkeiten im relationalen Modell führen zu folgendem Schema:

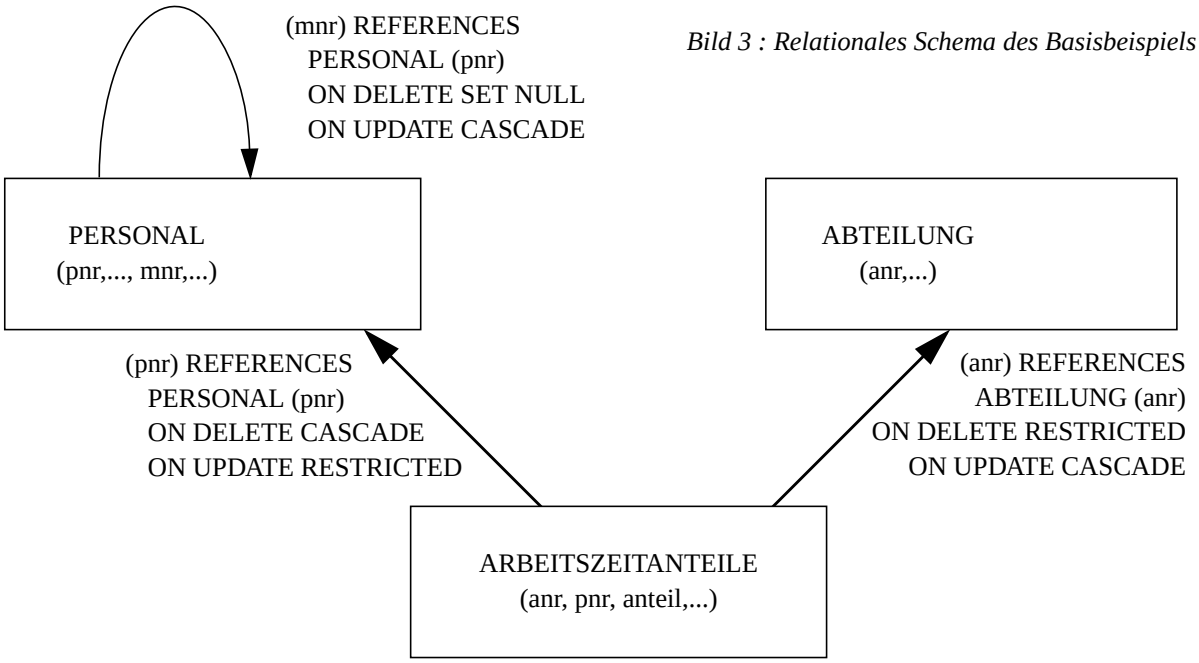


Bild 3 : Relationales Schema des Basisbeispiels

3.3 Lokale Semantik der referentiellen Integrität

Wegen der Definitionsform der referentiellen Integrität (lokal bei der referenzierenden Relation und nach einzelnen Referenzen getrennt) liegt es nahe, auch bei der Interpretation bzw. der Festlegung der Semantik einen lokalen Ansatz zu verfolgen. Allerdings kommt es dabei schon in diesem einfachen Beispiel zu Schwierigkeiten. Betrachtet man folgende Instanz des Schemas:

PERSONAL: (pnr,...,mnr,...):
(001,..., NULL,...)
(002,..., 001,...)

ABTEILUNG: (anr,...)
keine Tupel

ARBEITSZEITANTEILE: (anr, pnr, anteil,...)
keine Tupel

Wenn der Benutzer die Operation

```
DELETE FROM PERSONAL WHERE mnr = NULL
```

ausführt, so ist nicht sofort klar, wie das Ergebnis aussieht:
Wenn das Tupel (001,..., NULL,...) zuerst gelöscht wird und die Integritätssicherung sofort angestoßen wird, d.h., das Tupel (002,..., 001,...) wird zu (002,..., NULL,...), dann ist es prinzipiell denkbar, daß auch dieses Tupel gelöscht wird. Der Grund für diese Unsicherheit (im folgenden als *Abarbeitungs-Mehrdeutigkeit* bezeichnet) liegt in der Frage, ob die WHERE-Bedingung als Vor- oder als Nachbedingung der Operation interpretiert werden soll [Mar91]. Im folgenden wird immer davon ausgegangen, daß sich die Menge der Tupel, die für die Benutzeroperation selektiert wurden, nicht ändert, d.h., die WHERE-Bedingung ist eine *Vorbedingung* für die Operation. Dieses Vorgehen entspricht auch der Darstellung in der relationalen Algebra.

Wenn man von dieser, durch das Modell nicht besser greifbaren, Unklarheit einmal absieht, so stellt das obige relationale Schema ein sehr genaueres Abbild der Miniwelt dar.
Wesentlich größere Schwierigkeiten folgen aus dem generellen Ansatz, die Integritätsbedingungen lokal zu betrachten, denn durch die definierbaren Integritätssicherungsaktionen entsteht bei der Abarbeitung von Benutzeroperationen jedoch ein *globales* Netz von Auswirkungen. In diesem Netz kann es dann bei einer lokalen Abarbeitungsstrategie zu Widersprüchen kommen.

3.3.1 Schema-Mehrdeutigkeiten

Angenommen in der Miniwelt können die Mitarbeiter zwar für mehrere Abteilungen arbeiten, sind jedoch einer Abteilung zugeordnet, der die Verwaltungsaufgaben für die jeweiligen Mitarbeiter zukommen. Wenn eine Abteilung gelöscht wird, so werden die Mitarbeiter freigesetzt. Bei einer Änderung der Abteilung wird die Beziehung zu den Beschäftigten zunächst gelöst, ohne jedoch diese zu entlassen. Ins relationale Modell übertragen, könnte dies so aussehen[1]:

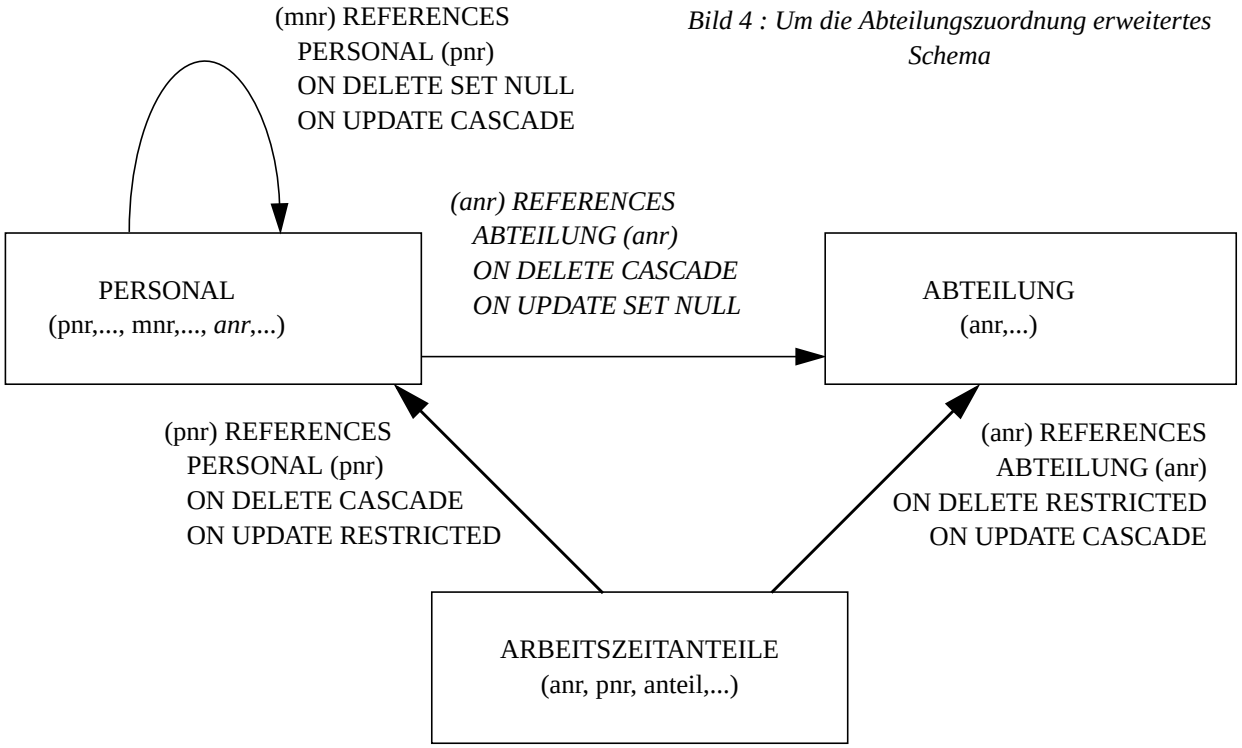


Bild 4 : Um die Abteilungszuordnung erweitertes Schema

Ein Problem dieses Netzes wird sichtbar, wenn man einmal folgende Instanz zugrunde legt:

PERSONAL: (pnr,...,mnr,..., anr,...)
(001,..., NULL,..., 4711,...)
(002,..., 001,..., 4712,...)

ABTEILUNG: (anr,...)
(4711,...)
(4712,...)

1. Erweiterungen und Abweichungen zu vorhergehenden Schemata werden hier und im folgenden jeweils kursiv gedruckt.

ARBEITSZEITANTEILE: (anr, pnr, anteil,...)
(4712, 002, 0.57,...)
(4711, 002, 0.33,...)

Wenn der Benutzer die Operation

```
DELETE FROM ABTEILUNG WHERE anr = 4712
```

ausführt, so kann das Ergebnis davon abhängen, in welcher Reihenfolge die referentiellen Integritätsbedingungen überprüft bzw. gesichert werden (die Auswirkungen pflanzen sich jeweils entgegen der Pfeilrichtung fort). Wird zunächst die Beziehung zur Tabelle ARBEITSZEITANTEILE betrachtet, so muß die Operation zurückgesetzt werden, da dort noch Tupel existieren, die die Abteilung 4712 referenzieren. Wird dagegen zunächst die Beziehung zur Tabelle PERSONAL überprüft, dann wird dort das Tupel (002,..., 001,..., 4712,...) gelöscht. Wird nun die Beziehung der Tabelle PERSONAL zur Tabelle ARBEITSZEITANTEILE betrachtet, so werden die Tupel (4712, 002, 0.57,...) und (4711, 002, 0.33,...) gelöscht. Wenn nun abschließend die Beziehung der ARBEITSZEITANTEILE zur ABTEILUNG geprüft wird, so ist alles in Ordnung und die Operation läuft vollständig durch. Diese Art von Fehler soll als *Schema-Mehrdeutigkeit* (oder einfacher Mehrdeutigkeit) bezeichnet werden. Zur Erkennung dieser Schema-Mehrdeutigkeit werden in einem weiteren Abschnitt einige Bedingungen entwickelt, durch deren Prüfung in einem aktuellen Schema erkannt werden kann, ob eine solche Schema-Mehrdeutigkeit vorliegt. Weitere Situationen von Schema-Mehrdeutigkeiten, die auch bei Änderungsoperationen mit überlappenden Fremdschlüsseln entstehen können, sind in [MR91] und [Mar91] enthalten.

Es gibt jedoch auch eine Art der Mehrdeutigkeit, die nicht durch sich widersprechende Optionen von referentiellen Integritätsbedingungen entstehen.

Das folgende Bild (Bild 5 : Darstellung der Projekte) ist das relationale Modell der wie folgt erweiterten Miniwelt:

- Eine Abteilung ist an der Finanzierung mehrerer Projekte beteiligt.
- Ein Projekt wird von mehreren Abteilungen getragen.
- Für Abrechnungszwecke sind folgende Faktoren zu ermitteln:
  - Der Anteil der Kosten, den eine Abteilung an einem bestimmten Projekt übernimmt
  - Zur Finanzierung eines Projektes gehören auch die Lohnkosten der Mitarbeiter. Deshalb müssen die Kosten erfaßt werden, die vom Lohnkostenanteil eines bestimmten Mitarbeiters einer bestimmten Abteilung zur Finanzierung eines bestimmten Projektes gerechnet werden sollen.

Daneben wurde die Semantik der Beziehung zwischen den ARBEITSZEITANTEILEn und der ABTEILUNG geändert, so daß nunmehr die entsprechenden Arbeitszeitanteile einer Abteilung ebenfalls gelöscht werden, wenn diese gelöscht wird. Für die folgenden Überlegungen sei folgende Instanz des Schemas zugrundegelegt:

PERSONAL: (pnr,...,mnr,..., anr,...):
(001,..., NULL,..., 4711,...)
(002,..., 001,..., 4712,...)

ABTEILUNG: (anr,...)
(4711,...)
(4712,...)

ARBEITSZEITANTEILE: (anr, pnr, anteil,...)
(4712, 002, 0.57,...)
(4711, 002, 0.33,...)

PROJEKTE: (pronr,...)
(0815,...)

PROJEKTFINANZIERUNG: (anr, pronr, anteil,...)
(4712, 0815, 0.22,...)

FINANZIERUNGSANTEILE: (pnr, anr, pronr, betrag,...)
(002, 4712, 0815, 152.000,--,...)

.

