

DBVO 1.10

Hintergrundspeicherverwaltung

Große Datenmengen: Daten finden nicht zur Gänze im Hauptspeicher Platz, auch ein virtueller Speicher ist zu klein oder zu ineffizient

Spezielle Techniken sind notwendig

- Puffern von Daten
- Clustern von Daten
- Indizieren von Daten
- Abfrageoptimierung

Physische Datenunabhängigkeit

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

Datenbanksystem:

Trennung zwischen den Anwendungsprogrammen und Daten

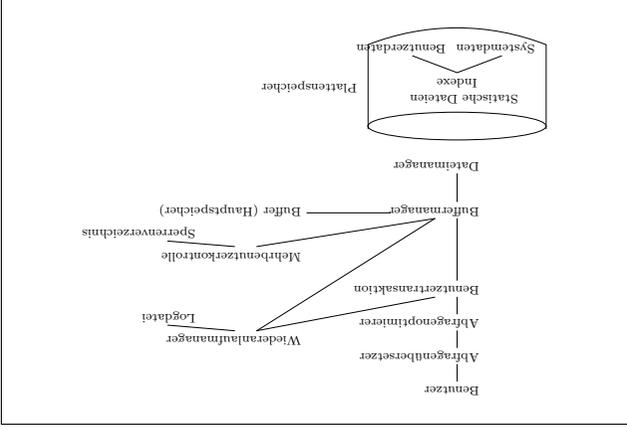
Die Programme arbeiten auf "logischen Daten"

die physische Datenorganisation ist für die Programme unsichtbar

Physische Datenunabhängigkeit: Programme und Adhoc-Abfragen sind von Speicher- oder Zugriffsmethoden unabhängig.

Logische Datenunabhängigkeit: Änderungen der Datenbasis sind möglich, ohne die darauf zugreifenden Programme signifikant zu beeinflussen.

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien



DBVO 1.11

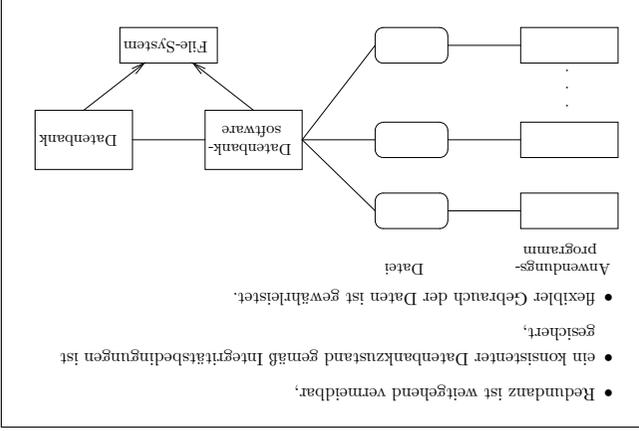
Transaktionen

Transaktionen führen einen konsistenten Datenbankzustand in einen anderen konsistenten Datenbankzustand über

Eigenschaften

- Atomarität
- Konsistenz (Serialisierbarkeit)
- Isolation
- Dauerhaftigkeit

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien



DBVO 1.9

Persistente Datenhaltung

- Daten überleben die Programmausführung
- Jede Art von Daten kann persistent sein
- Die Persistenz von Daten ist *implizit*; es ist nicht notwendig, Daten explizit vom Hintergrundspeicher zu lesen bzw. auf den Hintergrundspeicher zu schreiben

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

DBVO 1.12

Mehrbenutzerbetrieb

Daten werden von mehreren Benutzern gemeinsam verwendet

Jeder Benutzer kann das System benutzen, als ob er der Einzige wäre

- Spezielle Techniken sind notwendig, um zu verhindern, dass sich mehrere Benutzer gegenseitig stören (*concurrency control*)
- Traditionelles Korrektheitskriterium: *Serialisierbarkeit* von Transaktionen

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

Ziel: Herstellung eines konsistenten Datenbankzustands nach einem Fehler

- vom Anwendungsprogramm erkannte logische Fehler (z.B. Konto nicht gedeckt)
- vom System erkannte Fehler (z.B. zyklisches Warten mehrerer Programme auf die Freigabe einer Ressource)
- Hauptspeicherverlust (z.B. Stromausfall)
- Hintergrundspeicherverlust (z.B. disk crash)

Konzept: Transaktionen führen einen konsistenten Datenbankzustand in einen anderen konsistenten Datenbankzustand über

Wiederanlauf

- $Dom(SV_NUMMER) =$ Menge aller zehnstelligen Zahlen, die letzten sechs: *Datumsformat*.
 - $Dom(NAME) =$ Menge aller Namen.
 - $Dom(GEB_DATUM) =$ Menge der Datenschnittränge.
 - $Dom(TELEFON) =$ Menge aller vierstelligen Dezimalzahlen.
 - $Dom(GEHALT) = \{k \mid 500 \leq k \leq 10.000\}$
- Die Tabelle hat 3 Tupel. Eines von ihnen ist t mit: $t(NAME) =$ Huber, $t(SV_NUMMER) = 2424010163$, $t(GEB_DATUM) = 01.01.1963$, $t(TELEFON) = 4144$ und $t(GEHALT) = 2.600$.

Das Relationenmodell

- Nicht alle Benutzer sind gleich
- Klassifikation in Benutzergruppen
- Zugriffsrechte (ändern, lesen, ausführen)
- Granularität der Objekte, für die Zugriffsrechte definiert werden können: Datenbank, Relation, Attribut
- Weitergabe und Rücknahme von Rechten

Datenschutz

Relationenschema R : endliche Menge von **Attributen** $\{A_1, A_2, \dots, A_n\}$.

Wertebereich $Dom(A_i)$: eine Menge D_i , $1 \leq i \leq n$ zu jedem Attributnamen A_i .

Tupel t : Abbildung t von R nach $D = D_1 \cup D_2 \cup D_3 \cup \dots \cup D_n$, $t(A_i) \in D_i$.

Relation $r(R)$: endliche Menge von Tupeln $\{t_1, \dots, t_m\}$.

Relation ... Tabelle, Attribute ... Spalten, Tupel ... Zeilen.

Formalisierung des Relationenmodells

- **A-Wert** $t(A) = a$: Wert eines Tupels t für ein Attribut A .
 - **Einigungung** $t(X)$ eines Tupels t auf die Attributmenge X : Tupel t , das jedem $A \in X$ dieselben Werte aus $Dom(A)$ zuordnet wie t .
- Beispiel 2** $t(SV_NUMMER) = 2424010163$, $t(NAME) =$ Huber, $t(GEB_DATUM) = 01.01.1963$, $t(TELEFON) = 4144$ und $t(GEHALT) = 2.600$.
- Der **NAME-Wert** von t ist $t(NAME) =$ Huber.
 - Die **Einigungung** von t auf $\{NAME, GEB_DATUM\}$ ist t' mit $t'(NAME) =$ Huber, $t'(GEB_DATUM) = 01.01.1963$.

Nicht alle Abfragen sind a-priori bekannt

Es ist nicht notwendig, für allgemeine Abfragen spezielle Anwendungsprogramme zu schreiben

Direkte Abfragen an die Datenbank:

- Einfach formulierbar
- Deklarativ gestellt: es wird die Bedingung angegeben, die die Antwortmenge erfüllen soll, nicht aber das Verfahren, wie sie ermittelt wird
- Werden vom System optimiert

Ad-hoc Abfragen

Beispiel 1 Ein Relationenschema R :

angest. (SV_NUMMER, NAME, GEB_DATUM, TEL., GEHALT)

2424010163	Huber	01.01.1963	4144	2.600
9456030568	Meier	03.05.1968	4143	2.400
4923270866	Eder	27.08.1966	4140	2.400

Eine mögliche Relation:

ANGESTELLTER = {SV_NUMMER, NAME, GEB_DATUM, TELEFON, GEHALT}

Schlüssel (key): von $r(R)$ ist eine Teilmenge K von R mit: t_1, t_2 aus $r(R) \Rightarrow t_1(K) \neq t_2(K)$ und kein $K' \subset K$ hat diese Eigenschaft.

