Kapitel 8

Datenintegrität

8.1 Grundlagen

In diesem Kapitel werden *semantische Integritätsbedingungen* behandelt, also solche, die sich aus den Eigenschaften der modellierten Welt ableiten lassen. Wir unterscheiden statische und dynamische Integritätsbedingungen. Eine statische Bedingung muß von jedem Zustand der Datenbank erfüllt werden (z. B. Professoren haben entweder den Rang C2, C3 oder C4). Eine dynamische Bedingung betrifft eine Zustandsänderung (z. B. Professoren dürfen nur befördert, aber nicht degradiert werden).

Einige Integritätsbedingungen wurden schon behandelt:

- Die Definition des Schlüssels verhindert, daß zwei Studenten die gleiche Matrikelnummer haben.
- Die Modellierung der Beziehung *lesen* durch eine 1:N-Beziehung verhindert, daß eine Vorlesung von mehreren Dozenten gehalten wird.
- Durch Angabe einer Domäne für ein Attribut kann z. B. verlangt werden, daß eine Matrikelnummer aus maximal 5 Ziffern besteht (allerdings wird nicht verhindert, daß Matrikelnummern mit Vorlesungsnummern verglichen werden).

8.2 Referentielle Integrität

Seien R und S zwei Relationen mit den Schemata $\mathcal{R}$ und

$\mathcal{S}$. Sei κ Primärschlüssel von $\mathcal{R}$.

Dann ist $\alpha \subset \mathcal{S}$ ein Fremdschlüssel, wenn für alle Tupel $s \in S$ gilt:

1. s.α enthält entweder nur Nullwerte oder nur Werte ungleich Null
2. Enthält $s.\alpha$ keine Nullwerte, existiert ein Tupel $r \in R$ mit $s.\alpha = r.\kappa$

Die Erfüllung dieser Eigenschaft heißt *referentielle Integrität*. Die Attribute von Primär- und Fremdschlüssel haben jeweils dieselbe Bedeutung und oft auch dieselbe Bezeichnung (falls möglich). Ohne

Überprüfung der referentiellen Integrität kann man leicht einen inkonsistenten Zustand der Datenbasis erzeugen, indem z. B. eine Vorlesung mit nichtexistentem Dozenten eingefügt wird.

Zur Gewährleistung der referentiellen Integrität muß also beim Einfügen, Löschen und Ändern immer sichergestellt sein, daß gilt

$$\pi_\alpha(S) \subseteq \pi_\kappa(R)$$

Erlaubte Änderungen sind daher:

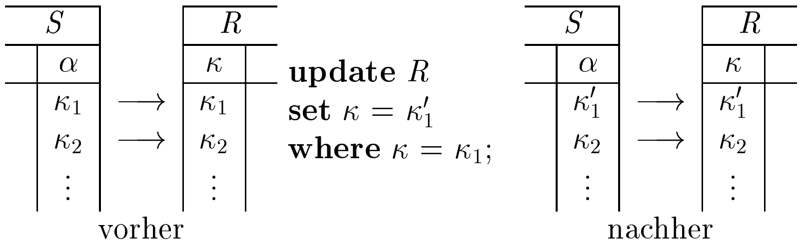
- Einfügen eines Tupels in S verlangt, daß der Fremdschlüssel auf ein existierendes Tupel in R verweist.
- Ändern eines Tupels in S verlangt, daß der neue Fremdschlüssel auf ein existierendes Tupel in R verweist.
- Ändern eines Primärschlüssels in R verlangt, daß kein Tupel aus S auf ihn verwiesen hat.
- Löschen eines Tupels in R verlangt, daß kein Tupel aus S auf ihn verwiesen hat.

8.3 Referentielle Integrität in SQL

SQL bietet folgende Sprachkonstrukte zur Gewährleistung der referentiellen Integrität:

