Datenintegrität

VO Datenmodellierung

Katrin Seyr

Institut für Informationssysteme
Technische Universität Wien

FAKULTÄT FÜR !NFORMATIK

Überblick

Integritätsbedingungen

Datenbankmanagementsysteme sind für die Erhaltung der Datenkonsistenz zuständig.

Konsistenzerhaltung nach Systemfehler (Recovery) bzw. bei Mehrbenutzerbetrieb (Concurrency Control) → LVA Datenbanksysteme.

Hier (Kapitel 5, 6): semantische Integritätsbedingungen, die aus den Eigenschaften der modellierten Miniwelt abgeleitet werden.

Beispiele

MatrNr muss eindeutig sein; - in AT: 7 stellig, die ersten zwei Ziffern leiten sich aus dem Immatrikulationsjahr ab, die nächsten zwei Ziffern codieren die Immatrikulationsuniversität.

PersNr ist eine 4-stellige Zahl

Integritätsbedingungen

Integritätsbedingungen werden eingeteilt in:

statische Integritätsbedingungen: Bedingungen (constraints) an den Zustand der Datenbasis, die in jedem Zustand der Datenbank erfüllt sein müssen.

Beispiele

Professoren haben Rang C2, C3, C4,
Matrikelnummer der Studierenden ist eindeutig
Anmeldung zur Übung nur bei aufrechter Inskription

dynamische Integritätsbedingungen: Bedingungen an Zustandsänderungen in der Datenbank.

Beispiel

Professoren dürfen nur befördert werden

Integritätsbedingungen

Bekannte implizite Anforderungen an die Datenintegrität

Schlüssel: müssen innerhalb einer Relation eindeutig sein.

Beziehungskardinalitäten: die im ER Modell definierten Kardinalitäten werden im Relationenmodell so umgesetzt, dass davon abweichende Daten nicht eingegeben werden können.

Attributdomänen: durch die Festlegung von Wertebereichen werden Bedingungen an die Daten gestellt.

Inklusion bei Generalisierung: Entities der Untertypen müssen auch in Obertypen enthalten sein

Statische Integritätsbedingungen in SQL

- Schlüsselkandidaten: `unique(Attributliste)`
- Primärschlüssel: `primary key(Attributliste)`
- Fremdschlüssel: `foreign key(Attributliste)`
- Keine Nullwerte erlaubt: `not null`
- Defaultwert angeben: `default`

Beispiel

```
create table Studenten(MatrNr    integer primary key,
                       Name      varchar(30) not null,
                       Semester  integer default 1);
```

Statische Integritätsbedingungen in SQL

Überprüfung allgemeiner statischer Integritätsbedingungen mittels `check`-Klausel, gefolgt von einer Bedingung

Auswertung der `check`-Klausel bei Änderung/Einfügung in die Tabelle

Änderungen auf einer Tabelle werden nur zugelassen, wenn `check` zu `true` auswertet.

Bedingungen können beliebig komplex sein, auch eine Anfrage enthalten - Achtung: nur sehr rudimentär implementiert!

Beispiel

```
create table Studenten
             (MatrNr    integer primary key,
              Name      varchar(30) not null,
              Semester  integer default 1
              check Semester between 1 and 13);
```

Statische Integritätsbedingungen in SQL

Beispiel

Studenten können sich nur über Vorlesungen prüfen lassen, die sie vorher gehört haben.

```
create table prüfen
(MatrNr integer,
 VorlNr integer,
 PersNr integer,
 Note   numeric(2,1) check(Note between 0.7 and 5.0),
 primary key (MatrNr, VorlNr),
 constraint vorherhören check
       (exists (select * from hören h
                where h.VorlNr = prüfen.VorlNr and
                      h.MatrNr = prüfen.MatrNr)));
```

Referentielle Integrität

Schlüssel identifizieren ein Tupel einer Relation.

Fremdschlüssel verweisen auf Tupel einer in Beziehung stehenden Relation.

Beispiel

`PersNr` ist Schlüssel von Professoren, `gelesenVon` in Vorlesungen verweist auf Tupel in Professoren und ist somit Fremdschlüssel. Es muss sichergestellt werden, dass in `gelesenVon` keine Werte stehen, die in `PersNr` nicht vorkommen.

Definition (referentielle Integrität)

Fremdschlüssel müssen entweder auf existierende Tupel einer anderen Relation verweisen oder einen Nullwert enthalten.

Referentielle Integrität

Definition (Dangling References)

Dangling References sind Verweise von Fremdschlüsseln auf nicht existierende Datensätze

Beispiele (Dangling References)

Einfügen des folgenden Datensatzes in der Tabelle Vorlesungen:

```
insert into Vorlesungen
       values(5100, 'Nihilismus', 40, 0007);
```

⇒ worauf verweist '0007'?