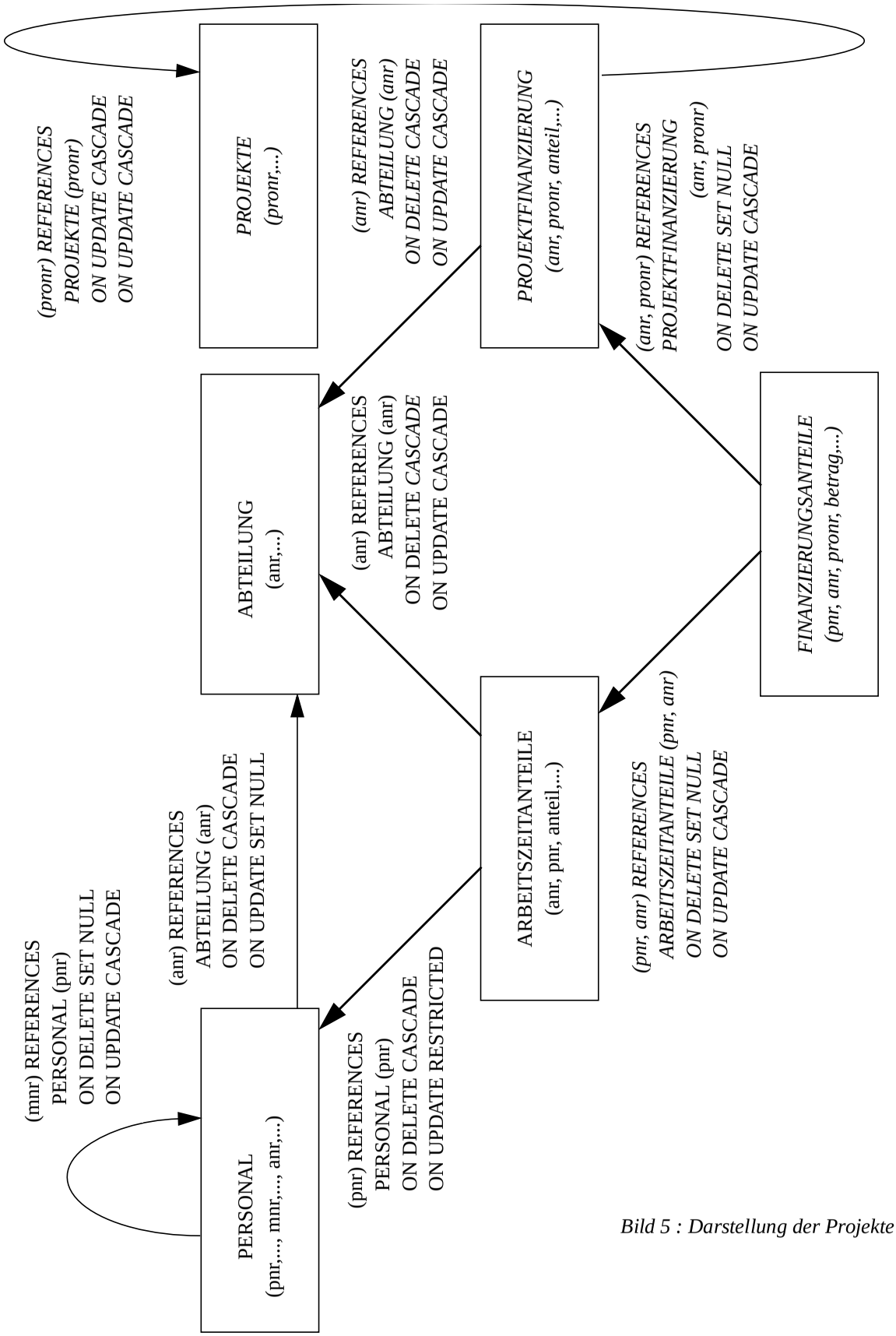


Bild 5 : Darstellung der Projekte

Eine Mehrdeutigkeit entsteht nun, wenn der Benutzer die Operation

```
DELETE FROM ABTEILUNG WHERE anr = 4712
```

ausführt. Der kritische Punkt ist die Tabelle FINANZIERUNGSANTEILE. Wenn zunächst der Pfad (entgegen der Pfeilrichtung) von ABTEILUNG über ARBEITSZEITANTEILE zur Tabelle FINANZIERUNGSANTEILE gewählt wird und der gesamte Fremdschlüssel, der FINANZIERUNGSANTEILE und ARBEITSZEITANTEILE verbindet, d.h. (pnr, anr) im Tupel (002, 4712, 0815, 152.000,--,...), auf den Null-Wert gesetzt wird, und damit das Tupel (NULL, NULL, 0815, 152.000,--,...) entsteht, dann ist dieses Tupel für den anderen Pfad (ABTEILUNG - PROJEKTFINANZIERUNG - FINANZIERUNGSANTEILE) nicht mehr erreichbar, denn der Fremdschlüssel, der die Beziehung aufbaut, ist durch die vorherige Operation zerstört. Wird der Pfad ABTEILUNG - PROJEKTFINANZIERUNG - FINANZIERUNGSANTEILE zuerst ausgewertet, so entsteht das analoge Problem.

Die Mehrdeutigkeiten haben ihren wesentlichen Ursprung darin, daß die vorgestellten Modellierungsmöglichkeiten lokal begrenzt sind und deshalb für eine Abarbeitung zunächst einigen Freiraum entstehen lassen. Allerdings ist klar, daß solche Freiräume für den Benutzer nicht tragbar sind, denn er kann aus den ihm zugänglichen Informationen (DB-Schema und die Operation) nicht entscheiden, welche Ergebnisvariante erreicht wird. Deshalb ist es notwendig, andere Semantiken der referentiellen Integrität zu entwickeln. Grundsätzlich läßt sich den Mehrdeutigkeiten durch ein Verbot begegnen, d.h., mehrdeutige Schemata werden nicht zugelassen. Da die referentielle Integrität jedoch ganz grundlegend mit dem relationalen Modell verbunden ist, muß versucht werden, das Konzept so weit als möglich zu realisieren. Dabei wird dann das Konzept selbst besser verstanden werden, so daß die heute noch unsicheren Grenzen sicherer abgesteckt werden können.

3.4 Globale Semantik der referentielle Integrität

Beim Versuch die Semantik der referentiellen Integrität aus einer globalen Sicht festzulegen, spielt der Grundsatz der *logischen Gleichzeitigkeit* eine wesentlich Rolle. Diese der Serialisierbarkeit ähnliche Eigenschaft soll wie folgt definiert werden:

Definition 1 (logische Gleichzeitigkeit[2]): Zwei Operationen o_1 und o_2 können *logisch gleichzeitig* auf einem Datenbankzustand R ausgeführt werden, wenn gilt, daß $o_1(o_2(R)) = o_2(o_1(R))$.

Ein Benutzer kann aus dem Schema nur die Existenz von referentiellen Integritätsbeziehungen ableiten, die für ihn jedoch alle gleichwertig sind. Damit ist es notwendig, die mit den Integritätsbedingungen verbundenen Aktionen in einer *logischen Gleichzeitigkeit* auszuführen. Um eine solche Gleichzeitigkeit zu erreichen, muß versucht werden, alle Operationen, die für ein Tupel bei der Wartung der referentiellen Integrität anfallen, beim ersten ändernden Zugriff auf ein Tupel durchzuführen oder bei jedem Zugriff soviel Information zu erhalten, daß eine weitere Bearbeitungen möglich ist.
Im vorangehenden Beispiel ist dies relativ einfach. Wenn man die Variante wählt, alle Operationen beim ersten Zugriff durchzuführen und man zuerst den Pfad ABTEILUNG - ARBEITSZEITANTEILE - FINANZIERUNGSANTEILE wählt, so sind für alle Tupel t der Tabelle FINANZIERUNGSANTEILE, die von der Löschung in der Tabelle ARBEITSZEITANTEILE betroffen sind, die Attribute pnr, anr und pronr auf den Null-Wert zu setzen, d.h., man führt die Zuweisung des Null-Wertes an pronr schon beim ersten Zugriff durch. In dieser Art und Weise kann die logische Gleichzeitigkeit jedoch nur unterbestimmten Voraussetzungen an das Datenbankschema (siehe Definition 2, Seite 16). In SQL2 [SQL2] wird ein anderer Weg zur Realisierung der logischen Gleichzeitigkeit gewählt: Alle Änderungen und Löschungen von Tupeln werden protokolliert bzw. auf einer Kopie der Datenbank ausgeführt. Beim Auftauchen einer Mehrdeutigkeit bei der aktuellen Operation wird die Operation abgebrochen, und alle schon durchgeführten Änderungen werden rückgängig gemacht bzw. die Kopien werden einfach “vergessen”. Damit wird die logische Gleichzeitigkeit in vollem Umfang ausgenutzt, um referentielle Integritätsbeziehungen möglichst umfassend zulassen zu können.

2. Der Begriff “logische Gleichzeitigkeit” soll die zeitliche unabhängigkeit betonen. Interpretiert man die Eigenschaft mehr in der Richtung der Algebra, so kann man von Kommutativität sprechen.

Damit kann dann jedoch die Funktionsweise eines Anwenderprogramms sehr stark reihenfolgeabhängig werden, und die Kosten der Auswertung einer Benutzeroperation, die referentielle Integritätsbedingungen berührt, werden durch den nun notwendigen Protokollaufwand wesentlich erhöht. Dies ist nicht nur ein Problem des Antwortzeitverhaltens, sondern zieht auch eine erhöhte Gefahr für Sperrkonflikte nach sich.
Im folgenden soll nun untersucht werden, in welchen Fällen es möglich ist, diesen zusätzlichen Aufwand bei der Abarbeitung einzusparen. Damit ist es dann möglich, die Realisierung einer globalen Semantik, wie sie in SQL2 auf Ausprägungsebene vorgestellt wird, zu möglichst geringen Kosten sicherzustellen.

4. Sichere referentielle Integritätsnetze

Schon sehr früh wurden bei der Untersuchung der referentiellen Integrität die Probleme erkannt, die im vorigen Kapitel geschildert wurden. Daraus wurden dann Anforderungen an die Datenbank-Schemata abgeleitet, die sicherstellen, daß es während der Abarbeitung keine Mehrdeutigkeiten gibt. Diese Anforderungen [IBM, Mar91] beziehen sich jeweils auf ganze Relationen als Betrachtungsgranulat. Dabei bleibt unberücksichtigt, daß die Probleme eigentlich durch Paare von Primär-/Fremdschlüsselbeziehungen entstehen und manche Situationen auf der Ebene der Relationen mehrdeutig erscheinen, es jedoch tatsächlich nicht sind und dies auch auf der Schemaebene erkennbar ist, wenn man die Ebene der Relationen verläßt und statt dessen die Fremd- und Primärschlüssel als Basis der Untersuchung benutzt. Dies soll im folgenden versucht. Ziel ist eine möglichst exakte Beschreibung von Datenbank-Schemata in denen die referentielle Integrität mit Hilfe eines “klassischen” Triggerkonzeptes ohne weitere Vorkehrungen unter Einhaltung der von SQL2 vorgeschriebenen Semantik möglich ist.
Um die problematischen Situationen exakt erfassen zu können, sind einige formale Definitionen hilfreich, die es dann auch ermöglichen, die Aussagen über ein- und mehrdeutige Situationen zu beweisen.

4.1 Darstellung der referentiellen Integritätsbedingungen

Für die referentiellen Integritätsbedingungen wird folgende Darstellung zugrundegelegt.
In der SQL2-Syntax wird die referentielle Integritätsbedingung in der Definition der referenzierenden (Sohn-) Tabelle in der Foreign-Key-Klausel definiert:

```
...FOREIGN KEY (a1,..., an) REFERENCES <tablenname> (b1,..., bn)
  ON UPDATE <update-regel>
  ON DELETE <delete-regel>
```

Dabei sind (a_1,..., a_n) Attribute der Sohn-Tabelle, der <tabellenname> der Bezeichner der referenzierten (Vater-) Tabelle und (b_1,..., b_n) Attribute der Vater-Tabelle.
Für die weitere Definitionen wird eine solche Integritätsbedingung durch ein Paar von Integritätsbedingungen wie folgt dargestellt (t_1 ist dabei die Sohn-, t_2 die Vater-Tabelle):

[($t_1.a_1$,..., $t_1.a_n$), ($t_2.b_1$,..., $t_2.b_n$), <delete-regel>], <delete-regel> ∈ {**D**elete **C**ascade (DC), **D**elete **R**estricted (DR), **D**elete **S**et-**N**ull (DSN)} und
[($t_1.a_1$,..., $t_1.a_n$), ($t_2.b_1$,..., $t_2.b_n$), <update-regel>], <update-regel> ∈ {**U**pdate **R**estricted (UR), **U**pdate **S**et-**N**ull (USN), **U**pdate **C**ascade (UC)}

4.2 Der referential-action-graph (RAG)

Für die Beschreibung der Semantik ist es notwendig, die möglichen Folgen einer Operation zu beschreiben. Dazu bietet es sich an, den folgenden Graphen (referential-action-graph, RAG; siehe auch [SQL2]) zu benutzen, der wie folgt definiert ist (dabei sei I die Menge der Integritätsbedingungen):

1. Der RAG ist ein gerichteter und beschrifteter Multigraph G=(V,E); V ist die Menge der Knoten und E die Menge der Kanten.
2. Es gibt einen speziellen Knoten USER $\in$ V
3. Wenn i = [($t_1.a_1$,..., $t_1.a_n$), ($t_2.b_1$,..., $t_2.b_n$),...] $\in$ I, dann gilt
   {$t_1.a_1$,..., $t_1.a_n$}, {$t_2.b_1$,..., $t_2.b_n$}[3] $\in$ V
   (USER, {$t_2.b_1$,..., $t_2.b_n$}, UC), (USER, {$t_2.b_1$,..., $t_2.b_n$}, DC) $\in$ E
4. Für v = {$t_2.b_1$,..., $t_2.b_n$} $\in$ V mit einer eingehenden Kante k, die mit o markiert ist, gilt:
   - o = DC: Für i $\in$ I mit i = [($t_3.e_1$,..., $t_3.e_l$), ($t_2.f_1$,..., $t_2.f_l$), <delete-regel>] gilt:
     ({$t_2.b_1$,..., $t_2.b_n$}, {$t_3.e_1$,..., $t_3.e_l$}, <delete-regel>) $\in$ E
   - o = UR/DR: keine Fortsetzung, da eine Propagierung spätestens bei einer Regel mit der Option RESTRICTED endet.
   - o = USN/DSN: Für i $\in$ I mit i = [($t_3.e_1$,..., $t_3.e_l$), ($t_2.f_1$,..., $t_2.f_l$), <update-regel>] mit
     {$t_2.b_1$,..., $t_2.b_n$}$\cap${$t_2.f_1$,..., $t_2.f_l$}$\neq\emptyset$ gilt:
     ({$t_2.b_1$,..., $t_2.b_n$}, {$t_3.e_1$,..., $t_3.e_l$}, UR) $\in$ E, falls <update-regel> = UR
     ({$t_2.b_1$,..., $t_2.b_n$}, {$t_3.e_1$,..., $t_3.e_l$}, USN) $\in$ E, sonst
   - o = UC: analog zu o = USN

Der USER-Knoten repräsentiert dabei die Operationen die ein Benutzer ausführen kann. Dies wird deutlich, wenn die Spezialisierungen des RAG's (s.u.) betrachtet werden. Durch die Regel unter 3. werden die einzelnen referentiellen Integritätsbedingungen des Schemas in den RAG aufgenommen. Die Regeln unter 4. und 5. sind für eine vollständige Propagierung aller möglichen Auswirkungen einer Operation verantwortlich.

