

## 5. Datenkontrolle

- **Sichtkonzept**

- Semantik von Sichten
- Abbildung von Sichten
- eingeschränkte Aktualisierungsmöglichkeiten

- **Semantische Integritätskontrolle**

- bisher in SQL schwach ausgebildet (z. B. NOT NULL, UNIQUE)
- Relationale Invarianten erst in SQL2 verbindlich
- benutzerdefinierte Integritätsbedingungen (*assertions*)
- Erweiterungen in SQL99 (Trigger)

- **Das Transaktionsparadigma**

- ACID-Prinzip
- SQL-Operationen: COMMIT WORK, ROLLBACK WORK  
(Beginn einer Transaktion implizit)

## Sichtkonzept

- **Ziel: Festlegung**

- welche Daten Benutzer sehen wollen  
(*Vereinfachung, leichtere Benutzung*)
- welche Daten sie nicht sehen dürfen (*Datenschutz*)
- einer zusätzlichen Abbildung  
(*erhöhte Datenunabhängigkeit*)

- **Sicht (View):** mit Namen bezeichnete, aus Tabellen abgeleitete, virtuelle Tabelle (Anfrage)

- **Korrespondenz zum externen Schema** bei ANSI/SPARC  
(Benutzer sieht jedoch i. allg. mehrere Sichten (Views) und Tabellen)

```
CREATE VIEW view [ (column-commlist ) ]  
AS table-exp  
[WITH [ CASCADED | LOCAL] CHECK OPTION]
```

- D2: Sicht, die alle Programmierer mit einem Gehalt < 30.000 umfasst**

**CREATE VIEW**

```
Arme_Programmierer (Pnr, Name, Beruf, Gehalt, Anr)  
AS SELECT Pnr, Name, Beruf, Gehalt, Anr  
FROM Pers  
WHERE Beruf = 'Programmierer' AND Gehalt < 30 000
```

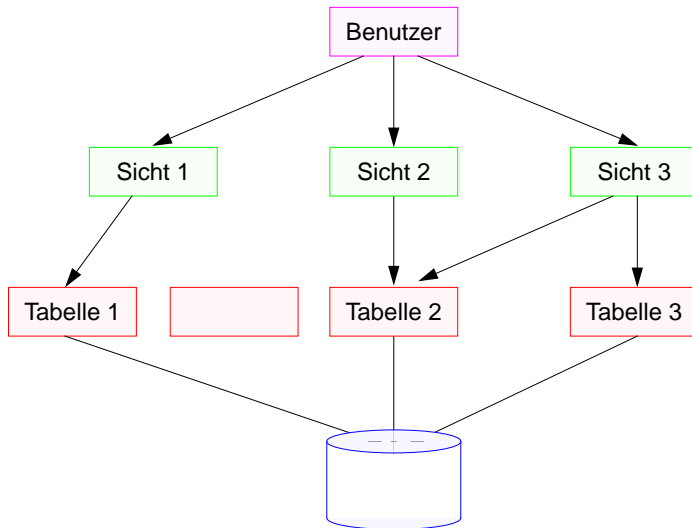
- D3: Sicht für den Datenschutz**

**CREATE VIEW** Statistik (Beruf, Gehalt)

```
AS SELECT Beruf, Gehalt  
FROM Pers
```

## Sichtkonzept (2)

- Sichten zur Gewährleistung von Datenunabhängigkeit

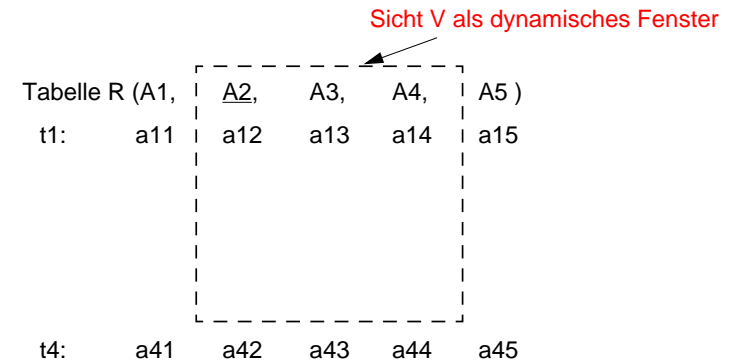


- Eigenschaften von Sichten**

- Sicht kann wie eine Tabelle behandelt werden
- Sichtsemantik: „**dynamisches Fenster**“ auf zugrundeliegende Tabellen
- Sichten auf Sichten sind möglich
- eingeschränkte Änderungen: aktualisierbare und nicht-aktualisierbare Sichten

