6. Datenintegrität

Integritätsbedingungen

- dienen zur Einschränkung der Datenbankzustände auf diejenigen, die es in der realen Welt tatsächlich gibt.
- sind aus dem erstellten Datenmodell ableitbar (semantisch) und können deshalb bei der Erstellung des Schemas bereits angegeben werden.
  - Konsistenzbedingungen werden nur einmal angegeben
  - AWPs sind befreit von der lästigen Überprüfung der Bedingungen
  - z. T. kann bei einer Masseneingabe aus Effizienzgründen auf eine Prüfung der Bedingungen verzichtet werden
- **statische** Bedingungen
  - sind definiert bzgl. Datenbankzuständen
- **dynamische** Bedingungen
  - sind definiert bzgl. Zustandsänderungen

6.1 Statische Integritätsbedingungen

Wertebereich-Bedingungen

- Festlegung von Ober- und Untergrenzen bei Attributen
  - Im vollen Standard von SQL'92 (wird nicht von Oracle unetrstützt) kann man explizit Wertebereiche definieren
- Aufzählung von Werten
  - Solche Aufzählungstypen lassen sich über eine eigene Relation mit einem Attribut modellieren.
- Nullwerte
  create table Kunde (KName char(20) *not null*, …)
  - Werte des entsprechenden Attribute müssen bei jedem Tupel vorliegen.

Tupel-Bedingungen

- Einschränkung der Werte, die ein Tupel bzgl. seiner verschiedenen Attribute annehmen darf.
  - Z. B. sollte in einer Relation Belegung, in der ein Tupel einen Platz von einem Bahnhof X zu einem Bahnhof Y reserviert, X ungleich Y sein.
  - In SQL kann eine Tuepl-Bedingung durch die check-Klausel beim Anlegen des Schemas angegeben werden.

Relationenbedingungen

- beziehen sich auf die Menge aller Tupel einer Relation:
  - Primärschlüssel und Schlüsselkandidaten
    In SQL kann diese Bedingung durch unique bzw. primary key bei dem Anlegen des Schemas angegeben werden.
  - Aggregat-Bedingungen
    Ein Aggregat soll über bzw. unter einer vorgebenen Schranke liegen.
  - Rekursive Bedingungen:
    In einer Relation Zugverbindung sollte jeder Bahnhof mit jedem anderen Bahnhof verbunden sein.

Referentielle Bedingungen

- beziehen sich auf mehrere Relationen

6.2 Referentielle Integrität

- sind Bedingungen an Relationen, die (insbesondere) eine Beziehung modellieren
- Seien R_1 und R_2 Relationen mit dem Schema RS_1 und RS_2. Sei K ($\subseteq$ RS_1). Primärschlüssel von R_1. Dann wird F $\subseteq$ RS_2 Fremdschlüssel von R2 genannt, falls zu jedem Datensatz s aus der Relation R_2 eine der folgenden Bedingungen gilt:
  - es gilt s[F] = NULL
  - es gibt einen Datensatz r aus R_1 gibt, so daß s[F] = r[K] gilt.
- Beispiel:
  - Relation Kunde
  - Relation Ware
  - Relation Bestellt (= m:n-Beziehung zwischen Kunde und Ware)

  mögliche Probleme, wenn referentielle Integrität nicht erfüllt ist:
  - Kunde bestellt eine Ware, die es nicht gibt.
  - Waren können von einem Kunden bestellt werden, der nicht existiert.

Einhaltung referentieller Integrität

- in einer Relation, die eine Beziehung modelliert, sollte gewährleistet sein, daß die Fremdschlüssel mit Werten belegt sind.
- relationale Algebra:
  - Relation R mit Primärschlüssel K
  - Relation S mit Fremdschlüssel F (bezieht sich auf K)

$$\pi_F(S) \subseteq \pi_K(R)$$

- erlaubte Änderungen
  - Einfügen eines Tupels s in S, wenn $s[F] \in \pi_K(R)$
  - Verändern eines Attributwerts eines Tupels s, wenn …
  - Verändern von $r[K]$ eines Tupels r, wenn $\sigma_{F = r[K]}(S) = \varnothing$
  - Löschen eines Tupels r aus R, wenn …

Referentielle Integrität in SQL

- folgende Möglichkeiten werden derzeit in ORACE angeboten:
  - Primärschlüssel: primary key
  - Fremdschlüssel: foreign key
- Beispiele:

  KNr muß bereits vorher als Primärschlüssel oder Schlüsselkandidat gekennzeichnet worden sein.

  - create relation Bestellt(KNr int,,…,
    foreign key(KNr) references Kunde(Knr))

  oder

  - create relation Bestellt(KNr int,,…,
    constraint test foreign key(KNr) references Kunde(Knr))

    In diesem Fall wird die Bedingung mit einem Namen vesehen, was insbesondere die Felxibilität bei Änderungsoperationen erhöht.

