

## 3. Tabellen und Sichten

### • Datendefinition nach SQL<sup>1</sup>

- Informations- und Definitionsschema
- Erzeugen von Basistabellen
- Integritätsbedingungen

### • Schemaevolution

- Änderung von Tabellen
- Löschen von Objekten

### • Indexierung

- Einrichtung und Nutzung von Indexstrukturen
- Indexstrukturen mit und ohne Clusterbildung
- Leistungsaspekte

### • Sichtkonzept

- Semantik von Sichten
- Abbildung von Sichten
- Aktualisierung von Sichten

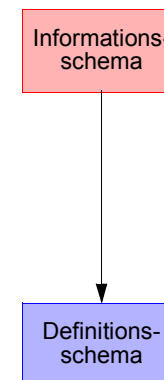
1. Synonyme: Relation – Tabelle, Tupel – Zeile, Attribut – Spalte, Attributwert – Zelle

## Informations- und Definitionsschema

### • Ziel der SQL-Normierung

- möglichst große Unabhängigkeit der DB-Anwendungen von speziellen DBS
- einheitliche Sprachschnittstelle genügt **nicht!**
- Beschreibung der gespeicherten Daten und ihrer Eigenschaften (**Metadaten**) nach einheitlichen und verbindlichen Richtlinien ist genauso wichtig

### • Zweischichtiges Definitionsmodell zur Beschreibung der Metadaten<sup>2</sup>



- bietet **einheitliche Sichten** in normkonformen Implementierungen
- ist **für den Benutzer zugänglich** und somit die definierte Schnittstelle zum Katalog

- beschreibt **hypothetische** Katalogstrukturen, also Meta-Metadaten
- erlaubt „**Altsysteme**“ mit abweichenden Implementierungen normkonform zu werden

### • Welche Meta-Metadaten enthält ein „generisches“ SQL-DBMS?<sup>3</sup>

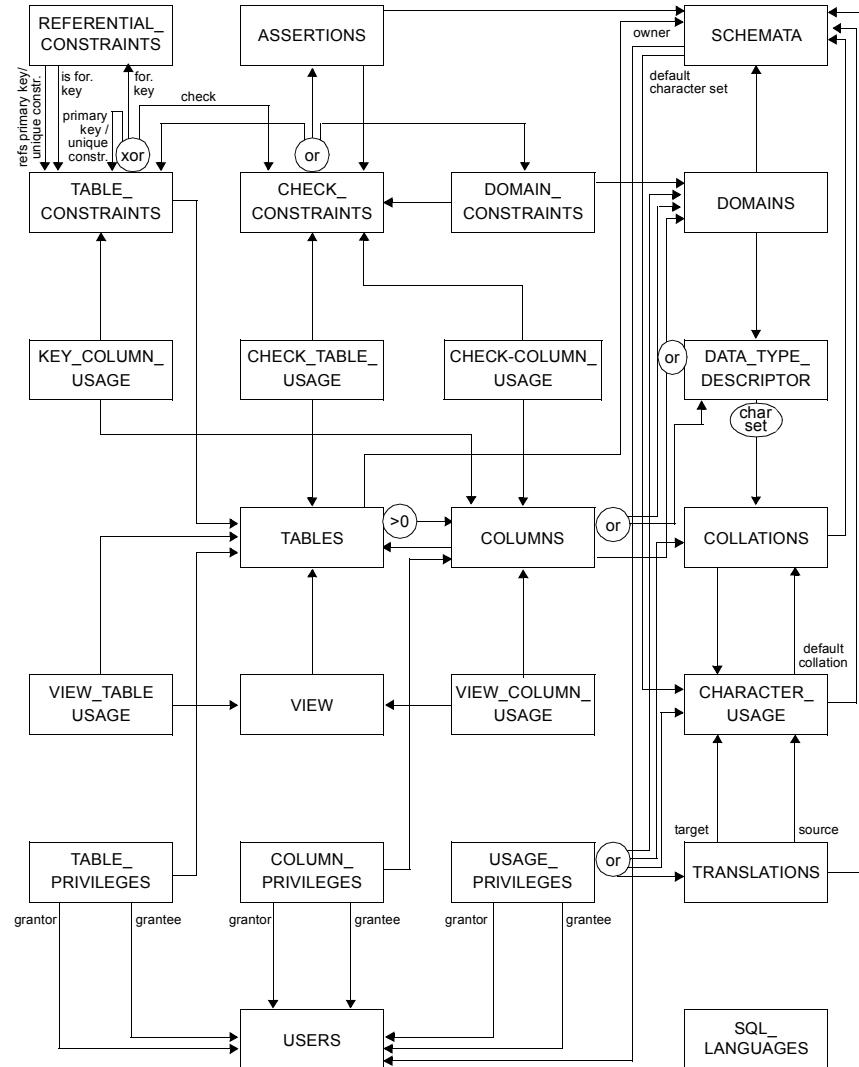
- DEFINITION\_SCHEMA umfasst 24 Basistabellen und 3 Zusicherungen
- In den Tabellendefinitionen werden ausschließlich 3 Domänen verwendet: SQL\_IDENTIFIER, CHARACTER\_DATA und CARDINAL\_NUMBER

2. Als Definitionsgrundlage für die Sichten des Informationsschemas spezifiziert die SQL-Norm das Definitionsschema, das sich auf ein ganzes Cluster von SQL-Katalogen bezieht und die Elemente aller darin enthaltenen SQL-Schemata beschreibt.