Oberschlüssel (superkey) K : R enthält einen Schlüssel.

Beispiel 3 Schlüsselkandidaten: $\{FLUGNR\}$, $\{VON, NACH\}$, $\{ZEITPUNKT\}$

Hänge (FLUGNR, VON, NACH, ABFLUG, ANKUNFT)

83	Frankfurt	London	11:30a	1:43p
84	London	Frankfurt	3:00p	5:55p
109	Frankfurt	Lissabon	9:50p	2:52a
213	Frankfurt	Wien	11:43a	12:45p
214	Wien	Frankfurt	2:20p	3:12p

ausgewählte Schlüssel (designated keys): Schlüssel, die explizit zu

einem Relationenschema angeführt sind

Primärschlüssel: ausgewählter Schlüssel, der im Relationenmodell

unterstützt wird.

Beispiel 4 FLÜGE (FLUGNR, VON, NACH, ANKUNFT, ABFLUG)

$$r \cup s = \begin{pmatrix} A & B & C \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix} \cup \begin{pmatrix} A & B & C \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix}$$

$$r \cap s = \begin{pmatrix} A & B & C \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix} \cap \begin{pmatrix} A & B & C \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix}$$

$$r - s = \begin{pmatrix} A & B & C \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix} - \begin{pmatrix} A & B & C \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix}$$

Erweiterung der Selektion

Sei θ ein beliebiger Vergleichsoperator:

$$\sigma_{A\theta a}(r) = \{t \in r \mid t(A)\theta a\}$$

$$\sigma_{A_1\theta A_2}(r) = \{t \in r \mid t(A_1)\theta t(A_2)\}$$

Beispiel 7 Selektion aller Flüge, die über 4 Stunden dauern.

$$\sigma_{(ABFLUG+4)>ANKUNFT}(flüge) =$$

(FLUGNR VON NACH ABFLUG ANKUNFT)
109 Frankfurt Lissabon 9:50p 2:52a

Operationen auf Relationen

Relationale Algebra (Codd 1972):

- Mengenoperationen
- Selektion
- Projektion
- Division
- Verbundoperation

Notation: r und s Relationen über den Relationenschemata R bzw S .

Auswahl von Zeilen, die einem bestimmten Kriterium entsprechen.

$$\sigma_{A=a}(r) = \{t \in r \mid t(A) = a\}$$

Die Selektion ist kommutativ bezüglich der Zusammensetzung:

$$\sigma_{A_1=a}(\sigma_{A_2=b}(r)) = \sigma_{A_2=b}(\sigma_{A_1=a}(r)).$$

Beispiel 8 Selektion aller Flüge, die vor 1:00p ankommen

$$\sigma_{ANKUNFT < 1:00p}(flüge) =$$

(FLUGNR VON NACH ABFLUG ANKUNFT)
109 Frankfurt Lissabon 9:50p 2:52a
117 Athen Wien 10:05p 12:43a
213 Frankfurt Wien 11:43a 12:45p

von $r(R), s(S)$ mit $R = S$:

- Durchschnitt (\cap),
- Vereinigung (\cup),
- Differenz ($-$ oder \setminus)

Beispiel 5

$$r \cap s = \begin{pmatrix} A & B & C \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix} \cap \begin{pmatrix} A & B & C \\ a_1 & b_2 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix}$$

Beispiel 6

Hänge (FLUGNR VON NACH ABFLUG ANKUNFT)
84 London Frankfurt 3:00p 5:55p
109 Frankfurt Lissabon 9:50p 2:52a
117 Athen Wien 10:05p 12:43a
213 Frankfurt Wien 11:43a 12:45p
214 Wien Frankfurt 2:20p 3:12p

$$\sigma_{VON=Frankfurt}(flüge) =$$

(FLUGNR VON NACH ABFLUG ANKUNFT)
109 Frankfurt Lissabon 9:50p 2:52a
213 Frankfurt Wien 11:43a 12:45p

Die Projektion

Auswahl von Spalten einer Tabelle: Projektion nach einer Teilmenge der

$$\pi_X(r) = \{t(X) \mid t \in r\} \text{ für } X \subseteq R$$

Der Projektionsoperator ist kommutativ bezüglich der Selektion, wenn die

Attribute, auf denen selektiert wird, in den projizierten Attributen enthalten

sind:

$$\pi_{A_1, A_2}(\sigma_{A_1=s}(r)) = \sigma_{A_1=s}(\pi_{A_1, A_2}(r))$$

Beispiel 9 Projektion der Relation Flüge nach den Attributen ABFLUG und ANKUNFT.

$$\pi_{ABFLUG, ANKUNFT}(flüge) =$$

3:00p	5:55p
9:50p	2:52a
10:05p	12:43a
11:43a	12:45p
2:20p	3:12p

optionen = benutzbar \bowtie zugelassen

optionen (FLUG MASCHINE PILOT)

83	727	Müller
83	727	Huber
83	747	Barth
83	747	Huber
84	727	Müller
84	727	Huber
84	747	Barth
84	747	Huber
109	707	Müller

Weitere Eigenschaften des Verbundoperators

Sei $q = s \bowtie r, r' = \pi_R(q)$, dann gilt: $r' \subseteq r$.

Beispiel 12

$$r \bowtie s = q \quad \begin{array}{c} a \quad b \quad c \\ (A \quad B \quad C) \end{array} \quad \begin{array}{c} a \quad b' \\ (A \quad B) \end{array} \quad \begin{array}{c} a \quad b \quad c \\ (B \quad C) \end{array} \quad s$$

$$\pi_{AB}(q) = r' \quad \begin{array}{c} a \quad b \\ (A \quad B) \end{array}$$

Der Natürliche Verbund (natural join)

Der Verbundoperator verknüpft zwei Relationen über ihre gemeinsamen Attribute:

$$r \bowtie s = \{t \mid \exists t_r \in r \text{ und } \exists t_s \in s : t_r = t(t) \text{ und } t_s = t(s)\}.$$

Der Verbundoperator ist kommutativ.

Das Kartesische Produkt

Falls $R \cap S = \{\}$, dann ist $r \bowtie s$ das kartesische Produkt, $r \times s$.

$$r \quad \begin{array}{c} a_1 \quad b_1 \\ (A \quad B) \end{array} \quad s \quad \begin{array}{c} c_1 \quad d_1 \\ (C \quad D) \end{array}$$

$$r \times s = r \bowtie s = \begin{array}{c} a_1 \quad b_1 \quad c_1 \quad d_1 \\ (A \quad B \quad C \quad D) \end{array}$$

Sei q eine Relation über RS und $q' = \pi_R(q) \bowtie \pi_S(q)$, so gilt: $q' \supseteq q$.

Wenn $q' = q \Rightarrow$ die Relation $q(RS)$ sei **verlustfrei zerlegbar (decomposes losslessly)** im $r(tR) = \pi_R(q)$ und $s(tS) = \pi_S(q)$.

Wenn die **Join-Attribute** zwischen Relationen r und s **Schlüssel** in r oder s sind,

\Rightarrow

die Relation $q = r \bowtie s$ ist **verlustfrei zerlegbar**.

