Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Prof. H.-J. Schek *A. Feßler, U. Röhm, H. Schuldt, R. Weber*

Informationssysteme IS-G SS 1997

Übung 5 Beispiellösung

Gegeben sei das aus früheren Übungen bekannte Schema einer Universitätsdatenbank:

Professor (P_Name, Abt_Nr, Gebäude, Raum, Tel)
Abteilung(Abt_Nr, A_Name, Vorsteher)
Gebäude (Gebäude, Hausmeister, Adresse)
Student (ETH_Nr, S_Name, Semester, Abt_Nr)
Prüfung (ETH_Nr, Fach, Prüfer, Note)

Aufgabe 1: Allgemeine Integritätsbedingungen

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

a) Studenten dürfen die Prüfung im Fach Informationssysteme erst ab dem 5. Semester ablegen.

```
CREATE ASSERTION IS_Klausur CHECK (
  NOT EXISTS (
      SELECT * FROM Prüfung p, Student s
      WHERE p.ETH_Nr = s.ETH_Nr
      AND Fach = ‘IS‘ AND Semester < 5))
  DEFERRABLE INITIALLY IMMEDIATE
```

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

```
CREATE ASSERTION Sem_15 CHECK (
  NOT EXISTS (
      SELECT ETH_Nr FROM Student
      WHERE Semester > 14
      EXCEPT
      SELECT ETH_Nr FROM Prüfung
      WHERE Note >= 4))
  DEFERRABLE INITIALLY IMMEDIATE
```

Eine andere Möglichkeit, diese Bedingung auszudrücken, ohne `EXCEPT`[1] zu verwenden, lautet:

```
CREATE ASSERTION Sem_15b CHECK (
  NOT EXISTS (SELECT * FROM Student s
              WHERE s.Sem > 14 and
              NOT EXISTS (SELECT * FROM Prüfung p
                          WHERE p.ETH_Nr = s.ETH_Nr
                          AND   p.Note >= 4)))
  DEFERRABLE INITIALLY IMMEDIATE
```

c) Professoren sollen sich ihr Büro nicht mit Kollegen teilen müssen. Allerdings kann der Fall auftreten, dass zwei Professoren ihre Räume miteinander tauschen.

```
CREATE ASSERTION Professor_Büro CHECK (
  1 >= ALL (SELECT COUNT(*)
           FROM Professor GROUP BY Gebäude, Raum))
  DEFERRABLE INITIALLY DEFERRED
```

Eine Variante ohne `GROUP BY` kann wie folgt formuliert werden:

```
CREATE ASSERTION 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))
  DEFERRABLE INITIALLY DEFERRED
```

d) Die ETH-Nr eines Studenten darf nie verändert werden.

```
CREATE TRIGGER Guard_ETH_Nr
  BEFORE UPDATE OF ETH_Nr ON STUDENT
  (<Fehlermeldung>; ROLLBACK WORK)
```

e) Die Note eines Studenten darf nur erhöht, nicht aber erniedrigt werden.

```
CREATE TRIGGER Noten_Update
  AFTER UPDATE OF Note ON Prüfung
  REFERENCING OLD AS old_pruef NEW AS new_pruef
  WHEN (new_pruef.Note < old_pruef.Note)
  (ROLLBACK WORK)
```

f) Studenten der Abteilung IIIc dürfen nur an die Abteilung IIIb wechseln, wenn sie in Elektrotechnik mindestens die Note 4.5 erreicht haben.

```
CREATE TRIGGER Abt_Wechsel
  AFTER UPDATE OF Abt_Nr ON Student
  REFERENCING OLD AS old_stud NEW AS new_stud
  WHEN (old_stud.Abt_Nr = ‘IIIc‘ AND new_stud.Abt_Nr = ‘IIIb‘ AND
        4.5 > (SELECT MAX(Note) FROM Prüfung p
               WHERE p.ETH_Nr = old_stud.ETH_Nr
```

[1] Die SQL'92-konforme Anweisung `EXCEPT` kennt Oracle 7.2 nicht; man muss für diese Datenbank anstelle von `EXCEPT` den Ausdruck `MINUS` verwenden.

```
              AND   Fach = 'Elektrotechnik')
   (<Fehlermeldung>; ROLLBACK WORK)
```

Aufgabe 2: Deferrability

a) In einer Datenbank mit 100 Tabellen sind dreitausend Assertions (alle mit der Deferrability `not deferrable`) spezifiziert, die laut Definition jederzeit gültig sein müssen. Wie kann das Datenbanksystem vermeiden, dass nach jeder Operation *alle* diese Assertions auf ihre Gültigkeit überprüft werden müssen? Welche Informationen sind hierzu nötig ?