3. Das nicht normkonforme Schema SYSCAT von DB2 enthält 37 Tabellen

## Definitionsschema

- Was ist alles zu definieren, um eine "leere DB" zu erhalten?



## Erzeugung von Basistabellen

- Definition einer Tabelle

- Definition aller zugehörigen Attribute mit Typfestlegung
- Spezifikation aller Integritätsbedingungen (Constraints)

### D1: Erzeugung der neuen Tabellen Pers und Abt

```
CREATE TABLE Pers
(Pnr          INT          PRIMARY KEY,
Beruf        CHAR (30),
PName       CHAR (30)    NOT NULL,
PAlder      Alter,      (* siehe Domaindefinition *)
Mgr         INT,
Anr         Abtnr       NOT NULL, (* Domaindef. *)
W-Ort      CHAR (25)    DEFAULT ' ',
Gehalt     DEC (9,2)    DEFAULT 0.00,
CHECK (Gehalt < 120000.00),
```

Constraint FK1 FOREIGN KEY (Anr) REFERENCES Abt  
ON UPDATE CASCADE ON DELETE CASCADE,  
Constraint FK2 FOREIGN KEY (Mgr) REFERENCES Pers (Pnr)  
ON UPDATE SET DEFAULT ON DELETE SET NULL)

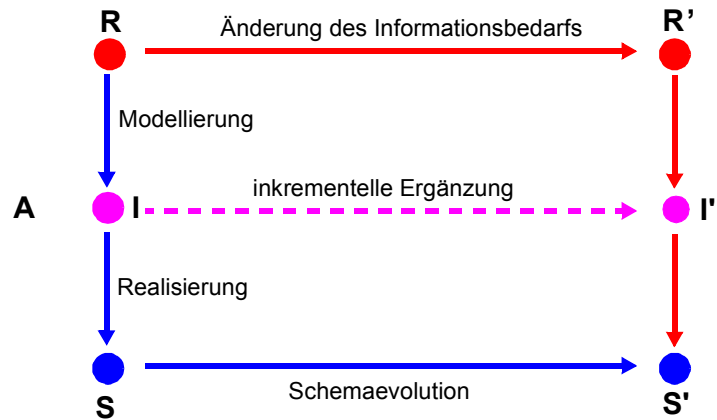
```
CREATE TABLE Abt
(Anr         Abtnr       PRIMARY KEY,
AName       CHAR (30)   NOT NULL,
Anzahl_Angest INT      NOT NULL,
...)
```

```
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?

## Evolution einer Miniwelt

- Grobe Zusammenhänge



R: Realitätsausschnitt (Miniwelt)

I: Informationsmodell  
(zur Analyse und Dokumentation der Miniwelt)

S: DB-Schema der Miniwelt  
(Beschreibung aller Objekt- und Beziehungstypen sowie aller Integritäts- und Zugriffskontrollbedingungen)

A: Abbildung aller wichtigen Objekte und Beziehungen sowie ihrer Integritäts- und Datenschutzaspekte

➔ Abstraktionsvorgang

- Schemaevolution:

- Änderung, Ergänzung oder Neudefinition von Typen und Regeln
- nicht alle Übergänge von S nach S' können automatisiert durch das DBS erfolgen

➔ gespeicherte Objekt- und Beziehungsmengen dürfen den geänderten oder neu spezifizierten Typen und Regeln nicht widersprechen

## Schemaevolution

- Wachsender oder sich ändernder Informationsbedarf

- Erzeugen/Löschen von Tabellen (und Sichten)
- Hinzufügen, Ändern und Löschen von Spalten
- Anlegen/Ändern von referentiellen Beziehungen
- Hinzufügen, Modifikation, Wegfall von Integritätsbedingungen

➔ Hoher Grad an logischer Datenunabhängigkeit ist sehr wichtig!

- Zusätzliche Änderungen im DB-Schema

durch veränderte Anforderungen bei der DB-Nutzung

- Dynamisches Anlegen von Zugriffspfaden
- Aktualisierung der Zugriffskontrollbedingungen

- Dynamische Änderung einer Tabelle

Bei Tabellen können dynamisch (während ihrer Lebenszeit) Schemaänderungen durchgeführt werden

```
ALTER TABLE base-table
{ ADD [COLUMN] column-def
| ALTER [COLUMN] column
  {SET default-def | DROP DEFAULT}
| DROP [COLUMN] column {RESTRICT | CASCADE}
| ADD base-table-constraint-def
| DROP CONSTRAINT constraint {RESTRICT | CASCADE}}
```

➔ Welche Probleme ergeben sich?

## Schemaevolution (2)

### E1: Erweiterung der Tabellen Abt und Pers durch neue Spalten