Löschoperationen

- Löschen eines Tupels r aus einer Relation R ist i.a. nicht möglich, falls es noch Tupel aus anderen Relationen gibt, die über einen Fremdschlüssel an r gebunden sind.

kaskadierendes Löschen

- Wenn ein Tupel r aus einer Relation R gelöscht wird, können auch Datensätze aus anderen Relationen automatisch gelöscht werden, die sich über einen Fremdschlüssel auf das Tupel r beziehen.
- bei der Definition des Fremdschlüssels kann diese Einstellung mitaufgenommen werden:

  create relation Bestellt(KNr NUMBER
  constraint fk_kunde references kunde(KNr) on delete cascade, …)

im SQL'92-Standard gibt es auch noch folgende Optionen:

- on delete set null
  - Es werden dann beim Löschen das entsprechende Attribut auf null gesetzt.
- on update cascade und on update set null

6.3 Verwaltung von Integritätsbedingungen

- Integritätsbedingungen können in SQL durch eine constraint implementiert werden.

Hinzufügen von Integritätsbedingungen

- alter table Bestellt
  add constraint plus_const check (Preis*Anzahl < 10000)
- alter table Kunde
  add constraint name_unique unique KName
- Leider wird der Befehl alter table <Name> drop constraint <CName> in Oracle nicht angeboten.

Erzwingen und Mißachten von Integritätsbedingungen

- alter table Kunde disable constraint name_unique
- alter table Kunde enable constraint name_unique
  exceptions into Doppel_Kunde

6.4 Komplexe Integritätsbedingungen

Trigger

- allgemeiner und mächtiger Mechanismus zur Wahrung der Datenkonsistenz
- Trigger bestehen aus einem Kopf und einem PL/SQL-Block
- Im Triggerkopf stehen Vorbedingungen zur Ausführung des Blocks.

Aufbau eines Triggerkopfs

- create [or replace] trigger <name>
  - Anlegen eines neuen Triggers bzw. Verändern eines bestehenden Triggers.
- Zeitpunkt: {before | after}
  - Hier wird nun der Zeitpunkt angegeben, ob der Triggerrumpf vor oder nach der Operation, die den Trigger ausgelöst hat, ausgeführt wird.
- Trigger-Ereignis: insert or update [of <Spalte1, Spalte2,…>] or delete on <Relationenname>
  - Ein Trigger kann für ein oder mehrere Ereignisse definiert sein. Bei mehreren Ereignissen kann im Rumpf durch folgende Klausel eine Fallunterscheidung vorgenommen werden:

a) if inserting then …
b) if updating [(Spaltei1, Spaltei2,…)] then
c) if deleting then

- Trigger-Typ: [for each row]
  - **befehlsorientierte** Trigger (default) werden genau einmal vor (before) bzw. nach (after) dem jeweiligen Ereignis ausgelöst.
  - **zeilenorientierte** Trigger werden dagegen für jedes geänderte Tupel einmal aufgerufen. Man hat dann im Rumpf die Möglichkeit (und nur diese) den alten bzw. neuen Wert der Tupel der Relation über *:neu* bzw. *:old* anzusprechen (bei insert nur den *:new* und bei delete nur *:old*). Ein anderer Zugriff auf die Relation ist nicht mehr möglich (auch wenn diese im zugehörigen Block angesprochen würde).
- Trigger-Restriktion: when <Prädikat>
  - Es können hier Bedingungen formuliert, welche die Ausführung des Rumpfs auslösen.
  - Hier kann man bei zeilenorientierten Trigger über *new* bzw. *old* sich auf die neuen bzw. alten Tupel der Relation beziehen.

Triggerrumpf

- Dieser besteht aus einem PL/SQL-Block, wobei zusätzlich die oben genannten Erweiterungen genutzt werden können.

Beispiel

- Der folgende Trigger protokolliert die Änderungen am Attribut *Sal* der Relation *Personen*:

```
create or replace trigger storeSalary
before update on Personen
for each row when (old.Sal  > 1500)
begin
        insert into diff values (:old.Sal, :new.Sal, sysdate);
end;
```

- Es soll nun beim Einfügen kontrolliert werden, daß die Gehaltserhöhung bei Personen mit einem hohen Gehalt entfällt:

```
create trigger testSalary
before update on Personen
when (new.stufe > 3000)
begin
        :new.Sal = :old.Sal;            // Zuweisung nur möglich bei before update
end;
```

Probleme bei der Verwendung von Triggern:

- Anwender muß die Widerspruchsfreiheit der Trigger kontrollieren
- Durch ein Trigger kann wieder ein Trigger ausgelöst werden. Dabei sind Zyklen immer zu vermeiden.
- Terminierung der Ereignisse

Ratschlag

- Wenn immer Konsistenzbedingungen durch constraints formuliert werden können, sollten keine Trigger benutzt werden.

6.5 Aktive Datenbanken

- Erweiterung des Trigger-Konzepts
- ECA-Regeln

  Event:
  - auslösendes Ereignis einer Aktion
  - Ereignisse sind dabei nicht nur auf die Datenbank beschränkt: Zeitereignisse, Anwendungsereignisse

  Condition:
  - Bedingung zum Auslösen der Aktion
  - Bedingungen sind nicht nur durch Datenbankanfragen definiert

  Action:
  - Angabe der Aktion (i.a. Folge von Datenbankoperationen)
- mögliche Anwendungen:
  - innerhalb eines heterogenen Informationssystems
  - Echtzeitproblemstellungen (teilweise möglich aber problematisch)