4 Datenintegrität und Datenschutz

4.1 Begriffsklärung

Unterschiedliche Bedeutungen für Begriff 'Integrität'

1. **semantische Integrität** (auch Konsistenz)

   Problem: Verletzung der logischen Korrektheit der Daten durch DB-Änderungen berechtigter Benutzer

   → Integrität(süberwachung) im engeren Sinne

2. **Zugriffsintegrität**

   Problem: Zugriffe unberechtigter Benutzer → Datenschutz

3. **operationale oder Ablauf-Integrität**

   Problem: konkurrierender Zugriff mehrerer Benutzer → Synchronisation von Transaktionen

4. **physische Integrität**

   Problem: Zerstörung von Daten durch technische Fehler → Datensicherung

In englischsprachiger Literatur unterschiedlich bezeichnet

1. integrity
2. security
3. concurrency control
4. recovery

Beispiele für Integritätsregeln zur KAL-Datenbank

KUNDE(KName,KAdr,Kto)

AUF(KName,Ware,Menge)

LIEF(LName,LAdr,Ware,Preis)

(1) Kein Kundenkonto darf unter -100 absinken; betrifft 1 Relation, alle Tupel (einzeln)

(2) Das Konto von Weiss darf nicht überzogen werden; betrifft 1 Relation, 1 Tupel

(3) Kunden sind durch ihre Namen identifiziert, d.h. kein Kundenname darf mehrfach vorkommen; betrifft 1 Relation, alle Tupel (paarweise)

(4) Der Durchschnittspreis für Karotten muss unter dem für Spargel liegen; betrifft 1 Relation, mehrere Tupel (2 Gruppen)

(5) Nur solche Waren dürfen in Auftrag gegeben werden, für die es mindestens einen Lieferanten gibt; betrifft mehrere Relationen

oben statischen IRen (betreffen 1 DB-Zustand), nun dynamische IRen (betreffen mehr als einen DB-Zustand)

(6) Der Brotpreis darf nicht erhöht werden

betrifft DB-Zustandsübergang

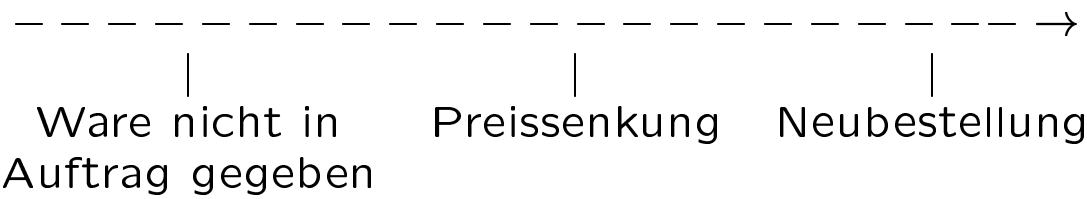
(7) Kunden dürfen nur gelöscht werden, wenn sie keine Aufträge haben

(8) Kunden müssen gelöscht werden, wenn sie keine Aufträge haben

Unterschied in den Formulierungen: ‘müssen gelöscht werden’ verschärft ‘dürfen nur gelöscht werden, wenn ...’

(9) Jeder Lieferant muss den Preis einer Ware, die nicht mehr in Auftrag gegeben ist, irgendwann senken, bevor sie wieder bestellt werden kann

betrifft sogar ganze Zeiträume; ‘Gesetze des Marktes’



Beispiele für temporale IBen

- wer einmal verheiratet ist, kann nie wieder ledig werden (nur geschieden oder verwitwet)

- jeder mehr als 3 Jahre alte PKW muss spätestens nach 2 Jahren wieder zum TÜV

- Steuererklärungen müssen spätestens im September des darauf folgenden Jahres abgegeben werden

4.2 Begriffe

Integritätsbedingung (IB) = Bedingung für die 'Zulässigkeit' ('Korrektheit') von

(i) (einzelnen) DB-Zuständen σ

(ii) DB-Änderungen, also Zustandsübergängen $\langle \sigma_{alt}, \sigma_{neu} \rangle$

(iii) ganzen DB-Verläufen, also Zustandsfolgen $\langle \sigma_{initial}, \sigma_1, \sigma_2, ..., \sigma_{aktuell}, ... \rangle$

Bezeichnungen: (i) statische IB; (ii) transitionale IB; (iii) temporale IB; (ii) und (iii) dynamische IB

