Integritätsbedingungen für komplexe Objekte in objektrelationalen Datenbanksystemen

Jens Lufter
Friedrich-Schiller-Universität Jena
lufter@informatik.uni-jena.de

Abstract: Ein Aspekt objektrelationaler Datenbanksysteme ist die Möglichkeit zur Definition von strukturierten und von Kollektionsdatentypen und die sich daraus ergebende Verwendbarkeit komplex geschachtelter Objekte. SQL-Norm und wichtige DBMS-Produkte haben jedoch noch deutliche Defizite in der Sicherung der Integrität komplex aufgebauter Attribute von Datenbanktabellen. Der vorliegende Beitrag diskutiert Anforderungen an eine adäquate Integritätssicherung, beschreibt den aktuellen Stand der Dinge in SQL-Norm und wichtigen Produkten und untersucht Möglichkeiten zur Ergänzung von SQL um Sprachmittel für entsprechende Integritätsbedingungen.

1 Einführung

Viele Anwendungssysteme halten ihre Daten heute in relationalen Datenbanksystemen, der Zugriff darauf erfolgt über die normierte Datenbanksprache SQL [ISO99] bzw. produktspezifische Dialekte derselben. Der Einfluss der Objektorientierung hat mittlerweile zur evolutionären Weiterentwicklung von relationalen zu objektrelationalen Datenbanksystemen geführt [DD98, Luf99], deren Grundlage erweiterbare Datentypsysteme sind.

Ein wichtiger Aspekt ist dabei die Schachtelbarkeit von Datentypen, insbesondere von strukturierten und von Kollektionsdatentypen. Damit können potentiell sehr tief geschachtelte Konstrukte entstehen. Noch nicht gut gelöst sind in diesem Zusammenhang u. a. Probleme der Integritätssicherung. In relationalen Datenbanksystemen ist diese Frage auf der Ebene von Datenbanktabellen und deren Attributen angesiedelt, spezialisierte Integritätsbedingungen (`NOT NULL`, Schlüssel, Fremdschlüssel) referenzieren diese Attribute, können jedoch nicht in ihren möglicherweise komplexen Aufbau „hineinschauen".

SQL-Norm und wichtige Produkte bieten heute die Möglichkeit, strukturierte und Kollektionsdatentypen zu definieren und orthogonal zueinander zu verwenden. Die Definition von Kollektionsdatentypen ist jedoch meist unzureichend und sehr heterogen gelöst und bedarf deutlicher Verbesserungen [Luf03b]. Wie die folgenden Ausführungen zeigen, ist es außerdem erforderlich, einen hinreichenden Umfang von Sprachmitteln zur Integritätssicherung anzubieten, die nicht nur auf Attributebene wirken, sondern tiefer in die Struktur komplex geschachtelter Objekte eindringen können. Das vorgestellte Thema ist Teil des Projekts *Object-Relational Database Features and Extensions: Data Model and Physical Aspects*, das durch einen *IBM Shared University Research Grant* unterstützt wird.

2 Integritätsbedingungen für komplexe Objekte

2.1 Datentypen in objektrelationalen Datenbanksystemen

Objektrelationale Datenbanksysteme erlauben die Definition von nutzerdefinierten Datentypen und Routinen und die Konstruktion von Datentypen mit Hilfe von Typgeneratoren. Zu den nutzerdefinierten Typen zählen dabei *Distinct types* zur strengen Typisierung anderer Datentypen und die strukturierten nutzerdefinierten Datentypen, die objektorientierte Eigenschaften wie Methoden, Objektidentität und -referenzen sowie Vererbung und Polymorphie aufweisen. Letztere können neben der üblichen Verwendung von Datentypen, z. B. zur Attributdefinition von Datenbanktabellen, auch zur Spezifikation typisierter Tabellen genutzt werden, deren Tupel Objekte dieses Typs inklusive Objektidentität und Referenzierbarkeit darstellen.