Löschen des folgenden Datensatzes in der Tabelle Professoren:

```
delete from Professoren where PersNr=2125;
```

⇒ worauf verweist '2125' in der Vorlesungs-Tabelle?

Gewährleistung der referentiellen Integrität

Theorem (referentielle Integrität)

Sei R eine Relation mit Primärschlüssel κ und S eine Relation mit Fremdschlüssel α auf R. Die referentielle Integrität ist gewährleistet, wenn

$$\Pi_{\alpha}(S) \subseteq \Pi_{\kappa}(R)$$

Erlaubte Änderungen sind also:

- Einfügen von s in S, wenn $s.\alpha \in \Pi_{\kappa}(R)$
- Verändern Tupels von s zu s' in S, wenn $s'.\alpha \in \Pi_{\kappa}(R)$
- Verändern von $r.\kappa$ in R, wenn $\sigma_{\alpha=r.\kappa} = \emptyset$, d.h. es gibt keine Verweise von S auf r
- Löschen von r in R, wenn $\sigma_{\alpha=r.\kappa} = \emptyset$

FAKULTÄT FÜR !NFORMATIK

Referentielle Integrität in SQL

Beschreibungsmöglichkeit für jeden der drei Schlüsselbegriffe wie folgt:

- (Kandidaten)-Schlüssel: `unique`
- Primärschlüssel: `primary key(Attributliste)`
- Fremdschlüssel: `foreign key(Attributliste)`

Kennzeichnung der Tabelle, auf die verwiesen wird:

- `references (Tabellenname)`

Einhaltung der Referenziellen Integrität bei Updates, Löschen:

- `on update{no action | cascade | set null | set default}`
- `on delete{no action | cascade | set null | set default}`

Achtung bei zyklischen Abhängigkeiten (Chicken-Egg Problem - siehe unten):

- `alter table` bzw. `deferred deferrable`

Referentielle Integrität in SQL

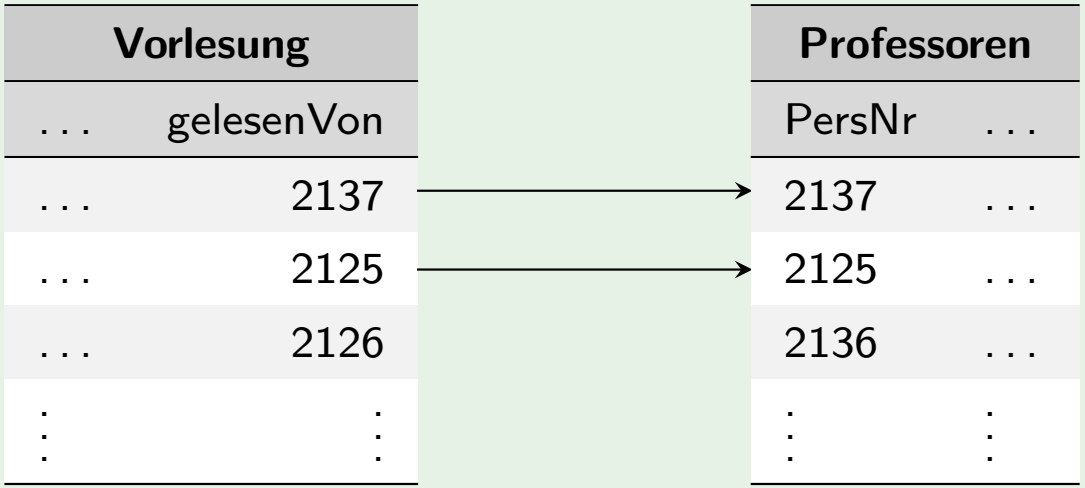
Beispiel

```
create table Professoren
(PersNr    integer primary key,
 Name      varchar(30) not null,
 ...        );

create table Vorlesungen
(VorlNr      integer primary key,
 Titel       varchar(30),
 SWS         integer,
 gelesenVon  integer [foreign key]
             references Professoren
             on update cascade on delete set null);
```

Referentielle Integrität in SQL

Beispiel (Fortsetzung)



Gewünschte Änderungen:

```
update Professoren set PersNr=1111 where PersNr=2137;
delete from Professoren where PersNr=2125;
```

Referentielle Integrität in SQL

Beispiel (Fortsetzung)

```
update Professoren set PersNr=1111 where PersNr=2137;
```

Einhaltung der referentiellen Integrität durch Kaskadieren (`on update cascade`)

