Kapitel 9: Integritätssicherung

9.1 Integritätsbedingungen

- **Constraints und Assertions**
- **Triggers**

9.2 Views

9.3 Zugriffskontrolle

Ziel: Integritätsregeln in DB verankern

→ effektivere Integritätssicherung

→ einfachere Anwendungsprogrammierung

Typen von Integritätsbedingungen

- Statische Integritätsbedingungen
  - datenmodellinhärente Invarianten
    - Primärschlüsselbedingung, Fremdschlüsselbedingung
  - anwendungsspezifische Invarianten
    - für ein Attribut eines Tupels
    - für ein Tupel
    - für mehrere Tupel einer Relation
    - für mehrere Relationen
- Dynamische Integritätsbedingungen

Zeitpunkt der Prüfung:

- am Ende einer SQL-Anweisung
- am Ende einer Transaktion

Reaktion auf Integritätsverletzungen:

- Nichtausführung bzw. Rückgängigmachen der Anweisung
- Abbruch der Transaktion
- Ausführung von Folgeänderungen

Beispiele

Statische Bedingungen:

1) Der Rabatt eines Kunden darf nicht über 50 % liegen.
2) Der Rabatt eines ausländischen Kunden darf nicht über 30 % liegen.
   (Annahme: Es gibt ein zusätzliches Kundenattribut Land.)
3) Der durchschnittliche Rabatt aller Kunden darf 30 % nicht überschreiten.
4) Der Gesamtwert aller Produkte im selben Lager darf 1 Mio. DM nicht überschreiten.
5) Es muß mindestens ein Produkt geben.
6) Die Rechnungssumme einer Bestellung ist das Produkt von Preis und bestellter Menge des bestellten Produkts abzüglich des Kundenrabatts.
7) Der Saldo eines Kunden ist die (negative) Summe der Rechnungssummen aller noch nicht bezahlten Bestellungen des Kunden.

Dynamische Bedingungen:

8) Der Rabatt eines Kunden darf nie reduziert werden.
9) Der Rabatt eines Kunden darf in 1 Jahr um max. 10 % erhöht werden.
10) Von Kunden, deren Saldo unter - 100000 DM liegt, werden keine Bestellungen mehr angenommen.
11) Der Status einer Bestellung darf sich nur in "geliefert" ändern, der Status einer gelieferten Bestellung nur in "bezahlt". Der Status einer bezahlten Bestellung darf sich nie mehr ändern.

9.1 Ausdrucksmittel zur Spezifikation von Integritätsbedingungen

- Constraints beim Create Table
- Create Assertion
- Create Trigger

} nur statische Integritätsbedingungen

Syntax von Constraints und Assertions

```
CREATE TABLE tablename
      ( colname datatype [DEFAULT {defaultconst | NULL}]
          [colconstraint {, colconstraint …}],
      {, colname datatype [DEFAULT {defaultconst | NULL}]
          [colconstraint {, colconstraint …}] …} )
      [tabconstraint {, tabconstraint …}]
colconstraint ::= NOT NULL |
      [CONSTRAINT constrname]
          { UNIQUE | PRIMARY KEY | CHECK (searchcond) |
            REFERENCES  tablename [(colname)] […]  }
tabconstraint ::=
      [CONSTRAINT constrname]
          { UNIQUE colname {, colname …} |
             PRIMARY KEY colname {, colname …} |
             CHECK (searchcond) |
             FOREIGN KEY colname {, colname …}
              REFERENCES  tablename [(colname)] […]  }
CREATE ASSERTION assertname CHECK (searchcond)
           [ INITIALLY {DEFERRED | IMMEDIATE} ]
```

Semantik von Constraints und Assertions

Semantik von CHECK (searchcond)
mit sql2trc‘[searchcond] = F(t1, ..., tn) mit freien Variablen t1, ..., tn:

$$\forall t1 \ldots \forall tn \; (\; (\; t1 \in R1 \wedge \ldots \wedge tn \in Rn \;) \Rightarrow F(t1, \ldots, tn) \;) \quad \Big\} \; Ij$$

Semantik der Datenbank aus logischer Sicht: $H \wedge I1 \wedge \ldots \wedge Im$
mit elementarer Formel (Fakten) H und Integritätsbed. I1, ..., Im