Forderung: Nach jeder Transaktion soll sich die DB in einem zulässigen Zustand bzgl. aller statischen und dynamischen IBen befinden

Eine **Integritätsregel** (IR) ⟨ O,B,A,R ⟩ besteht aus:

- B Integritätsbedingung
- O Menge von Objekt(typ)en, auf die sich B bezieht
- A Auslöser, wann B zu überprüfen ist
- R Reaktion, falls B verletzt ist

- Transaktion: zu einer Einheit zusammengefasste Folge von DB-Änderungsoperationen

- Beispiel zur Notwendigkeit von Transaktionen: Bibliotheks-DB mit BUCH(Doknr,Titel,...) und DESKRIPTOR(Doknr,Stichwort)

  (IB1) Alle Bücher haben mindestens ein Stichwort

  (IB2) Dokumentnummern aus DESKRIPTOR kommen auch in BUCH vor

- mögliche Operationalisierungen des Vorgangs 'Buch einfügen'

- Reihenfolge 1

```
insert into BUCH values (...) /* IB1 verletzt */
insert into DESKRIPTOR values (...)
```

- Reihenfolge 2

```
insert into DESKRIPTOR values (...) /* IB2 verletzt */
insert into BUCH values (...)
```

- Lösung: Operationsfolge explizit abschliessen

```
insert into DESKRIPTOR values (...)
insert into BUCH values (...)
commit
```

- Integritätsbedingungen nur nach abgeschlossenen Transaktionen prüfen; über Transaktionskonzept konsistente DB-Zustände und Übergänge dazwischen definierbar

4.3 Integritätsregeln in Datenbanksprachen

Integritätsregeln in SQL

Syntax: Vorschlag von 'System R' (1976)

grösstenteils so standardisiert, teilweise mit geringfügig anderer Syntax

assert ⟨Regelname⟩

[immediate | deferred] [on ⟨Operation⟩$^+$] (A)

[for ⟨Relation⟩$^+$] (O)

⟨Bedingung wie in where-Klausel⟩ (O,B)

[else] (⟨Folge von SQL-Anweisungen⟩) (R)

⟨Operation⟩ ::= { insertion | deletion } of ⟨Relation⟩ |

update of ⟨Relation⟩ [(⟨Attribut⟩$^+$)]

Erklärungen

- (A) = Auslöser; voreingestellt: immediate
  - immediate: sofort nach jeder bzw. nur nach den angegebenen Operationen
  - deferred: erst nach einer Transaktion, sofern angegebene Operationen vorgekommen sind
- $\langle$Relation$\rangle \mathrel{\widehat{=}} \forall$-quantifizierte Tupelvariable
- $\langle$Bedingung$\rangle \mathrel{\widehat{=}}$ Formel eines erweiterten Tupelkalküls; old/new bezeichnen die geänderten Tupel vor/nach der Operation/Transaktion
- (R) = Reaktion; voreingestellt: reject; reject = Zurückweisen und Rücksetzen der Operation/Transaktion

Weiteres Konzept: Trigger

trigger ... on $\langle$Operation$\rangle$: ($\langle$Reaktion$\rangle$)

$\mathrel{\widehat{=}}$

assert ... on $\langle$Operation$\rangle$: false
else ($\langle$Reaktion$\rangle$)

Trigger = Auslöser mit Folgeaktionen

Falls $\langle$Operation$\rangle$ angesprochen, überprüfe ob Bedingung gilt; Bedingung hier false, d.h. Bedingung gilt nicht; es folgt, dass $\langle$Reaktion$\rangle$ ausgeführt wird; also wird $\langle$Reaktion$\rangle$ immer ausgeführt, wenn $\langle$Operation$\rangle$ angesprochen wird

Beispiele für Integritätsregeln in SQL

1. ```
   assert IR1
     for KUNDE: Kto>=-100
   ```

   $\forall k : KUNDE(k.Kto >= -100)$

   äquivalent aber mit expliziter Angabe der auslösenden Operationen

   ```
   assert IR1'
     on insertion of KUNDE, update of KUNDE
     for KUNDE: Kto>=-100
   ```

   oder noch genauer

   ```
   assert IR1''
     on insertion of KUNDE, update of KUNDE(Kto)
     for KUNDE: Kto>=-100
   ```

   zweite Formulierung erlaubt effizientere Implementierung, da Prüfung nur nach den angegebenen Operationen und nicht z.B. nach
   delete from KUNDE oder
   update KUNDE set KAdr ...
   ```

