Max-Planck-Institut Saarbrücken
AG5: Datenbanken und Informationssysteme
Dozenten: Prof. Dr.-Ing. G. Weikum, Dr. Ralf Schenkel
Betreuer: Dipl.-Inform. Sergej Sizov, Dipl.-Inform. Christian Zimmer

Informationssysteme (SS 04)
Übungsblatt 6

Beispiellösungen

Aufgabe 1: Integritätsbedingungen

Gegeben sei das aus früheren Übungen bekannte Schema einer Universitätsdatenbank:

Professor	(P_Name, Fachrichtung_Nr, Gebäude, Raum, Tel)
Fachrichtung	(Fachrichtung_Nr, F_Name, Studiendekan)
Gebäude	(Gebäude, Hausmeister)
Student	(Matrikel_Nr, S_Name, Semester, Fachrichtung_Nr)
Prüfung	(Matrikel_Nr, Fach, Prüfer, Note)

a) Für die Relation Prüfung seien die folgenden Fremdschlüsselbedingungen spezifiziert:

```
CREATE TABLE Prüfung (
…
FOREIGN KEY Matrikel_Nr REFERENCES Student (Matrikel_Nr)
        ON DELETE CASCADE
FOREIGN KEY Prüfer REFERENCES Professor (P_Name)
        ON DELETE SET NULL)
```

Simulieren Sie mit Hilfe geeigneter Trigger-Spezifikationen den Effekt dieser Fremdschlüsselbedingungen.

```
CREATE TRIGGER LöschePrüfungen
AFTER DELETE ON Student
FOR EACH ROW
DELETE  FROM    Prüfung
        WHERE   Prüfung.Matrikel_Nr = Student.Matrikel_Nr

CREATE TRIGGER LöschePrüfer
AFTER DELETE ON Professor
FOR EACH ROW
UPDATE Prüfung
        SET         Prüfer = NULL
        WHERE       Prüfer = Professor.P_Name
```

b) Spezifizieren Sie die folgenden (nicht notwendigerweise mit der realen Welt übereinstimmenden) Integritätsbedingungen mit Hilfe von Assertion- oder Trigger-Deklarationen:

i) Studenten dürfen die Prüfung im Fach „Softwaretechnik“ erst ab dem 5. Semester ablegen.

```
CREATE ASSERTION CONSTRAINT Sem-5
CHECK NOT EXISTS  (SELECT *
```

```
        FROM    Prüfung p, Student s
        WHERE   p.Matrikel_Nr = s.Matrikel_Nr
          AND   Fach = 'Softwaretechnik'
          AND   Semester < 5)
```

ii) Studenten, die bereits im 15. Semester sind, müssen mindestens eine Prüfung abgelegt und bestanden haben.

```
CREATE ASSERTION CONSTRAINT Sem-15
CHECK NOT EXISTS  ((SELECT    Matrikel_Nr
                    FROM      Student
                    WHERE Semester > 14)
                    MINUS
                    (SELECT   Matrikel_Nr
                    FROM      Prüfung
                    WHERE Note ≤ 4.3))
```

Da MINUS in früheren Versionen einiger DBS nicht implementiert war, hier noch eine zweite Variante:

```
CREATE ASSERTION CONSTRAINT Sem-15b
CHECK 1 ≤ ALL       (SELECT COUNT(*)
                     FROM      Student s, Prüfung p
                     WHERE s.Semester > 14
                     AND       s.Matrikel_Nr = p.Matrikel_Nr
                     AND       p.Note ≤ 4.3
                     GROUP BY s.Matrikel_Nr)
```

iii) Professoren sollen sich ihr Büro nicht mit Kollegen teilen müssen.

```
CREATE ASSERTION CONSTRAINT Professor-Büro
CHECK 1 = ALL     (SELECT COUNT(*)
                   FROM      Professor
                   GROUP BY Gebäude, Raum)
```

Eine Variante ohne Group By kann wie folgt formuliert werden:

```
CREATE ASSERTION CONSTRAINT Professor-Büro-b
CHECK NOT EXISTS  (SELECT *
                   FROM      Professor p1, Professor p2
                   WHERE p1.P_Name ≠ p2.P_Name
                   AND       p1.Gebäude = p2.Gebäude
                   AND       p1.Raum = p2.Raum)
```

iv) Die Matrikel_Nr eines Studenten darf nie verändert werden.

```
CREATE TRIGGER MatNr
BEFORE UPDATE OF Matrikel_Nr ON Student
(<Fehlermeldung>; ROLLBACK WORK)
```

v) Die Note eines Studenten darf bei nachträglicher Änderung nur verbessert, nicht aber verschlechtert werden.

```
CREATE TRIGGER Noten-Update
AFTER UPDATE OF Note ON Prüfung
REFERENCING
OLD AS old-prüf NEW AS new-prüf
FOR EACH ROW
WHEN (new-prüf.Note > old-prüf.Note) (ROLLBACK WORK)
```