   Um zu vermeiden, dass nach jeder Operation alle Assertions geprüft werden müssen, kann das Datenbanksystem speichern, welche Tabellen an welcher Assertion beteiligt sind. Eine solche Tabelle mit diesen Meta-Informationen kann beispielsweise folgendermassen aussehen:
   `ASSERTION_CHECK(Table_Name, Assertion_Name)`
   Mit Hilfe dieser Information können nach einer Operation auf Tabelle `A` diejenigen Assertions (`Assertion_Name`) bestimmt werden, zu denen ein Eintrag in `ASSERTION_CHECK` mit Tabelle `A` existiert, so dass auch nur diese getestet werden müssen.

b) Wann ist es sinnvoll, dass Assertions bzw. Constraints die Deferrability `deferrable initially deferred` besitzen? Geben Sie hierzu ein Beispiel an.

   Sobald eine Assertion über mehrere Tabellen gelten soll ist die Deferrability `deferrable initially deferred` sinnvoll, da in der Regel erst nach mehreren Operationen (auf den beteiligten Tabellen) die Integritätsbedingung der Assertion wieder gültig ist.
   Ein Beispiel hierzu ist die Assertion Saldobedingung, die in der Vorlesung vorgestellt wurde (Skript, Seite 5-7). Wenn ein Kunde eine Lieferung bezahlt, dann muss sowohl der Status der Bestellung von `'geliefert'` auf `'bezahlt'` geändert werden und danach der Saldobetrag um den Betrag der Bestellung erhöht werden. Erst nach *beiden* Operationen ist die Saldobedingung wieder gültig.

c) Gegeben sei die aus der Vorlesung bekannte Beispieldatenbank:

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

   sowie die Assertion `Max_Wert`:

```
CREATE ASSERTION Max_Wert CHECK (
  NOT EXISTS (
     SELECT *
     FROM   Produkte
     WHERE  Preis*Vorrat > 500000))
DEFERRABLE INITIALLY IMMEDIATE
```

   Gegeben ist der folgende Ausschnitt einer Transaktion, in der u.a. ein neues Produkt sowie eine Bestellung dieses Produktes eingegeben wird, und die Bearbeitung der Lieferung erfolgt.

```
BEGIN OF TRANSACTION
.....
.....
.....
 SET CONSTRAINT Max_Wert DEFERRED;
INSERT INTO Produkte VALUES (8, 'Server', 7.500, 80000, 'Zug', 10);
.....
INSERT INTO Bestellungen VALUES (17, 4, 29, 5, 8, 5, 400000, 'neu');
.....
UPDATE Produkte SET Vorrat = Vorrat-5 WHERE PNr=8;
 SET CONSTRAINT Max_Wert IMMEDIATE;
.....
UPDATE Bestellungen SET Status = 'geliefert' WHERE BestNr=17;
.....
.....
.....
END OF TRANSACTION
```

Die Assertion `Max_Wert` soll so früh wie möglich innerhalb der Transaktion überprüft werden. Wie muss man obige Transaktion hierzu erweitern ?
Welche Nachteile kann ein Test der Assertion am Ende der Transaktion mit sich führen?

Stellt man erst am Ende einer Transaktion fest, dass eine Assertion verletzt ist, so muss man alle Operationen, die innerhalb dieser Transaktion ausgeführt wurden, rückgängig machen. Diesen Aufwand kann man reduzieren, wenn man Assertions so früh wie möglich innerhalb einer Transaktion auf ihre Gültigkeit überprüft. Stellt man bereits zu diesem Zeitpunkt fest, dass die Asssertion verletzt ist und wird die Assertion auch durch die restlichen Operationen der Transaktion nicht mehr beeinflusst, so kann man die Transaktion bereits zu diesem frühen Zeitpunkt abbrechen.