Zu den generierten (konstruierten) Datentypen zählen Kollektions-, Struktur- und Referenztypen (`ARRAY, SET, ROW, REF` usw.). Typgeneratoren ermöglichen die Definition von Datentypen, die ein gemeinsames allgemeines Verhalten, also vergleichbare Operationen, aufweisen. Parameter eines Typgenerators sind Datentypen, Feldnamen oder Längenangaben, das spezifische Verhalten des entstehenden Typs wird diesen Parametern automatisch angepasst. Generierte Datentypen sind im Gegensatz zu nutzerdefinierten Datentypen nicht Elemente des Datenbankschemas.

Nutzerdefinierte und generierte Typen können orthogonal zur Definition weiterer Datentypen und zur Attributdefinition von Datenbanktabellen genutzt werden. Dadurch können potentiell tief geschachtelte Typstrukturen entstehen, entsprechende Typinstanzen bezeichnen wir hier auch als komplexe Objekte. Um im Rahmen von Datenbankzugriffen auf innere Strukturen komplexer Objekte zugreifen zu können, müssen adäquate Operationen bereitstehen. So kann man Attribute von Strukturtypen mittels Punkt-Notation ansprechen, für Kollektionen gibt es Operationen zum Elementzugriff oder das Umwandeln in eine Tabelle (`UNNEST`), auf die dann relationale Operationen anwendbar sind.

2.2 Komplexe Attribute in Datenbanktabellen

Zur Illustration der Probleme bei der Integritätssicherung für komplexe Attribute von Datenbanktabellen soll die folgende Tabellendefinition dienen. Das Beispiel verwendet vordefinierte Datentypen wie `INTEGER`, generierte Strukturtypen (`ROW`), eine Reihe generierter Kollektionstypen (`ARRAY, SET, LIST`) und Referenzdatentypen zum Verweis auf Objekte in typisierten Tabellen (`REF`). Der Übersicht wegen abstrahieren wir von nutzerdefinierten Datentypen; im Hinblick auf die Integritätssicherung sind *Distinct types* zum jeweils zugrundeliegenden Datentyp äquivalent, nutzerdefinierte strukturierte Typen zu `ROW`. Die Kollektionsunterstützung in SQL-Norm und Produkten ist derzeit noch nicht befriedigend und sehr heterogen umgesetzt. Die Ausführungen beziehen sich daher auf ein von uns entwickeltes erweitertes Kollektionskonzept [Luf03b], dessen genauere Beschreibung hier jedoch zu weit führen würde.

```
CREATE TABLE Angestellter (
  ID         IdType PRIMARY KEY,
  Name       ROW ( Nachname CHAR(20),
                   Vornamen LIST ( CHAR(20) ) ),
  Adressen   ROW ( PLZ CHAR(5), Ort CHAR(20) ) ARRAY[3],
  Telefone   SET ( CHAR(20) ),
  MailListen SET ( LIST ( IdType ) ),
  Kinder     SET ( REF ( KindTyp ) ) )
```

Angestellte haben somit zunächst eine identifizierende Nummer und einen aus Nachname und Vornamen zusammengesetzten Namen. Ein Angestellter kann ggf. mehrere Vornamen haben, deren Ordnung wichtig ist, eine Liste bietet sich an. Weiterhin seien bis zu drei Adressen in einem Feld von zweiattributigen Strukturen verzeichnet. Außerdem möge ein Angestellter eine Menge von Telefonnummern haben. Geschachtelte Kollektionen illustriert die Menge der Mail-Listen eines Angestellten, jede Mail-Liste ist wiederum eine geordnete Kollektion von Angestellten-IDs. Die Kinder eines Angestellten ergeben sich aus einer Menge von Referenzen auf Objekte einer typisierten Tabelle. Bis auf den Primärschlüssel enthält die Tabelle noch keine Integritätsbedingungen.