Aufgabe 2: Integritätsbedingungen

Formulieren Sie die folgenden Integritätsbedingungen als Assertion oder Trigger in SQL:

a) Ein Buch kann frühestens ab seinem Erscheinungsdatum geliefert werden.
Standardlösung (mit Assertion bzw. Check-Klausel):

```
CREATE TABLE verkaeufe(
/* ... */
  CREATE ASSERTION CHECK (Lieferdatum>=ALL(SELECT edatum
  FROM Buecher WHERE Buecher.isbn=isbn))
);
```

Alternativlösung mit Trigger:

```
CREATE OR REPLACE TRIGGER A3a BEFORE INSERT OR UPDATE
  OF lieferdatum,isbn ON verkaeufe REFERENCING NEW AS new
  FOR EACH ROW
DECLARE ed DATE;
BEGIN
  SELECT edatum INTO ed FROM Buecher b WHERE b.isbn=:new.isbn;
  IF :new.lieferdatum<ed
  THEN
    ROLLBACK WORK;
  END IF;
END;
```

b) Für einen Kunden mit einer offenen Bestellung (also einem Bestelldatum ungleich NULL, aber Lieferdatum gleich NULL) darf die Bankverbindung nicht geändert werden.

```
CREATE OR REPLACE TRIGGER A3b BEFORE UPDATE
   OF bankverbindung ON kunden
   REFERENCING NEW AS new FOR EACH ROW
DECLARE num_best LONG;
BEGIN
  SELECT COUNT(*) INTO num_best FROM Verkaeufe V WHERE V.email=:new.email AND
  v.Bestelldatum IS NOT NULL AND v.Lieferdatum IS NULL;
  IF num_best>0
  THEN
    ROLLBACK WORK;
  END IF;
END;
```

Aufgabe 3: Views

Betrachten Sie das Beispielschema der Vorlesung mit den Relationen:

Kunden	(KNr, Name, Stadt, Saldo, Rabatt)
Produkte	(PNr, Bez, Gewicht, Preis, Lagerort, Vorrat)
Bestellungen	(BestNr, Monat, Tag, KNr, PNr, Menge, Summe, Status)

Nehmen Sie an, die Organisation des Unternehmens wird derart geändert, daß ein Produkt an verschiedenen Orten vorrätig gehalten werden kann. Bestellungen sollen grundsätzlich von dem Lager geliefert werden, das den größten Vorrat des entsprechenden Produkts hat.

Die Relation Produkte wird dazu in die zwei folgenden Relationen aufgespalten:

Prod	(PNr, Bez, Gewicht, Preis)
Lager	(PNr, Lagerort, Vorrat)

Wie können Sie erreichen, daß trotz dieser einschneidenden Änderung des Datenbankschemas und der Datenbank die für den Vertrieb essentielle Abfrage der Produktverfügbarkeit:

SELECT Vorrat FROM Produkte WHERE PNr = ...

und das Programm zur Erfassung von Lieferungen (siehe S. 88/89 Vorlesungsskript) unverändert weiterlaufen können?

Folgende UPDATE- und SELECT-Anweisungen müssen mindestens unterstützt werden:

```
SELECT  Vorrat
FROMProdukte
WHERE   PNr = ...

UPDATE Produkte
SET         Vorrat = Vorrat – :menge
WHERE       PNr = :pnr
AND         Vorrat ≥ :menge
```

Wenn möglich, sollten jedoch alle auf der Tabelle Produkte möglichen SQL-Abfragen auf die View Produkte anwendbar sein.

```
CREATE VIEW Produkte (PNr, Lagerort, Vorrat, Bez, Gewicht, Preis) AS
            SELECT  L1.PNr, L1.Lagerort, L1.Vorrat, Prod.Bez, Prod.Gewicht, Prod.Preis
            FROMProd, Lager L1
            WHERE   Prod.PNr = L1.PNr
            AND     L1.Lagerort =(SELECT MAX(L2.Lagerort)
                                  FROM      Lager L2
                                  WHERE L2.PNr = L1.PNr
                                  AND   L2.Vorrat =   (SELECT MAX(L3.Vorrat)
                                                       FROM      Lager L3
                                                       WHERE L2.PNr = L3.PNr));
```

Das Attributschema der View Produkte ist gleich dem Attributschema der Relation Produkte in der ursprünglichen Datenbank und daher gibt es keine syntaktsichen Probleme.