- Ein Schlüsselkandidat wird durch die Angabe von `unique` gekennzeichnet.
- Der Primärschlüssel wird mit `primary key` markiert. Seine Attribute sind automatisch `not null`.
- Ein Fremdschlüssel heißt `foreign key`. Seine Attribute können auch `null` sein, falls nicht explizit `not null` verlangt wird.
- Ein `unique foreign key` modelliert eine 1:1 Beziehung.
- Innerhalb der Tabellendefinition von S legt die Klausel α `integer references R` fest, daß der Fremdschlüssel α (hier vom Typ Integer) sich auf den Primärschlüssel von Tabelle R bezieht. Ein Löschen von Tupeln aus R wird also zurückgewiesen, solange noch Verweise aus S bestehen.
- Durch die Klausel `on update cascade` werden Veränderungen des Primärschlüssels auf den Fremdschlüssel propagiert (Abbildung 8.1a).
- Durch die Klausel `on delete cascade` zieht das Löschen eines Tupels in R das Entfernen des auf ihn verweisenden Tupels in S nach sich (Abbildung 8.1b).
- Durch die Klauseln `on update set null` und `on delete set null` erhalten beim Ändern bzw. Löschen eines Tupels in R die entsprechenden Tupel in S einen Nulleintrag (Abbildung 8.2).

(a) **create table** S (..., α **integer references** R **on update cascade**);



(b) **create table** S (..., α **integer references** R **on delete cascade**);

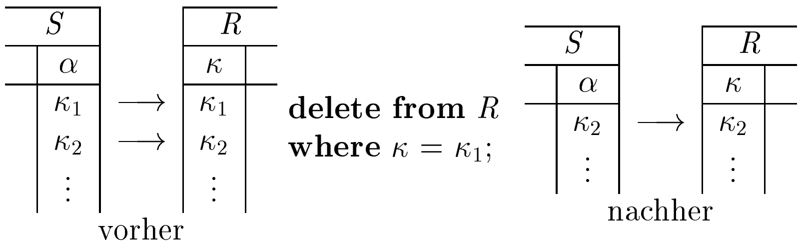
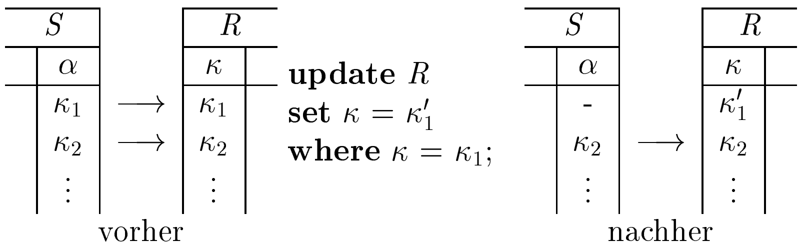


Abbildung 8.1: Referentielle Integrität durch Kaskadieren

a) **create table** S (..., α **integer references** R **on update set null**);



(b) **create table** S (..., α **integer references** R **on delete set null**);

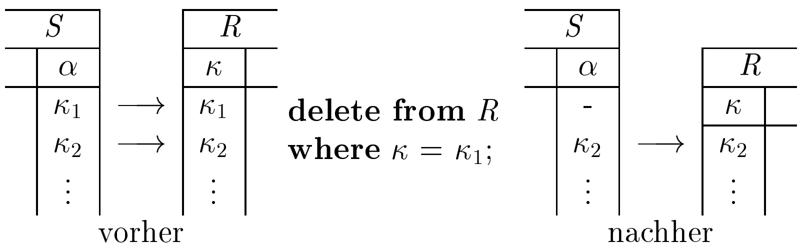
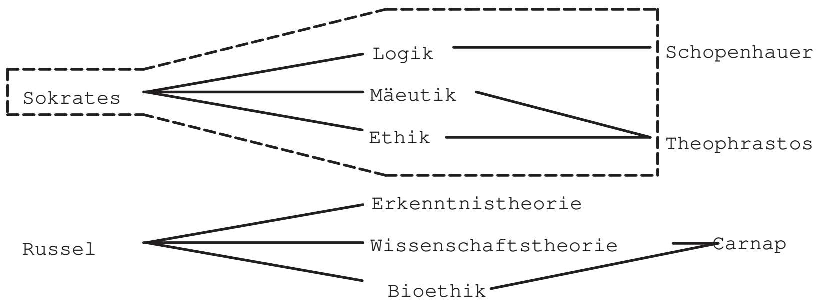


Abbildung 8.2: Referentielle Integrität durch Nullsetzen

Kaskadierendes Löschen kann ggf. eine Kettenreaktion nach sich ziehen. In Abbildung 8.3 wird durch das Löschen von `Sokrates` der gestrichelte Bereich mit drei Vorlesungen und drei *hören*-Beziehungen entfernt, weil der Fremdschlüssel *gelesenVon* die Tupel in *Professoren* mit `on delete cascade` referenziert und der Fremdschlüssel *VorlNr* in *hören* die Tupel in *Vorlesungen* mit `on delete cascade` referenziert.




```
create table Vorlesung
(...,
gelesenVon integer
           references Professoren
           on delete cascade);
```

```
create table hoeren
(...,
VorlNr integer
       references  Vorlesungen
       on delete cascade);
```

Abbildung 8.3: Kaskadierende Löschoperationen

8.4 Statische Integrität in SQL

Durch die *check-Klausel* können einem Attribut Bereichseinschränkungen auferlegt werden.