2. `assert IR2 for KUNDE:`
   `not (KName='Weiss') or (Kto>=0)`

   eigentlich `(KName='Weiss') implies (Kto>=0)` und unter Angabe der auslösenden Operationen

   ```
   assert IR2
     on insertion of KUNDE, update of KUNDE
     for KUNDE: (KName='Weiss') implies (Kto>=0)
   ```

   `update of R` ist äquivalent zu
   `update of R(A1), ..., update of R(An)` für
   `R(A1, ..., An)`

   ```
   assert IR2
     on insertion of KUNDE, update of KUNDE(KName),
        update of KUNDE(KAdr), update of KUNDE(Kto)
     for KUNDE: (KName='Weiss') implies (Kto>=0)
   ```

   `KAdr` kommt in der Bedingung nicht vor und braucht daher nicht betrachtet zu werden

   ```
   assert IR2
     on insertion of KUNDE, update of KUNDE(KName),
        update of KUNDE(Kto)
     for KUNDE: (KName='Weiss') implies (Kto>=0)
   ```

3.
```
assert IR3
  for KUNDE K1, KUNDE K2:
  not(K1.KName=K2.KName) or
    (K1.KAdr=K2.KAdr and K1.Kto=K2.Kto)
```

   äquivalent zur Angabe des Schlüssel; in besser lesbarer Syntax:

```
assert IR3
  for KUNDE K1, KUNDE K2:
  (K1.KName=K2.KName) implies
    (K1.KAdr=K2.KAdr and K1.Kto=K2.Kto)

assert IR3
  for KUNDE K1, KUNDE K2:
  (K1.KAdr<>K2.KAdr or K1.Kto<>K2.Kto) implies
    (K1.KName<>K2.KName)

assert IR3
  for KUNDE K1, KUNDE K2:
  (K1.KName=K2.KName) implies (K1=K2)

assert IR3
  for KUNDE K1, KUNDE K2:
  (K1<>K2) implies (K1.KName<>K2.KName)
```

   mit Angabe der Operationen die zum Anlass der Überprüfung der IB genommen werden sollen

```
assert IR3
  on insertion of KUNDE, update of KUNDE
  for KUNDE K1, KUNDE K2:
  (K1<>K2) implies (K1.KName<>K2.KName)
```

4. assert IR4 deferred:

```
(select avg (Preis) from LIEF
 where  Ware='Karotten')
<=
(select avg (Preis) from LIEF
 where  Ware='Spargel')
```

   deferred bedeutet Prüfung erfolgt erst nach Transaktion z.B. für Preisänderung für alle Waren eines Lieferanten

5. assert IR5

```
for AUF A:
exists (select * from LIEF where Ware = A.Ware)
```

   äquivalent

```
assert IR5'
  on insertion of AUF, update of AUF(Ware),
     deletion of LIEF, update of LIEF(Ware)
  for AUF A:
  exists (select * from LIEF where Ware = A.Ware)
```

6. assert IR6

```
on update of LIEF(Preis)
for LIEF: not (Ware='Brot') or
            (new Preis<=old Preis)
```

7.
```
assert IR7 deferred
  on deletion of KUNDE K:
  not exists (select * from AUF
              where  KName = old K.KName)
```

   old K bezieht sich nur auf gelöschte Tupel; deferred bedeutet, dass eine Transaktion erst einen Kunden und dann alle seine Aufträge löschen kann

8.
```
assert IR8
  on deletion of AUF A1:
  exists (select * from AUF A2
          where  A2.KName = old A1.KName)
else
  (delete from KUNDE K where K.KName = old A1.KName)
```

   - Kundenname des gerade gelöschten Auftrags kommt in der Relation AUF noch einmal vor; dann hat dieser Kunde noch einen Auftrag für eine Ware; IB erfüllt
   - Kundenname des gerade gelöschten Auftrags kommt in der Relation AUF nicht mehr vor; dann hat dieser Kunde keine Aufträge mehr; es erfolgt Löschung dieses Kunden, dessen letzter Auftrag gerade gelöscht wurde

   alternative, aber nicht äquivalente Formulierung nur als statische IB ohne aktiven Teil

```
assert IR8'
  for KUNDE K:
  exists (select * from AUF A where K.KName=A.KName)
```