2.3 Klassifikation von Integritätsbedingungen

In relationalen Datenbanksystemen gibt es traditionell drei wesentliche Klassen von Integritätsbedingungen: Schlüssel, Fremdschlüssel und allgemeine, über Anfragen definierte `CHECK`-Bedingungen. Hinzu kommt auf Grund der dreiwertigen Logik von SQL die Kurzschreibweise für `NOT NULL`-Bedingungen. Zur Unterscheidung von allgemeinen Bedingungen wollen wir Konstrukte wie Schlüssel, Fremdschlüssel oder `NOT NULL` im Folgenden auch als spezialisierte Integritätsbedingungen bezeichnen.

Schlüssel- und Fremdschlüsselbedingungen können auf allgemeine Bedingungen zurückgeführt werden, in der Tat macht das die SQL-Norm bei der Beschreibung der Semantik dieser Bedingungen genau so [ISO99]. Dennoch gibt es für diese beiden Fälle eine eigene Notation, die sich zum einen natürlich aus dem relationalen Datenmodell selbst ergibt. Zum zweiten ist die Notation deutlich bequemer zu handhaben und bietet einfachere Möglichkeiten für eine effiziente Implementierung. Schließlich gibt es noch semantische Zusätze. So kann eine Tabelle nur einen Primärschlüssel haben und die daran beteiligten Attribute können nicht `NULL` werden, ein Schlüssel ist zudem als Ziel von Fremdschlüsseln verwendbar. Wichtig sind die referentiellen Aktionen für Fremdschlüssel (`ON DELETE CASCADE` usw.), die aktives Verhalten in die Integritätssicherung einführen.

Objektrelationale Datenbanken ergänzen die genannten Sprachmittel um ein weiteres zum Absichern von Objektreferenzen. In unserem Beispiel ist `REF ( KindTyp )` ein Datentyp, dessen Werte auf Tupel von mit `KindTyp` typisierten Tabellen zeigen können, potentiell kann es mehrere solcher Tabellen geben. In Anlehnung an Fremdschlüssel kann man das Referenzziel auf eine (oder auch mehrere) Tabellen beschränken (`SCOPE`) und zudem, ebenfalls analog zu Fremdschlüsseln, referentielle Aktionen ergänzen.

Auf Grund der traditionell armen Typsysteme bisheriger relationaler Datenbanksysteme setzen Schlüssel und Fremdschlüssel, aber auch NOT NULL-Bedingungen, bislang auf der Attributebene von Tabellen an, können also nicht in komplexere Strukturen eindringen. Allgemeine Bedingungen können dagegen die ganze Anfragemächtigkeit von SQL ausnutzen. Bevor wir die Notwendigkeit spezialisierter Konstrukte für die Integritätssicherung unter Einbeziehung komplexer Objekte diskutieren, daher zwei Beispiele für allgemeine Bedingungen, die an sich einfache Aufgaben simulieren. Die erste der Regeln zeigt, wie Fremdschlüsselbedingungen simuliert werden können: Die Einträge der Mail-Listen gehören zu vorhandenen Angestellten. Die zweite Regel stellt sicher, dass die Menge der Telefone kein NULL-Element enthalten darf.

```
ALTER TABLE Angestellter
  ADD CONSTRAINT MailingIDs CHECK ( NOT EXISTS
    ( SELECT *
      FROM   Angestellter a,
             UNNEST ( a.MailListen ) AS ml ( Liste ),
             UNNEST ( ml.Liste ) AS l ( MailID )
      WHERE  l.MailID NOT IN
               ( SELECT ID FROM Angestellter ) )

ALTER TABLE Angestellter
  ADD CONSTRAINT TelefonNotNull CHECK ( NOT EXISTS
    ( SELECT *
      FROM   Angestellter a,
             UNNEST ( a.Telefone ) AS t ( Telefon ),
      WHERE  t.Telefon IS NULL )
```

UNNEST ist dabei immer die SQL-Operation, die eine Kollektion in eine relationale Tabelle verwandelt, der Rest der Beispiele sollte selbsterklärend sein. Schon der Formulierungsaufwand zeigt, dass für wichtige Regelarten spezialisierte Konstrukte eingeführt werden sollten. Zudem kann die Fremdschlüsselsimulation des ersten Beispiels nicht um referentielle Aktionen ergänzt werden.