Wichtig für korrekte Zerlegung von Relationenschemata

Beispiel 10

benutzbar (FLUGNR MASCHINE) \bowtie zugelassen (PILOT MASCHINE)

83	727	Müller	707
83	747	Müller	727
84	727	Barth	747
84	747	Barth	717
109	707	Huber	727
109	707	Huber	747

Kartesisches Produkt für Relationen mit gemeinsamen Attributen:

Umbenennen der Attribute.

Beispiel 11 $R = \{A, B, C\}$ und $S = \{A, B, D\}$,

\Rightarrow

Umbenennen der Attribute von S in $S = \{A', B', D\}$ oder Kennzeichnen durch Vorstellen des Relationennamens $S = \{S.A, S.B, D\}$.

Beispiel 13 $R = AB, S = BC$

$$q_1 \quad \begin{array}{c} a \quad b \quad c \\ (A \quad B \quad C) \end{array}$$

$$s \quad \begin{array}{c} a \quad b' \quad c' \\ (B \quad C) \end{array}$$

q_1'

$$r \quad \begin{array}{c} a \quad b \\ (A \quad B) \end{array} \quad \begin{array}{c} a \quad b \quad c \\ (A \quad B \quad C) \end{array}$$

Gegeben sind zwei Relationenschemata:

$$\text{Richter} \div q = q' \quad (\text{PILOT})$$

$$\text{Richter} \div s = s' \quad (\text{PILOT})$$

Pires
Truman
Pires

Beispielaufgabe wie vorher, aber um eine Spur schwieriger!

Gegeben sind wieder die zwei Relationenschemata:

$$\text{verkauft}(\overline{\text{WNR,FNR,DATUM,ANZAHL}})$$

$$\text{filiale}(\overline{\text{FNR,ADRESSE}})$$

(WNR) und bestimmte Verkaufstage (DATUM) ausgibt, so dass die entsprechende Ware am jeweiligen Tag in *jeder* Filiale zumindest gleich oft verkauft wurde wie an jedem anderen Tag in der selben Filiale.

Wenn an einem Tag DATUM=a die Ware mit der Warennummer WNR=b in Filiale FNR=c nicht verkauft wurde, dann gibt es *kein* Tupel (b,c,a,0) in der Relation verkauft.

Ziel: grundlegende Strukturierungsformen für die Datenmodellierung entsprechen den natürlichen Begriffsstrukturen.

Semantischen Strukturen, die in der Anwendungsumgebung auftreten, sollen im Datenbankschema erfaßt und dargestellt werden.

Bedeutung soll im Datenbankschema ausgedrückt werden können.

Beispielaufgabe

Gegeben sind zwei Relationenschemata:

$$\text{verkauft}(\overline{\text{WNR,FNR,DATUM,ANZAHL}})$$

$$\text{filiale}(\overline{\text{FNR,ADRESSE}})$$

Gesucht ist eine Abfrage in relationaler Algebra, die die Warennummern (WNR) und bestimmte Verkaufstage (DATUM) ausgibt, so dass am jeweiligen Tag in *jeder* Filiale zumindest 100 Stück von der entsprechenden Ware verkauft wurden.

Wenn an einem Tag DATUM=a die Ware mit der Warennummer WNR=b in Filiale FNR=c nicht verkauft wurde, dann gibt es *kein* Tupel (b,c,a,0) in der Relation verkauft.

Lösung:

$$\text{verkauft}(\overline{\text{WNR,FNR,DATUM,ANZAHL}})$$

$$\text{filiale}(\overline{\text{FNR,ADRESSE}})$$

Gesucht ist eine Abfrage in relationaler Algebra, die die Warennummern (WNR) und bestimmte Verkaufstage (DATUM) ausgibt, so dass die entsprechenden Ware am jeweiligen Tag in *jeder* Filiale zumindest gleich oft verkauft wurde wie an jedem anderen Tag in der selben Filiale.

$$Y = \pi_{\text{FNR}}(\text{filiale})$$

$$V = \text{verkauft}[WNR = WNR' \wedge FNR = FNR' \wedge \text{ANZAHL} > \text{ANZAHL}']_{\text{verkauft}}$$

$$U = \pi_{\text{WNR,FNR,DATUM}}(V) - \pi_{\text{WNR,FNR,DATUM}}(V)$$

Resultat: $U \div Y$

Die wichtigsten semantischen Datenmodelle:

- Entity-Relationship-Model, (Chen 1976) erweitert zum Extended-Entity-Relation-ship-Modell, (Teorey, Yang und Fy 1986)
- SDM, Hammer, (McLeod 1978)
- RM/T, (Codd 1979)
- Objektorientierte Datenmodelle

Weitere: DAPLEX (Shipman 1979), TAXIS (Mylopoulos, Bernstein und Wong 1980) und IFO (Abitbolul, Hull 1984).

Lösung:

$$\text{verkauft}(\overline{\text{WNR,FNR,DATUM,ANZAHL}})$$

$$\text{filiale}(\overline{\text{FNR,ADRESSE}})$$

Gesucht ist eine Abfrage in relationaler Algebra, die die Warennummern (WNR) und bestimmte Verkaufstage (DATUM) ausgibt, so dass am jeweiligen Tag in *jeder* Filiale zumindest 100 Stück von der entsprechenden Ware verkauft wurden.

$$Y = \pi_{\text{FNR}}(\text{filiale})$$

$$Z = \sigma_{\text{ANZAHL} \geq 100}(\text{verkauft})$$

Resultat: $\pi_{\text{FNR,WNR,DATUM}}(\sigma_{\text{ANZAHL} \geq 100}(\text{verkauft})) \div \pi_{\text{FNR}}(\text{filiale})$

Semantische Datenmodelle

Modellierungskonzepte semantischer Datenmodelle

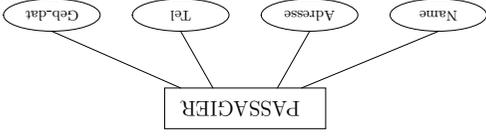
Äquivalente Begriffe: Objekttypenebene = Objektklassenebene = Schema (Schema-level), Objektenebene = Exemplarebene (Instance-level)

Klassifikation: Objekte mit gleichen Eigenschaften werden zu Objekttypen (Objektklassen) zusammengefaßt.

Aggregation: Eine Beziehung zwischen Komponenten-Objekten wird als Objekt höherer Stufe (Aggregationsobjekt) betrachtet.

Generalisierung: Mehrere Objekttypen werden als ein allgemeinerer Objekttyp betrachtet, indem Gemeinsamkeiten hervorgehoben und Unterschiede vernachlässigt werden.
 Entlang der Generalisierungshierarchie werden Attribute verteilt.
Gruppierung: Eine Beziehung zwischen mehreren Objekten wird als Mengenobjekt höherer Stufe betrachtet.

Aggregation im ER Modell
 Aggregation von Attributen zu Entites: Kennzeichnung der Entity durch Satz von Attributen (Merkmale), Attribute nehmen Werte (values) aus einem Wertebereich (value set) an.



Beispiel 1

Beziehungskomplexität:
 1:1 jede Objektinstanz von A zu genau einer Instanz von B und umgekehrt
 1:n jede Objektinstanz von A zu einer oder mehreren von B, aber jede Instanz von B nur zu einer von A.
 m:n jede Objektinstanz von A zu einer oder mehreren in B und umgekehrt.

Weitere Modellierungskonzepte

Welt: besteht aus Entites (Objekten) und Relationships (Beziehungen) zwischen den Objekten.