Zur präzisen Betrachtung einzelner Operationen werden für jede Tabelle t der t-Update- und der t-Delete-Teilgraph des RAG eingeführt. Diese Teilgraphen beschreiben genau die von einer Benutzer-Operation möglichen Auswirkungen. Dabei bedeutet *Teilgraph*, daß die Menge der Knoten eine Teilmenge der Knoten des RAG's und die Menge der Kanten eine Teilmenge der Kanten des RAG's ist, wobei jedoch die Markierung der Kante geändert werden kann:

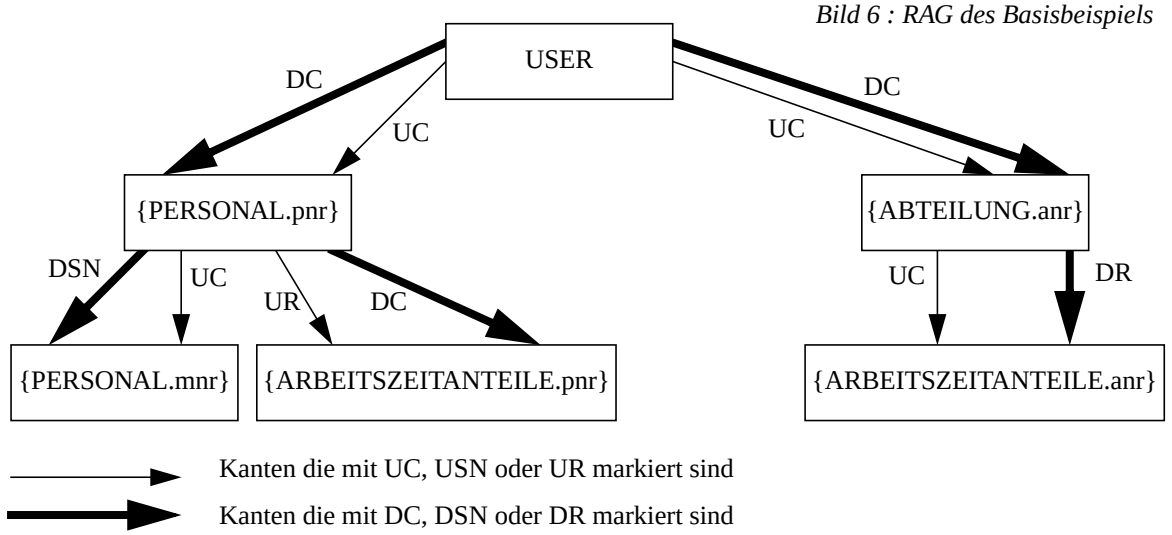
1. Der t-Update-Teilgraph ist ein Teilgraph tUT = (V_{tUT},E_{tUT}) des RAG's G=(V,E), für den gilt:
   - Der Knoten USER $\in$ V_{tUT}.
   - Für alle Knoten v = {$t.a_1$,..., $t.a_n$} $\in$ V gilt, v $\in$ V_{tUT} und (USER, v, UC) $\in$ E_{tUT}.
   - Sei v_1 $\in$ V_{tUT}\{USER}und v_2 $\in$ V, so daß es eine Kante e = (v_1, v_2, <option>) mit <option> $\in$ {UC, USN, UR} in E gibt, dann gilt:
     • v_2 $\in$ V_{tUT}, falls es einen gerichteten Weg w vom Knoten USER nach v_1 in tUT gibt, der keine Kante mit der Markierung UR enthält
     • (v_1, v_2, <option'>) $\in$ E_{tUT} wobei gilt
       <option'> = USN, falls alle Eingangskanten von v_1 im tUT mit USN markiert sind[4]
       <option'> = <option>, sonst
2. Der t-Delete-Teilgraph ist ein Teilgraph tDT = (V_{tDT},E_{tDT}) des RAG's G=(V,E) für den gilt:

3. Es ist dabei zu beachten, daß {$t_1.a_1$,..., $t_1.a_n$} hier nicht als Menge von Knoten verstanden wird, sondern die "Beschriftung eines Knoten im Graphen darstellt.
4. Diese Eigenschaft ist erst nach Erreichen des Fixpunktes für den tUT zu entscheiden!

- Die Knoten USER $\in V_{tDT}$.
- Für alle Knoten $v = \{t.a_1,..., t.a_n\} \in V$ gilt, $v \in V_{tUT}$ und $(USER, v, DC) \in E_{tDT}$.
- Sei $v_1 \in V_{tDT}\backslash\{USER\}$ und $v_2 \in V$, so daß es eine Kante $e = (v_1, v_2, <option>)$ mit <option> $\in$ {DC, DR, DSN, UC, USN, UR} in E gibt, dann gilt:
  - $v_2 \in V_{tDT}$, falls gilt:
    - es gibt einen gerichteten Weg w vom Knoten USER über v_1 nach v_2 in tDT, der keine Kante mit der Markierung UR oder DR enthält
    - wenn <option> $\in$ {DC, DR, DSN} und v_1 mindestens eine Eingangskante mit der Markierung DC besitzt
    - wenn <option> $\in$ {UC, USN, UR} und v_1 mindestens eine Eingangskante mit der Markierung DSN, USN oder UC besitzt
  - $(v_1, v_2, <option'>) \in E_{tDT}$, wobei gilt
    <option'> = USN, falls alle Eingangskanten von v_1 die Markierung USN oder DSN haben und <option> $\in$ {USN, UC}[5]
    <option'> = <option>, sonst

Für das Basisbeispiel (siehe Relationales Schema des Basisbeispiels, Seite 6) ergibt sich folgender RAG:

Bild 6 : RAG des Basisbeispiels



Die wesentlichen Unterschiede zwischen dem relationalen Schema (RS) und dem RAG kann man in zwei Punkten zusammenfassen:

- Die Richtung der Wege der referentiellen Integritätsbedingungen beschreiben im RS die funktionalen Abhängigkeiten. Die Wege im RAG beschreiben die Auswirkungen von Operationen.
- Im RS wird lokal, d.h. für ein Paar von Primär- (respektive Schlüsselkandidaten)/Fremdschlüssel, die Beziehung dargestellt. Im RAG werden die Auswirkungen von Änderungen bzw. Löschungen eines bestimmten Primärschlüssels (respektive Schlüsselkandidaten) erfaßt. Der RAG für das Basisbeispiel ist in Abbildung 6 dargestellt. Man kann daran erkennen, daß Benutzeroperationen auf PERSONAL sowohl Auswirkungen auf PERSONAL selbst als auch auf ARBEITSZEITANTEILE besitzen können.

Wie in obigem Bild (Bild 6 : RAG des Basisbeispiels) deutlich wird, entsteht auch bei sehr kleinen relationalen Schemata ein sehr komplexer RAG. Deshalb wird in den folgenden Beispielen darauf verzichtet, den kompletten Graphen

5. Siehe Anmerkung 5., Seite 14

darzustellen. Es werden dabei nur die wesentlichen Elemente dargestellt.

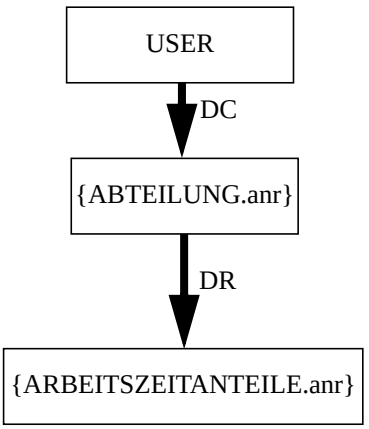


Bild 7 : ABTEILUNG-Delete-Teilgraph des Basisbeispiels

Für das, um die Abteilungszuordnung erweiterte RS entsteht ein ABTEILUNGS-Delete-Teilgraph (Bild 8 : ABTEILUNG-Delete-Teilgraph des erweiterten Basisbeispiels), in dem man deutlich erkennt, daß beim Löschen einer Abteilung eine Mehrdeutigkeit entstehen kann, denn an den Knoten {ARBEITSZEITANTEILE.anr} und {ARBEITSZEITANTEILE.pnr} enden Kanten mit unterschiedlichen Optionen.,

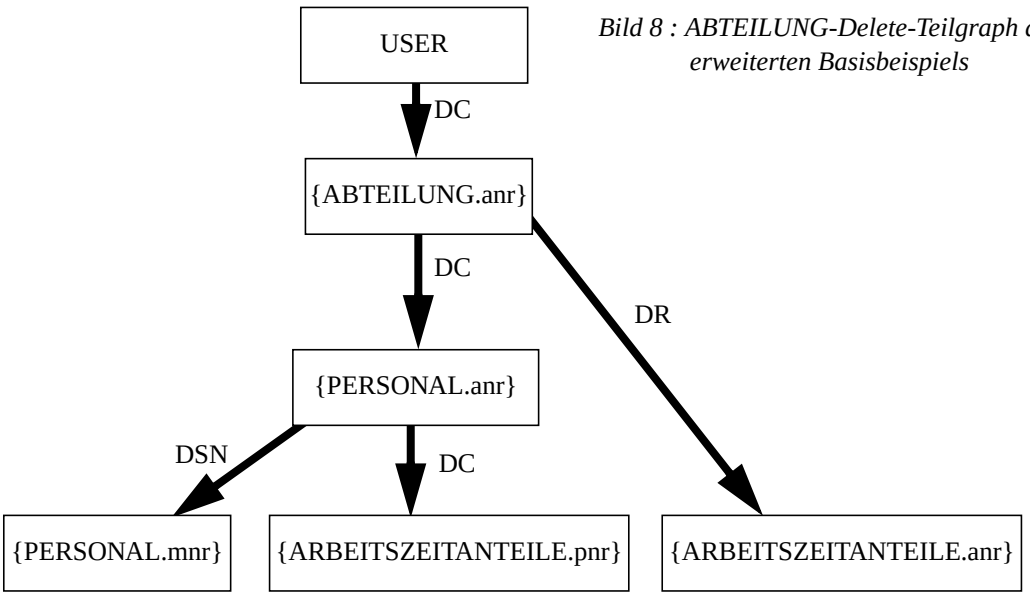


Bild 8 : ABTEILUNG-Delete-Teilgraph des erweiterten Basisbeispiels

Kanten die mit DC, DSN oder DR markiert sind

Diese Eigenschaft soll nun näher beleuchtet und schließlich zur Definition von Bedingungen benutzt werden, die es erlauben, sichere Datenbankschemata von solchen zu unterscheiden, die noch Mehrdeutigkeiten enthalten.

4.3 Sichere Schemata für Delete-Operationen

Das Problem, das in diesem Abschnitt untersucht werden soll, ist die Frage, wann ein gegebenes Schema ohne Mehrdeutigkeiten bezüglich dem Ergebnis einer beliebigen vom Benutzer initiierten Delete-Operation sicher ist, ohne weitere Voraussetzung an die Abarbeitungsstrategien zu machen.

Es wurde bereits erwähnt, daß bei Folge-Update-Operationen einer Delete-Operation vor allem überlappende Fremdschlüssel Probleme erzeugen und daß es dabei auch notwendig sein würde, spezielle Forderungen an das relationale Schema zu stellen, damit nicht auch gleichlautende Optionen an einem Knoten zu Problemen führen (alle Änderungsoperationen zum Zeitpunkt des ersten Zugriffs durchführen). Die formale Grundlage im Fall der Delete-Operation liefert nun dafür die folgende Definition:

Definition 2 (Eigenschaft V): Sei R=(T, A, I) ein relationales Schema mit T einer Menge von Relationennamen, A einer Menge von Attributen und I einer Menge von Integritätsbedingungen. R hat die Eigenschaft V genau dann, wenn für alle $t \in T$ gilt:
Sei $G=(V_{tDT}, E_{tDT})$ der t-Delete-Teilgraph von t, $v_1, v_2 \in V_{tDT}$ mit $v_1 \cap v_2 = m \neq \emptyset$ und Eingangskanten $k_{v_1}, k_{v_2} \in E_{tDT}$, $k_{v_1} \neq k_{v_2}$ die mit DSN oder USN markiert sind.
Wenn ein Tupel l der zu v_1 gehörenden Tabelle als Folge einer Benutzeraktion über die Kante k_{v_1} (Attribute v_1) respektive k_{v_2} (Attribute v_2) geändert werden soll, dann ist zu jedem Zeitpunkt der Abarbeitung entscheidbar, ob l ebenfalls über die Kante k_{v_2} (respektive k_{v_1}) geändert werden wird oder nicht.