Für komplexe Objekte können neue Arten von Integritätsbedingungen erforderlich sein [The96], wesentlich sind im objektrelationalen Fall aber vor allem Erweiterungen für Schlüssel-, Fremdschlüssel- und NOT NULL-Bedingungen, so dass sie in geschachtelte Datentypen eindringen können. Wichtige Beispiele für Integritätsbedingungen unter Einbeziehung geschachtelte Strukturen sind:

- Elemente einer Kollektion dürfen nicht NULL werden

  Es ist natürlich ein Unterschied, ob ein kollektionswertiger Ausdruck NULL werden kann, oder ob einzelne Elemente einer Kollektion NULL sind.

- Felder einer Struktur dürfen nicht NULL werden

  Mit ROW (...) IS NOT NULL kann man in SQL zwar schon ganze Strukturen ansprechen, dieses Konstrukt wird aber so umgesetzt, dass dann alle Felder der Struktur nicht NULL werden dürfen. Sinnvoll wäre auch ein selektives Verhalten.

- Schlüsselbedingungen für Datenbanktabellen

  Schlüsselbedingungen können eine vielfältige Gestalt haben. So kann der Schlüssel einer Datenbanktabelle z. B. auch Felder einer Struktur enthalten, etwa den Nachnamen in unserem Beispiel, oder auch einzelne Elemente einer Kollektion wie den ersten Vornamen aus der Vornamensliste eines Angestellten.

- Fremdschlüsselbedingungen

  Eine einfache Fremdschlüsselbedingung kann sich z. B. auf ein mengenwertiges Tabellenattribut beziehen und fordern, dass die Elemente der Menge sich auf den Schlüssel einer anderen Tabelle beziehen: `...SET ( INTEGER REFERENCES AndereTabelle )...` Ähnliches gilt für Felder von Strukturen innerhalb eines komplexen Attributs. Fremdschlüssel können referentielle Aktionen umfassen.

- Gesicherte Objektreferenzen

  Das Ziel von Objektreferenzen soll auf ausgewählte typisierte Tabellen einschränkbar sein, analog zu Fremdschlüsseln sollen referentielle Aktionen erlaubt sein.

- Duplikatverbot und andere Eindeutigkeitsbedingungen

  Für ungeordnete Kollektionen ohne Duplikate gibt es Mengentypen, Duplikatfreiheit kann aber auch für geordnete Kollektionen wichtig sein. Beispiele für Eindeutigkeitsbedingungen in (ggf. tiefer geschachtelten) Kollektionen sind etwa Schlüssel für Attributkombinationen geschachtelter Tabellen.

3 SQL-Norm und Produkte

Die objektrelationalen Möglichkeiten zur Nutzung komplexer Objekte in SQL-Norm und wichtigen Produkten unterscheiden sich bislang noch erheblich. Neben einer vernünftigen Unterstützung von Kollektionen mangelt es vor allem noch an der Integritätssicherung für geschachtelte Strukturen.

An für unsere Untersuchungen relevanten objektrelationalen Sprachmitteln erlaubt SQL in der aktuellen Version von 1999 [ISO99] die Nutzung generierter und nutzerdefinierter Strukturtypen (`ROW`, *Structured type*) und von generierten, allerdings nicht schachtelbaren, Feld-Typen (`ARRAY`). Außerdem gibt es Objektreferenzen zum Referenzieren von Tupeln typisierter Tabellen (`REF`). Die Folgenorm SQL:200x [ISO02] wird Multimengen ergänzen und Kollektionen schachtelbar machen.

Mit SQL:1999 kann man in Schlüsseln, Fremdschlüsseln und `NOT NULL`-Bedingungen nur ganze Attribute einer Datenbanktabelle referenzieren, innere Teile komplex strukturierter Attribute sind nicht ansprechbar. Man kann sie allenfalls so wie in den Beispielen aus Abschnitt 2.3 mit `CHECK`-Bedingungen simulieren.