```
ALTER TABLE Pers
  ADD Svrnr INT UNIQUE
```

```
ALTER TABLE Abt
  ADD Geh-Summe INT
```

Abt	Anr	Aname	Ort
	K51	PLANUNG	KAISERSLAUTERN
	K53	EINKAUF	FRANKFURT
	K55	VERTRIEB	FRANKFURT

Pers	Pnr	Name	Alter	Gehalt	Anr	Mnr
	406	COY	47	50 700	K55	123
	123	MÜLLER	32	43 500	K51	-
	829	SCHMID	36	45 200	K53	777
	574	ABEL	28	36 000	K55	123

### E2: Verkürzung der Tabelle Pers um eine Spalte

```
ALTER TABLE Pers
  DROP COLUMN Alter RESTRICT
```

- Wenn die Spalte die einzige der Tabelle ist, wird die Operation zurückgewiesen.
- Da RESTRICT spezifiziert ist, wird die Operation zurückgewiesen, wenn die Spalte in einer Sicht oder einer Integritätsbedingung (Check) referenziert wird.
- CASCADE dagegen erzwingt die Folgelöschung aller Sichten und Check-Klauseln, die von der Spalte abhängen.

## Schemaevolution (3)

### • Löschen von Objekten

```
DROP {TABLE base-table | VIEW view |
      DOMAIN domain | SCHEMA schema }
      {RESTRICT | CASCADE}
```

- Falls Objekte (Tabellen, Sichten, ...) nicht mehr benötigt werden, können sie durch die DROP-Anweisung aus dem System entfernt werden.
- Mit der CASCADE-Option können 'abhängige' Objekte (z. B. Sichten auf Tabellen oder anderen Sichten) mitentfernt werden
- RESTRICT verhindert Löschen, wenn die zu löschende Tabelle noch durch Sichten oder Integritätsbedingungen referenziert wird

### E3: Löschen von Tabelle Pers

```
DROP TABLE Pers RESTRICT
```

PersConstraint sei definiert auf Pers:

1. ALTER TABLE Pers  
DROP CONSTRAINT PersConstraint CASCADE
2. DROP TABLE Pers RESTRICT

### • Durchführung der Schemaevolution

- Aktualisierung von Tabellenzeilen des SQL-Definitionsschemas
- „tabellengetriebene“ Verarbeitung der Metadaten durch das DBS

## Indexierung

- Einsatz von Indexstrukturen

- Beschleunigung der Suche: Zugriff über Spalten (Schlüsselattribute)
- Kontrolle von Integritätsbedingungen (relationale Invarianten)
- Zeilenzugriff in der logischen Ordnung der Schlüsselwerte
- Gewährleistung der Clustereigenschaft für Tabellen

➔ aber: erhöhter Aktualisierungsaufwand und Speicherplatzbedarf

- Einrichtung von Indexstrukturen

- Datenunabhängigkeit erlaubt Hinzufügen und Löschen
- jederzeit möglich, um z. B. bei veränderten Benutzerprofilen das Leistungsverhalten zu optimieren
- "beliebig" viele Indexstrukturen pro Tabelle und mit unterschiedlichen Spaltenkombinationen als Schlüssel möglich
- Steuerung der **Eindeutigkeit** der Schlüsselwerte, der Clusterbildung

Minimale Anzahl von Indexen:

- Freiplatzanteil (PCTFREE) pro Seite beim Anlegen erleichtert Wachstum

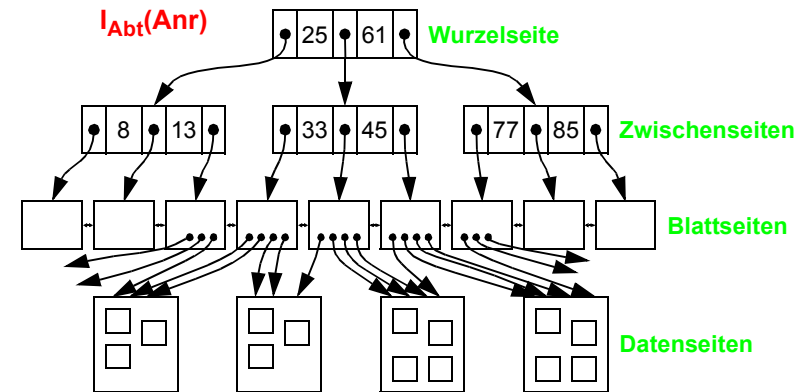
➔ Spezifikation: DBA oder Benutzer

- Im SQL-Standard nicht vorgesehen, jedoch in realen Systemen (z. B. DB2):

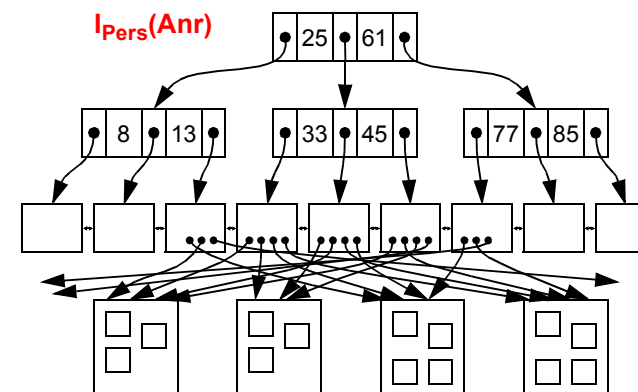
```
CREATE [UNIQUE] INDEX index
ON base-table (column [ORDER] [,column[ORDER]] ...)
[CLUSTER] [PCTFREE]
```

## Indexierung (2)

- Index mit Clusterbildung



- Index ohne Clusterbildung



## Indexierung (3)

### E4: Erzeugung einer Indexstruktur mit Clusterbildung auf der Spalte Anr von Abt