Vorlesung	
...	gelesenVon
...	1111
...	2125
...	2126
⋮	⋮

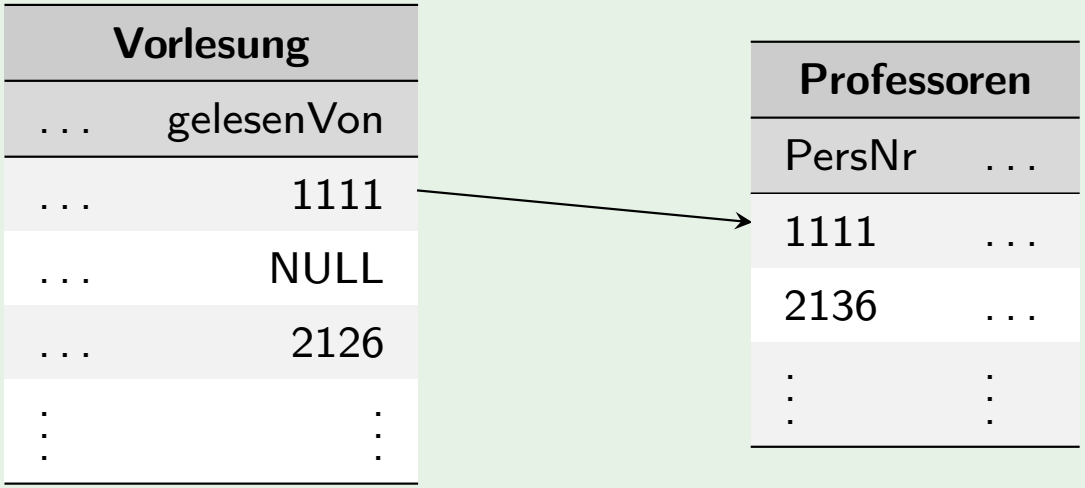
Professoren	
PersNr	...
1111	...
2125	...
2136	...
⋮	⋮

Referentielle Integrität in SQL

Beispiel (Fortsetzung)

```
delete from Professoren where PersNr=2125;
```

Einhaltung der referentiellen Integrität durch "auf NULL setzen" (`on delete set NULL`)

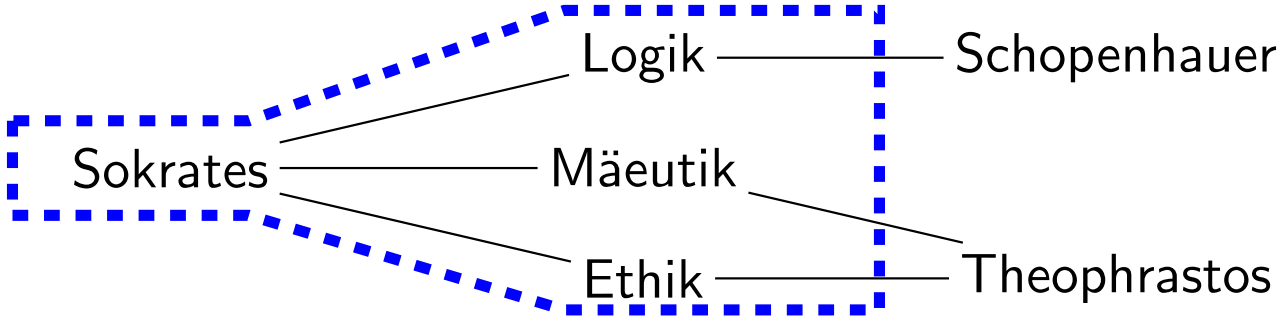


Kaskadierendes Löschen

Vorsicht bei der Verwendung von `on delete cascade`. Es kann zu kaskadierdendem Löschen kommen.

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

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



Universitätsschema mit Integritätsbedingungen

```
create table Studenten
(MatrNr    integer primary key,
 Name      varchar(30) not null,
 Semester  integer check (Semester between 1 and 13));

create table Professoren
(PersNr integer primary key,
 Name   varchar(30) not null,
 Rang   char(2) check(Rang in ('C2','C3','C4')),
 Raum   integer unique);

create table Assistenten
(PersNr        integer primary key,
 Name          varchar(30) not null,
 Fachgebiet    varchar(30),
 Boss          integer references Professoren
               on delete set null);
```

Universitätsschema mit Integritätsbedingungen

```
create table Vorlesungen
(VorlNr integer primary key,
 Titel  varchar(30),
 SWS    integer,
 gelesenVon integer references Professoren
            on delete set null);

create table hören
(MatrNr integer references Studenten
        on delete cascade,
 VorlNr integer references Vorlesungen
        on delete cascade,
 primary key(MatrNr, VorlNr));
```