Objektreferenzen lassen sich auf eine Zieltabelle beschränken, außerdem sind referentielle Aktionen angebbar. Dies gilt nicht nur für Referenzen auf Attributebene, sondern auch für eingeschachtelte Referenzen in Feldern von Strukturtypen (`ROW`, nutzerdefinierte struktu-

rierte Typen). Als Elementtyp von Kollektionen lässt sich Referenztypen zwar die Zieltabelle mitgeben, es sind aber keine referentiellen Aktionen angebbar:

```
CREATE TABLE myTable (
  att1 REF ( type1 ) SCOPE table1
         REFERENCES ARE CHECKED ON DELETE CASCADE,
  att2 LIST ( ROW ( field1 REF ( type2 ) SCOPE table2
                      REFERENCES ARE CHECKED
                        ON DELETE NO ACTION ) ),
  att3 SET ( REF ( type3 ) SCOPE table3 ) )
```

Die Semantik-Beschreibung der Norm wandelt dabei Referenz-Bedingungen auf oberster Ebene (`att1`) in Fremdschlüssel um, Bedingungen eingeschachtelter Referenzen in `CHECK`-Bedingungen. Das impliziert auch, dass im zweiten Fall nur `NO ACTION` angebbar ist, da referentielle Aktionen nur für Fremdschlüssel definiert sind.

Das nicht orthogonale Verhalten ergibt sich aus einem Design-Fehler der Norm [LFP02]. Insgesamt ist festzustellen, dass SQL bislang keine adäquaten Konzepte zur Integritätssicherung komplexer Objekte umfasst, das wird sich nach dem derzeitigen Stand der Dinge auch mit der Folgenorm nicht ändern [ISO02].

Wichtige aktuelle Produkte wie Oracle 9i, Informix Dynamic Server 9.3 oder DB2 UDB 7.3 erlauben in sehr unterschiedlichem Umfang die Nutzung von Struktur- und Kollektionsdatentypen zur Definition komplexer Datenstrukturen. Möglichkeiten zur Definition adäquater Integritätsbedingungen für komplexe Objekte gibt es so gut wie nicht. Aus Platzgründen kann hier kein entsprechender Überblick gegeben werden, der interessierte Leser sei auf [Luf03a] verwiesen.

4 Wege zur Ergänzung der SQL-Norm

Im folgenden stellen wir einen Ansatz zur Integration spezialisierter Integritätsbedingungen für komplexe Objekte in die SQL-Norm vor. Dabei liegt der Schwerpunkt auf syntaktischen Überlegungen, semantische Fragen und Details zur Einbindung in das Normdokument können hier nicht näher erläutert werden.

4.1 Zugriff auf die innere Struktur geschachtelter Datentypen

Für die gewünschten Integritätsbedingungen muss man die innere Struktur geschachtelter Datentypen ansprechen können, das geschieht durch eine Erweiterung der Punkt-Notation, mit der man in SQL auf Felder von Strukturtypen zugreifen kann. Im Gegensatz zu Feldern von Strukturtypen sind die Elementtypen von Kollektionsdatentypen unbenannt. Um sie ansprechen zu können, nutzen wir das Schlüsselwort `ELEMENT`, die Punkt-Notation erlaubt dann die Angabe von Pfaden zum Eindringen in eine komplexe Struktur.