## Sichtkonzept (3)

- Zum Aspekt: Semantik von Sichten

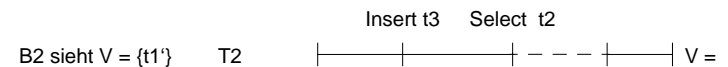
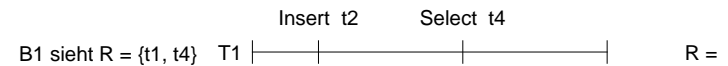


- Sichtbarkeit von Änderungen – Wann und Was?**

Wann werden welche geänderten Daten in der **Tabelle/Sicht** für die **anderen Benutzer** sichtbar?

**Vor BOT**  
von T1, T2

**Nach EOT**  
von T1, T2



## Sichtkonzept (4)

### • Abbildung von Sicht-Operationen auf Tabellen

- Sichten werden i. allg. nicht explizit und permanent gespeichert, sondern Sicht-Operationen werden in äquivalente Operationen auf Tabellen umgesetzt
- Umsetzung ist für Leseoperationen meist unproblematisch

#### Anfrage (Sichtreferenz):

```
SELECT Name, Gehalt
FROM Arme_Programmierer
WHERE Anr = 'K55'
```

#### Ersetzung durch:

```
SELECT Name, Gehalt
FROM
WHERE Anr = 'K55'
```

### • Abbildungsprozess auch über mehrere Stufen durchführbar

#### Sichtdefinitionen:

```
CREATE VIEW V AS SELECT ... FROM R WHERE P
CREATE VIEW W AS SELECT ... FROM V WHERE Q
```

#### Anfrage:

```
SELECT ... FROM W WHERE C
```

#### Ersetzung durch

```
SELECT ... FROM V WHERE Q AND C
```

und

```
SELECT ... FROM R WHERE Q AND P AND C
```

## Sichtkonzept (5)

### • Einschränkungen der Abbildungsmächtigkeit

- keine Schachtelung von Aggregat-Funktionen und Gruppenbildung (GROUP-BY)
- keine Aggregat-Funktionen in WHERE-Klausel möglich

#### Sichtdefinition:

```
CREATE VIEW Abtinfo (Anr, Gsumme) AS
SELECT Anr, SUM (Gehalt)
FROM Pers
GROUP BY Anr
```

#### Anfrage:

```
SELECT AVG (Gsumme) FROM Abtinfo
```

#### Ersetzung durch

```
SELECT
FROM Pers
GROUP BY Anr
```

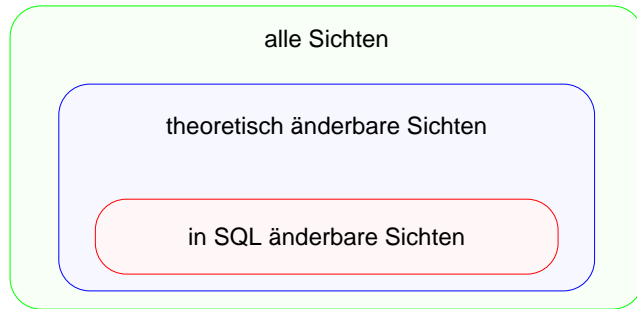
### D4: Löschen von Sichten:

```
DROP VIEW Arme_Programmierer
CASCADE
```

- Alle referenzierenden Sichtdefinitionen und Integritätsbedingungen werden mitgelöscht
- RESTRICT würde eine Löschung zurückweisen, wenn die Sicht in weiteren Sichtdefinitionen oder CHECK-Constraints referenziert werden würde.

## Sichtkonzept (6)

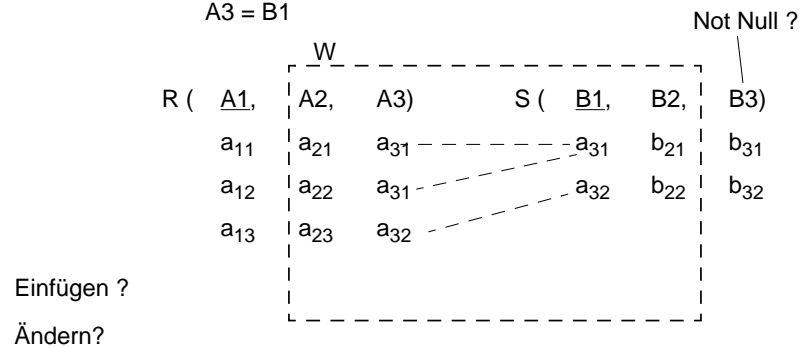
- **Änderbarkeit von Sichten**



- Sichten über mehr als eine Tabelle sind i. allg. **nicht aktualisierbar!**

$$W = \Pi_{A2,A3,B1,B2} (R \bowtie S)$$

A3 = B1



- **Änderbarkeit in SQL**

- nur eine Tabelle (Basistabelle oder Sicht)
- Schlüssel muß vorhanden sein
- keine Aggregatfunktionen, Gruppierung und Duplikateliminierung