9. Im ‘System R’ Vorschlag nicht formulierbar;
   Erweiterung um temporale Logik

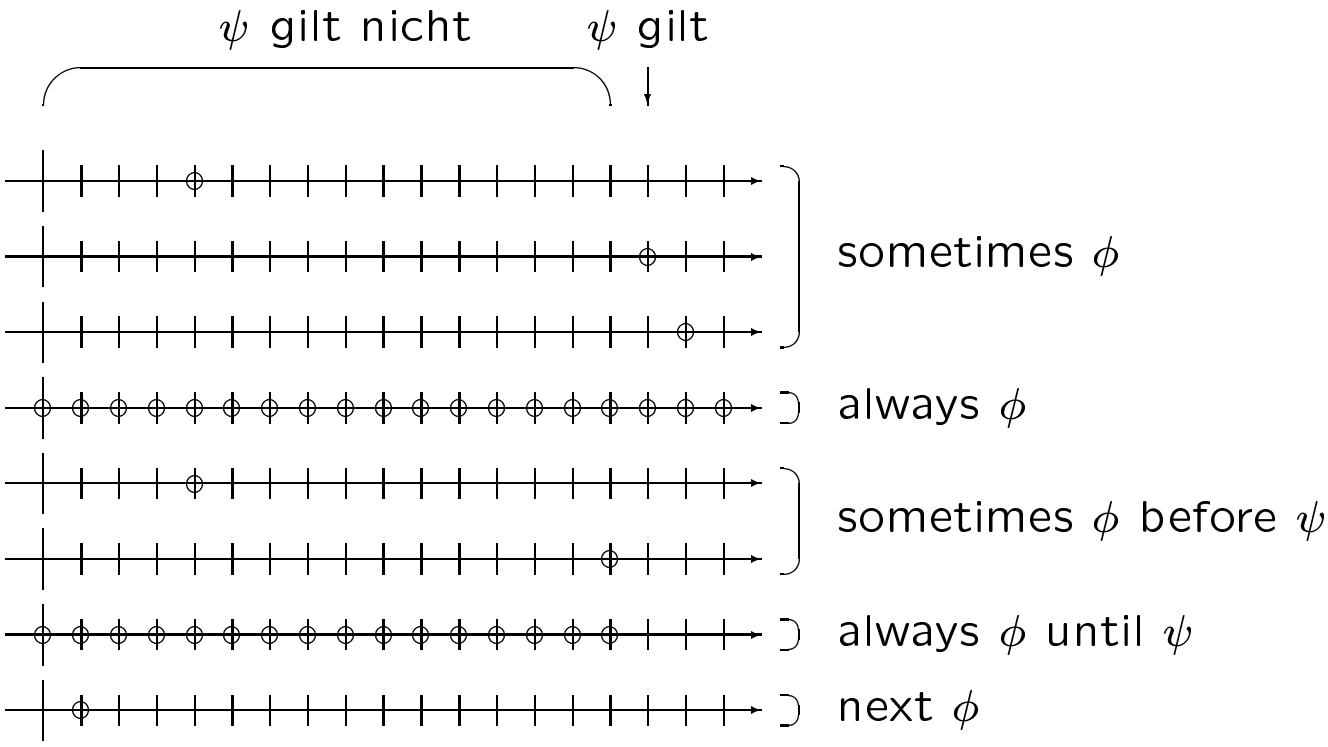
```
assert IR9
  for LIEF L:
  ( 'nicht-in-Auftrag-gegeben(L.Ware)' )
  implies
  ( sometimes
      'Preissenkung(L.LName,L.Ware)'
    before
      'Neu-Bestellung(L.Ware)' )

assert IR9
  for LIEF L
  ( L.Ware not in (select Ware from AUF) )
  implies
  ( sometimes
      ( exists ( select *
                 from   LIEF
                 where  LName=L.LName and
                        Ware=L.Ware and
                        Preis<L.Preis ) )
    before
      ( L.Ware in (select Ware from AUF) ) )
```

`L.Preis` in IR9 bezieht sich auf den ‘alten’ Preis

allgemeine Form der Quantoren der temporalen Logik

- always ϕ
- sometimes ϕ
- always ϕ until ψ
- sometimes ϕ before ψ
- next ϕ



$\circ \equiv \varphi$ gilt

- alternative Formulierung von IR8

```
trigger IR8 on deletion of AUF:
  ( delete from KUNDE K
    where not exists
            (select * from AUF where KName=K.KName) )
```