1. SELECT-Anweisungen: Es wird zu jedem Produkt nur ein Tupel mit maximalem Vorrat und zugehörigem Lagerort ausgegeben. Falls es mehrere Lagerorte mit maximalem Vorrat gibt, wird einfach der alphabetisch letzte Lagerort ausgewählt. Daher liefert eine SQL-Anfrage sowohl mit der View Produkte, als auch mit der ursprünglichen Relation Produkte vergleichbare Ergebnisse.

2. UPDATE-Anweisungen: Das Attribut Vorrat der View Produkte ist nicht abgeleitet. Durch den Join werden keine Werte des Attributs dupliziert, die Unterabfragen selektieren lediglich Datensätze aus dem Join. Also ist ein Update des Attributs Vorrat prinzipiell möglich, doch wird in kommerziellen DBS meist der konservative Weg beschritten und ein Update einer solchen View grundsätzlich verboten. Alle anderen Attribute werden durch den Join dupliziert und sind daher nicht über die View Produkte änderbar.

Aufgabe 4: DB-Entwurf mit ODL und Anfragen in OQL

a) Entwerfen Sie zur – bisher relational spezifizierten – Musikdatenbank aus Übung 4 ein entsprechendes ODL-Schema unter Ausnutzung der „relationship“-Klausel.

Disk	(DiskID, DiskTitel, Preis)
Musikstück	(DiskID, StückID, Titel, Länge)
Person	(PID, Name, Nationalität)
Interpret	(PID, DiskID, StückID, Funktion, Instrument)
Autor	(PID, DiskID, StückID, Tätigkeit)

```
class Disk {
  extent Disks;
  attribute String DiskTitel;
  attribute Float Preis;
  relationship list<Musikstück> hatStücke inverse Musikstück::aufDisk;
}

class Musikstück {
  extent Musikstücke;
  relationship Disk aufDisk inverse Disk::hatStücke;
  relationship set<Interpret> hatInterpreten inverse Interpret::musikstück;
  relationship set<Interpret> hatAutoren inverse Autor::musikstück;
  attribute String Titel;
  attribute Integer Länge;
}

class Person {
  extent Personen;
  attribute String Name;
  attribute String Nationalität;
  relationship set<Interpret> interpretIn inverse Interpret::person;
  relationship set<Autor> autorVon inverse Autor::person;
}

class Interpret {
  extent Interpreten;
  relationship Person person inverse Person::interpretIn;
  relationship Musikstück musikstück inverse Musikstück::hatInterpreten;
  attribute String Funktion;
  attribute String Instrument;
}

class Autor {
  extent Autoren;
  relationship Person person inverse Person::autorVon;
  relationship Musikstück musikstück inverse Musikstück::hatAutoren;
  attribute String Tätigkeit;
}
```

b) Erweitern Sie das ODL-Schema von a) um:

- Attribute zur Speicherung von Musikaufnahmen als digitales Audio (z.B. im MP3-Format), sowie zur Speicherung von Fotos von Musikern (z.B. im JPEG-Format), wobei die Audios und Fotos selbst einfach vom Typ Binary sein sollen (dies entspricht BLOB in SQL),

```
class Musikstück {
  extent Musikstücke;
  relationship Disk aufDisk inverse Disk::hatStücke;
  relationship set<Interpret> hatInterpreten inverse Interpret::musikstück;
  relationship set<Interpret> hatAutoren inverse Autor::musikstück;
  attribute String Titel;
  attribute Integer Länge;
  relationship Audio audio inverse Audio::zuHören;
}

class Audio {
  extent Audios;
  attribute Binary audio;
  relationship Musikstück zuHören inverse Musikstück::audio;
}

class Person {
  extent Personen;
  attribute String Name;
  attribute String Natinalität;
  relationship Bild bild inverse Bild::zeigtPerson;
}

class Bild {
  extent Bilder;
```

```
  attribute Binary bild;
  relationship Person zeigtPerson inverse Person::bild;
}
```