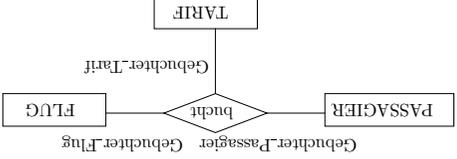
Entity: Modell eines Dings, das in der Umwelt erkannt und eindeutig identifiziert werden kann.

- real (z.B. eine Person)
- ideell (z.B. eine Dienstleistung)

Das Entity-Relationship Modell

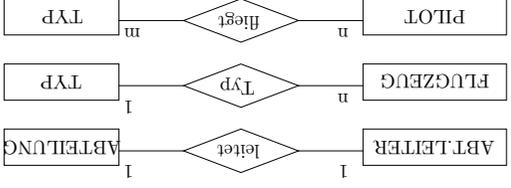
Aggregation von Gegenständen zu Relationships:

- eine Beziehung zwischen zwei oder mehreren Objekten: Element einer mathematischen Relation
- die Funktion, die ein Objekt in einer Beziehung erfüllt: seine Rolle
- Der Grad einer Beziehung: die Anzahl der an der Beziehung beteiligten Objekte



Beispiel 2

Beispiel 4

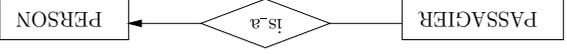


Klassifikation im ER Modell

Entites werden zu entity sets (Objekttypen), und Relationships zu relationship sets (Beziehungstypen) zusammengefaßt.
 Im Weiteren lassen wir das Wort set weg, da aus dem Zusammenhang klar ist, was gemeint ist.

Generalisierung im ER

Die Generalisierung wird explizit durch eine spezielle Relationship dargestellt:



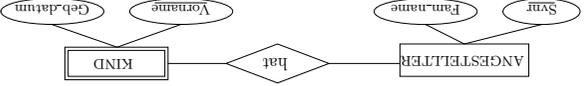
Beispiel 3

Attribute:

- Identifiers (identifizierende Attribute) auch mehrere
- Descriptors (beschreibende Attribute).

Weak Entity: Objekt, das nur mit Hilfe einer (bzw. mehrerer) anderen Entity(es), mit der (denen) es in Beziehung steht, identifiziert werden kann.

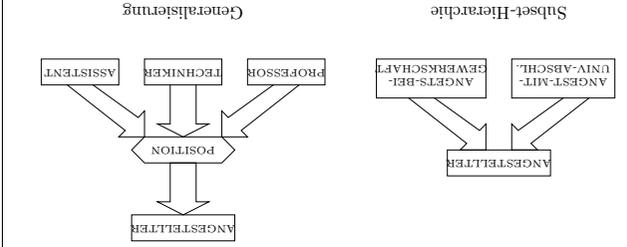
Beispiel 5



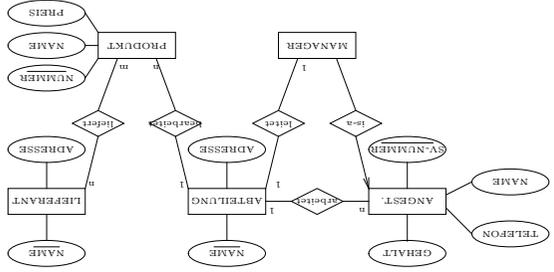
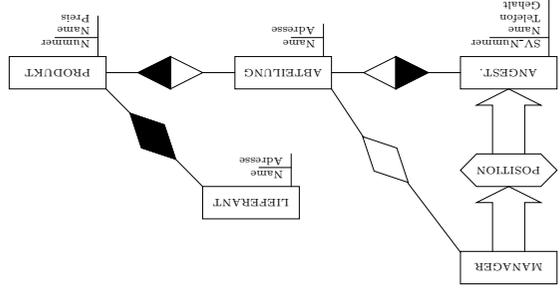
Beispiel 6 In der Datenbank werden Daten einer Firma gespeichert.

Entites: Abteilung, Angestellter, Manager, Lieferant, Produkt

- Eine Abteilung wird eindeutig beschrieben durch ihren Namen; die Adresse ist ein weiteres Attribut.
- Der Angestellte wird eindeutig beschrieben durch seine Sozialversicherungsnummer, weitere Attribute sind Name, Gehalt und Telefonnummer. Er ist einer Abteilung zugeordnet.
- Ein Manager ist ein Angestellter, der eine Abteilung leitet.
- Der Lieferant ist eindeutig bestimmt durch seinen Namen, als weiteres Attribut gibt es noch seine Adresse.
- Die Produkte werden eindeutig beschrieben durch ihre Nummer, außerdem haben sie einen Namen und einen Preis. Sie werden von einer Abteilung bearbeitet und von verschiedenen Lieferanten geliefert.



Beispiel 8



Erweiterung der Modellierungsmöglichkeiten der Relationships

Eine Beziehung heißt *n*-äre Beziehung, wenn sie vom Grad *n* ist. Unäre, binäre und ternäre Relationships sind Spezialfälle.

- Komplexität: Einfarben der Rante, die die Relation darstellt.
- optionale Beziehungen: ein Ring an der Kante der Relation.

Übersetzung des EER in das Relationenmodell

Entites und Relationships (Beziehungen) haben **keine** direkte Entsprechung im Relationenmodell.

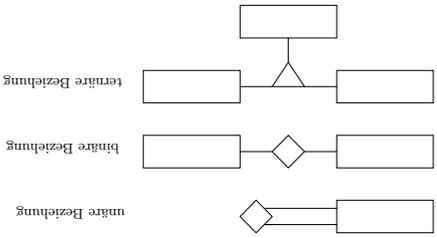
Alle Sachverhalte werden einheitlich durch Relationen (Beziehungen) zwischen Attributen) dargestellt.

Beschreibung der Auflösung der Beziehungen. Alle anderen Konstrukte: Artikel "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship-Model" von Teorey, T.J., Yang, D. and Fry, J.P. im Anhang des Übungsskriptums.

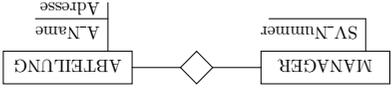
Das Extended Entity Relationship Modell

Erweiterung des ER

- **Subset-Hierarchie:** eine Entity E_1 ist ein Subset der Entity E_2 , wenn jedes Vorkommen der Entity E_1 auch ein Vorkommen der Entity E_2 ist.
- **Generalisierungshierarchie:** eine Entity E ist eine Generalisierung der Entites E_1, \dots, E_n , wenn jedes Vorkommen der Entity E auch das Vorkommen *genau eines* aus den der Entites E_1, \dots, E_n ist.



1:1 Beziehung



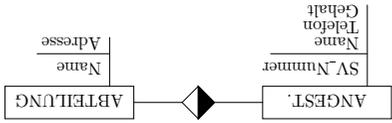
Abteilung = {A_NAME, A_ADRESSE}
 Manager = {SV_NUMMER, A_NAME}

Weitere Sprachen für das Relationenmodell

Einteilung der Sprachtypen für das Relationenmodell:
 nach Beschreibungsart
 Prozedurale Sprachen
 Nichtprozedurale Sprachen
 nach der Art der Resultate
 Typorientierte Sprachen
 Mengenorientierte Sprachen

Angestellter = {SV.NUMBER, NAME, TELEFON, GEGHALT, A.NAME}
 Abteilung = {A.NAME, A.ADDRESS}
 Manager = {SV.NUMBER, P.NAME}
 Produkt = {P.NUMBER, P.NAME, PREIS, A.NAME}
 Lieferant = {L.NAME, L.ADDRESS, L.NAME, P.NUMBER}
 liefert =

Abteilung = {ABT.NAME, ABT.ADDRESS, TELEFON, GEHALT, ABT.NAME}



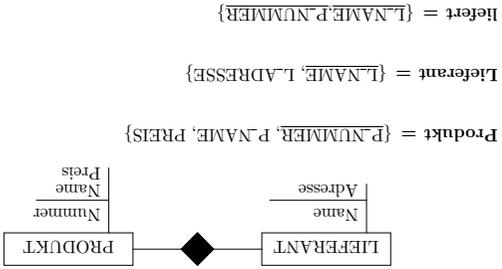
1:n Beziehung

Einteilung der Sprachtypen für das Relationenmodell:

nach Beschreibungsart
 Prozedurale Sprachen
 Nichtprozedurale Sprachen
 nach der Art der Resultate
 Typorientierte Sprachen
 Mengenorientierte Sprachen

NF²: Non First Normal Form Relations (Relational Model with Relation-Valued Attributes).
 Erweiterung des Relationalen Modelles
 Erlauben komplexere Kombination der Konzepte: Attributwerte können wieder Relationen sein.
 Komplexe Objekte und hierarchische Strukturen können leichter dargestellt werden (z.B.: CAD, Bitumengebung; ...).