```
CREATE UNIQUE INDEX Persind1
ON Abt (Anr) CLUSTER
```

- **UNIQUE:** keine Schlüsselduplikate in der Indexstruktur
- **CLUSTER:** zeitoptimale sortiert-sequentielle Verarbeitung (Scan-Operation)

### E5: Erzeugung einer Indexstruktur auf den Spalten Anr (absteigend) und Gehalt (aufsteigend) von Pers

```
CREATE INDEX Persind2
ON Pers (Anr DESC, Gehalt ASC)
```

#### • Wie viele Indexstrukturen sollten angelegt werden?

- Heuristik 1:
  - auf allen Primär- und Fremdschlüsselattributen
  - auf Attributen vom Typ DATE
  - auf Attributen, die in (häufigen) Anfragen in Gleichheits- oder IN-Prädikaten vorkommen
- Heuristik 2:
  - Indexstrukturen werden auf Primärschlüssel- und (möglicherweise) auf Fremdschlüsselattributen angelegt
  - Zusätzliche Indexstrukturen werden nur angelegt, wenn für eine aktuelle Anfrage der neue Index zehnmal weniger Sätze liefert als irgendein existierender Index

#### • Nutzung einer vorhandenen Indexstruktur

➔ Entscheidung durch DBS-Optimierer

## Indexierung (4)

#### • Realisierung

- sortierte (sequentielle) Tabelle
- Suchbaum (vor allem Mehrwegbaum)
- Hash-Tabelle (mit verminderter Funktionalität!)

#### • Typische Implementierung einer Indexstruktur: B\*-Baum (wird von allen DBS angeboten!)

➔ dynamische Reorganisation durch Aufteilen (Split) und Mischen von Seiten

#### • Wesentliche Funktionen

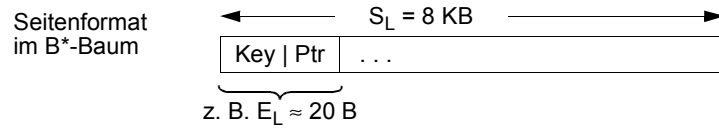
- direkter Schlüsselzugriff auf einen indexierten Satz
- sortiert sequentieller Zugriff auf alle Sätze (unterstützt Bereichsanfragen, Verbundoperation usw.)

#### • Balancierte Struktur

- unabhängig von Schlüsselmenge
- unabhängig von Einfügereihenfolge

## Indexierung (5)

### • Vereinfachtes Zahlenbeispiel zum B\*-Baum



$$ES = \frac{S_L}{E_L} = \text{max. \# Einträge/Seite} (\approx 400)$$

$h_B$  = Baumhöhe  
 $N_T$  = #Zeilenverweise im B\*-Baum  
 $N_B$  = #Blattseiten im B\*-Baum

$$N_{Tmin} = 2 \cdot \left(\frac{ES}{2}\right)^{h_B - 1} \leq N_T \leq ES^{h_B} = N_{Tmax}$$

→ Welche Werte ergeben sich für  $h_B = 3$  und  $E_L = 20 \text{ B}$ ?

$S_L$	ES	$N_{Tmin}$	$N_{Tmax}$
500 B	25	$2 \cdot 13^2 = 338$	$25^3 = 15.625$
8 KB	400	$8 \cdot 10^4$	$400^3 = 64 \cdot 10^6$
32 KB	1600	$128 \cdot 10^4$	$\approx 4 \cdot 10^9$

## 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-commalist ) ]
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
```