Aufgabe 3: Fremdschlüsselbedingungen

a) Für die Relation *Student* ist die folgende Fremdschlüsselbedingung definiert:

```
CREATE TABLE Student (
...
FOREIGN KEY Abt_Nr REFERENCES Abteilung(Abt_Nr),
...
)
```

„Simulieren“ Sie diese Fremdschlüsselbedingung

i. als „gewöhnlichen“ Table Constraint in der Relation *Student*

```
CREATE TABLE Student (
...
CONSTRAINT Fremdschluessel
CHECK ( Abt_Nr IS NULL OR EXISTS ( SELECT * FROM Abteilung
                 WHERE  Abteilung.Abt_Nr = Abt_Nr ) ),
...
)
```

Bemerkung: In einem Table-Constraint einer Relation *TableName* bezieht sich *ColumnName* immer auf *TableName.ColumnName*! Alle anderen Referenzen auf Attribute müssen explizit qualifiziert werden.

ii. als allgemeine Assertion (Zusatzfrage: Wie sähe diese Assertion im Relationenkalkül aus?)

```
CREATE ASSERTION Stud_Ref_Abteilung
CHECK ( NOT EXISTS (
        SELECT * FROM Student
        WHERE  NOT EXISTS (
               SELECT * FROM Abteilung
               WHERE  Abteilung.Abt_Nr = Student.Abt_Nr )
        AND Student.Abt_Nr IS NOT NULL
) )
DEFERRABLE INITIALLY IMMEDIATE
```

Bemerkung: Allg. Assertions müssen oft mit doppelter Negation ausgedrückt werden, da SQL den Allquantor nicht unterstützt. Weil

$$\forall\, F \;\equiv\; \nexists \neg\, F,$$

gilt:

$\forall s \in Student(\exists a \in Abteilung(a.Abt\_Nr = s.Abt\_Nr) \vee s.Abt\_Nr = NULL)$

iii. als Trigger

```
CREATE TRIGGER Abteilung_Vorhanden1
  AFTER INSERT ON Student
```

```
  REFERENCING NEW AS s
  WHEN ( s.Abt_Nr IS NOT NULL AND
         NOT EXISTS ( SELECT * FROM Abteilung a
                      WHERE  a.Abt_Nr = s.Abt_Nr ) )
  ( ROLLBACK WORK )

CREATE TRIGGER Abteilung_Vorhanden2
  AFTER UPDATE OF Abt_Nr ON Student
  REFERENCING NEW AS s
  WHEN ( s.Abt_Nr IS NOT NULL AND
         NOT EXISTS ( SELECT * FROM Abteilung a
                      WHERE  a.Abt_Nr = s.Abt_Nr ) )
  ( ROLLBACK WORK )

CREATE TRIGGER Noch_Studenten
  AFTER DELETE ON Abteilung
  REFERENCING OLD AS a
  WHEN ( EXISTS ( SELECT * FROM Student s
                  WHERE  s.Abt_Nr = a.Abt_Nr ) )
  ( ROLLBACK WORK )

CREATE TRIGGER Abteilungs_Umnummerierung
  AFTER UPDATE OF Abt_Nr ON Abteilung
  REFERENCING OLD AS a
  WHEN ( EXISTS ( SELECT * FROM Student s
                  WHERE  s.Abt_Nr = a.Abt_Nr ) )
  ( ROLLBACK WORK )
```

Bemerkung: Anstelle der vier `AFTER`-Trigger können auch die entsprechenden `BEFORE`-Trigger der Art

```
CREATE TRIGGER Abteilung_Vorhanden1
  BEFORE INSERT ON Student
  REFERENCING NEW AS s
  WHEN ( s.Abt_Nr IS NOT NULL AND
         NOT EXISTS ( SELECT * FROM Abteilung a
                      WHERE  a.Abt_Nr = s.Abt_Nr ) )
  ( ROLLBACK WORK )
```

verwendet werden.

Es ist umstritten, ob das Ändern von Primärschlüsseln erlaubt sein soll. Mittels Triggern ist es möglich, dies entweder zu verhindern oder die Änderung zu propagieren.

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

```
CREATE TABLE Prüfung (
...
FOREIGN KEY ETH_Nr REFERENCES Student (ETH_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
  REFERENCING OLD AS S
  ( DELETE FROM Prüfung
    WHERE Prüfung.ETH_Nr = S.ETH_Nr )

CREATE TRIGGER LöschePrüfer
  AFTER DELETE ON Professor
  REFERENCING OLD AS P
  ( UPDATE Prüfung
    SET Prüfer = NULL
    WHERE Prüfer = P.P_Name )
```

Dazu gehören noch Trigger für die Einhaltung der Fremdschlüsselbedingungen bei INSERT und UPDATE Operationen, die analog zur vorherigen Aufgabe gestaltet werden.