Zum Beispiel erzwingen

```
... check Semester between 1 and 13 ...
... check Rang in ('C2', 'C3', 'C4') ...
```

gewisse Vorgaben für die Semesterzahl bzw. den Professorenrang.

Listing 8.1 zeigt die Formulierung der Uni-Datenbank mit den Klauseln zur Überwachung von statischer und referentieller Integrität. Hierbei wurden folgende Restriktionen verlangt:

- Ein Professor darf solange nicht entfernt oder sein Primärschlüssel geändert werden, wie noch Verweise auf ihn existieren.
- Eine Vorlesung darf solange nicht entfernt werden, wie noch Verweise auf sie existieren. Eine Änderung ihres Primärschlüssels ist erlaubt und zieht das Ändern der Sekundärschlüssel nach sich.
- Ein Student darf entfernt werden und zieht dabei das Entfernen der Zeilen nach sich, die über Sekundärschlüssel auf ihn verweisen. Auch sein Primärschlüssel darf geändert werden und zieht das Ändern der Sekundarschlüssel nach sich.

```
CREATE TABLE Studenten (
        MatrNr          INTEGER PRIMARY KEY,
        Name            VARCHAR(20) NOT NULL,
        Semester        INTEGER,
        GebDatum        DATETIME)

CREATE TABLE Professoren (
        PersNr          INTEGER PRIMARY KEY,
        Name            VARCHAR(20) NOT NULL,
        Rang            CHAR(2) CHECK (Rang in ('C2', 'C3', 'C4')),
        Raum            INTEGER UNIQUE,
        Gebdatum        DATETIME)

CREATE TABLE Assistenten (
        PersNr          INTEGER PRIMARY KEY,
        Name            VARCHAR(20) NOT NULL,
        Fachgebiet      VARCHAR(20),
        Boss            INTEGER REFERENCES Professoren,
        GebDatum        DATETIME)

CREATE TABLE Vorlesungen (
        VorlNr          INTEGER PRIMARY KEY,
        Titel           VARCHAR(20),
        SWS             INTEGER,
        gelesenVon      INTEGER REFERENCES Professoren)

CREATE TABLE hoeren (
        MatrNr          INTEGER REFERENCES Studenten   ON UPDATE CASCADE
                                                       ON DELETE CASCADE,
        VorlNr          INTEGER REFERENCES Vorlesungen ON UPDATE CASCADE,
        PRIMARY KEY     (MatrNr, VorlNr))

CREATE TABLE voraussetzen (
        Vorgaenger      INTEGER REFERENCES Vorlesungen ON UPDATE CASCADE,
        Nachfolger      INTEGER REFERENCES Vorlesungen,
        PRIMARY KEY     (Vorgaenger, Nachfolger))

CREATE TABLE pruefen (
        MatrNr          INTEGER REFERENCES Studenten   ON UPDATE CASCADE
                                                       ON DELETE CASCADE,
        VorlNr          INTEGER REFERENCES Vorlesungen ON UPDATE CASCADE,
        PersNr          INTEGER REFERENCES Professoren,
        Note            NUMERIC(3,1) CHECK (Note between 0.7 and 5.0),
        PRIMARY KEY     (MatrNr, VorlNr))
```

Listing 8.1: Universitätsschema mit Integritätsbedingungen

8.5 Trigger

Die allgemeinste Konsistenzsicherung geschieht durch einen *Trigger*. Dies ist eine benutzerdefinierte Prozedur, die automatisch bei Erfüllung einer bestimmten Bedingung vom DBMS gestartet wird. Hilfreich sind zwei vom System gefüllte Tabellen `DELETED` und `INSERTED`, in der solche Zeilen gespeichert sind, auf die sich das beabsichtigte Löschen bzw. Einfügen beziehen. In diesem Zusammenhang ist die Operation `UPDATE` als hintereinander ausgeführte Kombination von Löschen und Einfügen zu sehen.