## Integritätskontrolle

- **ZIEL<sup>1</sup>**

- Nur DB-Änderungen zulassen, die allen definierten *Constraints* entsprechen (offensichtlich 'falsche' Änderungen zurückweisen!)
- Möglichst hohe Übereinstimmung von DB-Inhalt und Miniwelt (Datenqualität)

➔ *Integritätsbedingungen der Miniwelt sind explizit bekannt zu machen, um automatische Überwachung zu ermöglichen.*

- **Konsistenz der Transaktionsverarbeitung**

- Bei COMMIT müssen alle semantischen Integritätsbedingungen erfüllt sein.
- Zentrale Spezifikation/Überwachung im DBS: „*system enforced integrity*“

- **Physische Konsistenz**

der DB ist Voraussetzung für logische Konsistenz

- Gerätekonsistenz
- Speicherkonsistenz (Speicherungsstrukturen/Zugriffspfade/Zeiger sind konsistent)

- **Logische Konsistenz**

(TA-Konsistenz)

- modellinhärente Bedingungen (z. B. Relationale Invarianten)
- benutzerdefinierte Bedingungen aus der Miniwelt

1. „**Golden Rule**“ nach C. J. Date: No update operation must ever be allowed to leave any relation or view (relvar) in a state that violates its own predicate. Likewise no update transaction must ever be allowed to leave the database in a state that violates its own predicate.

## Klassifikation semantischer Integritätsbedingungen

- **Reichweite**

	Beispiel
a) Attributbedingungen	GEB-JAHR ist numerisch, 4-stellig
b) Bedingungen bezüglich einer Satzausprägung	ABT.GEHALTSSUMME < ABT.JAHRESETAT
c) Satztypbedingungen	PNR ist eindeutig
d) Satztypübergreifende Bedingungen	ABT.GEHALTSSUMME ist Summe aller Angestelltegehälter

- **Zeitpunkt der Überprüfbarkeit**

- unverzögert (sofort bei Änderungsoperation) z.B. a)
- verzögert (am Transaktionsende)



zyklische  
Fremdschlüsselbedingungen

- **Art der Überprüfbarkeit**

- Zustandsbedingungen
- Übergangsbedingungen

Beispiele:

- Übergang von FAM-STAND von 'ledig' nach 'geschieden' ist unzulässig
- Gehalt darf nicht kleiner werden

## Integritätsbedingungen in SQL

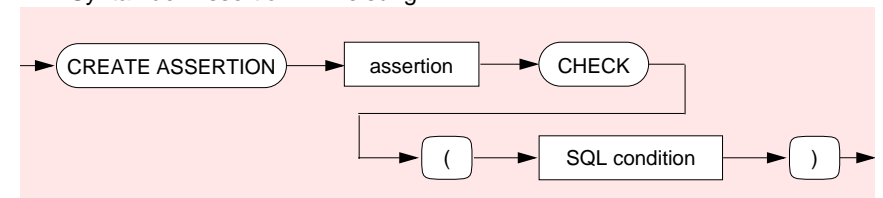
- **Bereits eingeführt** (siehe Datendefinition)

- CHECK-Bedingungen bei CREATE DOMAIN, CREATE TABLE, Attributdefinition
- UNIQUE, PRIMARY KEY, Verbot von Nullwerten
- Fremdschlüsselbedingungen (FOREIGN-KEY-Klausel)

➔ Diese Integritätsbedingungen sind an DB-Objekte gebunden

- **Allgemeine Integritätsbedingungen**

- beziehen sich typischerweise auf mehrere Relationen
- lassen sich als eigenständige DB-Objekte definieren
- erlauben die Verschiebung ihres Überprüfungszeitpunktes
- Syntax der Assertion-Anweisung



- **Beispiel**

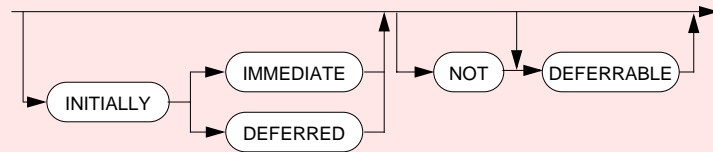
Die Relation Abt enthält ein Attribut, in dem (redundant) die Anzahl der Angestellten einer Abteilung geführt wird. Es gilt folgende Zusicherung:

```
CREATE ASSERTION A1
CHECK (NOT EXISTS
(SELECT * FROM Abt A
WHERE A.Anzahl_Angest <>
(SELECT COUNT (*) FROM Pers P
WHERE P.Anr = A.Anr)));
```

➔ Bei welchen Operationen und wann muss überprüft werden?