Mögliche Implementierung:
bei potentieller Verletzung von Ij Ausführen von

SELECT t1, ..., tn FROM R1, ..., Rn WHERE NOT searchcond

und Test auf Leerheit des Resultats

Zeitpunkt der Integritätsprüfung:
nach der SQL-Anweisung (... IMMEDIATE) oder
am Ende der Transaktion (... DEFERRED)

Reaktion bei Integritätsverletzung: Rollback

Beispiele: Constraints und Assertions

Bedingungen 1, 2, 3:

```
CREATE TABLE Kunden ( ... Rabatt FLOAT
    CONSTRAINT Rabattbedingung CHECK (Rabatt BETWEEN 0.0 AND 0.5)
    CONSTRAINT Auslandsrabattbedingung CHECK (Land = 'D' OR Rabatt <= 0.3), ...,
    CONSTRAINT Durchschnittsrabattbedingung CHECK
                    (0.3 >= (SELECT AVG(Rabatt) FROM Kunden))  )
```

Bedingung 4:

```
CREATE TABLE Produkte ( ...,
    CONSTRAINT Lagerwertbedingung CHECK (1000000.0 >=ALL
                    (SELECT SUM(Vorrat*Preis) FROM Produkte GROUP BY Lager))  )
```

Bedingung 5:

```
CREATE ASSERTION Produktexistenzbedingung CHECK
    (EXISTS (SELECT * FROM Produkte) )
```

Bedingung 6:

```
CREATE ASSERTION Rechnungssummenbedingung CHECK (Bestellungen.Summe =
    (SELECT B.Menge * Produkte.Preis * (1.0 - Kunden.Rabatt)
    FROM Bestellungen B, Produkte, Kunden WHERE B.PNr=Produkte.PNr
    AND B.KNr=Kunden.KNr AND B.BestNr=Bestellungen.BestNr ) )
```

Bedingung 7:

```
CREATE ASSERTION Saldobedingung CHECK ( Kunden.Saldo =
    (SELECT SUM(Summe) FROM Bestellungen WHERE
    Bestellungen.KNr=Kunden.KNr AND Status <> 'bezahlt') )
```

Optionen für Fremdschlüsselbedingung

Grobsyntax:

... FOREIGN KEY (column {, column …})
REFERENCES [user .] table [(column {, column …})]
[ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]
[ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]
[INITIALLY {IMMEDIATE | DEFERRED}] …

Semantik von
CREATE TABLE R ...
FOREIGN KEY A1, …, Am REFERENCES R1 (B1), …, Rm (Bm):

$$\forall t\,(\, t \in R \Rightarrow (\,(t.A1=\omega \wedge \ldots \wedge t.Am=\omega) \vee (\exists t1 \ldots \exists tm\,(t1 \in R1 \wedge \ldots \wedge tm \in Rm \wedge t1.B1=t.A1 \wedge \ldots \wedge tm.Bm=t.Am))\,)\,)$$

Reaktion bei Integritätsverletzung:

- Zurückweisung der Löschung/Änderung (bei NO ACTION)
- Löschen/Ändern aller "abhängigen" Tupel (bei CASCADE)
- Fremdschlüssel in "abhängigen" Tupeln auf Default-/Nullwert setzen

Beispiel: Fremdschlüsselbedingung

Kunden

KNr	…
1	
2	
...	

Produkte

PNr	…
1	
2	
...	

Bestellungen

BestNr	Monat	Tag	KNr	Summe	Status
1001			1		
1002			2		
1003			1		
...					

Bestellposten

BestNr	PNr	Menge
1001	1	
1002	2	
1003	1	
1003	2	
...		

```
CREATE TABLE Bestellungen ( ... ,
    FOREIGN KEY KNr REFERENCES Kunden (KNr) ON DELETE SET NULL )
CREATE TABLE Bestellposten ( ... ,
    FOREIGN KEY PNr REFERENCES Produkte (PNr) ON DELETE CASCADE,
    FOREIGN KEY BestNr REFERENCES Bestellungen (BestNr)
        ON DELETE CASCADE )
```

Beispiel: Fremdschlüsselbedingung

Kunden

KNr	...
1	
2	
...	