Um also den Elementtyp der Vornamensliste unseres Angestellten-Beispiels aus Abschnitt 2.2 in einer Integritätsbedingung ansprechen zu können, muss man über das Vornamensfeld der Namensstruktur gehen, analog kann man auf Strukturen in Kollektionstypen (Ort einer Adresse) oder geschachtelte Kollektionen (ID in einer Mail-Liste) zugreifen:

```
Name.Vornamen.ELEMENT
Adressen.ELEMENT.Ort
MailListen.ELEMENT.ELEMENT
```

Abgedeckt ist natürlich auch das Ansprechen ganzer Kollektionen oder Strukturen, etwa der Vornamensliste oder der Adressstruktur:

```
Name.Vornamen
Adressen.ELEMENT
```

Semantisch muss man sich die Nutzung von `ELEMENT` wie ein implizites `UNNEST` vorstellen. Grundlage von Integritätsbedingungen sind ja letztlich Anfragen, auch die SQL-Norm spezifiziert die Semantik spezialisierter Integritätsbedingungen durch deren Umwandlung in allgemeine, anfragebasierte `CHECK`-Bedingungen. Wenn man also verlangen will, dass die Vornamen nicht `NULL` werden dürfen (`Name.Vornamen.ELEMENT NOT NULL`), führt das zu einer äquivalenten allgemeinen Bedingung:

```
CHECK ( NOT EXISTS
  ( SELECT *
    FROM   Angestellter a,
           UNNEST ( a.Name.Vornamen ) AS v ( Vorname ),
    WHERE  v.Vorname IS NULL ) )
```

Neben der neuen `ELEMENT`-Syntax, die eine Kollektion sozusagen „flachklopft“ und alle ihre Elemente (einzeln) anspricht, erlaubt SQL:1999 bereits den Zugriff auf einzelne Elemente von Felddatentypen innerhalb von Pfadausdrücken, unsere erweiterte Kollektionsunterstützung [Luf03b] erlaubt das auch für Listen. Damit ist es möglich, Integritätsbedingungen an spezifische Elemente einer Liste zu knüpfen statt an alle, etwa an den ersten Vornamen eines Angestellten oder den Ort seiner zweiten Adresse:

```
Name.Vornamen[1]
Adressen[2].Ort
```

Man kann somit mit einer einfachen und klaren Syntax Teile komplexer Datenstrukturen ansprechen, die Änderungen in den Semantik-Teilen der Norm lassen sich aber nichtsdestotrotz in Grenzen halten.

4.2 Erzeugen und Löschen von Integritätsbedingungen

Die im vorigen Abschnitt eingeführte Syntax macht es auf einfache Weise möglich, Bedingungen für komplexe Strukturen auf Attribut- und Tabellenebene zu definieren bzw.

bei Tabellenänderungen zu löschen: Auf Attributebene würde man z. B. bei der Definition des Namensattributs mit `Vornamen.ELEMENT` den Elementtyp der Vornamensliste referenzieren, auf Tabellenebene schriebe man `Name.Vornamen.ELEMENT`.

Ein Aspekt soll hier noch angesprochen werden, nämlich das Aufheben der Vermischung von intensionalen und extensionalen Fragen, wie sie in Abschnitt 3 kurz diskutiert wurden: Für Objektreferenzen werden in der SQL-Norm Zieltabelle und dazugehörige Integritätsbedingungen teilweise intensional, also auf Datentypebene definiert. Das führt neben Problemen mit der Orthogonalität auch zu Inkonsistenzen bei Änderungen: Man kann mit `ALTER TABLE` schlecht Datentypen verändern [LFP02]. Solche Bedingungen werden mit unserem Ansatz ebenfalls auf die Attribut- bzw. Tabellenebene verlagert.

4.3 Spezifizierbare Integritätsbedingungen

Das von uns entwickelte Konzept erlaubt derzeit die Einbeziehung von komplex strukturierten Attributen in vier Arten spezialisierter Integritätsbedingungen: `NOT NULL`-Bedingungen, Schlüssel, Fremdschlüssel und gesicherte Objektreferenzen. Das Mischen von einfach und komplex strukturierten Attributen ist dabei (wo sinnvoll) möglich.