Ob ein relationales Schema die Eigenschaft V besitzt (d.h. die Entscheidbarkeit des obigen Problems), hängt sowohl von der Struktur des Datenbankschemas als auch vom Wissen über bestimmte Zugriffsreihenfolgen ab. Dabei ist es möglich, zunächst von einer reinen Schemabedingung (ohne Voraussetzungen an die Abarbeitungs- und Zugriffsreihenfolge für die referentielle Integritätssicherung) auszugehen und diese durch die Kenntnis des Plangenerators bzw. Optimierers schrittweise zu erweitern. Eine hinreichende Schemabedingung gibt der folgende Satz wieder.

Satz 1: Sei R=(T, A, I) ein relationales Schema mit T einer Menge von Relationennamen, A einer Menge von Attributen und I einer Menge von Integritätsbedingungen. R besitzt die Eigenschaft V, wenn für alle t-Delete-Teilgraphen, $G=(V_{tDT}, E_{tDT})$ (für $t \in T$) gilt:
Für alle Knoten $v_1 \in V_{tDT}$, gibt es einen Knoten $\bar{v}_1 \in V_{tDT}$, so daß für alle Knoten $v_2 \in V_{tDT}$ mit $v_1 \cap v_2 \neq \emptyset$ und Eingangskanten $k_1 = (..., v_1, SN)$ und $k_2 = (..., v_2, SN)$ $(k_1 \neq k_2)$ folgende Bedingung erfüllt ist: Für alle Pfade $p = (k_0, ..., k_n)$ mit $E_{tDT} \ni k_h = (e_h, e_{h+1}, <markierung>_h)$ für $0 \leq h \leq n$, $<markierung>_n \in \{USN, DSN\}$, $e_0 = USER$ und $e_n = v_1$ oder $e_n = v_2$ gilt:

1. - Es gibt $i_1, ..., i_n \in I$ mit $i_h = (u_h, w_h, <regel>_h)$ für $1 \leq h \leq n$ mit $w_h \subseteq u_{h+1}$
2. - $e_h \subseteq w_h$, $e_{h+1} \subseteq u_h$, $<regel>_h = <markierung>_h$ für $0 \leq h \leq n$
3. - Es gibt ein i_0 so, daß $e_{i_o} = \bar{v}_1$

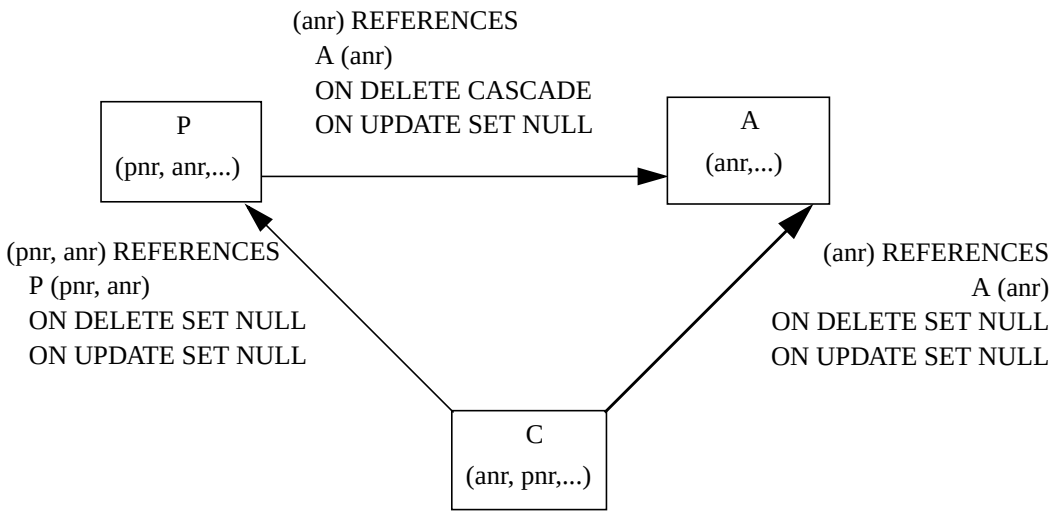
Beweisskizze: Sei ein relationales Schema wie es im Satz spezifiziert wird gegeben und es gibt in diesem Schema einen t-Delete-Teilgraphen in dem es Knoten $v_1, v_2 \in V_{tDT}$ mit $v_1 \cap v_2 \neq \emptyset$ und Eingangskanten $k_1 = (..., v_1, SN)$ und $k_2 = (..., v_2, SN)$ $(k_1 \neq k_2)$. Weiter sei ein Tupel l der Relation t' gegeben, die den Knoten v_1 in den Graphen einführt. Dieses Tupel werde über die Kante k_{v_1} geändert und die Attribute die v_2 betreffen seine alle $\neq$ dem Null-Wert (sonst findet in keinem Fall ein Update über k_{v_2} statt). Wegen der Bedingung 3 des Satzes gibt es einen Grund r für diesen Update, der allen Pfaden, die in v_1 oder v_2 enden gemeinsam ist. Die Bedingung 1 sichert, daß wenn r durchgeführt wird und v1 schließlich geändert wird, daß dann v2 ebenfall geändert wird (sonst ist mindestens eine Referentielle Integritätsbedingung die in einem der Pfade wegen Bedingung 2 enthalten ist nicht erfüllt).

Intuitiv besagt dieser Satz, daß über verschiedene Wege $(k_1 \neq k_2)$ mögliche gleichzeitige $(v_1 \cap v_2 \neq \emptyset)$ Änderungen an einem Tupel dann unkritisch sind, d.h. beim ersten Zugriff vollständig durchgeführt werden können, wenn alle diese Wege von einem Knoten $(\bar{v}_1)$ ausgehen (der einen vollständigen Schlüsselkandidaten darstellt) und der Pfad der referentiellen Integritätsbedingungen ohne Unterbrechung $(w_h \subseteq u_{h+1})$ im entsprechenden Graphen enthalten ist.
Diese Eigenschaft sichert nun, daß gleichlautende Änderungsoperationen bei der Sicherung der referentiellen Integrität völlig unabhängig von eventuellen Zugriffsreihenfolgen auf die Tupel oder Reihenfolgen der Integritätsbedingungen zugelassen werden können. Dazu muß nun bei der Abarbeitung nur bekannt sein, welcher RAG abgearbeitet wird und ob als Strategie für überlappende Fremdschlüssel, die Änderungsoperationen beim ersten Zugriff komplett ge-

macht werden sollen oder ob jeweils genügend Restinformation bestehen bleiben soll.
Allerdings schränkt die Eigenschaft V die zulässigen Schemata ein. Deshalb ist es, um den Wartungsaufwand der SQL2-Semantik von referentieller Integrität während der Compilezeit schon so weit wie möglich optimieren zu können, hier, wie auch später bei der Darstellung der Ermittlung der generellen Schemaeigenschaften auch, sinnvoll, möglichst viel Wissen über die Zugriffsreihenfolgen, die der Optimierer bzw. das Zugriffsssytem erzeugt, einfließen zu lassen. Durch dieses Wissen kann dann die Eigenschaft V, nun allerdings *system-* und *implementierungsspezifisch*, für eine größere Klasse von Schemata zugesichert werden.

Die folgenden Beispiele sind sehr künstlich. Dies wird dadurch bedingt, daß die Probleme nur in Randbereichen auftreten, für die keine gängigen Beispiele vorhanden sind. Trotzdem muß ein Datenbanksystem auch solche Fälle erkennen und darauf korrekt reagieren können.
Im Schema, das in folgendem Bild (Bild 9 : Schema mit Eigenschaft V) dargestellt ist, muß berücksichtigt werden, daß die Attributkombination (anr, pnr) kein Primärschlüssel in C ist, da sonst die Optionen SET NULL nicht zulässig sind.

Bild 9 : Schema mit Eigenschaft V



Kritisch sind Löschoperationen für die Tabelle A. Der A-Delete-Teilgraph für dieses Schema sieht wie folgt aus:

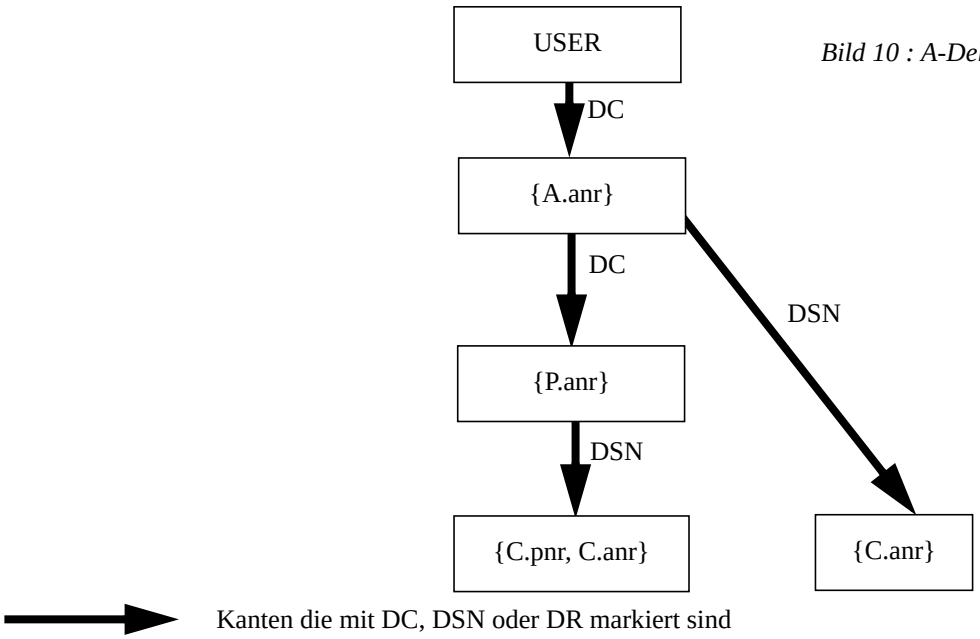


Bild 10 : A-Delete-Teilgraph I

Wenn nun ein C-Tupel t als Folge einer Löschung in der Tabelle A über den Weg (USER, {A.anr}, {C.anr}) geändert werden soll und t.pnr ebenfalls definiert ist, kann t.pnr ebenfalls auf den Null-Wert gesetzt werden, da der Weg (USER,{A.anr}, {P.anr}, {C.pnr,C.anr}) in diesem Fall auch beschritten werden muß.

Betrachtet man allerdings folgendes, leicht veränderte Schema, so ist diese Schlußfolgerung falsch. Zu beachten ist dabei noch, daß der Primärschlüssel von P die Attributkombination (pnr, anr) sein soll.

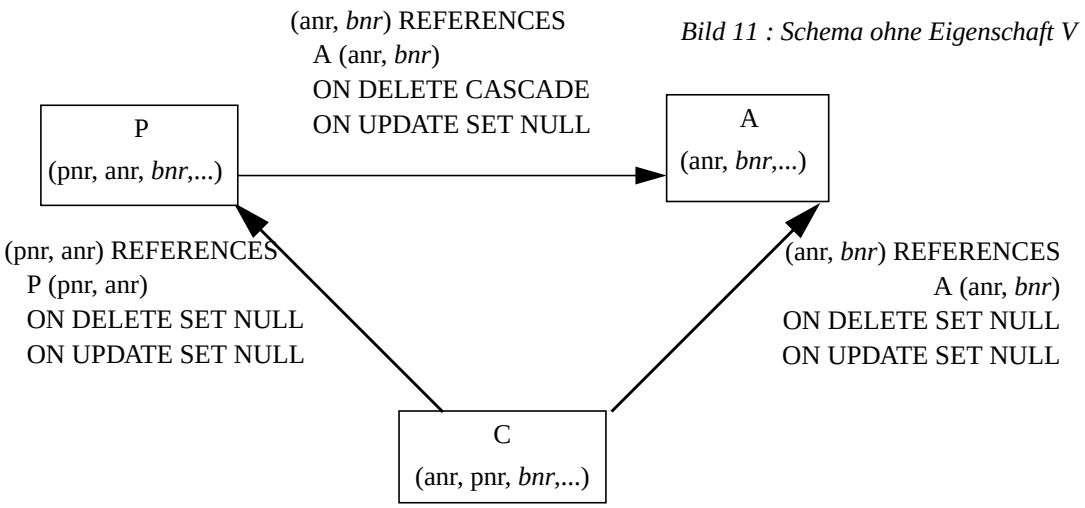


Bild 11 : Schema ohne Eigenschaft V

Der zugehörige A-Delete-Teilgraph sieht wie folgt aus:

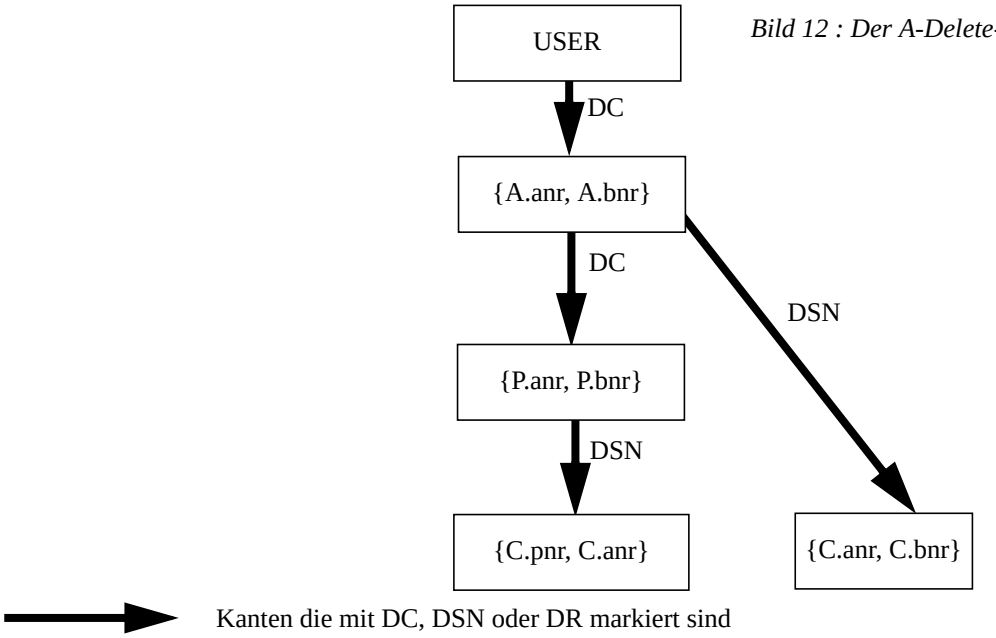


Bild 12 : Der A-Delete-Teilgraph II

Wenn man nun folgende Instanz des Schemas betrachtet, so wird die Mehrdeutigkeit bei einer beliebigen Zugriffsfolge deutlich:

A: (anr, bnr,...)
(01, 02,...)
P: (pnr, anr, bnr,...)
(22, 01, NULL,...)
C: (pnr, anr, bnr,...)
(22, 01, 02,...)