- zu Relationen KUNDE(KName, ..., AnzAuf) und AUF(KName, ...)

```
trigger Auftragszaehlung+ on insertion of AUF A:
  ( update KUNDE set AnzAuf = AnzAuf+1
    where  KName = new A.KName )

trigger Auftragszaehlung- on deletion of AUF A:
  ( update KUNDE set AnzAuf = AnzAuf-1
    where  KName = old A.KName )

trigger Auftragszaehlung-+ on update of AUF(KName):
  ( update KUNDE set AnzAuf = AnzAuf+1
    where  KName = new A.KName;
    update KUNDE set AnzAuf = AnzAuf-1
    where  KName = old A.KName )
```

Garantieren diese Trigger die folgende IB?

```
assert Auftragszaehlung deferred for KUNDE:
  AnzAuf = ( select count(*)
             from   AUF
             where  KUNDE.KName=AUF.KName )
```

Ja, falls Attribut AnzAuf korrekt initialisiert und durch andere Operationen bzw. Transaktion nicht modifiziert wird

4.4 Klassifikationen von Integritätsregeln

- nach **betroffenen Objektmengen** (O):
  einzelne Attribute versus mehrere Relationen
- nach Art der **Bedingung** (B):
  statisch versus dynamisch
- nach Art des **Auslösers** (A):
  primär (sofortig) versus sekundär (verzögert)
  operationsunabhängig versus operationsabhängig
- nach Art der **Reaktion** (R):
  passiv versus aktiv
- **Abgrenzung vom Datenmodell:**
  (modell-)inhärent versus explizit

Regeln, die sich bereits durch Konzepte des Datenmodells ausdrücken lassen oder sich daraus ergeben, heißen inhärent, alle anderen explizit

Inhärente Integritätsbedingungen des Relationenmodells

nicht von allen relationalen DBMS unterstützt

1. **Typintegrität**
   (a) Datentypen der Attribute
   (b) Zulässigkeit von Nullwerten
   jedoch explizite Einschränkungen von Wertebereichen

2. **Schlüsselintegrität**
   in frühen SQL-Implementierungen nur mittels Indexdefinition

3. **Referentielle Integrität**
   Fremdschlüssel: z.B. Attribut A in einer Relation verweist auf genau ein Tupel in einer zweiten Relation in der der Schlüssel aus Attribut A besteht

   Motivation für referentielle Integrität: ER-Modell und Relationen-Modell 'rücken näher' aneinander

   ER-inhärente IBen bzgl. Relationshiptypen können nun übersetzt werden

- 1. Typintegrität

  a) Datentypen der Attribute

  z.B. KAdr:string bzw. in SQL: KAdr varchar(50)

  b) Zulässigkeit von Nullwerten

  z.B. KName ... not null

  jedoch explizit Einschränkungen von Wertebereichen, z.B. IR1

- 2. Schlüsselintegrität

  z.B. KUNDE(KName,...)

  $\forall k_1, k_2 : KUNDE$
  $(k_1.KName = k_2.KName \Rightarrow k_1 = k_2)$

  in SQL-89 nur mittels Indexdefinition

- 3. Referentielle Integrität: Fremdschlüssel

  z.B. AUF(KName $\rightarrow$ KUNDE.KName, Ware, ...)

  $\forall\ a : AUF\ \exists\ k : KUNDE\ (a.KName = k.KName)$

  $\pi_{KName}(AUF) \subseteq \pi_{KName}(KUNDE)$

- Übersetzung von ER-Schemata in relationale DB-Schemata

  Beziehungstyp $R(E_1, ..., E_l; A_1, ..., A_m) \Rightarrow$

  Relationenschema
  $R(X_1 \rightarrow E_1.X_1, ..., X_l \rightarrow E_l.X_l; A_1, ...)$ wobei X_i Schlüssel von E_i und Pfeile Fremdschlüsselbeziehungen repräsentieren

- AUSLEIHE(BUCH, STUDENT; Datum) beinhaltet als ER-inhärente IB

  $\sigma(AUSLEIHE) \subseteq \sigma(BUCH) \times \sigma(STUDENT)$

  Übersetzung:

  AUSLEIHE (Doknr $\rightarrow$ BUCH.Doknr, Matrnr $\rightarrow$ STUDENT.Matnr; Datum)

  aus Fremdschlüsselangaben folgt

  $\pi_{Doknr,Matrnr}(AUSLEIHE) \subseteq$
  $\pi_{Doknr}(BUCH) \times \pi_{Matrnr}(STUDENT)$