NF²-Relationen

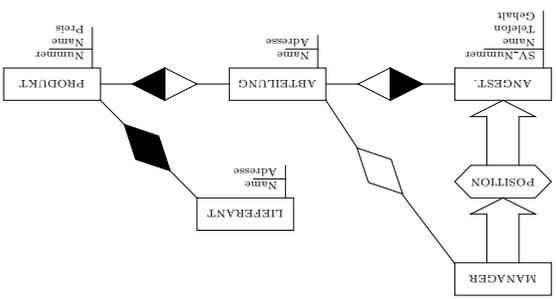


n:m Beziehung

Prozedurale Sprachen:

- operational,
 - Abarbeitungsreihenfolge ist vorgegeben.
 - Beispiele: DAML für Netzwerkmodell, Hierarchisches Modell und Relationale Algebra.
- Nichtprozedurale Sprachen:**
- deskriptiv (oder deklarativ),
 - geben eine Beschreibung der Auswahlbedingung für ein Datenemnt.
 - Beispiele: Relationalkalkül, QBE, und SQL.

ANR		ANNAME		BÜROANG		TNR		T-ARBETT		KNR		JAHR	
51	Programmierer	1	75	76	79	2	76	5	79	1	75	2	76
52	Grundlagen	1	82	79	79	2	82	2	79	1	82	2	76
78	Planung	2	75	75	75	4	75	4	77	2	75	4	77
50	Entwurf	4	75	81	81	2	81	2	81	4	75	2	81
80	Wartung	3	77	77	77	3	77	3	77	3	77	3	77
81	Planung	1	82	82	85	2	82	1	82	1	82	2	85



Typorientierte Sprachen:

- Einzelne Datensätze werden als Resultat geliefert.
- mehrere Datensätze als Resultat \Rightarrow explizite sequentielle Abarbeitung.
- Beispiele: DML für Netzwerkmodell und Hierarchisches Modell.

Mengenorientierte Sprachen:

- Mengen von Datensätzen werden als Resultat geliefert.
- Cursor-Konzept zur Umsetzung in tupelorientierte Darstellung.
- Beispiele: Alle Relationalen Abfragesprachen (SQL, QUEL, ...)

benutzt	(TEIL, TYP, ANZAHL)	lager	(TEIL, ORT, MENGE)
211	707	86	211
2114	727	134	211
2114	707	86	211
2114	727	134	2114
2116	707	244	2114
2116	727	296	2116
21164	707	488	2116
21164	707	488	2116
21164	727	592	21164
36.391			Athen

QBE - Query By Example

- am IBM Watson Research Center entwickelt;
- basiert auf Relationenkalkül mit Bereichsvariablen.
- Interface: **Tabellengerüste** – Kopfzeile mit Attributen
- **Bereichsvariable:** unterstrichene Zeichenketten, aus dem Wertebereich des Attributes
- **Konstante:** nicht unterstrichene Zeichenketten
- **Ausgabewerte:** gekennzeichnet durch vorgestelltes **P.**
- Eintragungen in derselben Zeile werden UND-verknüpft.

Der Relationenkalkül

Der Relationenkalkül beruht auf Prädikatenlogik 1. Stufe.

Beispiel 1

$$\forall x \forall y (P(x, y) \rightarrow \exists z Q(x, z))$$

wobei x, y, z Variable, P und Q Prädikate sind.

Wertebereich der Variablen

- **Typel** \Rightarrow Relationenkalkül mit Typelvariablen (tupel relational calculus)
- **Attributwerte** \Rightarrow Relationenkalkül mit Bereichsvariablen (domain relational calculus)

Der Relationenkalkül mit Typelvariablen

$$\{x(R)|f(x)\}$$

$x \dots$ Typelvariable, $R \dots$ Menge von Attributen
 $f \dots$ eine Funktion über der Typelvariable nach $\{wahr, falsch\}$

Ergebnisrelation $r(R)$ besteht aus Tupeln $t(R)$ mit $f(t) = wahr$

Beispiel 3 *Gesucht sind die Bestandteile des Teiles mit der Nummer 211.*
 $\{x(TEIL_TEILNAME) \mid x \in teilinfo \text{ und } x(BESTANDTEIL_VON) = 211\}$

(TEIL, TEILNAME)	Sitzbezug	2116	Gurt

Beispiel 5 *Wo ist der Teil 211 verfügbar?*

lager	TEIL	LAGEORT	MENGE
	211	P. Chicago	
	211	P.	

oder

lager	TEIL	LAGEORT	MENGE
lager	LAGERORT	Frankfurt	Wien
		London	

Beispiel 2

(TEIL, BESTANDTEIL_VON, TEILNAME)	Sitz	211	0
		211	211
		2116	Gurtschloß
		2116	Gurtbefestigung
		21164	eine Art Schraube
		0	Gepäckfach
		206	Leselampe
		206	Lichtschalter
		206	Belüftungsschalter

$$\frac{134}{(\text{ANZAHL})}$$

$\forall y \exists x (TEILNAME) = y(\text{Sitz})$
 $\forall \exists y \in teilinfo, x(TEIL) = y(TEIL)$

$\{x(\text{ANZAHL}) \mid x \in benutzt \wedge x(TYP) = 727\}$

Beispiel 4 *Wieviele Sitze werden für eine 727 gebraucht?*

Selektionen: Vergleichsoperatoren als Präfix von Konstanten oder Variablen.

Beispiel 6 *Welche Teile werden mehr als 100 mal für ein Flugzeug, das keine 727 ist, gebraucht?*

benutzt	TEIL	ANZAHL
		$\neq 727$
		≥ 100

benutzt	TEIL
	2116
	21164

Grundkonstruktion einer SQL-Abfrage

"SELECT attributes FROM relations WHERE condition"

Auswertung:

- kartesisches Produkt der relations
- Auswahl der Tupel, die condition erfüllen
- Projektion auf attributes

condition: zusammengesetzt aus mehreren, durch logische Operatoren AND, OR und NOT verbundenen Teilbedingungen.

Teilbedingungen: Vergleiche zwischen Attributwerten oder zwischen Attributwerten und Konstanten.

Ausgabe: Duplikate werden mit SELECT DISTINCT eliminiert, die Reihenfolge der Ausgabe: ORDER BY attributes ASC|DESC

- Aliasbezeichner in FROM-Klausel
- SELECT expression AS attribut.n

Beispiel 13 *Selektiere paarweise alle Geschwister, die in verschiedenen Jahren geboren sind, mit deren Altersdifferenz*

SELECT k1.vorname, k2.vorname,
FROM kinder k1, kinder k2
(k1.gbjahr - k2.gbjahr) AS altersdifferenz

WHERE k1.pnr = k2.pnr
AND k1.gbjahr > k2.gbjahr;

Aggregatfunktionen (außer COUNT): angewandt auf arithmetische Ausdrücke über Attributen.