- Methoden für das Ermitteln
  - der Gesamtdauer einer CD
  - der verschiedenen Nationen, aus denen die Interpreten eines Musikstücks kommen, sowie
  - der verschiedenen Nationen, aus denen die Interpreten einer Disk kommen.

```
class Disk {
  ...
  Real Gesamtlänge();
  Set<String> verschiedeneNationen();
}

class Musikstück {
  ...
  set<String> verschiedeneNationen();
}

class Disk::Gesamtlänge {
  Integer l = 0;
  list<Ref<Musikstück>> DiskStücke = this->hatStücke;
  Iterator<Ref<Musikstück>> it = DiskStücke->create_iterator();
  Ref<Musikstück> m;
  while (m = it.next())
  {
    l += m->Länge;
  }
  return l;
}


class Musikstück::verschiedeneNationen {
  set<String> nationen;
  list<Ref<Interpret>> interpreten = this->hatInterpreten;
  Iterator<Ref<Interpret>> it_i = interpreten->create_iterator();
  Ref<Interpret> I;
  while (i = it_i.next())
  {
    nationen->insert_element(i->person->Nationalität);
  }
  return nationen;
}

class Disk::verschiedeneNationen {
  set<String> nationen;
  list<Ref<Musikstück>> diskStücke = this->hatStücke;
  Iterator<Ref<Musikstück>> it_m = diskStücke->create_iterator();
  Ref<Musikstück> m;
  While (m = it_m.next())
  {
    nationen->union(m->verschiedeneNationen());
  }
  return nationen;
}
```

c) Formulieren Sie die folgenden Anfragen in OQL:

- Welche Stücke hat F. Chopin komponiert?

```
SELECT    DISTINCT (M.Titel)
FROM      Musikstücke M
WHERE     (EXISTS A IN M.hatAutoren :  A.person.Name= “F.Chopin” AND
                                       A.Tätigkeit = “Komponist”)
```

alternativ:

```
SELECT   DISTINCT (A.musikstück.Titel)
FROM     Autoren A
WHERE    A.person.Name="F.Chopin"
AND      A.Tätigkeit="Komponist"
```

aber nicht:

```
SELECT   DISTINCT (M.Titel)
FROM     Musikstücke M
WHERE    M.hatAutoren.person.Name="F.Chopin"
AND      M.hatAutoren.Tätigkeit="Komponist"
```

- Welche Disks enthalten kein Stück, das länger als 60 (Sekunden) dauert?

```
SELECT   DISTINCT (D.DiskTitel)
FROM     Disks D
WHERE    (FOR ALL M IN D.hatStücke: M.Länge ≤ 60)
```

- Welchen Durchschnittspreis haben Disks, auf denen Interpreten aus über drei Nationen zu hören sind und mindestens ein Interpret aus Deutschland kommt?

```
SELECT   AVG(D.Preis)
FROM     Disks D
WHERE    D.verschiedeneNationen()->cardinality ≥ 3
AND      D.verschiedeneNationen()->contains_element(„Deutschland")
```

- Auf welchen Disks sind Trompeten zu hören?

```
SELECT   I.musikstück.aufDisk.diskTitel
FROM     Interpreten I
WHERE    I.Instrument="Trompete"
```

- Auf welchen Disks gibt es ein Musikstück mit einer Dauer von weniger als 10 (Sekunden)?

```
SELECT   D.DiskTitel
FROM     Disks D
WHERE    (EXISTS M IN D.musikstücke: M.Länge < 10)
```

oder:

```
SELECT   D.DiskTitel
FROM     Disks D
WHERE    D.musikstücke.Länge > 10
```

- Wieviele verschiedene Nationalitäten haben im Durchschnitt die Interpreten einer Disk?

```
SELECT   AVG(L.cardinality)
FROM     (SELECT D.verschiedeneNationen() AS L)
```