Wenn nun das Tupel (01, 02,...) aus A gelöscht wird und danach zunächst der Pfad A-C eingeschlagen wird, so ist im Tupel (22, 01, 02,...) nicht zu erkennen, daß dieses Tupel über den Weg A-P-C nicht erreicht wird und deshalb maximal der Fremdschlüssel (anr, bnr) den Null-Wert erhalten darf. Der in obigem Schema (Bild 12 : Der A-Delete-Teilgraph II) dargestellte A-Delete-Teilgraph widerspricht der im Satz 1 aufgestellten Bedingung in dem Punkt, daß es für den Weg (USER, {A.anr, A.bnr}, {P.anr, P.bnr}, {C.pnr, C.anr}) keine entsprechende Folge von Integritätsbedingungen gibt, die die Inklusionsbedingung erfüllen.

Mit der Eigenschaft V beginnt man, zunächst auf abstrakter und implementierungsunabhäniger Ebene, die Abarbeitungsstrategie explizit in die Betrachtung von sicheren Integritätsnetzen mit einzubeziehen. Dieser Ansatz kann bei einer konkreten Implementierung durch Wissen über den Optimierer weiter verfeinert werden. Wenn der Optimierer beispielsweise für die Änderungen in einer Tabelle immer zuerst die Änderungen aller Eingangskanten in einem Zwischenspeicher (Table-Queues) sammelt und erst im Anschluß daran die Operation anwendet, dann besitzen natürlich mehr Schemata die Eigenschaft V (beispielsweise auch das Schema aus Bild 11 : Schema ohne Eigenschaft V). Wie schon bemerkt, liegt hier das Optimierungspotential im Optimierer bzw. in der Plangenerierung.

Für die anschließenden Betrachtungen wird folgende Abarbeitungsstrategie zugrunde gelegt:

Abarbeitungsstrategie UEZ: Alle **U**pdate-Operationen, die eine konkrete Überlappung betreffen, werden beim **e**rsten **Z**ugriff auf das Tupel über einen der Pfade, der diese Überlappung betrifft, vollständig durchgeführt. Sind bei diesem Zugriff weitere Fremdschlüssel (die zu der Überlappung gehören) vollständig definiert, so wird auch dieser Fremdschlüssel auf den Null-Wert gesetzt.

Es geht nun um die Frage, welchen Bedingungen ein Schema genügen muß, damit keine Mehrdeutigkeiten durch widersprüchliche Operationen entstehen. Dazu werden wieder die t-Delete-Teilgraphen benutzt. In diesen Graphen sind alle Auswirkungen, die von einer Delete-Operation des Benutzer ausgehen können, erfaßt.

Definition 3 (induzierter Fremdschlüssel): Sei t eine Relation mit dem t-Delete-Teilgraphen $G=(V_{tDT}, E_{tDT})$, $v \in V_{tDT}$ und t' die Tabelle zu der die Attribute von v gehören. Dann heißt der Fremdschlüssel fk von t' der von v induzierte Fremdschlüssel bezüglich V_{tDT} (oder kurz: induzierter Fremdschlüssel), wenn es eine Integritätsbedingung $I \ni i = (fk, t''.sk, <regel>)$ gibt, so daß i ein Grund für die Aufnahme von v in V_{tDT} ist.

Definition 4 (Delete-Mengen): Sei t eine Relation mit dem t-Delete-Teilgraphen $G=(V_{tDT}, E_{tDT})$. Dann werden folgende Mengen für die Relationen t', die in G referenziert werden, definiert:

- $D_t(t')=\{t'.a \in Attribute(t') \mid \exists v \in V_{tDT}, t'.fk$ der von v induzierte Fremdschlüssel, $t'.a \in t'.fk$ und $\exists k \in E_{tDT}$ mit $k = (\overline{v}, v, <mark>)$, $\overline{v} \neq USER$ und $<mark> = DC\}$
- $SN_t(t')=\{t'.a \in Attribute(t') \mid \exists v \in V_{tDT}, t'.fk$ der von v induzierte Fremdschlüssel, $t'.a \in t'.fk$ und $\exists k \in E_{tDT}$ mit $k = (\overline{v}, v, <mark>)$ und $<mark> = USN$ oder $DSN\}$
- $R_t(t')=\{t'.a \in Attribute(t') \mid \exists v \in V_{tDT}, t'.fk$ der von v induzierte Fremdschlüssel, $t'.a \in t'.fk$ und $\exists k \in E_{tDT}$ mit $k = (\overline{v}, v, <mark>)$ und $<mark> = UR$ oder $DR\}$

In der Menge $D_t(t')$ werden die Kanten, die vom Knoten USER ausgehen, nicht betrachtet, da von folgender Verarbeitungssemantik ausgegangen wird: Es werden zunächst die sich qualifizierenden Tupel selektiert, d.h., die Querybedingung stellt die Vorbedingung für die Operation dar. Die so ermittelte Menge von Tupel, die von der Löschung *direkt* betroffen sind, ändert sich *während* der Verarbeitung *nicht* mehr. Damit können Löschungen, die direkt vom Benutzer verlangt werden, nicht mit Folgeänderungen/-löschungen kollidieren und werden deshalb nicht weiter betrachtet.

In [Mar91] wird eine hinreichende Schema-Bedingung angegeben, die beschreibt, wann relationale Schemata ohne Mehrdeutigkeiten sind, die aus referentiellen Integritätsbedingungen resultieren können. Diese Bedingung formalisiert, ähnlich dem RAG, die Abhängigkeiten, die durch die referentielle Integrität auftreten können. Allerdings ist dabei die kleinste Einheit, die betrachtet wird, eine Relation als Ganzes und nicht wie beim RAG bzw. den davon abgeleiteten Teilgraphen der Fremdschlüssel. Durch dieses feinere Betrachtungsgranulat ist es möglich, eine größere Klasse von relationalen Schemata als widerspruchsfrei zu erkennen und damit die, für die von SQL2 vorgeschriebene Semantik notwendigen Sicherungsmaßnahmen in mehr Fällen zu optimieren. Außerdem wurde durch die Formalisierung der Eigenschaft V die Möglichkeit geschaffen, auch überlappende Fremdschlüssel mit der gleichen Aktion zu erlauben.
Der folgende Satz beschreibt nun das von uns entwickelte hinreichende Kriterium:

Satz 2: Sei R=(T, A, I) ein relationales Schema, mit einer Menge T von Relationennamen, A einer Menge von Attributen der Form t.a mit $t \in T$ und I einer Menge von referentiellen Integritätsbedingungen der Form
$[(t_1.a_1,..., t_1.a_n), (t_2.b_1,..., t_2.b_n), <delete-regel>]$, $<delete-regel> \in \{DC, DR, DSN\}$ bzw.
$[(t_1.a_1,..., t_1.a_n), (t_2.b_1,..., t_2.b_n), <update-regel>]$, $<update-regel> \in \{UC, UR USN\}$, das die Eigenschaft V besitzt und als Abarbeitungsstrategie wird UEZ benutzt, dann gilt:
Wenn für alle t-Delete-Teilgraphen die Bedingungen
1. $R_t(t)=\emptyset$
2. $(\forall t' \in R).D_t(t') \neq \emptyset \rightarrow R_t(t') = \emptyset$
3. $(\forall t' \in R).D_t(t') \cap SN_t(t') = \emptyset$
4. $(\forall t' \in R).R_t(t') \cap SN_t(t') = \emptyset$

gelten, dann sind die Operationen o des Typs

```
DELETE FROM t̂ WHERE <prädikat>
```

für alle Instanzen $\hat{R}$ von R von der Zugriffsreihenfolge auf Tupel in $\hat{R}$ oder der Reihenfolge der Überprüfung und Sicherung von referentiellen Integritätsbedingungen unabhängig.

Beweis:

(Widerspruchsbeweis)
Annahme: Es gibt ein relationales Schema R=(T, A, I), wie oben definiert, und eine Instanz $\hat{R}$ von R und eine Benutzer-Operation des Typs o, so daß das Ergebnis von o von der Zugriffsreihenfolge und der Sicherungsreihenfolge der referentiellen Integritätsbedingungen abhängig ist, jedoch die Bedingungen 1. - 4. von R erfüllt werden.
Dann gilt ($o(\hat{R})$ ist dabei das Ergebnis der Ausführung von o auf den Datenbankzustand $\hat{R}$):

1. $o(\hat{R})=\hat{R}$ (es tritt ein RESTRICTED auf) und $o(\hat{R}) = \hat{R}' \neq \hat{R}$

   Da $o(\hat{R})=\hat{R}$ möglich ist, existiert im t-Delete-Teilgraphen $G=(V_{tDT}, E_{tDT})$ ein Knoten v und ein Knoten v_1, so daß

   (v_1, v, UR) oder (v_1, v, DR) in E_{tDT} und die Instanz $\hat{t}'$ der Relation $t' \in T$, die zu $v=\{t'.a_1,..., t'.a_m\}$ gehört ist nicht leer und es gibt Integritätsbedingungen $i_1,..., i_{n-1}$der Form
   $i_j = [t_j.fk, t_{j-1}.sk, DC]$ oder $i_j=[t_j.fk, t_{j-1}.sk, UC]$ oder $i_j = [t_j.fk, t_{j-1}.sk, DSN]$ und
   $i_n = [t'.fk, t_{n-1}.sk, DR]$ oder $i_j=[t'.fk, t_{n-1}.sk, UR]$, so daß gilt:
   (i) $t_0=t$ und entweder
   (ii) $i_1,..., i_{n-1} = [..., DC]$ und $i_n = [..., DR]$ oder
   (iii) $i_1,..., i_k = [..., DC]$ und $i_k = [..., DSN]$ und $i_{k+1},...,i_{n-1} = [..., DC$ oder $UC]$ und
   für $kJ\leq n$ gilt $t_{i_j}.fk \cap t_{i_j}.sk \neq \emptyset$ und $i_n = [..., UR]$

   Fallunterscheidung:
   a. $t_n = tDT$: $R_t(t)\neq\emptyset$. $\perp$
   b. Es gelte (ii) von oben, d.h. es gibt entsprechende $i_1,..., i_{n-1} = [..., DC]$ und $i_n = [..., DR]$, und in $\hat{R}$ wird diese Integritätsbedingung auch tatsächlich überprüft, und es wird noch ein abhängiges Tupel gefunden (da RESTRICTED eintritt). Deshalb gibt es eine Folge von Tupeln $l_0,..., l_n$, für die gilt
      <prädikat> (l_1) = TRUE (<prädikat> (l) bedeutet, daß das Prädikat <prädikat> auf l angewendet wird)
      Null-Wert $\notin l_i.sk$ für $0 \leq i \leq n-1$
      Null-Wert $\notin l_i.fk$ für $1 \leq i \leq n$
      Für die andere Folge von Operationen, in der RESTRICTED nicht ausgelöst wird, gilt, daß mindestens das Tupel l_0 ebenfalls gelöscht wird. Damit folgt:
      α. $l_n \notin \hat{R}'$: Dann gilt nach der Konstruktion von G, daß $D_t(t')\neq\emptyset$ und $R_t(t')\neq\emptyset$. $\perp$
      β. $l_n \in \hat{R}'$: Betrachte nun j minimal, so daß $l_{j-1} \notin \hat{R}'$, aber $l_j \in \hat{R}'$. Dann gilt, daß die Aktion der Integritätsbedingung i_j nicht ausgeführt wird. Damit folgt
      • entweder $SN_t(t_n) \cap R_t(t_n) \neq\emptyset$ (falls j = n). $\perp$
      • oder $SN_t(t_n) \cap D_t(t_n) \neq\emptyset$ (falls j < n). $\perp$
   c. Es gelte nun (iii), d.h., es gibt entsprechende $i_1,..., i_{k-1} = [..., DC]$ und $i_k = [..., DSN]$ und $i_{k+1},...,i_{n-1} = [..., DC$ oder $UC]$ und für $i_k<j\leq n$ gilt $t_{i_j}.fk \cap t_{i_j}.sk \neq \emptyset$ und $i_n = [..., UR]$. Zu dieser Folge von Integritätsbedingungen gibt es in $\hat{R}$ eine Folge von Tupeln $l_0,..., l_n$ (wie im Fall b). Damit gilt nun:
      α. $l_n \notin \hat{R}'$: Dann gilt nach der Konstruktion von G, daß $D_t(t')\neq\emptyset$ und $R_t(t')\neq\emptyset$. $\perp$
      β. $l_n \in \hat{R}'$:
      - $\exists$ j minimal, j<k, so daß $l_j \in \hat{R}'$. Dann gilt aber $l_{j-1} \notin \hat{R}'$ und die Löschoperation kann nicht mehr ausgeführt werden, dann gilt aber $SN_t(t_j) \cap D_t(t_j) \neq\emptyset$. $\perp$
      - $\exists$ j maximal, $k \leq j < n$, so daß $l_j \notin \hat{R}'$. Dann gilt nach der Konstruktion des tDT, daß es Knoten v_1, v_2, v_3, $v_4 \in V_{tDT}$ gibt mit
        • $v_1 \subseteq t_j.fk$ (der Fremdschlüssel, der bei der ersten Operationsfolge benutzt wird), v_1 hat eine Eingangskante die mit USN oder DSN markiert ist.
        • $v_2 \subseteq t_j.fk'$ (der Fremdschlüssel, der bei der zweiten Operationsfolge benutzt wird), v_2 hat eine Eingangskante die mit DC markiert ist.