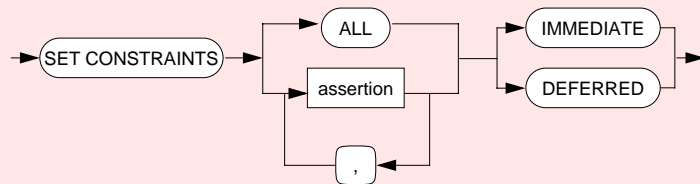
## Integritätsbedingungen in SQL (2)

### Festlegung des Überprüfungszeitpunktes:



- IMMEDIATE: am Ende der Änderungsoperation (Default)
- DEFERRED: am Transaktionsende (COMMIT)

### Überprüfung kann durch Constraint-Modus gesteuert werden



- Zuordnung gilt für die aktuelle Transaktion
- Bei benannten Constraints ist eine selektive Steuerung der Überprüfung möglich; so können gezielt Zeitpunkte vor COMMIT ausgewählt werden.

### Beispiel

IB	Zustand bei BOT
IB1	IMMEDIATE, DEFERRABLE
IB2	DEFERRED, DEFERRABLE

TA |-----|

## Beispiel-DB

Abt	Anr	Aname	Ort	Anzahl_Angest
	K51	PLANUNG	KL	1
	K53	EINKAUF	F	1
	K55	VERTRIEB	F	2

Pers	Pnr	Name	Alter	Gehalt	Anr	Mnr
	406	COY	47	50 000	K55	123
	123	MÜLLER	32	43 500	K51	-
	829	SCHMID	36	45 200	K53	777
	574	ABEL	28	30 000	K55	123

- Gehaltssumme an Abt anhängen
- Gehaltssumme mit Werten füllen
- Einfügen eines neuen Angestellten

Wann wird Constraint A2 überprüft?

```

CREATE ASSERTION A2
CHECK (NOT EXISTS
(SELECT * FROM Abt A
WHERE A.Geh_Summe <>
(SELECT SUM (P.Gehalt) FROM Pers P
WHERE P.Anr = A.Anr)))
INITIALLY DEFERRED;
  
```

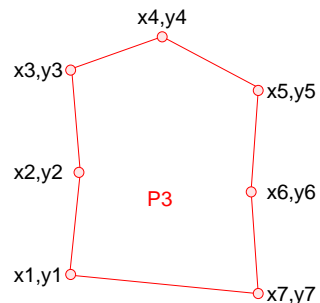
## Beispiel-DB (2)

### • Integritätsbedingungen auf einer Relation

```
CREATE TABLE Polygon
(Id CHAR(5) NOT NULL,
 PktNr INTEGER NOT NULL,
 X DECIMAL NOT NULL,
 Y DECIMAL NOT NULL,
 PRIMARY KEY (Id, PktNr));
```

- Es sollen mehrere Polygone (Id) in Relation Polygon enthalten sein.
- Die Position jedes Punktes (PktNr) in der „Liste“ der Polygon-Punkte ist erforderlich!

Polygon P3 und seine relationale Darstellung



Polygon			
Id	PktNr	X	Y
...			
P3	1	x1	y1
P3	2	x2	y2
P3	3	x3	y3
P3	4	x4	y4
P3	5	x5	y5
P3	6	x6	y6
P3	7	x7	y7
...			

### • Wie lassen sich folgende Integritätsbedingungen formulieren?

- Folgeänderungen bei „Füge neuen Eckpunkt von P3 zwischen (x2,y2) und (x3,y3) ein!“
- Die Linien eines Polygons sollen sich nicht kreuzen!
- Der Umfang U eines Polygons darf nicht größer als Konstante c sein!

$$U = \sum_{i=2}^n \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} + \sqrt{(x_1 - x_n)^2 + (y_1 - y_n)^2}$$

➔ Deskriptive Formulierungen sind hier sehr schwierig!

## Das Transaktionsparadigma

### • DB-bezogene Definition der Transaktion:

Eine TA ist eine ununterbrechbare Folge von DML-Befehlen, welche die Datenbank von einem logisch konsistenten Zustand in einen neuen logisch konsistenten Zustand überführt.