Listing 8.2 zeigt einen AFTER-UPDATE-Trigger für die Tabelle Professoren, der nach jedem Update aufgerufen wird und im Falle einer Degradierung diese wieder rückgängig macht.

```
create trigger korrigieredegradierung -- definiere einen Trigger
on Professoren after update           -- nach update von Professoren
as
if update(Rang)                       -- falls Rang veraendert wurde
begin
  update professoren                  -- aendere Professoren erneut
  set    rang = d.rang                -- setze Rang auf alten Rang
  from   professoren p, deleted d     -- mit Hilfe der Tabelle deleted
  where  p.persnr = d.persnr          -- bei uebereinstimmender PersNr
  and    p.rang < d.rang              -- falls neuer Rang < alter Rang
end
```

Listing 8.2: Trigger zur Korrektur einer Degradierung

Listing 8.3 zeigt die Lösung für dasselbe Problem durch einen INSTEAD-OF-UPDATE-Trigger, der statt des Update durchgeführt wird. Durch Vergleich der Einträge in den Systemtabellen `DELETED` und `INSERTED` kann die beabsichtigte Beförderung für solche Professoren durchgeführt werden, die zuvor noch keine Rangangabe aufwiesen oder eine kleinere Rangangabe hatten.

```
create trigger verhinderedegradierung  -- definiere einen Trigger
on Professoren instead of update       -- statt eines updates
as
  update professoren                   -- aendere nun Professoren doch
  set    rang = i.rang                 -- setze auf eingefuegten Rang
  from   deleted d, inserted i         -- unter Verwendung von d und i
  where  professoren.persnr = d.persnr -- bei passender Personalnummer
  and    d.persnr = i.persnr           -- bei passender Personalnummer
  and    (d.rang=null                  -- falls vorher kein Rang vorhanden
         or d.rang < i.rang)           -- oder alter Rang < neuer Rang
```

Listing 8.3: Trigger zur Verhinderung einer Degradierung

Listing 8.4 zeigt einen AFTER-INSERT-Trigger, der immer nach dem Einfügen eines Tupels in die Tabelle *hoeren* einen Professor sucht, der jetzt mehr als 10 Hörer hat und ihn dann nach C4 befördert.

```
create trigger befoerderung         -- definiere den Trigger befoerderung
on hoeren after insert, update      -- nach insert oder update bei hoeren
as
update professoren set rang='C4'    -- befoerdere Professor nach C4
where persnr in                     -- falls seine Personalnummer in
  (select persnr                    -- der Menge der Personalnummern liegt
   from   professoren p,            -- die mehr als 10 Hoerer haben
          vorlesungen v,
          hoeren h
   where  p.persnr = v.gelesenvon
   and    v.vorlnr = h.vorlnr
   group by p.persnr
   having count(*) > 10)
```

Listing 8.4: Trigger zum Auslösen einer Beförderung

Listings 8.5 und 8.6 zeigen die Verwendung eines INSTEAD-OF-INSERT-Triggers im Zusammenhang mit Sichten, in die nur unter sehr eingeschränkten Bedingungen eingefügt werden kann. Auf der Tabelle *Person* sei eine Sicht *Geburtstagsliste* gegeben. In dieser Sicht wird das momentane Lebensalter in Jahren anhand der Systemzeit und des gespeicherten Geburtsdatums errechnet. Natürlich ist diese Sicht nicht update-fähig bei Einfügen eines Namens mit Lebensalter. Durch den Trigger *geburtstag* läßt sich das Einfügen trotzdem erreichen, da nach Umrechnung des Alters in ein Geburtsdatum ein Tupel in die Tabelle *Person* eingefügt werden kann.

```
create view Geburtstagsliste                          -- lege Sicht an
as                                                    -- bestehend aus
select Name,                                          -- Name
       datediff(year,gebdatum,getdate()) as Jahre     -- Lebensalter
from Person                                           -- von Tabelle Person
```

Listing 8.5: Nichtupdatefähige Sicht

```
create trigger Geburtstag                        -- lege Trigger an bzgl.
on Geburtstagsliste                              -- Geburtstagsliste
instead of insert                                -- statt einzufuegen
as
insert into Person (name,gebdatum)               -- fuege in Person ein
select i.name,                                   -- eingefuegter Name
       dateadd(year, -i.jahre, getdate())        -- errechnetes Geburtsjahr
from inserted i                                  -- aus Tabelle inserted
```

Listing 8.6: Trigger zum Einfügen eines errechneten Geburtsdatums

Das Schlüsselwort `drop` entfernt einen Trigger: `drop trigger Geburtstag`