Produkte

PNr	...
1	
2	
...	

Bestellungen

BestNr	Monat	Tag	KNr	Summe	Status
1001			1 Null		
1002			2		
1003			1 Null		
...					

Bestellposten

BestNr	PNr	Menge
1001	1 Null	
1002	2	
1003	1 Null	
1003	2	
...		

```
CREATE TABLE Bestellungen ( ... ,
    FOREIGN KEY KNr REFERENCES Kunden (KNr) ON DELETE SET NULL )
CREATE TABLE Bestellposten ( ... ,
    FOREIGN KEY PNr REFERENCES Produkte (PNr) ON DELETE CASCADE,
    FOREIGN KEY BestNr REFERENCES Bestellungen (BestNr)
        ON DELETE CASCADE )
```

Syntax und Semantik von CREATE TRIGGER

Grobsyntax:

CREATE TRIGGER trigger-name {BEFORE | AFTER }
{ DELETE | INSERT | UPDATE [OF column {, column …}] }
ON table [REFERENCING OLD AS corrvar NEW AS corrvar]
[FOR EACH ROW | FOR EACH STATEMENT]
[WHEN (condition)] (statement-sequence)

Semantik mit sql2trc‘[condition] = F:

Werte F bei spez. Ereignis aus und starte Aktion, falls F wahr

- Fall 1 (FOR EACH STATEMENT):
  F hat keine freien Variablen
- Fall 2 (FOR EACH ROW):
  F hat eine oder zwei freie Variablen – told und tnew -, die an den alten bzw. neuen Wert des jeweiligen Tupels gebunden werden

Beispiele: Trigger

Bedingung 7:

```
CREATE TRIGGER Saldoeintrag
AFTER INSERT ON Bestellungen FOR EACH ROW WHEN ( Status = 'neu' )
( UPDATE Kunden SET Saldo = Saldo - Summe
  WHERE Kunden.KNr=Bestellungen.KNr );
CREATE TRIGGER Saldoausgleich
AFTER UPDATE OF Status ON Bestellungen
REFERENCING OLD AS BOld  NEW AS BNew FOR EACH ROW
WHEN ( BNew.Status = 'bezahlt' AND BOld.Status <> 'bezahlt' )
( UPDATE Kunden SET Saldo = Saldo + Summe
  WHERE Kunden.KNr=Bestellungen.KNr )
```

Bedingung 8:

```
CREATE TRIGGER Rabattmonotonie
AFTER UPDATE OF Rabatt ON Kunden
REFERENCING OLD AS KOld  NEW AS KNew FOR EACH ROW
WHEN ( KNew.Rabatt < KOld.Rabatt )  ( ROLLBACK WORK )
```

Bedingung 10:

```
CREATE TRIGGER Kundensperrung
BEFORE INSERT ON Bestellungen FOR EACH ROW
WHEN ( (SELECT Saldo FROM Kunden
            WHERE Kunden.KNr=Bestellungen.KNr) < -100000.0 )
( <Fehlermeldung ausgeben>; ROLLBACK WORK )
```

Trigger zur Steuerung der “Business-Logik”

Beispiel:

```
CREATE TRIGGER Kundenbeförderung
AFTER INSERT ON Bestellungen
REFERENCING NEW AS BNew FOR EACH ROW
WHEN ( ( (SELECT SUM(Summe) FROM Bestellungen
                WHERE Bestellungen.KNr=BNew.KNr) > 100000.0 )
           AND
            ( (SELECT SUM(Summe) - BNew.Summe FROM Bestellungen
                WHERE Bestellungen.KNr=BNew.KNr) <= 100000.0 ) )
( UPDATE Kunden SET Rabatt = Rabatt + 0.05
      WHERE Kunden.KNr=BNew.KNr;
  INSERT INTO Stammkunden
      SELECT * FROM Kunden WHERE Kunden.KNr=BNew.KNr )
```

Aber Achtung:
Die Effekte von Triggern, die selbst andere Trigger in bestimmten Reihenfolge „feuern“, sind u.U. sehr schwer vorhersagbar!