- $v_3 \subseteq t_{j+1}.fk$ (der Fremdschlüssel, der bei der ersten Operationsfolge benutzt wird), (v_1, v_3, USN oder UR) $\in E_{tDT}$, da t_j nicht die letzte Tabelle der Folge sein kann.
- $v_4 = t_{j+1}.fk$ und (v_1, v_3, <mark>) $\in E_{tDT}$, da es eine Verbindung von t_j nach t_{j+1} gibt (über v_1 und v_3) und deshalb laut Konstruktion v_4 eingefügt werden muß. Insbesondere gilt $v_4 \cap v_3 \neq \emptyset$. Damit muß <mark> mit USN oder UR kompatibel sein, sonst entsteht direkt ein Widerspruch.
  (i) <mark> = DSN: Da nun jedoch gilt, daß alle Tupel l_m, m > j, in $\hat{R}'$ enthalten sind, kann die Verbindung zu l_n nur dadurch aufgebrochen werden, daß ein Teil eines notwendigen Fremdschlüssels vorzeitig auf den Null-Wert gesetzt wird. Dies widerspricht jedoch der Abarbeitungsstrategie UEZ. $\perp$
  (ii) <mark> = DR: Da l_n noch existiert, muß ein Teil des notwendigen Fremdschlüssel durch Null-Werte ersetzt werden, damit gilt jedoch $SN_t(t_n) \cap R_t(t_n) \neq \emptyset$. $\perp$

- Damit gilt nun, daß $l_0,..., l_{k-1} \notin \hat{R}'$ und $l_k,..., l_n \in \hat{R}'$. Das bedeutet jedoch, daß eine Unterbrechung der Kette, damit die RESTRICTED-Option nicht erreicht wird, nur über ein teilweises Null-setzen eines notwendigen Fremdschlüssels geschehen kann. Dies widerspricht jedoch der Abarbeitungsstrategie UEZ (nämlich alles, was einen Fremdschlüssel betrifft, beim ersten Zugriff zu tun). $\perp$

2. $o(\hat{R})=\hat{R}_1 \neq \hat{R}$, $o(\hat{R})=\hat{R}_2 \neq \hat{R}$ und $\hat{R}_1 \neq \hat{R}_2$

   a. $\exists\, l \in \hat{R}_1$ und $l \notin \hat{R}_2$: In $\hat{R}$ existiert eine Folge von Tupeln $l_1,..., l_n = l$ mit $l_i.sk = l_{i+1}.fk \neq$ (NULL,..., NULL) (konstruktion wie im Fall 1). Betrachte nun j minimal mit $l_j \notin \hat{R}_1$ und $l_j \in \hat{R}_2$, dann gilt $l_{j+1}.fk \neq$ (NULL,..., NULL), sonst müßte l_j gelöscht werden. Damit gilt jedoch $SN_t(t_{j+1}) \cap D_t(t_{j+1}) \neq \emptyset$. $\perp$

   b. $\exists\, l \notin \hat{R}_1$ und $l \in \hat{R}_2$: Analog zu Fall a. $\perp$

   c. $\exists\, l_{\hat{R}_1} \in \hat{R}_1$ und $l_{\hat{R}_2} \in \hat{R}_2$ und $l_{\hat{R}_1} \neq l_{\hat{R}_1}$: Da $l_{\hat{R}_1}$ und $l_{\hat{R}_2}$ existieren, besteht nur die Möglichkeit, daß sich in den Werten eines Attributes unterscheiden. Dabei besteht nur die Möglichkeit, daß der Wert eines Attributes in einem Tupel gleich dem Null-Wert ist, während das Attribut im anderen Tupel den ursprünglichen Wert besitzt. Damit überlappen sich jedoch mehrere Fremdschlüssel, wobei in den Tupeln $l_{\hat{R}_1}$ und $l_{\hat{R}_2}$ unterschiedliche Teile der Fremdschlüssel auf den Null-Wert gesetzt werden. Da jedoch in der Voraussetzung festgelegt wurde, daß das Schema die Eigenschaft V besitzt und die Abarbeitungsstrategie UEZ beachtet werden muß, kann dieser Fall nicht auftreten (beim ersten Zugriff werden alle betroffenen Fremdschlüssel auf den Null-Wert gesetzt). Damit wird dieser Fall ebenfalls zum Widerspruch geführt. $\perp$

Damit ist alles bewiesen. □

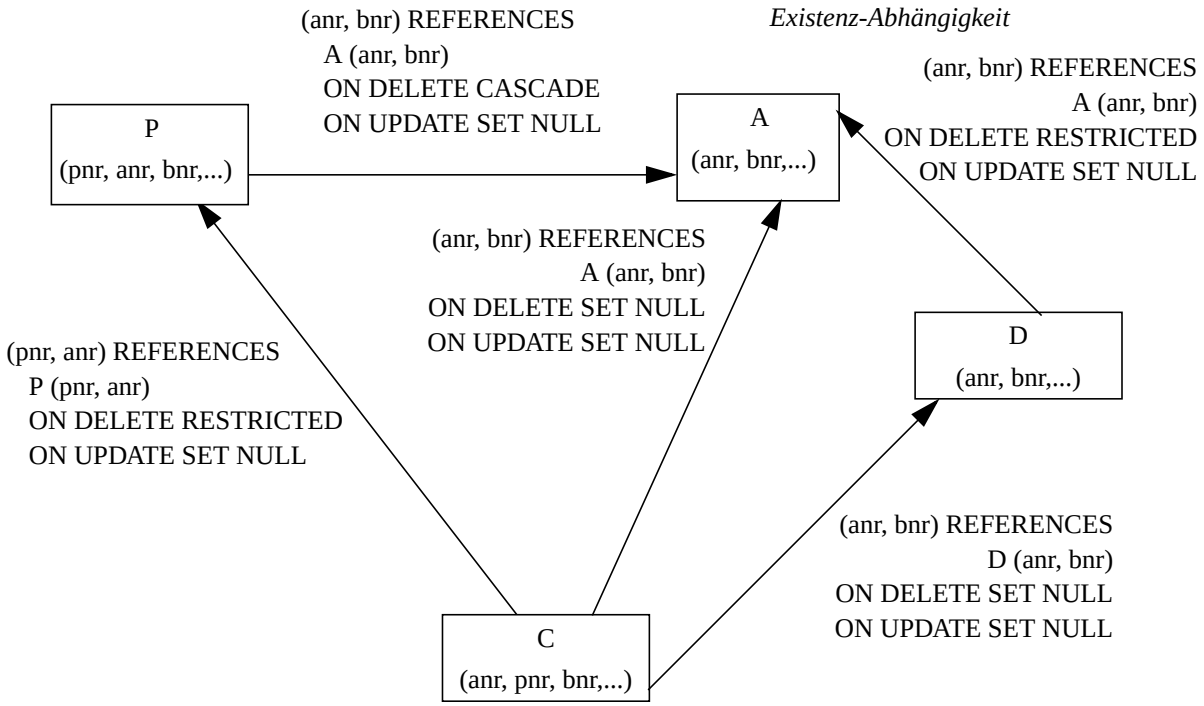
Die Entwicklung eines notwendigen Kriteriums stellt sich schwierig dar, denn die Eigenschaft, “widerspruchsfrei” zu sein, hängt nicht nur von den referentiellen Integritätsbedingungen ab, die durch eine Operation direkt betroffen sind, sondern auch von solchen, die gewisse Existenzen von Tupeln sicherstellen.

4.4 Existenzabhängigkeiten in DB-Schemata

Zwischen Tupeln t_1 und t_2 besteht eine Existenzabhängigkeit, wenn t_1 nur dann in einem Datenbankzustand, der alle Integritätsbedingungen erfüllt, enthalten sein kann, wenn t_2 im gleichen Datenbankzustand enthalten ist.
Im Falle von referentiellen Integritätsbedingungen herrschen solche Existenzabhängigkeiten im wesentlich nicht zwischen Tupeln selbst, sondern die Eigenschaft wird darauf eingeschränkt, daß für t_2 kein komplettes Tupel vorgeschrieben wird, sondern daß nur ausgesagt wird, daß ein Tupel mit einem bestimmten Primärschlüssel bzw. Schlüsselkandidaten vorhanden sein muß. Jede referentielle Integritätsbedingung stellt damit natürlich auch eine Existenzabhängigkeit dar. Es gibt auch dabei eine Kaskadierung, ähnlich den t-Delete- und t-Update-Teilgraphen. Allerdings ist diese nicht mit den referentiellen Aktionen verknüpft, sondern hängt nur von strukturellen Bedingungen ab. So ist es z.B. notwendig, daß Fremdschlüssel und Primärschlüssel bzw.Schlüsselkandidaten (die in einer referentiellen Integritätsbedingung referenziert werden) einer Relation sich entweder nicht überlappen oder aber der Primärschlüssel respektive der Schlüsselkandidat umfaßt den Fremdschlüssel. Ansonsten geht die Eigenschaft der Existenabhängigkeit verloren.

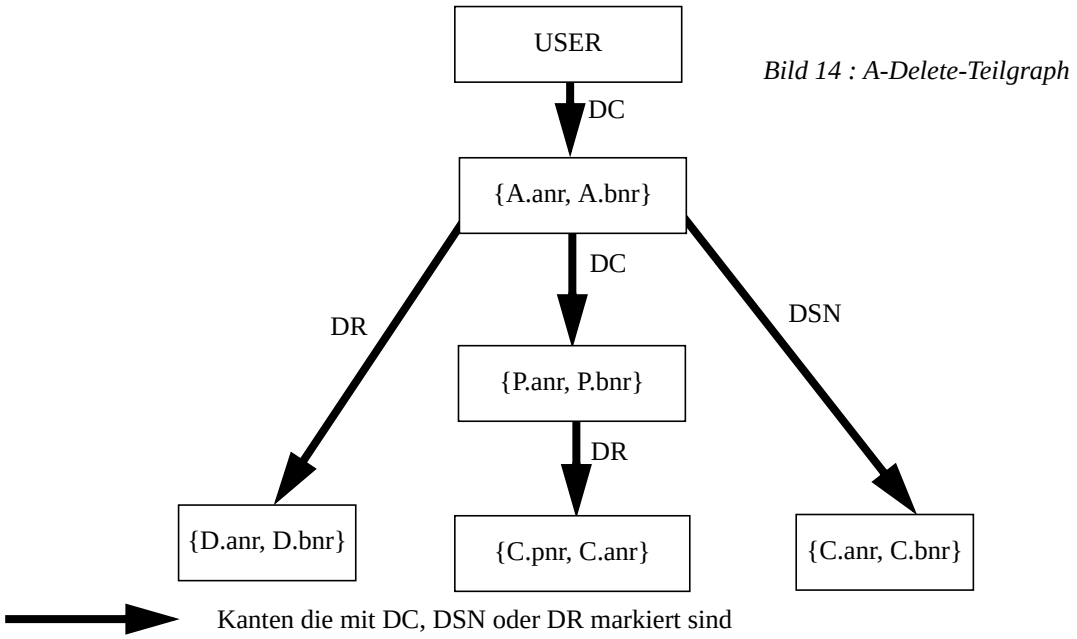
Ein Beispiel für den Einfluß von Existenzabhängigkeiten wird in folgendem Schema deutlich:

Bild 13 : Beispielschema für Existenz-Abhängigkeit



Ein mögliches Problem liegt in der Tabelle C bei einer Delete-Operation in der Tabelle A.

Bild 14 : A-Delete-Teilgraph



Dort überlappen sich die Fremdschlüssel, die C mit A und C mit P verbinden, so, daß die Bedingung von Satz 2 für den A-Delete-Teilgraphen verletzt ist. Allerdings ist das Schema insgesamt nicht mehrdeutig, denn es gilt folgender Zusammenhang:
Eine Mehrdeutigkeit kann dann auftreten, wenn in der Tabelle C der Fremdschlüssel (anr, bnr) eines Tupels auf den Null-Wert gesetzt werden soll und gleichzeitig dieses Tupel für die Auslösung des Restricted-Falles notwendig ist. Da-

für müssen die Fremdschlüssel (C.anr,C. bnr) und (C.pnr, C.anr) vollständig definiert sein. Durch die Existenz der Verbindung von C über (anr, bnr) nach D gilt jedoch, daß in D ein Tupel mit den entsprechenden Attributwerten existieren muß, denn sonst ist die referentielle Integrität verletzt. Dieses “bedingte” Tupel wird nun jedoch von einer Löschung in der Tabelle C ebenfalls betroffen und deshalb wird die notwendige Restricted-Aktion, die in C zerstört werden kann, auf jeden Fall durch das Restricted in D ausgeführt. Solche Abhängigkeiten sollen im folgenden als *Existenz-Abhängigkeiten* bezeichnet werden.

Mit solchen Existenz-Abhängigkeiten kann die Abwesenheit von Mehrdeutigkeiten nicht mehr lokal bei einer Relation, bzw. den konstituierenden Fremdschlüsseln entschieden werden, sondern das Problem ist nur entscheidbar, wenn alle referentiellen Integritätsbedingungen in einem Schema mit berücksichtigt werden. Eine weitere Steigerung der Komplexität, aber auch der Optimierungsmöglichkeiten, entsteht, wenn neben den referentiellen Integritätsbedingungen auch andere Integritätsbedingungen (beispielsweise Assertions) mit berücksichtigt werden. Um dafür dann die Abwesenheit von Mehrdeutigkeiten zu beweisen, wird im Datenbanksystem eine Komponente notwendig, die Beweise in einem Prädikatenkalkül führen kann [Bi82]. Vom Ansatz her müssen neben den Assertions auch die referentiellen Integritätsbedingungen in Prädikate übersetzt werden, und über der entstehenden Formelmenge muß dann ein Beweis geführt werden. Die Probleme sind offensichtlich, vor allem wenn man bedenkt, daß in ersten Formalisierungen nur die Endlichkeit jedes Datenbankschemas verhindert, daß ein logisches Kalkül 2. Ordnung notwendig ist.