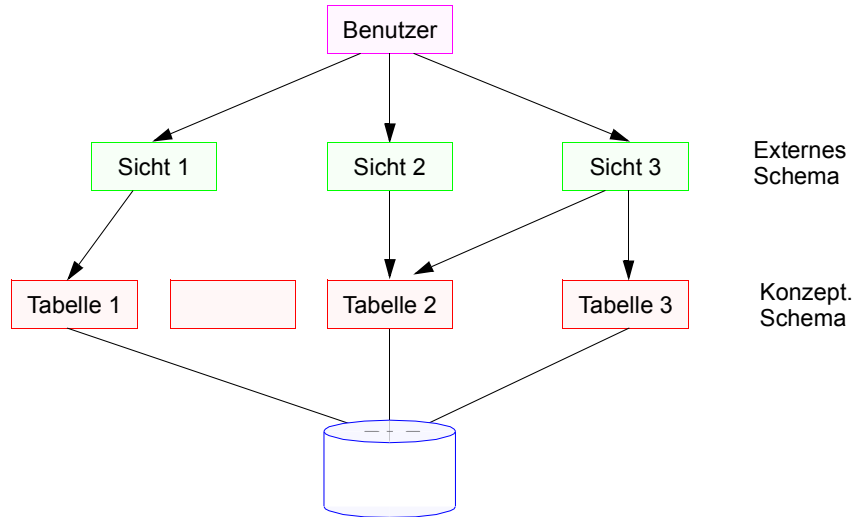
**D3: Sicht für den Datenschutz**

### CREATE VIEW Statistik (Beruf, Gehalt)

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

## Sichtkonzept (2)

- Sichten zur Gewährleistung von Datenunabhängigkeit



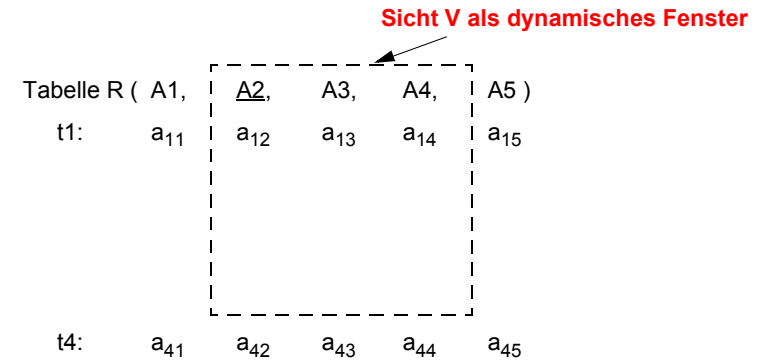
Der initiale DB-Entwurf enthalte Tabelle 1. Diese werde später aufgeteilt in Tabelle 11 und Tabelle 12.

### Eigenschaften von Sichten

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

## Sichtkonzept (3)

- Zum Aspekt: Semantik von Sichten

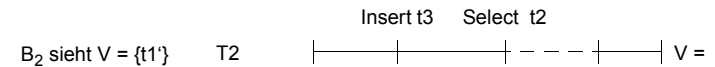
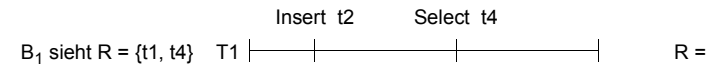


### Sichtbarkeit von Änderungen – Wann und Was?

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'
```

#### Realisierung durch Anfragemodifikation:

```
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
```

#### Anfragemodifikation:

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

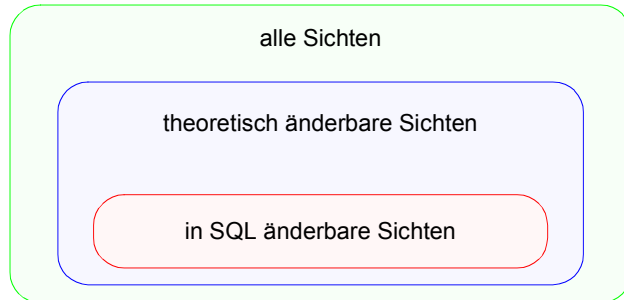
### D4: Löschen von Sichten:

```
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)

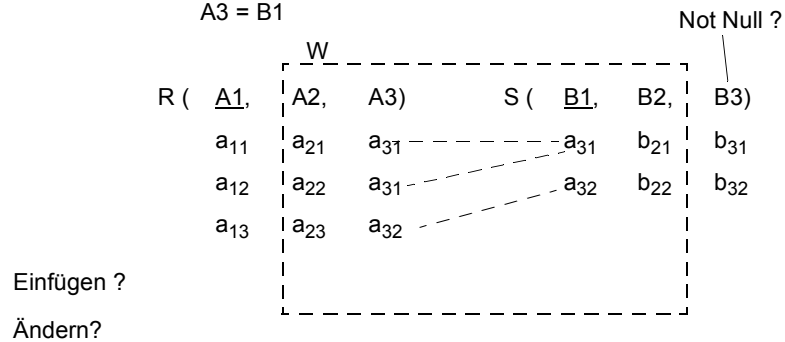
- **Änderbarkeit von Sichten**



- 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-Sichten**

- **beschränkt auf nur eine Tabelle** (Basistabelle oder Sicht)
- Schlüssel muss vorhanden sein
- keine Aggregatfunktionen, Gruppierung und Duplikateliminierung