Die genaue Syntaxspezifikation und Fragen der exakten Umsetzung in der Norm würden den Umfang dieses Beitrags sprengen und sind teilweise auch noch Gegenstand laufender Arbeiten. Statt dessen sollen Anwendung und Mächtigkeit anhand des aus Abschnitt 2.2 bekannten Angestellten-Beispiels erläutert werden:

```
CREATE TABLE Angestellter (
  ID          IdType PRIMARY KEY,
  Name        ROW ( Nachname CHAR(20),
                    Vornamen LIST ( CHAR(20) ) )
                CONSTRAINT c1 NOT NULL
                CONSTRAINT c2 Vornamen.ELEMENT NOT NULL,
  Adressen    ROW ( PLZ CHAR(5), Ort CHAR(20) ) ARRAY[3]
                CONSTRAINT c3 ELEMENT.Ort NOT NULL,
  Telefone    SET ( CHAR(20) ),
  MailListen SET ( LIST ( IdType ) )
                CONSTRAINT c4 ELEMENT NOT NULL
                CONSTRAINT c5
                  ELEMENT.ELEMENT REFERENCES Angestellter,
  Kinder      SET ( REF ( KindTyp ) )
                CONSTRAINT c6
                  ELEMENT SCOPE KindTabelle
                          REFERENCES ARE CHECKED
                            ON DELETE CASCADE,
  CONSTRAINT c7
    UNIQUE ( Name.Nachname, Name.Vornamen[1],
             Adressen[1] ) )
```

Es sei noch einmal darauf hingewiesen, dass das Beispiel neben den neuen Integritätsbedingungen auch Bezug auf eine erweiterte Kollektionsunterstützung nimmt [Luf03b]. Die Praxisrelevanz der spezifizierten Integritätsbedingungen sei Nebensache.

Zunächst zu den `NOT NULL`-Bedingungen. Bedingung c1 ist eine herkömmliche SQL-Bedingung, sie gilt für das ganze Attribut. Da dieses strukturiert ist, impliziert die SQL-Semantik, dass weder der Nachname noch die Vornamensliste `NULL` sein dürfen. Was noch fehlt, ist eine Bedingung, dass auch die einzelnen Vornamen dieser Liste bekannt sein müssen (c2). Bedingung c3 verlangt, dass die Orte von Adressen spezifiziert sein müssen. Im Gegensatz dazu können sowohl einzelne Postleitzahlen als auch einzelne Adressen und ebenso das ganze Adressen-Attribut `NULL` werden. Schließlich besagt c4, dass die Menge der Mail-Listen keine undefinierten Elemente enthalten darf.

Einen einfachen Fremdschlüssel spezifiziert Bedingung c5: Die Elemente einer Mail-Liste referenzieren wiederum Angestellte. Bedingung c6 sichert schließlich Objektreferenzen ab. Im Vergleich zu SQL:1999 gibt es hier drei Neuheiten: Die Norm kann zum einen direkt in Kollektionen enthaltene Referenzen bislang gar nicht absichern, das ist jetzt möglich. Zum zweiten sind für eingeschachtelte Referenzen nun auch beliebige referentielle Aktionen spezifizierbar. Und zum dritten sind die Angabe der Zieltabelle und die Bedingung jetzt Teil der Tabellendefinition, nicht mehr der Typdefinition.

Bedingung c7 ist ein Beispiel für einen zusätzlichen Schlüssel, der zudem die Verwendung von Element-Referenzen demonstriert, wie sie SQL:1999 bereits eingeführt hat. Die Bedingung besagt, dass eine Kombination aus Nachname, erstem Vornamen und erster Adresse eines Angestellten eindeutig sein soll.

4.4 Integration in die Norm

Unsere Arbeiten zur genaueren Spezifikation der erforderlichen Normerweiterungen orientieren sich bislang an SQL:1999, werden aber anhand der jeweils aktuellen Arbeitsversionen zur Folgenorm SQL:200x verifiziert. Neben der vergleichsweise einfachen syntaktischen Umsetzung liegt dabei der Schwerpunkt auf der Einbindung der Semantik in die bestehende Infrastruktur für Integritätsbedingungen in der Norm.