An diesem Punkt ist noch nicht klar, welches Optimierungspotential vorliegt. Sicher ist jedoch, daß durch die explizite Formulierung von Wissen und einem geeigneten Ableitungsprozeß ein großer Performancegewinn für die Laufzeit erzielt werden kann. Ebenso offensichtlich ist, daß, wenn die Formulierung einem Ableitungsprozeß nicht zugänglich ist (z.B. weil das Wissen prozedural in Form von datenbank-externem Code vorliegt), keine Aussagen mehr möglich sind.

5. Wartung referentieller Integritätsbedingungen durch ein Trigger-Subsystem

Bei der Überlegung, wie referentielle Integritätsbedingungen in einem konkreten System realisiert werden können, kommen sogenannte Trigger (z.B. [Es76]) als Grundlage in betracht, denn die Struktur der referentiellen Integritätsbeziehungen läßt sich auf abstrakter Ebene in einen Bedingungs- und einen Aktionsteil aufgliedern. Die Bedingung beschreibt die Attributgruppen, zwischen denen die Mengeninklusionsbeziehung gilt, und der Aktionsteil enthält die spezifizierte Reaktion auf die Verletzung der Bedingung. Um nun noch unterschiedliche Reaktionen auf verschiedene Operationen wie Update, Insert oder Delete zu ermöglichen, ist es sinnvoll, die Definition noch nach diesen Operationen zu unterscheiden. Damit ergibt sich eine referentielle Integritätsbeziehung als Tripel (<Operation>, <Bedingung>, <Aktion>). Gerade für solche Art der Information (Wissen) wurde im Datenbankbereich das Konzept der Trigger entwickelt. Die Vorschläge von [Es76] wurden in den letzten Jahre durch die Forschungsanstrengungen im Bereich der aktiven Datenbanken [Da88, Cha89] erweitert. Die wesentlichen Teile von Triggern sind:

- Ein Ereignisteil: Das Ereigniskonzept bildet den zentralen Unterschied zwischen Triggern und Regeln, wie sie beispielsweise in der künstlichen Intelligenz [Ni82] verwendet werden. Es gibt dem ganzen Ansatz auch seinen Namen. In diesem Teil wird beschrieben, *wann* die weiteren Elemente des Triggers überhaupt betrachtet werden, d.h., wann diese Elemente getriggert werden. Im Fall der referentiellen Integrität kann die Operation (z.B. DELETE ABTEILUNG) als Ereignis aufgefaßt werden.
- Ein Bedingungsteil: An dieser Stelle wird beschrieben, welcher Datenbankzustand bzw. Zustandsübergang zwischen Datenbankzuständen Voraussetzung für die Ausführung des Aktionsteils ist. Damit wird der Bezug zu den verfügbaren Daten hergestellt. Im Unterschied zum Ereignisteil ist der wesentliche Bezug also nicht zu einem Zeitpunkt (Ereignis), sondern zu einem Zustand. Bei referentiellen Integritätsbedingungen würde hier die negierte Inklusionsbedingung formuliert werden (falls nicht über den gleichen Mechanismus auch die Eindeutigkeit des referenzierten Tupels überprüft werden soll).

- Ein Aktionsteil: Hier werden nun die Aktionen beschrieben, die ausgeführt werden sollen, wenn zu einem bestimmten Zeitpunkt (Ereignis) ein bestimmter Zustand (Bedingung) gegeben ist. Damit lassen sich die definierten Aktionen der referentiellen Integritätsbedingungen hier erfassen.

Es ist nun sehr leicht, die Schema-Definitionen für referentielle Integritätsbedingungen in Triggerspezifikationen umsetzen lassen. Damit sind alle Ergebnisse bezüglich Abarbeitungsstrategien und Implementierungen aus diesem Gebiet für die Betrachtung referentieller Integrität anwendbar. Allerdings ist das Problem der logischen Gleichzeitigkeit im Bereich der Trigger nur ungenügend gelöst. Es wurden zwar Modelle entwickelt, wie eine Abarbeitung mehrerer Trigger, die von einem Ereignis ausgelöst wurden, realisiert werden kann. So wird beispielsweise in [WF90, WCL91] eine stufenweise Verarbeitung mit der Selektion jeweils eines Triggers vorgeschlagen, während [Da88] eine konkurrente, allein durch die Synchronisationsmechanismen eines unterliegenden Datenbanksystem gesteuerte, Ausführung aller ausgelösten Trigger vorschlägt.
Für das Problem der logischen Gleichzeitigkeit bieten beide Ansätze keine Lösung. In [Me91] werden für diesen Bereich Konzepte vorgeschlagen, wie Erweiterungen eines gewöhnlichen Triggeransatzes aussehen können, die dieses Problem lösen. Damit bietet es sich an, ein Triggerkonzept als Implementierungsgrundlage einzusetzen, denn viele Probleme der Integrität lassen sich mit Triggern relativ einfach beschreiben, so daß es fragwürdig ist, für die Wartung der referentiellen Integrität eine selbständige Realisierung zu wählen. In jedem Fall können Trigger in den herkömmlichen Konzepten mehr oder weniger unkontrolliert eingesetzt werden, wenn es sich um ein sicheres referentielles Integritätsnetz handelt.

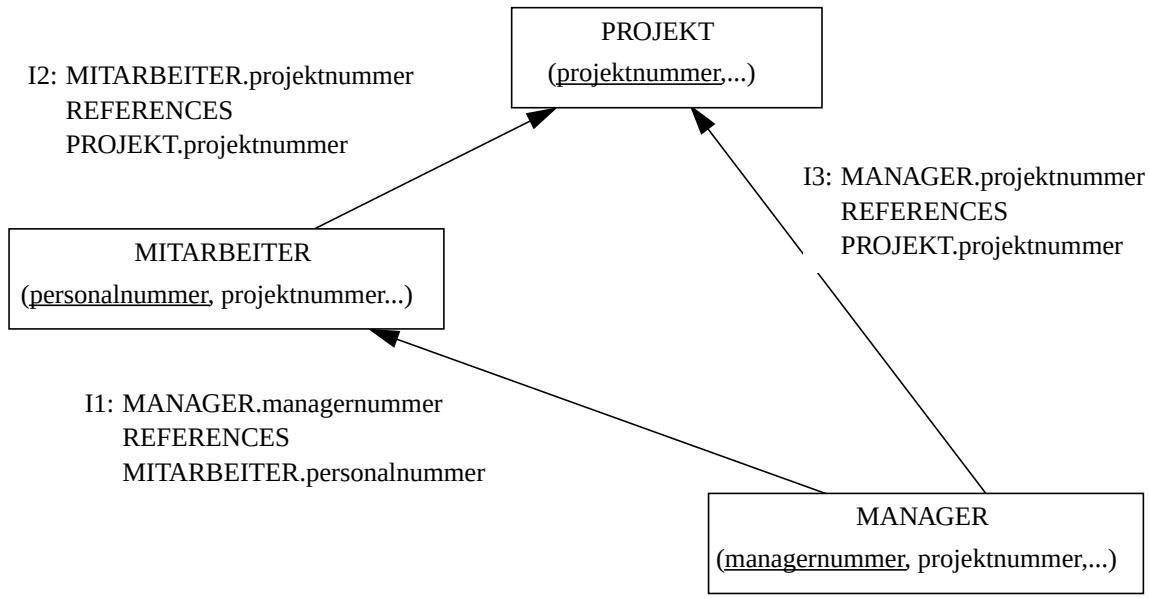
6. Vergleich und Grenzen

In diesem Abschnitt soll durch ein Beispiel gezeigt werden, wo sich das oben vorgestellte Vorgehen von Ansätzen unterscheidet, wie sie in der Literatur [Mar91] vorgeschlagen werden bzw. in konkreten Systemen [DB2] verwirklicht sind, um die Prüfung auf Mehrdeutigkeiten anhand des Datenbankschemas durchführen. Außerdem soll die Grenze zum SQL2-Standard ebenfalls an einem Beispiel dargestellt werden. Da das vorgestellte Verfahren im Augenblick, wie auch der Ansatz von Markowitz und DB2, auf die Delete-Operation beschränkt ist, soll auch nur diese Grundlage für den Vergleich sein.

6.1 Vergleich

Markowitz stellt in [Mar91] ein Beispiel vor, in dem es um die Verwaltung von Mitarbeitern, Manangern und Projekten geht. Abstrakt gesehen wird von folgendem Schema ausgegangen:

Bild 15 : Schema des Vergleichsbeispiels



Um eine übersichtliche Form für den Vergleich zu haben, wurde eine etwas andere Form für die Schemadarstellung gewählt als bisher.
Die Mächtigkeit der Konzepte entscheidet sich nun an den Optionen, die für die einzelnen Integritätsbeziehungen zulässig sind. Für jede Integritätsbeziehung gibt es drei mögliche Delete-Regeln (Update-Regeln spielen in diesem Beispiel keine Rolle, da es keine überlappenden Fremdschlüssel gibt und Update-Operationen an sich nicht betrachtet werden), so daß insgesamt siebenundzwanzig alternative Kombinationen entstehen.

Table 1: Mögliche Kombinationen der Delete-Regeln im Beispielschema

Fall	I1	I2	I3	Vergleich der Auswirkungen
1-3	R	R	*	Alle Verfahren lassen diese Kombination zu, da sie völlig unkritisch ist.
4	R	C	R	dto.
5	R	C	C	In allen Verfahren verboten
6	R	C	S	*In DB2 und in [Mar91] verboten*
7	R	S	R	Alle Verfahren lassen diese Kombination zu
8	R	S	C	dto.
9	R	S	S	dto.
10	C	R	R	dto.

Table 1: Mögliche Kombinationen der Delete-Regeln im Beispielschema

Fall	I1	I2	I3	Vergleich der Auswirkungen
11	C	R	C	dto.
12	C	R	S	dto.
13	C	C	R	In allen Verfahren verboten
14	C	C	C	Alle Verfahren lassen diese Kombination zu
15	C	C	S	In DB2 verboten
16	C	S	R	Alle Verfahren lassen diese Kombination zu
17	C	S	C	dto.
18	C	S	S	dto.
19-27	S	*	*	Nicht möglich, da MANAGER.managernummer Primärschlüssel

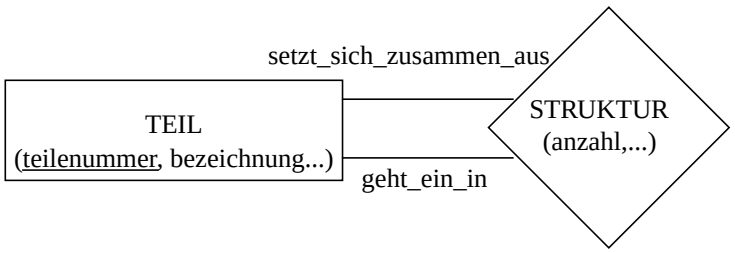
An diesem Beispiel wird der wesentliche Unterschied der Verfahren deutlich:

- Die Fall, der nur in DB2 verboten ist (15), wird dort deshalb ausgeschlossen, weil in DB2 auch überlappende Fremdschlüssel zugelassen werden und bei der Prüfung des Schemas dies nicht unterschieden wird. Außerdem gibt es nichts, was der Eigenschaft V entspricht.
- Der Fall, der auch vom Ansatz aus [Mar91] nicht erfaßt wird (6), wird in unserem Ansatz durch das feinere Betrachtungsgranulat gelöst.

6.2 Grenzen der Schemaprüfung

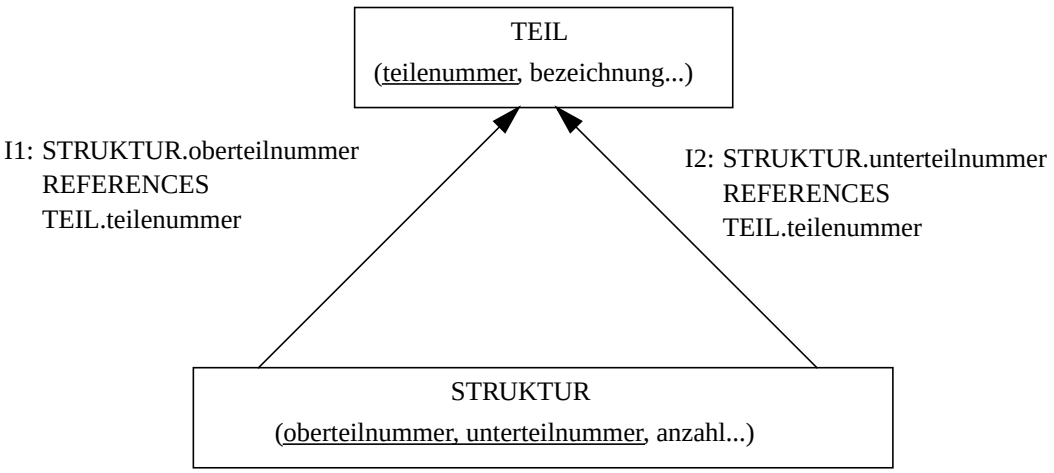
Um die Grenzen des von uns entwickelten Ansatzes, bzw. die Grenzen der referentiellen Integrität allgemein, aufzuzeigen, soll das sehr bekannte und wichtige Beispiel des Stücklistenproblems aufgegriffen werden. Dieses Problem besteht darin, daß Teile, z.B. Maschinenelemente, zum einen bezüglich der Stammdaten erfaßt werden sollen und zum anderen soll sowohl die Zusammensetzung (setzt_sich_zusammen_aus) von Teilen aus anderen Teilen als auch die Verwendung (geht_ein_in) von Teilen in anderen Teilen erfaßt werden. In einem ER-Diagramm läßt sich dies so darstellen:

Bild 16 : ER-Diagramm des Stücklistenproblems



Für die Umsetzung dieses ER-Diagramms in ein relationales Schema werden zwei Relationen benötigt (n:m-Beziehung), die über zwei Primär-Fremdschlüssel-Beziehungen miteinander verbunden sind.