COUNT ausgewertet über Booleschen Ausdrücken – zählt jene Tupel, die den Booleschen Ausdruck erfüllen.

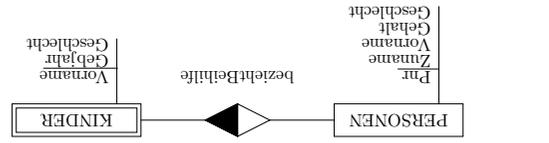
In der SELECT-Klausel und in der HAVING-Klausel dürfen jene Attribute, nach denen gruppiert wurde, nur ohne

Aggregatfunktion selektiert werden.

Nach jenen Attributen, die in der SELECT-Klausel ohne

Beispiel 11 *Relationenschema R für die weiteren Beispiele:*

personen (PNB, ZUNAME, VORNAME, GEBJAHR, GESCHLECHT).
kinder (PNR, VORNAME, GEBJAHR, GESCHLECHT).



Das EBR dazu sieht folgendermaßen aus:

Gruppierung und Aggregatfunktionen

"SELECT attributes FROM relations WHERE condition-1 GROUP BY attributList HAVING condition-2"

attributes besteht aus einer Liste von Attributen mit AVG, SUM, MAX, MIN, COUNT.

Auswertung:

bildet Gruppen von Tupeln und wendet auf diese Aggregatfunktionen an

- Selektion der entsprechenden Tupel aus dem kartesischen Produkt
- Gruppierung dieser nach gemeinsamen Werten von attributList
- Selektion der Gruppen nach condition-2
- Anwendung der Aggregatfunktionen auf jede der verbleibenden Gruppen.

Keine GROUP-BY und HAVING-Klauseln vorhanden:

- Aggregatfunktionen müssen auf alle Attribute im SELECT angewendet werden...

- ...oder auf keines.

⇒ alle in der WHERE-Klausel ausgewählten Tupel bilden eine Gruppe.

Beispiel 12 *Selektiere Personnummer und Gehalt jener Personen, die für ein Kind, das nach 1984 geboren wurde, Familienbeihilfe beziehen. Duplikate werden eliminiert, die Ausgabe wird nach Personnummer sortiert.*

SELECT DISTINCT personen.pnr, personen.gehalt
FROM personen, kinder
WHERE personen.pnr = kinder.pnr
AND kinder.gbjahr > 1984
ORDER BY personen.pnr ASC;

Beispiel 14 *Schreibe eine SQL-Abfrage, die die Personnummern jener Personen liefert, die pro Familienmitglied (Eltern und Kinder mit einem Geburtsjahr nach 1984) weniger als 500 verdienen. Weiters wird jeweils die Anzahl der Kinder sowie ihr Durchschnittsalter ausgegeben.*

SELECT p.pnr, count(*), avg(2002 - k.gbjahr)
FROM personen p, kinder k
WHERE p.pnr=k.pnr
GROUP BY p.pnr, p.gehalt
HAVING (p.gehalt / (count(k.gbjahr > 1984) + 2) < 500;

Teilafragen

Beispiel 15 *Selektiere die Personnummern jener Personen, die mehr Familienbeihilfe für Mädchen als für Burschen beziehen, und für die das Durchschnittsalter der Mädchen höher als jenes der Burschen ist.*

SELECT p.pnr
FROM personen p
WHERE (SELECT COUNT(*) , AVG(2002-k.gbjahr)
FROM kinder k
WHERE p.pnr=k.pnr AND k.geschlecht='w')
>
(SELECT COUNT(*) , AVG(2002-k.gbjahr)
FROM kinder k
WHERE p.pnr=k.pnr AND k.geschlecht='m');

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

- SQL-Abfrage (Teilabfrage) in der WHERE-Klausel
- Wird in der Teilabfrage auf eine Relation der Hauptabfrage Bezug genommen, so wird die Teilabfrage für jedes Tupel der Hauptabfrage ausgewertet.
- Vergleich von gleich langen Listen positionswise (wahr, wenn der Vergleich für jede Position der Liste wahr liefert).
- Ein einzelner Ausdruck statt Ausdruckliste \Rightarrow die Abfrage darf nur ein Attribut selektieren.

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

Division

Welcher Pilot fliegt (pilot, typ) kann alle Maschinen in Maschine(typ) fliegen?

```

SELECT f1.pilot
FROM fliegt f1
WHERE NOT EXISTS (SELECT typ
FROM maschine
WHERE typ NOT IN (SELECT f2.typ
FROM fliegt f2
WHERE f1.pilot = f2.pilot));

```

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

Erzeugen von Relationenschemata

- Wertebereich zu jedem Attribut,
- Integritätsbedingungen zu jedem Attribut,
- PRIMARY KEY Primärschlüssel,
- NOT NULL kein Nullwert, default für Schlüssel,
- UNIQUE eindeutig, default für Schlüssel,
- CHECK (*condition*) Wertebereichseinschränkung;
- 'DEFAULT value' Defaultwert.

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

Spezielle Formen von Teilbedingungen

- "EXISTS query" liefert wahr, wenn die Abfrage query mindestens ein Tupel selektiert.
- "expressionList comparison ANY query" liefert wahr, wenn der Vergleich der Ausdruckliste expressionList mit irgendeinem von der Abfrage query erzeugten Tupel wahr liefert.
- "expressionList comparison ALL query" liefert wahr, wenn der Vergleich der Ausdruckliste expressionList mit allen von der Abfrage query erzeugten Tupel wahr liefert.
- "expressionList IN query", wenn der Vergleich der Ausdruckliste expressionList mit einem Tupel der Abfrage query identisch ist.
- "expressionList NOT IN query", wenn der Vergleich der Ausdruckliste expressionList mit keinem Tupel der Abfrage query identisch ist.

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

Mengenoperationen

- UNION (Mengenvereinigung);
- UNION ALL (Multimengenvereinigung, d. h. gleiche Tupel werden nicht eliminiert);
- INTERSECT (Durchschnitt);
- EXCEPT (Mengendifferenz)

für typkompatible Relationen

Beispiel 17 *Selektiere die Vornamen aller Personen und Kinder.*

```

(SELECT vorname FROM personen) UNION
(SELECT vorname FROM kinder);

```

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

Beispiel 18 *Erzeuge ein Relationenschema mit den Attributen für Personen aus Beispiel 11 und dazu eine leere Relation mit dem Namen personen.*

```

CREATE TABLE personen
(pnr          INTEGER PRIMARY KEY,
vorname      VARCHAR(15) NOT NULL,
geschlecht   CHAR(1) CHECK (geschlecht IN ('w','m')),
gehalt       REAL DEFAULT 2000);

```

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

Beispiel 16 *Selektiere die Personennummern jener Personen, die ein Kind haben, dessen Vorname identisch mit dem Vornamen eines Einzelkindes einer anderen Person ist.*

```

SELECT k1.pnr
FROM kinder k1
WHERE k1.vorname
= ANY
(SELECT k2.vorname
FROM kinder k2
WHERE k1.pnr <> k2.pnr AND NOT EXISTS
(SELECT k3.vorname
FROM kinder k3
WHERE k2.vorname <> k3.vorname AND
k2.pnr = k3.pnr);

```

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

DDL (Data Definition Language)

- Erzeugen von Relationen;
- Festlegung von globalen Integritätsbedingungen;
- Ändern von Relationen;
- Entfernen von Relationen.