- Allgemeiner Fall

  ER-Schema: E1(<u>A1</u>,A1'); E2(<u>A2</u>,A2'); R(E1,E2;A)

  Relationales DB-Schema: E1(<u>A1</u>,A1'); E2(<u>A2</u>,A2'); R(<u>A1</u>$\rightarrow$E1.A1,<u>A2</u>$\rightarrow$E2.A2,A)

4.5 Datenschutz

Datenschutz = Schutz der DB vor mißbräuchlicher Benutzung; gewährleistet Zugriffskontrolle

Generelle Schutzmaßnahmen

- physische Schutzvorrichtungen (z.B. Personenkontrolle) oder Verschlüsselung
- Benutzer-Identifikation z.B. durch Paßwörter
- Zugriffsrechte: Welche Benutzer dürfen auf welche Systemkomponenten wie zugreifen?

DB-Zugriffsrechte (Autorisierungsregeln):

- Aufbau:
  ⟨Benutzer, DB-Ausschnitt, erlaubte Operation⟩
- Verwaltung und Überwachung durch DBMS

Vergabe: (in SQL-89)

grant | all / ⟨Operation⟩$^+$ / all but ⟨Operation⟩$^+$ | on | ⟨RelName⟩ / ⟨SichtName⟩ | ...

... to | public / ⟨Benutzer⟩$^+$ | [with grant option]

⟨Operation⟩ ::=
select | insert | delete | update[(⟨Attribut⟩$^+$)] | alter

Rücknahme:

revoke ⟨Operation⟩$^+$ on ... from ⟨Benutzer⟩$^+$

Anmerkungen:

- public: alle Benutzer
- grant option: Recht auf Weitergabe von Zugriffsrechten

Voreinstellungen:

- DBA hat alle Rechte an allen Relationen
- Jeder Benutzer hat alle Rechte an 'eigenen', d.h. selbst definierten Relationen
- Rechte an Sichten ergeben sich aus Rechten an Basisrelationen

Weitergehende Differenzierung bei der Rechtevergabe für Operationen (wie z.B. nur für bestimmte Werte oder Parameter erlaubt) nicht direkt, sondern nur wieder über Sichten realisierbar

Beispiele für Rechtevergabe

- Rechte an Selektionssicht für bestimmten Kunden

```
create view WEISS-AUFTRAEGE
  as select Ware, Menge
     from   AUF
     where  KName='Weiss'
grant select, insert, update(Menge)
  on WEISS-AUFTRAEGE to 'Weiss'
```

  Weiss sieht nur seine eigenen Aufträge; Weiss darf lesen, einfügen, Menge ändern, aber nichts löschen; Operationen nicht stark differenzierbar; es lässt sich nicht formulieren, dass Menge nur erhöht oder nur erniedrigt werden kann oder dass die Mengenangabe nur z.B. in Hunderterschritten erfolgt; derartiges liesse sich über IBen realisieren

- ```
  create view KUNDENLISTE
    as select KName, KAdr
       from   KUNDE
  grant all on KUNDENLISTE
    to KAL-ADRESSVERWALTUNG
  ```

- ```
  create view GUTE-KUNDEN
    as select KName, Kto
       from   KUNDE
       where  Kto>0
  grant select on GUTE-KUNDEN
    to KAL-EMPFANG
  ```

- berechnete Sicht mit aggregierter Information mit lesendem Zugriff

```
create view BEDARF
  as select   Ware, sum(Menge)
     from     AUF
     group by Ware
grant select on BEDARF
  to KAL-LAGER
```

- ```
  revoke update(Menge)
    on WEISS-AUFTRAEGE from 'Weiss'
  ```

- Problematisch: Kombination mehrerer Sichten durch einen Benutzer; im Beispiel KUNDENLISTE und GUTE-KUNDEN; Kombination erlaubt Rückschluss auf Kunden mit nicht-positivem Kontostand unter Kenntnis des konzeptionellen Gesamtschemas

```
select KName
from   KUNDENLISTE
where  KName not in ( select KName from GUTE-KUNDEN )

KUNDENLISTE = (select KName from KUNDE)
-
GUTE-KUNDEN = (select KName from KUNDE where  Kto>0)

select KName
from   KUNDE
where  Kto<=0
```
```