## Sichtkonzept (7)

- **Problem**

- Sichtdefinierendes Prädikat wird durch Aktualisierungsoperation verletzt
- Beispiel:  
Insert Into Arme\_Programmierer  
(4711, 'Maier', 'Programmierer', **50 000**, 'K55')

- **Überprüfung der Sichtdefinition: WITH CHECK OPTION**

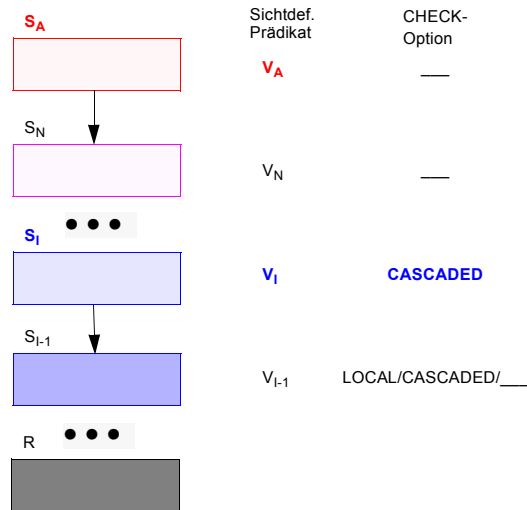
- Einfügungen und Änderungen müssen das die Sicht definierende Prädikat erfüllen. Sonst: Zurückweisung
- nur auf aktualisierbaren Sichten definierbar

- **Zur Kontrolle der Aktualisierung von Sichten, die wiederum auf Sichten aufbauen**, wurde die CHECK-Option verfeinert. Für jede Sicht sind drei Spezifikationen möglich:

- Weglassen der CHECK-Option
- WITH CASCADED CHECK OPTION oder äquivalent WITH CHECK OPTION
- WITH LOCAL CHECK OPTION

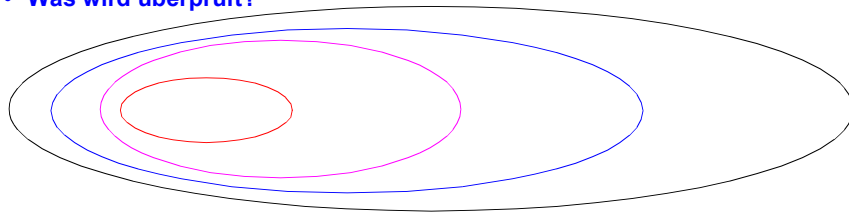
## Sichtkonzept (8)

- **Annahmen**
- Sicht  $S_A$  mit dem die Sicht definierenden Prädikat  $V_A$  wird aktualisiert
- $S_I$  ist die höchste Sicht im Abstammungspfad von  $S_A$ , welche die Option CASCADED besitzt
- Oberhalb von  $S_I$  tritt keine LOCAL-Bedingung auf
- **Vererbung der Prüfbedingung durch CASCADED**



Einfügung mit Prädikat  $P_A$  in Sicht  $S_A$ :  
Welches sichtdefinierende Prädikat wird überprüft?

- **Was wird überprüft?**



3 - 21

## Sichtkonzept (9)

- **Aktualisierung von  $S_A$**
- Als Prüfbedingung wird von  $S_I$  aus an  $S_A$  "vererbt":  
 $V = V_I \wedge V_{I-1} \wedge \dots \wedge V_1$
- ↳ **Erscheint irgendeine aktualisierte Zeile von  $S_A$  nicht in  $S_I$ , so wird die Operation zurückgesetzt**
- Es ist möglich, dass Zeilen aufgrund von gültigen Einfüge- oder Änderungsoperationen aus  $S_A$  verschwinden

- **Aktualisierte Sicht besitzt WITH CHECK OPTION**

- Default ist CASCADED
- Als Prüfbedingung bei Aktualisierungen in  $S_A$  ergibt sich  
 $V = V_A \wedge V_N \wedge \dots \wedge V_I \wedge \dots \wedge V_1$
- Zeilen können jetzt aufgrund von gültigen Einfüge- oder Änderungsoperationen nicht aus  $S_A$  verschwinden

- **LOCAL hat eine undurchsichtige Semantik**

- LOCAL bei  $S_A$ : Aktualisierungen und Einfügungen auf  $S_A$  lassen entweder keine Zeilen aus  $S_A$  verschwinden oder die betroffenen Zeilen verschwinden aus  $S_A$  und  $S_N$
- Empfehlung: nur Verwendung von CASCADED