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

Integritätsbedingungen zur Relation

- CONSTRAINT-Klauseln mit eigenem Namen;
- REFERENCES-Klausel: Fremdschlüsselbedingung;
- "ON UPDATE *action*" und "ON DELETE *action*": Aktion, was beim Ändern oder Löschen des referenzierten Schlüssels geschehen soll.
- *action*:
- NO ACTION (nicht löschen);
- CASCADE (Ändern oder Löschen propagieren);
- SET NULL (Wert des referenzierenden Attributes auf einen Nullwert setzen);
- SET DEFAULT (Wert des referenzierenden Attributes auf den Defaultwert setzen).

Beispiel 19 Im nächsten SQL-Befehl erzeugen wir ein Relationenschema mit den Attributen für Kinder aus Beispiel 11 und dazu eine leere Relation mit dem Namen kinder.

```
CREATE TABLE kinder
  (pnr
    INTEGER,
    vorname
    VARCHAR(15),
    geschlecht
    CHAR(1),
    gebjahr
    INTEGER,
    CONSTRAINT kinder_pk PRIMARY KEY (pnr, vorname)
    CONSTRAINT pnr_fk FOREIGN KEY (pnr) REFERENCES personen (pnr)
    ON UPDATE CASCADE
    ON DELETE NO ACTION
    CONSTRAINT geschlecht_v CHECK (geschlecht IN ('w','m')));
```

Ändern und Löschen von Relationenschemata

- "ALTER TABLE relationName ADD attribute"
- "ALTER TABLE relationName DROP attribute"
- Beispiel 21 Im folgenden SQL-Befehl fügen wir das Attribut Geburtsort zur Relation personen hinzu.
ALTER TABLE personen ADD (geburtsort VARCHAR(12))
- "DROP TABLE relationName"

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 4.46

Beispiel 23 Füge alle Frauen aus der Relation personen in die Relation Frauen ein.

```
INSERT INTO Frauen (pnr, zugame, vorname, gehalt)
SELECT pnr, zugame, vorname, gehalt
FROM personen
WHERE geschlecht = 'w';
```

Alle Ergebnistupel einer beliebigen Abfrage werden eingefügt.

Beispiel 28 Füge alle Frauen aus der Relation personen in die Relation Frauen ein.

• "INSERT INTO relationName (attribute-1, ..., attribute-n) query";

Globale Integritätsbedingungen

"CREATE ASSERTION ass_name bool_expression [DEFERRABLE]"

- "bool-expression" muß in jedem Datenbankzustand erfüllt sein.
- DEFERRABLE:
 - Dynamische Überprüfung der Integritätsbedingung kann mit dem Befehl "SET CONSTRAINT constraintName DEFERRED" in der Relation, die ass_name verwendet, erst am Ende einer Transaktion ausgeführt werden.
 - "SET CONSTRAINT constraintName IMMEDIATE" andauernde Überprüfung der Integritätsbedingung.

DML (Data Manipulation Language)

- Einfügen von Tupeln in Relationen.
- Ändern von Attributwerten von Tupeln.
- Löschen von Tupeln aus Relationen.

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 4.47

Löschen

- "DELETE FROM relationName" löscht alle Tupel aus der Relation relationName ;
- "DELETE FROM relationName WHERE condition" löscht alle Tupel aus der Relation relationName für die die angegebene Bedingung condition erfüllt ist.

Beispiel 24 Lösche das zuvor eingefügte Tupel aus der Relation personen.

```
DELETE FROM personen
WHERE pnr=4;
```

Beispiel 20 Definiere die globale Integritätsbedingung, daß Personen mit einem Gehalt größer als 10.000 nur dann Familienbeihilfe für Kinder beziehen dürfen, wenn sie diese für mehr als zwei Kinder beziehen. Die Integritätsbedingung: DEFERRABLE, da während des Einfügens des ersten und zweiten von drei Kindern einer Person die Integritätsbedingung kurzfristig verletzt sein kann.

```
CREATE ASSERTION min_kinder
  (2 > ALL (SELECT COUNT(*)
    FROM person p, kinder k
    WHERE p.pnr = k.pnr AND p.gehalt > 10000
    GROUP BY p.pnr))
  DEFERRABLE;
```

Einfügen

- "INSERT INTO relationName (attribute-1, ..., attribute-n) VALUES (value-1, ..., value-n)"

Beispiel 22 Füge ein neues Tupel, das Information über Frau Fischer repräsentiert, in die Relation personen ein.

```
INSERT INTO personen (pnr, zugame, vorname, geschlecht, gehalt)
VALUES (4, 'Fischer', 'Ulrike', 'w', 1.800);
```

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 4.48

Ändern

- "UPDATE relationName SET attribute = expression WHERE condition"

Beispiel 25 Mit dem nächsten SQL-Befehl erhöhen wir das Gehalt aller Frauen um 10%.

```
UPDATE personen
SET gehalt = gehalt*1.1
WHERE geschlecht = 'w';
```

Views (Benutzeransichten)

- Logische Datenunabhängigkeit in SQL durch die Definition von Views (Sichtrelationen, Benutzeransichten)
- definiert auf Grund einer Abfrage über Basisrelationen und zuvor definierten Views.
- Die Typel von Basisrelationen sind gespeichert, die Typel von Views sind **virtuell**
→ sie werden auf Grund der bei der Definition der View angegebenen Abfrage berechnet.

Updaten über Views / Löschen von Views

- USER verweist auf den augenblicklichen Benutzer.
- Personaldaten.*
- Beispiel 27** Die View *persoensich* enthält für jeden Benutzer seine *Personaldaten*.
- ```
CREATE VIEW persoensich AS
SELECT *
FROM personen p
WHERE p.zuname = USER;
Löschen: "DROP VIEW relationName"
```

**Beispiel 28** Gewähre dem Benutzer Huber alle Rechte.

```
GRANT ALL PRIVILEGES
ON kleinverdiener
TO Huber
WITH GRANT OPTION;
```

**Beispiel 29** Jeder Benutzer lese seine *Personaldaten*.

```
GRANT SELECT
ON persoensich
TO PUBLIC;
```

**Views anlegen**

- "CREATE VIEW relationName (attribute-1, ..., attribute-n) AS query-expression"

**Beispiel 26** Mit dem folgenden SQL-Befehl definieren wir eine View *kleinverdiener*, die die *Personalmummer*, den *Zunamen* und das *Gehalt* jener *Personen* enthält, die *weniger als 400,- verdienen*.

```
CREATE VIEW kleinverdiener (pnr, zuname, vorname, gehalt) AS
SELECT pnr, zuname, vorname, gehalt
FROM personen
WHERE gehalt < 400
WITH CHECK OPTION;
```

**Zugriffskontrolle**

**Besitzer** einer Relation: der Erzeuger.

"GRANT privilege ON relationName TO authID"

"WITH GRANT OPTION": Recht auf Weitergabe der Rechte

*authID* = PUBLIC; Recht an alle (auch zukünftige) Benutzer.

**Zugriffskontrolle: Rechte zurücknehmen**

"REVOKE privilege ON relationName FROM authID"

rekursiv; auch für alle weitergegebenen Rechte.

**Beispiel 30** Benutzer Huber darf *Personalmummern* und *Vornamen* von *kleinverdiener* nicht mehr ändern.

```
REVOKE UPDATE(pnr, vorname)
ON kleinverdiener
FROM Huber;
```

**Updaten über Views**

- SQL erlaubt nur Änderungen über Attributen von Views, die auf einer Basisrelation definiert sind.
- Problematisch: Änderung eines Typels einer View, sodass dieses danach nicht mehr zur View gehört.
- "WITH CHECK OPTION" verhindert solche Änderungen.