➔ Diese Definition eignet sich insbesondere für relativ kurze TA, die auch als ACID-Transaktionen bezeichnet werden.

### • Wesentliche Abstraktionen aus Sicht der DB-Anwendung

- Alle Auswirkungen auftretender Fehler bleiben der Anwendung verborgen (*failure transparency*)
- Es sind keine anwendungsseitigen Vorkehrungen zu treffen, um Effekte der Nebenläufigkeit beim DB-Zugriff auszuschließen (*concurrency transparency*)

➔ Gewährleistung einer fehlerfreien Sicht auf die Datenbank im logischen Einbenutzerbetrieb

### • ACID-Prinzip

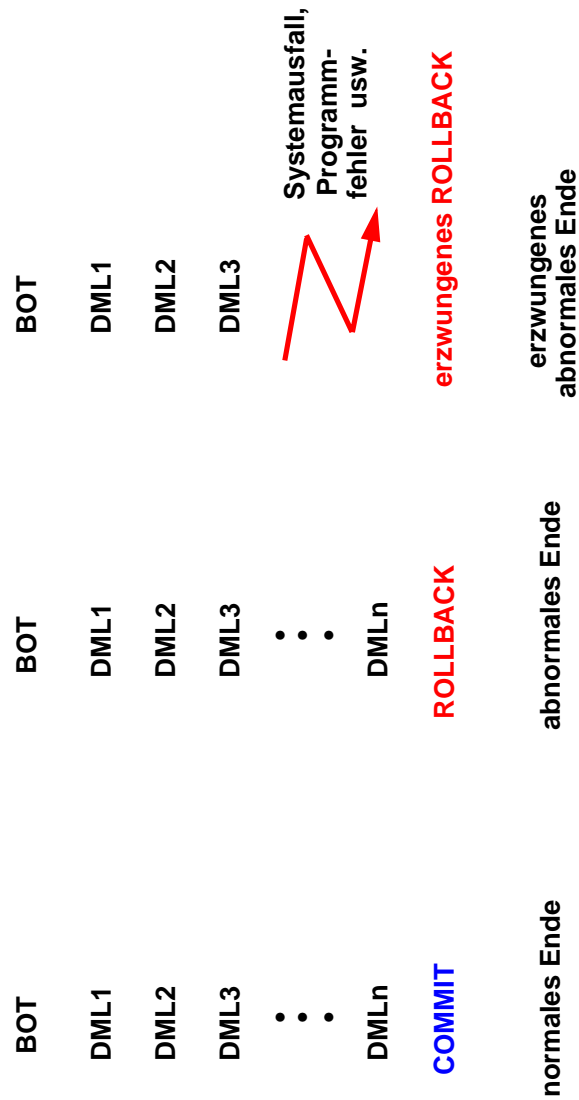
**A**tomicity: 'Alles oder Nichts'-Eigenschaft (Fehlerisolierung)

**C**onsistency: eine erfolgreiche TA erhält die DB-Konsistenz (Menge der definierten Integritätsbedingungen)

**I**solation: alle Aktionen innerhalb einer TA müssen vor parallel ablaufenden TA verborgen werden

**D**urability: sobald eine TA ihre Änderungen freigegeben hat, muß das System das Überleben dieser Änderungen trotz beliebiger (erwarteter) Fehler garantieren (*Persistenz*)

## Mögliche Ausgänge einer Transaktion



## Zusammenfassung

- **Sichtenkonzept**
  - Erhöhung der Benutzerfreundlichkeit,
  - Flexibler Datenschutz
  - Erhöhte Datenunabhängigkeit
  - Rekursive Anwendbarkeit,
  - Eingeschränkte Aktualisierungsmöglichkeiten
  
- **Semantische Integritätskontrolle**
  - Relationale Invarianten, referentielle Integrität und Aktionen
  - Benutzerdefinierte Integritätsbedingungen (*assertions*)
    - ↳ zentrale Spezifikation/Überwachung im DBS wird immer wichtiger
  
- **Das Transaktionsparadigma**
  - Verarbeitungsklammer für die Einhaltung von semantischen Integritätsbedingungen
  - Verdeckung der Nebenläufigkeit (*concurrency isolation*)
    - ↳ Synchronisation
  - Verdeckung von (erwarteten) Fehlerfällen (*failure isolation*)
    - ↳ Logging und Recovery
  - im SQL-Standard: COMMIT WORK, ROLLBACK WORK
    - ↳ Beginn einer Transaktion implizit