ECA-Regeln für Aktive Datenbanken

on event **if** condition **do** action
(mit allgemeineren Formen von Events und Actions)

Beispiele:

1) Auffüllen des Lagers für knappe Produkte automatisch initiieren:
   on after update Produkte.Vorrat
   if Produkte.Vorrat < 100
   do execute LagerAuffüllen(...)
2) Werbebriefe drucken für Kunden,
   die seit einem Jahr nichts mehr bestellt haben:
   on 0:00 daily **do execute** JunkMail(...)
3) Verschicken eines Mahnbriefs an Kunden,
   die dreimal hintereinander die Zahlungsfrist überschreiten.
4) Ausgabe einer Meldung, wenn der Kurs einer Aktie
   innerhalb der letzten 5 Minuten um mehr als 50% variiert.

9.2 Views (Sichten, Virtuelle Relationen)

simplifizieren Integritätsssicherung, Anfragen und Schemaänderungen

Grobsyntax:

CREATE VIEW view-name [(column {, column …})]
AS select-block [WITH CHECK OPTION]

Beispiel:

gespeicherte Relationen:

Kunden (KNr, Name, Stadt, Rabatt)

Bestellungen (BestNr, Monat, Tag, KNr, PNr, Menge, Summe, Status)

zusätzliche View:

```
CREATE VIEW KundenInfo ( KNr, Name, Stadt, Rabatt, Saldo ) AS
SELECT K.KNr, Name,Stadt,Rabatt, SUM(Summe) FROM Kunden K, Bestellungen
WHERE K.KNr = Bestellungen.KNr AND Status <> 'bezahlt'
GROUP BY K.KNr, Name, Stadt, Rabatt
```

Abfrage des Saldos:

```
SELECT Saldo FROM KundenInfo WHERE KNr=1
```

oder z.B. auch:

```
SELECT Saldo FROM KundenInfo, Bestellungen
WHERE KundenInfo.KNr=Bestellungen.KNr AND BestNr=1001
```

Views auf Views

1) CREATE VIEW BestellungsInfo
   (BestNr, Monat, Tag, KNr, Kundenname, Rabatt,
   PNr, Produktbez, Menge, Summe, Status) AS
   SELECT BestNr, Monat, Tag, Kunden.KNr, Name, Rabatt,
   Produkte.PNr, Bez, Menge, Summe, Status
   FROM Bestellungen, Kunden, Produkte
   WHERE Bestellungen.KNr=Kunden.KNr
   AND Bestellungen.PNr=Produkte.PNr

2) CREATE VIEW SuperBestellungsInfo AS
   SELECT * FROM BestellungsInfo
   WHERE Summe > 10000.0

Semantik von Views

Für CREATE VIEW VIRT AS viewquery ist
sql2ra [SELECT A1, …, Am
FROM TAB1, …, TABk, VIRT
WHERE F]
= π[A1, ..., Am] (sql2ra' [F] (TAB1 $\times$ … $\times$ TABk $\times$ sql2ra[viewquery]))

Beispiel:
sql2ra [SELECT * FROM SuperBestellungsInfo WHERE Rabatt>0.3]
= σ[Rabatt>0.3] (SuperBestellungsInfo)
= σ[Rabatt>0.3] (σ[Summe>10000.0] (BestellungsInfo))
= σ[Rabatt>0.3] (σ[Summe>10000.0]
(π[BestNr, ...] (Kunden |x| Bestellungen |x| Produkte)))

Einfachere Queries mit Views

Beispiel:
Prüfungen (Fach, Student, Note)
Welches ist das Fach mit der besten Durchschnittsnote?

a)
```
SELECT Fach FROM Prüfungen GROUP BY Fach
  HAVING AVG(Note) <= ALL
                        ( SELECT AVG(Note) FROM Prüfungen
                          GROUP BY Fach )
```