Bild 17 : Schema des Stücklistenproblems



Wie bisher müssen auch hier die Eigenschaften der einzelnen Integritätsbeziehungen noch näher beschrieben werden. Als Verarbeitungssemantik (“business-rules”) wird folgendes zugrundegelegt:

- Ein Element aus TEIL ist weder direkt noch indirekt Komponente von sich selbst.
- Ein Element aus TEIL darf nur dann gelöscht werden, wenn es keine Komponente eines anderen Elementes aus TEIL ist. Gelöscht werden dürfen jedoch auch “oberste” Elemente einer Hierarchie.

Damit wird folgende Instanz des Schemas möglich:

TEIL: (01, 4-Zylinder-Motor,...)
(02, Zylinder,...)
(03, Stirn-Getriebe,...)
(04, Nockenwelle)
(05, Zahnrad,...)
(27, 2-Zylinder-Motor,...)
(32, Planeten-Getriebe,...)

STRUKTUR: (01, 02, 4,...)
(01, 32, 1,...)
(01, 04, 1,...)
(03, 05, 2,...)
(27, 02, 2,...)
(27,03, 1,...)
(27, 04, 1,...)
(32, 05, 4,...)

Die geforderte Semantik läßt sich eigentlich mit referentiellen Integritätsbedingungen ausdrücken. Dazu wird I1 mit der Delete-Regel “ON DELETE CASCADE” (Löschung eines “obersten” Elementes ist erlaubt) versehen und I2 mit der Delete-Regel “ON DELETE RESTRICTED” (Teile, die noch verwendet werden, dürfen nicht gelöscht werden). Damit würde sich nun in der Beispiel-Instanz der 2-Zylinder-Motor löschen lassen, nicht jedoch das Stirn-Getriebe. Beim Versuch, das Getriebe zu löschen, könnte allerdings zu einem konkreten Zeitpunkt während der Abarbeitung der

referentiellen Integritätsbedingungen das Tupel (03, 05, 2,...) gelöscht werden. Erst das Auffinden des Tupels (27, 03, 1,...) setzt die ganze Operation zurück.
Allerdings ist die angegebene Formalisierung der Miniwelt in *keinem* der vorgestellten Ansätze erlaubt, denn es kann prinzipiell zu Mehrdeutigkeiten kommen. Aber die Tatsache, daß ein Teil nie Komponente von sich selbst sein kann, verhindert, daß in einer konkreten Ausprägung des Schemas eine wirkliche Mehrdeutigkeit enthalten ist, solange man die Benutzeraktion auf das Löschen eines Teiles einschränkt (wird diese Bedingung aufgegeben, so kann man die Eigenschaft noch dadurch sicherstellen, daß Benutzeraktionen vorrangig behandelt werden), denn in Instanzen des Schemas entstehen nur dann Probleme, wenn ein Tupel aufgrund der gleichen Aktion gelöscht und diese Löschung verhindern soll. Solche Situationen können (mit obiger Einschränkung) nur dann auftreten, wenn ein Tupel in der Instanz möglich ist, in dem die Oberteilnummer und die Unterteilnummer identisch sind. Die Bedingung, daß solche Beziehungen *nicht* erlaubt sind, ist zwar in den business-rules enthalten, sie ist jedoch im DB-Schema nicht explizit formalisiert.
Im SQL2-Standard ist obiges Schema mit den angegebenen Regeln für die Integritätsbedingungen erlaubt, da dort eine Prüfung auf Ausprägungsebene stattfindet und es dort, wegen der business-rules, nie zu Problemen kommt. Dies ist sicherlich ein Gewinn an Flexibilität, allerdings ist dieser Ansatz aus der Sicht des Schemaentwurfes nicht befriedigend, denn eigentlich sollten im Schema *alle* Informationen der Miniwelt, soweit möglich, enthalten sein.

6.3 Grenzen der referentiellen Integrität

Kommt nun noch als weitere business-rule hinzu, wenn die Löschung eines Elementes aus TEIL andere Elemente ohne Verwendung (geht_ein_in) entstehen läßt, daß dann auch diese Elemente gelöscht werden sollen, dann stößt man an die Grenzen der Mächtigkeit der referentiellen Integritätsbeziehungen. Denn dann müßte sich die Löschung eines STRUKTUR-Elementes wieder auf die Tabelle TEIL auswirken. Dazu wäre es jedoch notwendig, daß der Primärschlüssel der STRUKTUR-Tabelle als Fremdschlüssel in der TEIL-Tabelle enthalten ist. Dies ist jedoch wegen der n:m-Beziehung nicht möglich! Um auch diese Regel verwirklichen zu können, wird im Normungsvorschlag von SQL3 [SQL3] eine Option PENDANT zur weiteren Spezifizierung von referentiellen Integritätsbedingungen eingeführt. Diese Option führt eine Art symmetrische referentielle Integrität ein und besagt, daß ein Tupel t der referenzierten Tabelle (in diesem Fall der Tabelle TEIL) gelöscht wird, wenn das letzte Tupel der referenzierenden Tabelle (hier STRUTUR) gelöscht wird, das t referenziert hat. Damit würde dann die Integritätsbeziehung I2 mit der Option PENDANT versehen und würde dann genau diese business-rule realisieren.
Es ist jedoch fragwürdig, ob der Ansatz, für wichtige Anwendungsbeispiele immer neue Optionen einzuführen und andere Tatsachen unausgesprochen, d.h. nicht im Schema formalisiert, zu lassen, nicht letztlich zu ähnlichen Entwurfsfehlern der Norm führt, wie das in der Entwicklung des CODASYL-Datenmodells [CODA71, CODA73, CODA76, CODA78] geschehen ist. Vom Standpunkt des Schemaentwurfes wäre es vielleicht hilfreicher, ein relativ strenges Konzept der referentiellen Integrität zu verwirklichen und deskriptive Integritätsbedingungen an allgemeine DB-Zustände und -Übergänge zur Verfügung zu stellen.

7. Zusammenfassung und Ausblick

Im vorliegenden Aufsatz wurde ausgehend von der Semantik, die im SQL2-Standard für referentielle Integritätsbedingungen vorgeschlagen wird, versucht, Bedingungen für relationale Schemata anzugeben, an Hand derer zum Übersetzungszeitpunkt einer Anfrage entschieden werden kann, ob für die Überwachung der referentiellen Integritätsbedingungen bestimmte Sicherungsmaßnahmen (z.B. Anlegen von temporären Relationen) notwendig sind, oder diese Bedingungen bezüglich ihrer Abarbeitung unabhängig voneinander sind. Es wurde eine, im Vergleich zu [Mar91] verfeinerte, hinreichende Bedingung formuliert und es wurden die Schwierigkeiten der Bestimmung einer notwendigen Bedingung aufgezeigt.

Im weiteren Verlauf der Untersuchungen soll der Gedanken eines allgemeinen Beweisers für Schema-Eigenschaften und eine spezielle Formulierung einer notwendigen Bedingung für referentielle Integritätsbedingungen unter dem Ausschluß aller anderen Möglichkeiten zur Formulierung von Integritätsbedingungen weiter verfolgt werden.
Außerdem soll eine konkrete Formulierung und Implementierung der Algorithmen zeigen, ob der Mehraufwand zum Übersetzungszeitpunkt tragbar ist und an welchen Stellen der allgemeine Ansatz verlassen werden muß, um beispielsweise ad-hoc-queries nicht durch einen prohibitiven Übersetzungsaufwand von der Benutzung auszuschließen.
Abschließend wurde an Beispielen die Erweiterung der Mächtigkeit im Vergleich zu Vorschlägen in der Literatur sowie die Grenzen des hier vorgestellten Ansatzes dargestellt. Zusätzlich wurden am Beispiel des Stücklistenproblems auch die Grenzen der referentiellen Integrität ganz allgemein aufgezeigt.

Dank
Ich möchte an dieser Stelle S. Deßloch, T. Härder und S. Meybrink danken, die in langen Diskussionen dazu beigetragen haben die Konzepte der Referentiellen Integrität besser zu verstehen und für die Verbesserung früherer Version diese Papiers zahlreiche Hinweise gaben.

Literaturverzeichnis

[Bi82]..... Bibel, W. Automated Theorem Proving
Textbuch, Viewg-Verlag, Braunschweig 1982

[Ch76]..... Chen, P.P.-S.: The Entity-Relationship Model - Toward a Unified View of Data,
in: ACM TODS, Vol. 1, No. 1, 1976, pp. 9-36.

[Cha89]..... Chakravarthy, S. et. al.: HIPAC: A Research Project in Active Time-Constrained Database Management. Final Technical Report.
Xerox Advanced Information Technology, Cambridge, Mass., July 1989

[Co70]..... Codd, E.F.: A Relational Model of Data for Large Shared Databases
in: CACM, Vol. 13, No. 6, 06.1970, pp. 337-387

[Co84]..... Codd, E.F.: A Relational Model - Version 2

[CODA71] ..... CODASYL DATABASE TASK GROUP (DBTG) Report, April 1971

[CODA73] ..... CODASYL DDL Journal of Development, June 73 Report

[CODA76] ..... Manola, F.: The CODASYL Data Description Language: Status and Activities 1976
in: Jardine, D.A. (ed.): The ANSI/SPARC DBMS Model, North Holland Publ. Co., Amsterdam 1976

[CODA78] ..... Report of the COASYL Data Description Language Committee
in: Information Systems, Vol.3, No. 4, 1978, pp. 247-320

[Da81]..... Date,C.J.: Referential Integrity,
in: Proc. VLDB'81, IEEE, CH1701-2/81/0000/0002$00.75, 03.1981, pp. 2-12.

[Da86]..... Date,C.J.: Relational Databases: Selected Writings
Addison-Wesley Publishing Company, 1986 (ISBN: 0-201-14196-5)

[Da90]..... Date,C.J.: Relational Databases: Selected Writings 1985-1989
Addison-Wesley Publishing Company, 1990 (ISBN: 0-201-50881-8)

[Da88]..... Dayal,U.: Active Database Management Systems
in: 3rd Int. Conf. on Data and Knowledge Bases: Improving Usability and Responsiveness, Jerusalem, Israel, June 28-30, (Hrsg.: Beeri,C.; Schmidt,J.W.; Dayal,U.) Morgan Kaufmann Publishers, Inc., San Matheo, California 94403, 06.1988, pp. 150-169

[EC75] ..... Eswaran, K.P.; Chamberlin,D.D.: Functional Specification of a Subsystem for Data Base Integrity
in: Proc. 1st VLDB, Framingham, 09.1975, pp. 48-68

[Es76] ..... Eswaran, K.P.: Specifications, Implementations and Interactions of a Trigger Subsystem in an Integrated Database System, IBM Research Division, San Jose, Research Report RJ 1820 (26414), 11.08.1976

[HM75]..... Hammer,M.M.; McLeod,D.J.: Semantic Integrity in a Relational Data Base System
in: Proc. VLDB '75, Framingham, Ma., 1975, pp. 25-47

[IBM] ..... IBM DB2 Version 2, SQL Reference (SC 26-4380-1)

[Mar91] ..... Safe Referential Integrity Structures in Relational Databases
in: Proceedings of the 17th VLDB Conference, Barcelona, 1991

[Me92] ..... Meybrink, S.:Entwurf eines Architekturkonzeptes für ein Integritäts-Subsystem
unveröffentlichter Interner Bericht

[Mi88] ..... Mitschang, B.: Ein Molekül-Atom-Datenmodell für Non-Standard-Anwendungen - Anwendungsanalyse, Datenmodellentwurf und Implementierungskonzepte, Springer-Verlag Berlin Heidelberg, Informatik Fachberichte Nr. 185, Dissertation, 1985.

[MR91] ..... Meybrink, S.; Reinert, J.: Referentielle Integrität in relationalen Datenbanksystemen - Bestandsaufnahme und Realiesierungsstudie
Zwischenbericht zur Forschungskooperation der Universität Kaiserslautern und SNI AG, München

[Ni82] ..... Nilsson, N.: Principles of Artificial Intelligence
Textbuch, Springer Verlag 1982

[SQL] ..... ISO/IEC 9075:1989, Database Language SQL

[SQL2] ..... ISO/IEC JTC1/SC21 Information Retrieval, Transfer and Management for OSI: International Standard ISO/IEC 9075:1992, Revised Text of CD 9075.2, Information Technology - Database Languages - SQL2, for DIS registration and letter ballot, 10.04.1991, 522 pp

[SQL3] ..... X3H2-91-183 DBL-KAW-003 ISO/IEC JTC1/SC21/WG3 N1223: ISO/ANSI Working Draft - Database Language SQL3, 07.1991, 772 pp

[WCL91] ..... Widom, J.; Cochrane, R. J.; Lindsay, B.G.: Implementing Set-Oriented Production Rules as an Extension to Starburst
in: Proceedings of the 17th VLDB Conference, Barcelona, 1991, pp. 275-285

[WF90] ..... Widom,J.; Finkelstein,S.J.: Set-Oriented Production Rules in Relational Database Systems
in: Proc. ACM SIGMOD, ACM SIGMOD Record, Vol. 19, No. 2, (Hrsg.: Garcia-Molina,H.; Jagadish,H.V.) , 06.1990, pp. 259-270