Aufgabe 2: OQL

Gegeben ist das folgende ODL-Schema einer extrem vereinfachten Klimadatenbank.

```
class Land {
   extent Laender;
   key Landesbez;
   attribute String Landesbez;
   relationship Set<Stadt> HatStaedte inverse Stadt::GehoertZuL;
   relationship Set<See> HatSeen inverse See::GehoertZuL;
}

class Stadt {
   extent Staedte;
   key Stadtname;
   attribute String Stadtname;
   attribute Integer Einwohnerzahl;
   relationship Land GehoertZuL inverse Land::HatStaedte;
   relationship Set<Fluss> LiegtAnF inverse Fluss::FliesstDurchS;
   attribute Zeitreihe Temperatur;
   attribute Zeitreihe Niederschlag;
}

class Fluss {
   extent Fluesse;
   key Flussbez;
   attribute String Flussbez;
   relationship Set<Stadt> FliesstDurchS inverse Stadt::LiegtAnF;
   relationship Set<See> HatSeen inverse See::GehoertZuF;
}

class See {
   extent See;
   key Seebez;
   attribute String Seebez;
   relationship Fluss GehoertZuF inverse Fluss::HatSeen;
   relationship Set<Land> GehoertZuL inverse Land::HatSeen;
   attribute Zeitreihe Pegelstand;
}

class Zeitreihe {
   attribute Array<Struct<Tagesdaten: Date; Messwerte: Real>>;
   Real Messwert (in Date);
   Real Durchschnittswert (in Date, in Date);
            // Durchschnitt von Datum1 bis Datum2
   Real Gleitdurchschnittswert (in Integer);
            // Gleitender Durchschnittswert für N aufeinander
            // folgende Tage
}
```

Formulieren Sie die folgenden Anfragen in OQL:

a) Welche Seen wären von einer Wasserverschmutzung in einer Schweizer Stadt betroffen?

```
SELECT DISTINCT (FLATTEN(L.HatStaedte.LiegtAnF.HatSeen))
FROM       Laender L
WHERE L.Landesbez="Schweiz"
```

Bemerkung zu FLATTEN:
L.HatStaedte liefert eine Menge (Menge='set' ist Subklasse von ‚collection') von Städten, L.HatStaedte.LiegtAnF liefert dann eine Menge von Mengen von Flüssen und L.HatStaedte.LiegtAnF.HatSeen liefert schließlich eine Menge von Mengen von Mengen von Seen. Wir nehmen zur Vereinfachung an, daß die Funktion FLATTEN beliebig tief verschachtelte Collections von Collections von ... von Collections mit Instanzen vom Typ T in eine (flache) Collection mit Instanzen vom Typ T konvertieren kann. Laut ODGM 2.0 ist die Funktion nur auf eine Collection einer Collection vom Typ T anwendbar, so daß FLATTEN demnach eigentlich mehrfach angewendet werden müßte (siehe R.G.G. Cattel et al.: The Object Database Standard: ODMG 2.0, p 110f).

b) Welche Seen wären von einer Wasserverschmutzung in einer Schweizer Großstadt (d.h. mit mehr als 100.000 Einwohnern) betroffen?

```
SELECT  DISTINCT(FLATTEN(S.LiegtAnF.HatSeen))
FROMStaedte S
WHERE   S.GehoertZuL.Landesbez="Schweiz"
AND     S.Einwohnerzahl > 100.000
```

c) Wie hoch waren am 11. November 1999 die Pegelstände der Seen in Bangladesh?

```
SELECT  S.Pegelstand.Messwert(11.Nov 1999)
FROMSeen S
WHERE   S.GehoertZuL.Landesbez="Bangladesh"
```

oder

```
SELECT  S.Pegelstand.Messwert(11.Nov 1999)
FROMSeen S
WHERE   FLATTEN(S.GehoertZuL.Landesbez)->contains_element(„Bangladesh")
```

oder

```
SELECT  S.Pegelstand.Messwert(11.Nov 1999)
FROMSeen S
WHERE   (EXISTS L IN S.GehoertZuL: L.Landesbez="Bangladesh")
```

d) In welchen anderen Ländern könnte sich ein starker Regen in Bangladesh potentiell auswirken?

```
SELECT  DISTINCT (L.Landesbez)
FROMLaender L
WHERE   L.HatStaedte.LiegtAnF.FliesstDurchS.GehoertZuL.Landesbez="Bangladesh"
OR      L.HatSeen.GehoertZuL.Landesbez="Bangladesh"
```

Alternativen gemäß c)

e) Wie hoch ist der 7-Tages-Durchschnitt der Temperatur in den am Strom Sambesi liegenden Städte?

```
SELECT  S.Stadtname, S.Temperatur.Gleitdurchschnittswert(7)
FROMStaedte S
WHERE   S.LiegtAnF.Flussbez="Sambesi"
```

Alternativen gemäß c)

f) Wie hoch war am 15. November 1999 die Durchschnittstemperatur der Großstädte (d.h. mit mehr als 100.000 Einwohnern) in China?

```
SELECT AVG(S.Temperatur.Messwert(15.Nov 1999))
FROM        Staedte S
WHERE S.Einwohnerzahl > 100.000
AND         S.GehoertZuL.Landesbez="China"
```