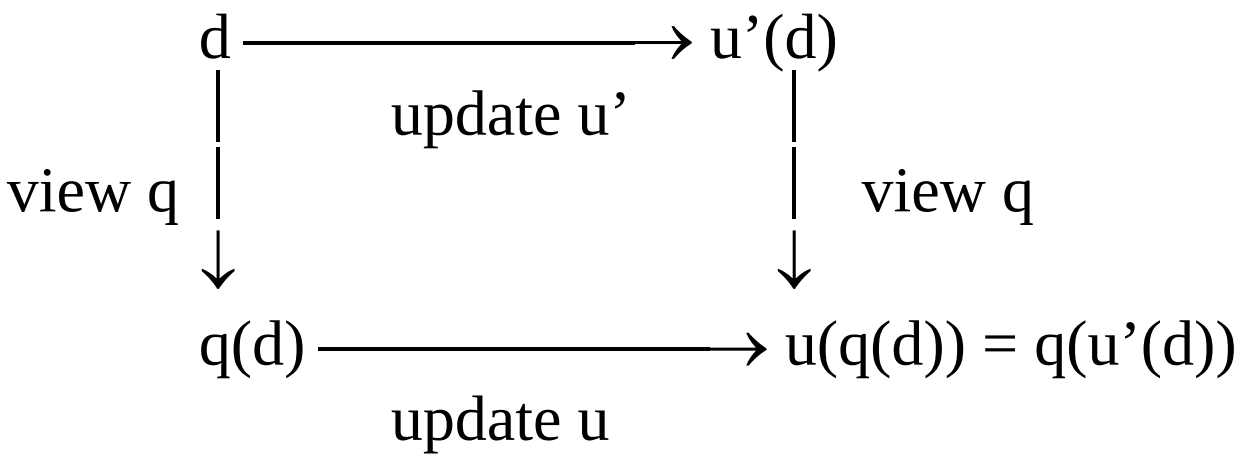
oder einfacher:

b)
```
CREATE VIEW Fachstatistik AS
      SELECT Fach, AVG(Note) AS Durchschnitt FROM Prüfungen
      GROUP BY Fach;
  SELECT Fach FROM Fachstatistik
  WHERE Durchschnitt =
            (SELECT MIN(Durchschnitt) FROM Fachstatistik )
```

Updates auf Views

Sei D die Menge aller Datenbanken.
Views und Updates sind partielle Funktionen q: D $\rightarrow$ D, u: D $\rightarrow$ D.

Eine View q heißt *änderbar* genau dann,
wenn es für jede Datenbank d, auf der q definiert ist,
und jeden auf q(d) definierten Update u
einen eindeutigen Update u' gibt,
der auf d definiert ist und für den **u(q(d)) = q(u'(d))** gilt.



Beispiele: View-Updates

```
CREATE VIEW KundenInfo ( KNr, Name, Stadt, Rabatt, Saldo ) AS
    SELECT K.KNr, Name,Stadt,Rabatt, SUM(Summe) FROM Kunden K, Bestellungen
    WHERE Kunden.KNr = Bestellungen.KNr AND Status <> 'bezahlt'
    GROUP BY KNr, Name, Stadt, Rabatt;

CREATE VIEW BestellungsInfo (BestNr, Monat, Tag, KNr, Kundenname, Rabatt,
    PNr, Produktbez, Menge, Summe, Status)  AS
    SELECT BestNr, Monat, Tag, Kunden.KNr, Name, Rabatt, Produkte.PNr, Bez,
                Menge, Summe, Status FROM Bestellungen, Kunden, Produkte
    WHERE Bestellungen.KNr=Kunden.KNr AND Bestellungen.PNr=Produkte.PNr
```

1) `UPDATE KundenInfo SET Stadt='Homburg' WHERE KNr=1`
2) `UPDATE BestellungsInfo SET Produktbez = 'Druckerpapier'  WHERE Monat=11`
3) `UPDATE BestellungsInfo SET Produktbez = 'Druckerpapier' WHERE PNr=1`
4) `UPDATE BestellungsInfo SET Produktbez = 'Druckerpapier' WHERE BestNr=1`
5)
```
CREATE VIEW BestellungsKurzInfo (BestNr, Kundenname,Produktbez, Menge) AS
   SELECT BestNr, Name, Bez, Menge FROM Kunden, Bestellungen, Produkte
   WHERE Bestellungen.KNr=Kunden.KNr AND Bestellungen.PNr=Produkte.PNr;
   INSERT INTO BestellungsKurzInfo VALUES (1111, 'Hempel', 'Maus', 5)
```
6) `UPDATE KundenInfo SET Saldo = Saldo + 1000.0 WHERE KNr=1`