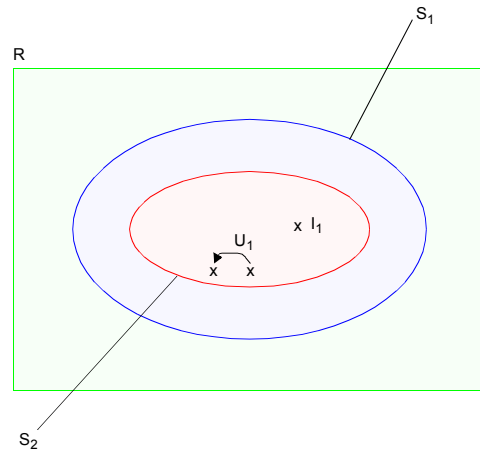
3 - 22

## Sichtbarkeit von Änderungen

### Sichtenhierarchie auf R:

$S_2$  mit  $V_1 \wedge V_2$

$S_1$  mit  $V_1$  und CASCADE



### Aktualisierungsoperationen in $S_2$

$I_1$  und  $U_1$  erfüllen das  $S_2$ -definierende Prädikat  $V_1 \wedge V_2$

$I_2$  und  $U_2$  erfüllen das  $S_1$ -definierende Prädikat  $V_1$

$I_3$  und  $U_3$  erfüllen das  $S_1$ -definierende Prädikat  $V_1$  nicht

### Welche Operationen sind erlaubt?

Insert in  $S_2$ :  
 $I_1$  ✓  
 $I_2$   
 $I_3$

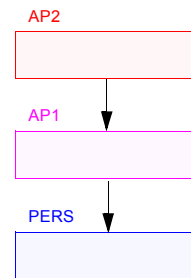
Update in  $S_2$ :  
 $U_1$  ✓  
 $U_2$   
 $U_3$

Ohne Check-Option werden alle Operationen akzeptiert!

## Sichtkonzept (10)

### • Beispiel

Tabelle	Pers
Sicht 1 auf Pers	AP1 mit Beruf = 'Prog.' AND Gehalt < '30K'
Sicht 2 auf AP1	AP2 mit Gehalt > '20K'



Sichtdef. Prädikat	CHECK-Optionen			
	1	2	3	4
> '20K'	—	—	CASC	CASC
< '30K'	—	CASC	—	CASC

### • Operationen

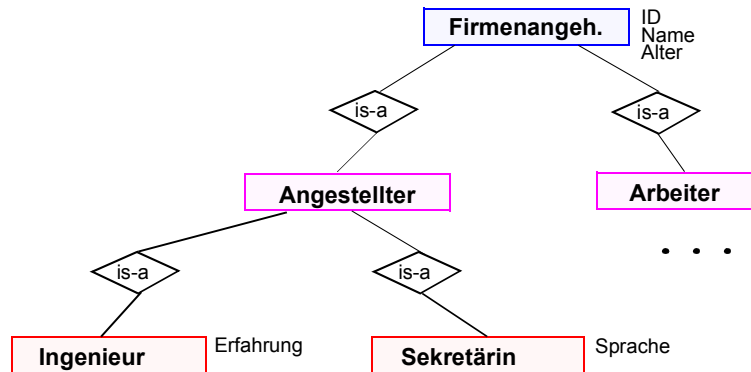
- INSERT INTO AP2  
VALUES (. . . , '15K')
- UPDATE AP2  
SET Gehalt = Gehalt + '5K'  
WHERE Anr = 'K55'
- UPDATE AP2  
SET Gehalt = Gehalt - '3K'

### • Welche Operationen sind bei den verschiedenen CHECK-Optionen gültig?

	1	2	3	4
a				
b				
c				

## Generalisierung mit Sichtkonzept

- Ziel: Simulation einiger Aspekte der Generalisierung



- Einsatz des Sichtkonzeptes

```

CREATE TABLE Sekretärin
  (ID      INT,
   Name    CHAR(20),
   Alter   INT,
   Sprache CHAR(15)
   ...);
  
```

```

INSERT INTO Sekretärin
VALUES (436, 'Daisy', 21, 'Englisch');
  
```

```

CREATE TABLE Ingenieur
  (ID      INT,
   Name    CHAR(20),
   Alter   INT,
   Erfahrung CHAR(15)
   ...);
  
```

```

INSERT INTO Ingenieur
VALUES (123, 'Donald', 37, 'SUN');
  
```

```

CREATE VIEW Angestellter
AS SELECT ID, Name, Alter
   FROM Sekretärin
   UNION
   SELECT ID, Name, Alter
   FROM Ingenieur;
  
```

```

CREATE VIEW Firmenangehöriger
AS SELECT ID, Name, Alter
   FROM Angestellter
   UNION
   SELECT ID, Name, Alter
   FROM Arbeiter;
  
```

## Zusammenfassung

- Datendefinition

- Zweischichtiges Definitionsmodell für die Beschreibung der Daten: Informationsschema und Definitionsschema
- Erzeugung von Tabellen
- Spezifikation von referentieller Integrität und referentiellen Aktionen
- CHECK-Bedingungen für Wertebereiche, Attribute und Tabellen

- Schemaevolution

Änderung/Erweiterung von Spalten, Tabellen, Integritätsbedingungen, ...

- Indexstrukturen als B\*-Bäume

- mit und ohne Clusterbildung spezifizierbar
- Balancierte Struktur unabhängig von Schlüsselmenge und Einfügereihenfolge
- ➔ dynamische Reorganisation durch Aufteilen (Split) und Mischen von Seiten
- direkter Schlüsselzugriff auf einen indexierten Satz
- sortiert sequentieller Zugriff auf alle Sätze (unterstützt Bereichsanfragen, Verbundoperation usw.)
- ➔ Wie viele Indexstrukturen/Tabellen?