Universitätsschema mit Integritätsbedingungen

```
create table voraussetzen
(Vorgänger  integer references Vorlesungen
            on delete cascade,
 Nachfolger integer references Vorlesungen
            on delete cascade,
 primary key(Vorgänger, Nachfolger));

create table prüfen
(MatrNr integer references Studenten
        on delete cascade,
 VorlNr integer references Vorlesungen,
 PersNr integer references Professoren
        on delete set null,
 Note   numeric(2,1) check(Note between 0.7 and 5.0),
 primary key(MatrNr, VorlNr));
```

Zyklische Abhängigkeiten

Beispiel (chicken/egg)

Legen Sie die Tabellen chicken und egg an, wobei für jedes Huhn vermerkt wird, aus welchem Ei es geschlüpft ist, und umgekehrt:

```
create table chicken
(cID integer primary key,
 eID integer references egg);
```

```
ORA-00942: table or view does not exist
```

```
create table egg
(eID integer primary key,
 cID integer references chicken);
```

Zyklische Abhängigkeiten

Lösung: verwenden Sie den `alter table` Befehl:

Beispiel (chicken/egg)

```
create table chicken(cID integer primary key,
                     eID integer);

create table egg(eID integer primary key,
                 cID integer references chicken);

alter table chicken add constraint chickenrefegg
foreign key (eID) references egg;
```

Zyklische Abhängigkeiten

Beispiel (chicken/egg)

Das Anlegen der Tabellen funktioniert, aber:

```
insert into chicken values (1,11);
```

```
ORA-02291: integrity constraint (SEYR.CHICKENREFEGG) violated -
parent key not found
```

Zyklische Abhängigkeiten

Lösung: verzögern Sie die Überprüfung der Bedingung bis zum Ende der Transaktion (Definition Transaktion in LVA Datenbanksysteme).

Beispiel (chicken/egg)

```
alter table chicken add constraint chickenrefegg
foreign key (eID) references egg
initially deferred deferrable;

begin;
insert into chicken values(1,11);
insert into egg values(11,1);
commit;
```

Trigger

Standardisierung erst im SQL-99 Standard, waren allerdings schon vorher in kommerziellen DBMS enthalten. Daher ist die Implementierung in den meisten DBMS verschieden.

Verwendung zur Sicherstellung Datenintegrität

Arbeiten nach dem Event - Condition - Action Modell:

- beim Eintreffen eines bestimmten Ereignisses (event)
- unter bestimmten Bedingungen (condition)
- werden vom DBMS Aktionen automatisch ausgeführt (action).

Achtung: zyklische Trigger oder Endlosschleifen!

Anwendungsgebiete

Trigger finden Verwendung bei:

- Berechnung gespeicherter abgeleiteter Attribute

Beispiel

Berechnung des Bruttopreises bei gespeichertem Nettopreis und Umsatzsteuersatz

- allgemeinen Bedingungen

Beispiele

Alle Mitarbeiter, die mehr als 5 SWS Vorlesungen abhalten, bekommen einen Gehaltsbonus (+5%).
Professoren können nicht degradiert werden

Syntax

- Definition des Triggers: `create trigger`
- Auslösende Ereignisse: `insert`, `update`, `delete`
- "feuert" vor oder nach Ausführung: `before`/`after insert`
- auf Tupel- bzw. Queryebene: `for each row`/`statement`
- einschränkende Bedingungen mittels `when`
- kann auf die Werte vor und nach dem Ereignis zugreifen: `referencing old`/`new` (nur bei `for each row`)
- Verwendung von Prozeduren in jeweiligen DBMS Syntax

Beispiele

Beispiel

Alle Mitarbeiter, die mehr als 5 SWS Vorlesungen abhalten, bekommen einen Gehaltsbonus (+5%).

```
create trigger viellehre
after insert on Vorlesungen
referencing new as vo_neu
for each row
when group by vo_neu.gelesenvon
     having sum(Vorlesungen.SWS) > 5 and
            sum(Vorlesungen.SWS) - vo_neu.SWS <= 5
update Professoren
set Professoren.gehalt = Professoren.gehalt * 1.05
where Professoren.PersNr = vo_neu.gelesenvon;
```

Beispiele

Beispiel

Professoren können nicht degradiert werden (ORACLE Syntax in Prozedur)

```
create trigger keineDegradierung
before update on Professoren
for each row
when (old.Rang is not null)
begin
 if :old.Rang = 'C3' and :new.Rang = 'C2'
    then :new.Rang :='C3';
    end if;
 if :old.Rang = 'C4' then :new.Rang := 'C4';
    end if;
 if :new.Rang is null then :new.Rang := :old.Rang;
    end if;
end
```