

10. GI-Fachtagung
Datenbanksysteme in Büro, Technik und Wissenschaft

**Integritätsbedingungen für komplexe Objekte
in objektrelationalen Datenbanksystemen**

Jens Luffer

Friedrich-Schiller-Universität Jena
Institut für Informatik
luffer@informatik.uni-jena.de

Leipzig, 26. Februar 2003

Gliederung

Einführung

Integritätsbedingungen für komplexe Objekte

- Arten von Integritätsbedingungen
- Nutzung komplexer Datenstrukturen
- Integritätssicherung für komplexe Objekte

Wege zur Ergänzung der SQL-Norm

- Zugriff auf die innere Struktur komplexer Objekte
- Spezifikation von Integritätsbedingungen
- Sprachmöglichkeiten am Beispiel

Zusammenfassung und Ausblick

Motivation

Objektrelationale Datenbanksysteme

- erweiterbare Typsysteme, objektorientierte Konzepte
 - in Norm und Produkten bislang recht heterogen umgesetzt
- ein Aspekt sind komplex geschachtelte Datenstrukturen
 - Strukturen: Tupeltypen (ROW), strukturierte Typen (Objekttypen)
 - Kollektionsdatentypen für Felder, Mengen, Listen usw.
- Problem: Integritätssicherung für derart komplexe Objekte
 - in Norm und Produkten bislang kaum existent

Zielsetzung

- Integritätskonzept für komplexe Datenstrukturen in ORDBMS
 - NOT NULL, Schlüssel, Fremdschlüssel, gesicherte Objektreferenzen
- Integration der neuen Sprachmittel in die SQL-Norm
 - Basis ist SQL:2003

Integritätsbedingungen in (O)RDBS

Arten von Integritätsbedingungen

- allgemeine Bedingungen
 - nutzen Anfragemächtigkeit von SQL
 - CHECK-Bedingungen, Zusicherungen (in der Norm: tabellenübergreifend)
- spezialisierte Bedingungen
 - spezifische Notation, effizienter implementierbar
 - semantische Zusätze, u. U. aktives Verhalten
 - NOT NULL, Schlüssel, Fremdschlüssel, gesicherte Objektreferenzen
- setzen bislang i. w. auf der Attributebene von Tabellen an

Integritätssicherung für komplexe Objekte

- durch allgemeine Bedingungen realisierbar, aber:
 - selbst einfache Aufgaben (NOT NULL) schwer definierbar
 - kein aktives Verhalten (ON DELETE CASCADE)
- Folgerung: Spezialisierte Bedingungen nötig

Nutzung komplexer Datenstrukturen

Beispiel (mit erweiterter Kollektionsunterstützung)

```
CREATE TABLE Angestellter (  
  ID          IdTyp PRIMARY KEY,  
  Name        ROW ( Nachname CHAR(20), Vornamen LIST ( CHAR(20) ) ),  
  Adressen    ROW ( PLZ CHAR(5), Ort CHAR(20) ) ARRAY[3],  
  MailListen SET ( LIST ( IdTyp ) ),  
  Kinder      SET ( REF ( KindTyp ) SCOPE KindTabelle ),  
  
  CONSTRAINT MailIDsSimulatedForeignKey CHECK ( NOT EXISTS (  
    SELECT *  
    FROM   Angestellter a,  
          UNNEST ( a.MailListen ) AS ml ( Liste ),  
          UNNEST ( ml.Liste ) AS l ( ID )  
    WHERE l.ID IN ( SELECT ID FROM Angestellter ) ) ),  
  
  CONSTRAINT VornameNotNull CHECK ( NOT EXISTS (  
    SELECT *  
    FROM   Angestellter a,  
          UNNEST ( a.Name.Vornamen ) AS vl ( Vorname )  
    WHERE vl.Vorname IS NULL ) ) )
```

Integritätssicherung für komplexe Objekte

Neue spezialisierte Integritätsbedingungen

- angelehnt an vorhandene Konstrukte der SQL-Norm
 - NOT NULL, Schlüssel, Fremdschlüssel, gesicherte Objektreferenzen
 - akzeptabler Integrationsaufwand
- können in komplexe Datenstrukturen „eindringen“
- weitere Arten von Integritätsbedingungen wären denkbar

Klare Trennung von der Typspezifikation

- auf Tabellenebene definiert, änderbar per ALTER TABLE
- problematische Gegenbeispiele:
 - Informix: LIST (INTEGER NOT NULL)
 - SQL:1999: ROW (Feld1 SCOPE Tab1 REFERENCES ARE CHECKED ON DELETE NO ACTION)

Viele Probleme, daher in SQL:2003 nicht mehr spezifizierbar.
- vermeidet Probleme: Orthogonalität, Änderbarkeit, Wiederverwendung

Zugriff auf komplexe Datenstrukturen

Derzeitige Sprachmöglichkeiten der SQL-Norm

- Zugriff auf Attribute von Strukturtypen (Punkt-Notation)
- Zugriff auf einzelne Elemente geordneter Kollektionen
- Beispiele: `Name.Vornamen[1]` oder `Adressen[2].Ort`

Ansprechen aller Elemente einer Kollektion

- Elementtypen sind unbenannt \implies erweiterte Punktnotation
Beispiele: `Name.Vornamen.ELEMENT` oder `Adressen.ELEMENT.Ort`
- semantisch äquivalent zum Entschachteln
- Beispiel: `Name.Vornamen.ELEMENT NOT NULL` entspricht

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

Erzeugen und Löschen von Integritätsbedingungen

Spezifikation bei der Definition von Datenbanktabellen

- wie gewohnt auf Tabellen- bzw. Attributebene nutzbar:
 - auf Attributebene z. B. `Vornamen.ELEMENT`
 - auf Tabellenebene z. B. `Name.Vornamen.ELEMENT`
- Erzeugen und Löschen auch mit `ALTER TABLE`
- orthogonale Erweiterung bestehender Integritätsmechanismen
- wichtig: keine Vermischung mit der Definition von Datentypen

Spezifizierbare Integritätsbedingungen

- `NOT NULL`-Bedingungen, beliebig feingranular
- Schlüssel für Datenbanktabellen
- Fremdschlüssel inkl. referentieller Aktionen
- gesicherte Objektreferenzen inkl. referentieller Aktionen

Sprachmöglichkeiten am Beispiel

```
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,  
  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] ) )
```

Zusammenfassung und Ausblick

Schlußfolgerungen

- ORDBS erlauben komplex geschachtelte Datenstrukturen
 - in Norm und Produkten bislang recht heterogen realisiert
 - Integritätssicherung ist unterentwickelt
- Realisierung spezialisierter Integritätskonzepte erforderlich
 - Ergänzung der SQL-Norm mit vertretbarem Aufwand möglich
 - Konzept orientiert sich an vorhandenen Sprachmitteln

Weitere Arbeiten

- detailliertere Untersetzung und Verifikation der Semantik
 - Anpassung von Syntax und Semantik an SQL:2003
 - Erarbeitung konkreter Vorschläge für SQL5
- prototypische Umsetzung der neuen Sprachmittel