- Sichtenkonzept

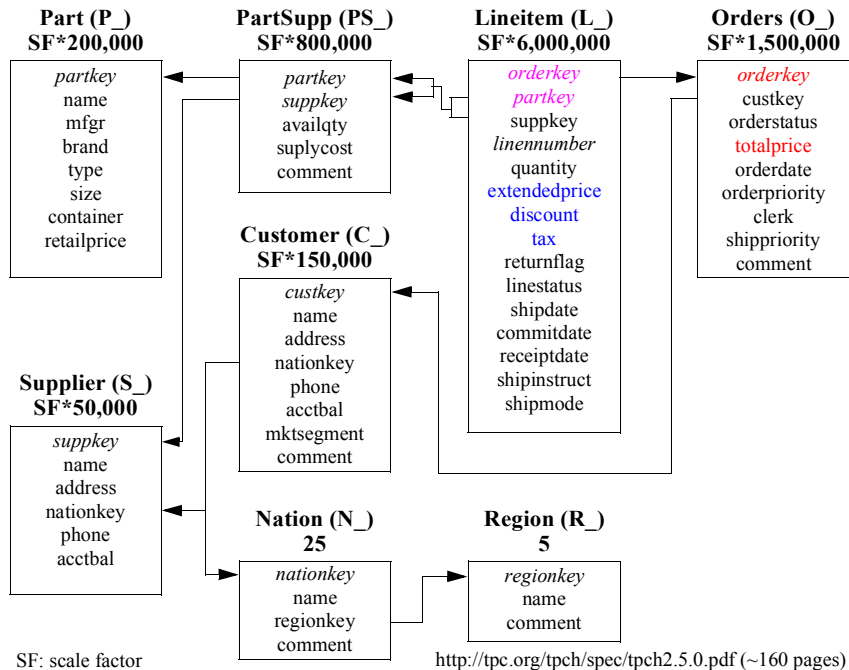
- Erhöhung der Benutzerfreundlichkeit
- Flexibler Datenschutz
- Erhöhte Datenunabhängigkeit
- Rekursive Anwendbarkeit
- Eingeschränkte Aktualisierungsmöglichkeiten

# TPC-R Benchmark

- Summary

The TPC Benchmark™H (TPC-H) is a *decision support benchmark* and consists of >20 business-oriented queries and concurrent data modifications. It represents decision support environments where users *run ad-hoc queries* against a database system. TPC-R (Decision Support, Business Reporting) is similar to TPC-H, but it allows additional optimizations based on advance knowledge of the queries. In this environment, *pre-knowledge of the queries is assumed and may be used for optimization* to run these standard queries very rapidly.

The *performance metric reported by TPC-R* is called the TPC-R Composite Query-per-Hour Performance Metric (*QphR@Size*), and reflects multiple aspects of the capability of the system to process queries. These aspects include the selected database size against which the queries are executed, the query processing power when queries are submitted by a single stream, and the query throughput when queries are submitted by multiple concurrent users. The *TPC-R Price/Performance metric* is expressed as  $\$/QphR@Size$ .



# Materialized Views

- Use of materialized views

- they are a powerful tool for **improving the performance** of complex queries
- **efficient online maintenance** is needed
  - but only known for views defined by SQL queries composed of only Select, Project, Join, Group-by operators (SPJG views)
  - and limited to aggregation functions such as sum or count that can be computed incrementally
- example uses a 10GB TPC-R database

**Example** Query Q against the TPC-R database contains two levels of aggregation and attempts to **find important parts**; a part is important if it contributed more than 90 % to the value of an order. To save space, np (net price) is used as a short-hand for  $(l\_extendedprice * (1 + l\_tax) * (1 - l\_discount))$ . The query cannot be well supported using only SPJG because of the two-level aggregation but allowing views to be stacked (views on views) changes the situation.

```
Q: select l_partkey, count(*) ocnt, sum(sp) oval
    from orders as o,
         (select sum(np) sp, l_orderkey, l_partkey
          from lineitem
          group by l_orderkey, l_partkey) as l
    where o_orderkey = l_orderkey
          and o_totalprice * 0.9 < sp
    group by l_partkey
```

Without any views: **9877 secs**

```
V1: select sum(np) sp, l_orderkey, l_partkey
     from lineitem
     group by l_orderkey, l_partkey
```

Using materialized view V1: **349 secs**

```
V2: select l_partkey, count(*) ocnt, sum(sp) oval
     from orders as o, V1 as l
     where o_orderkey = l_orderkey
           and o_totalprice * 0.9 < sp
     group by l_partkey
```

Using the concept of *stacked views (S-SPJG views)*, we can create a materialized view V2 that combines orders and V1.

The query reduces to a simple scan of V2: **4.5 secs**