Die hier vorgestellten Konzepte sollen zudem anhand einer derzeit prototypisch entwickelten Transformationsschicht validiert werden, die erweiterte objektrelationale Sprachmittel auf reale Datenbanksysteme abbildet [KLS02, Her03].

5 Zusammenfassung und Ausblick

Objektrelationale Datenbanksysteme bieten heute die Möglichkeit zur Definition komplex geschachtelter Objekte mit Hilfe von strukturierten und von Kollektionsdatentypen, wenn auch auf recht heterogene Art und Weise. Potentiell tief geschachtelte Datenstrukturen erfordern aber auch neue Möglichkeiten der Integritätssicherung, hier sind noch deutliche

Verbesserungen in SQL-Norm und Produkten nötig. Der vorliegende Beitrag hat versucht, auf Basis von Anforderungen an solche Integritätsbedingungen und den gegenwärtigen Stand der Technik Wege zur Ergänzung der aktuellen SQL-Norm vorzustellen, die diesen Erfordernissen genügen.

Ende 2003 wird voraussichtlich die nächste Version der SQL-Norm verabschiedet. Der aktuelle Arbeitsstand (*Final Committee Draft*, [ISO02]) macht eine Aufnahme der Vorschläge bereits in diese Version der Norm unwahrscheinlich, das ändert aber nichts an deren Validität und Notwendigkeit. Neben entsprechenden Zuarbeiten für das Normungsgremium [FLP02, LFP02] sowie einer detaillierteren Spezifikation und ggf. Ergänzung der vorgestellten Norm-Erweiterungen ist es das Ziel weiterer Arbeiten, die vorgestellten Sprachmittel in Verbindung mit einer erweiterten Unterstützung für Kollektionen prototypisch umzusetzen und zu validieren.

Literatur

[DD98] C. J. Date and H. Darwen. *Foundation for Object/Relational Databases: The Third Manifesto*. Addison-Wesley, Reading, MA, 1998.

[FLP02] T. Fanghänel, J. Lufter, and P. Pistor. ISO/IEC JTC 1/SC32/WG3 DRS-091: Introducing Some Indicators Missing from Definition Schema, August 2002.

[Her03] A. Hermann. Prototypische Umsetzung erweiterter objektrelationaler Datenbankkonzepte durch Abbildung auf reale DBMS. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2003. In Vorbereitung.

[ISO99] ISO/IEC 9075:1999. *Information Technology – Database Languages – SQL*, 1999.

[ISO02] ISO/IEC Final Committee Draft 9075-2:200x. *Information Technology – Database Languages – SQL – Part 2: Foundation (SQL/Foundation)*, January 2002.

[KLS02] C. Kauhaus, J. Lufter, and S. Skatulla. Eine Transformationsschicht zur Realisierung objektrelationaler Datenbankkonzepte mit erweiterter Kollektionsunterstützung. *Datenbank-Spektrum*, 2(4):49–58, November 2002.

[LFP02] J. Lufter, T. Fanghänel, and P. Pistor. ISO/IEC JTC 1/SC32/WG3 DRS-089: Problems related to <scope clause> and <reference scope check>, September 2002.

[Luf99] J. Lufter. Objektrelationale Datenbanksysteme (Aktuelles Schlagwort). *Informatik Spektrum*, 22(4):288–290, August 1999.

[Luf03a] J. Lufter. Integritätsbedingungen für komplexe Objekte in objektrelationalen Datenbanksystemen. Jenaer Schriften zur Mathematik und Informatik, Institut für Informatik, Friedrich-Schiller-Universität Jena, January 2003.

[Luf03b] J. Lufter. Kollektionsunterstützung für SQL:1999. *Informatik - Forschung und Entwicklung*, 2003. Angenommen zur Veröffentlichung.

[The96] S. Thelemann. Semantische Anreicherung eines Datenmodells für komplexe Objekte. Dissertation, FB 17 Mathematik/Informatik, Universität Gesamthochschule Kassel, June 1996.