Views bei Schemaänderungen

bisherige Anfrage:

```
SELECT * FROM Kunden WHERE KNr=1
```

Schemaänderung (plus Datenbank aktualisieren oder neu laden):

```
1) ALTER TABLE Kunden
     ADD Vorname CHAR(20), Nachname CHAR(20)
2) UPDATE Kunden  SET Vorname = SUBSTR (Name, ...),
                                Nachname = SUBSTR (Name, ...)
3) ALTER TABLE Kunden DROP Name
```

Vorgehen zur "Bewahrung" der bisherigen Anfrage:

```
A) Kunden in KundenDaten umbenennen
B) CREATE VIEW Kunden (KNr, Name, Stadt, Rabatt, Saldo) AS
     SELECT KNr, Vorname || ' ' || Nachname, Stadt, Rabatt, Saldo
     FROM KundenDaten
```

9.3 Zugriffskontrolle

Begriffsklärung:

- Datenschutz (engl.: data privacy):
  Einschränkungen bei Speicherung und Verarbeitung "kritischer" Daten
- Zugriffskontrolle / Autorisierung (engl.: data security, authorization):
  Verhinderung von unbefugten Zugriffen auf gespeicherte Daten

Maßnahmen der Zugriffskontrolle:

1) Organisatorische Maßnahmen
   (Rechner-/Netzzugang)
2) Technische Maßnahmen
   (Verschlüsselung, Krypto-Protokolle)
3) Maßnahmen des Betriebssystems
   (Firewalls, File-/Prozessisolation)
4) Authentikation des (DB-) Benutzers
5) Prüfung der Zugriffsrechte des DB-Benutzers beim Zugriff auf Daten

Rechtevergabe in SQL

Prinzipien:
Subjekte haben *Rechte* (zur Ausführung von Operationen) auf *Objekten*.
Der Erzeuger einer Tabelle ist deren Eigentümer.
Rechte sind minimal, d.h. beschränkt auf den Eigentümer und Subjekte, denen explizit Rechte erteilt wurden.

Vergabe von Rechten mit der GRANT-Anweisung in SQL.
Grobsyntax:

GRANT { ALL | privilege {, privilege ...} } ON { table | view }
TO { PUBLIC | user {, user …} } [WITH GRANT OPTION]

Mögliche Rechte sind:

SELECT	lesender Zugriff auf eine Relation
INSERT	Einfügen in eine Relation
UPDATE	Ändern von Tupeln einer Relation (ggf. nur bestimmte Attribute)
DELETE	Löschen von Tupeln einer Relation
CONNECT	Verbindung zum DBS aufnehmen ("Login"-Recht)
RESOURCE	Anlegen neuer Relationen (ggf. mit Limit für den Plattenplatz)
DBA	Datenbankadministration (z.B. Aufruf von Dienstprogrammen)
EXECUTE	Ausführung eines Anwendungsprogramms
IO_LIMIT	Beschränkung des Ressourcenverbrauchs für SQL-Anweisungen

Beispiele: Zugriffskontrolle mit SQL

1) Meier hat das SELECT-Recht auf Bestellungen.
   GRANT SELECT ON Bestellungen TO Meier
2) Meier hat das Recht zur Ausführung des Programms Lieferung.
   GRANT EXECUTE Lieferung TO Meier
3) Programm Lieferung hat das UPDATE-Recht auf Bestellungen.
   GRANT UPDATE Status ON Bestellungen TO Lieferung
4) Meier hat das Recht zum Lesen der Kundendaten der Stadt Homburg.
   CREATE VIEW KundenHOM AS SELECT * FROM Kunden
   WHERE Stadt='Homburg‘;
   GRANT SELECT ON KundenHOM TO Meier

5) Meier sei der Eigentümer der Relation MeierTabelle

Meier: GRANT SELECT ON MeierTabelle TO Schmid WITH GRANT OPTION
Schmid: GRANT SELECT ON MeierTabelle TO Schmitz WITH GRANT OPTION
Meier: REVOKE SELECT ON MeierTabelle FROM Schmid
Schmitz: GRANT SELECT ON MeierTabelle TO Schmid
REVOKE zieht (transitiv) auch das Recht von Schmitz zurück