**Beispiel 26** (fortgesetzt) Änderungen des Gehalts mit einem neuen Gehalt größer gleich 400,- sind über die View *kleinverdiener* nicht zugelassen.

*privilege:*

- SELECT bzw. SELECT(attribute-1, ..., attribute-n): Lesen von Typeln bzw. ausgewählten Attributwerten,
- INSERT: Einfügen,
- DELETE: Löschen,
- UPDATE bzw. UPDATE(attribute): Ändern von Typeln bzw. ausgewählten Attributwerten,
- REFERENCES(attribute-1, ..., attribute-n): Definition von Fremdschlüsseln auf Attributen,
- ALL PRIVILEGES: alle Rechte.

**Transaktionsverwaltung**

- Die meisten SQL-Befehle werden im Rahmen einer Transaktion verwendet.
- INSERT, UPDATE, SELECT, CREATE TABLE
- Transaktionen werden automatisch gestartet (Interaktiv), sie werden mit COMMIT erfolgreich abgeschlossen, mit ROLLBACK abgebrochen.



Viel verdienen die Mitarbeiter an Gehalt, Provision und insgesamt?  
 FROM mitarbeiter;  
 SELECT vname, name, gehalt, prov, gehalt + prov AS sum

| VNAME | NNAME  | GEHALT | PROV | SUM   |
|-------|--------|--------|------|-------|
| Gerti | Müller | 20000  | 1000 | 21000 |
| Anton | Eder   | 19000  | 500  | 19500 |
| Max   | Master | 13000  |      |       |
| Edith | Huber  | 15000  |      |       |
| Rita  | Master | 18000  |      |       |
| Lise  | Hofer  | 17000  | 900  | 17900 |
| Rudi  | Maier  | 21000  |      |       |
| Theo  | Novak  | 20000  |      |       |
| Karin | Meinl  | 25000  | 1200 | 26200 |

Wir wollen nun wissen, wieviel Gehalt und Provision die Mitarbeiter in jedem Restaurant bekommen und wieviele Mitarbeiter es gibt.  
 SELECT mnr, COUNT (gehalt+prov), SUM(gehalt+prov)  
 FROM mitarbeiter  
 GROUP BY mnr;

| MNR | COUNT(GEHALT+PROV) | SUM(GEHALT+PROV) |
|-----|--------------------|------------------|
| 1   | 2                  | 40500            |
| 2   | 0                  |                  |
| 3   | 2                  | 44100            |

Beispiel 31 Betrachte abt und ang

| abt (ABT# NAME) | ang (PERS# ABT# GEHALT) |
|-----------------|-------------------------|
| 2 Buchhaltung   | 1 50,000                |
| 3 Verkauf       | 2 20,000                |
| 4 Einkauf       | 3 10,000                |
|                 | 4 20,000                |

ang \* abt (ABT# NAME PERS# GEHALT)

|               |          |
|---------------|----------|
| 2 Buchhaltung | 1 50,000 |
| 2 Buchhaltung | 2 20,000 |
| 3 Verkauf     | 3 10,000 |
| 4 Einkauf     | 4 20,000 |
|               | ?        |
|               | ?        |
|               | ?        |

Wir wollen wissen, welche Mitarbeiter 500 ATS, 1.200 ATS Provision bekommen, oder von welchen Mitarbeitern keine Information vorhanden ist.  
 SELECT vname, name, prov  
 FROM mitarbeiter  
 WHERE prov IN (500, 1200, NULL);

| VNAME | NNAME | PROV |
|-------|-------|------|
| Anton | Eder  | 500  |
| Karin | Meinl | 1200 |

SELECT vname, name, prov  
 FROM mitarbeiter  
 WHERE prov IN (500, 1200) OR  
 prov IS NULL;

„normaler“ Verbund: jene Tupel von Relationen r, s werden verbunden, die in beiden Relationen auf den gemeinsamen Attributen denselben Wert annehmen.  
**Äußerer Verbund:** die Tupel, die nicht verbindbar (joinable) sind, werden mit Nullwerten aufgefüllt und zum Ergebnis dazugefügt.

Möglichkeit zur Angabe, welche der beiden Seiten beim Joinen mit Nullwerten aufgefüllt werden soll.  
**Outer-Join:** "SELECT attributes FROM rname1 [FULL | LEFT | RIGHT] OUTER JOIN rname2 ON joinCondition WHERE condition"  
 Beispiel 32 Selektiere die Personalnummern aller jener Personen, für die kein Gehalt bekannt ist, als auch die Vornamen gegebenenfalls vorhandener Kinder.  
 SELECT p.pnr, k.vorname  
 FROM personen p OUTER JOIN kinder k ON p.pnr=k.pnr  
 WHERE gehalt IS NULL;

Nun wollen wir eine Liste aller Mitarbeiter, deren Provision nicht 500 ATS oder 1.200 ATS beträgt, oder deren Provision kein Nullwert ist.  
 SELECT vname, name, prov  
 FROM mitarbeiter  
 WHERE prov NOT IN (500, 1200, NULL);

no rows selected

SELECT vname, name, prov  
 FROM mitarbeiter  
 WHERE prov NOT IN (500, 1200) AND  
 prov IS NOT NULL;

Seien  $r(R), s(S)$  Relationen mit  $R \cap S = X \neq \{\}$ , und  $?(X)$  das Tupel über  $X$ , bestehend nur aus Nullwerten.  
 $g = r \bowtie s, \quad r' = r \setminus \pi_R(g), \quad s' = s \setminus \pi_S(g)$   
 Der Äußere Verbund von  $r$  und  $s$  (outer join of  $r$  and  $s$ ) ist definiert als  
 $r * s = q \cup (r' \times \{?\} \cup (S \setminus X)) \cup (s' \times \{?\} \cup (R \setminus X))$   
 Der Äußere Verbund ist im Gegensatz zum Verbund nicht assoziativ!

turing-vollständige Sprache für die Definition und Wartung von persistenten, komplexen Objekten.  
**ANSI und ISO**  
 • Rekursive Abfragen  
 • Objektorientiertes Datenmanagement (object identifiers, Objekthierarchien, Vererbung (Inheritance), Datenkapselung, ...)  
 • benutzeredefinierte Datentypen, abstrakte Datentypen (ADT's)  
 • Triggers und Assertions  
 • Unterstützung für wissensbasierte Systeme

**Teil 1:** ANSI/ISO/IEC 9075-1-1999 Information Technology – Database Language – SQL Part 1: Framework (SQL/Framework). Überblick über den Standard und beschreibt das Zusammenspiel der einzelnen Komponenten von SQL.

**Teil 2:** ANSI/ISO/IEC 9075-2-1999 Information Technology – Database Language – SQL Part 2: Foundation (SQL/Foundation). Spezifikation aller Datentypen, Beschreibung der DDL, der DML, die Befehle zur Transaktionskontrolle und zum Verbindungsaufbau mit der Datenbank.

**Teil 3:** ANSI/ISO/IEC 9075-3-1999 Information Technology – Database Language – SQL Part 3: SQL/CLI (Call Level Interface). Schnittstelle zu DBMS, Beschreibung von Datenstrukturen und Prozeduren zur Ausführung von SQL-Statements aus Anwendungen.

## SQL-3 Standard

- **BOOLEAN** nimmt die Werte TRUE oder FALSE an.
- **LARGE BINARY OBJECT (BLOB)** und **LARGE CHARACTER OBJECT (CLOB)**: Objekte, die entweder nur aus Binärzeichen oder aus Characters bestehen.
- **ROW** („Zeile“) zur Darstellung komplexer Attribute.
- **ROW** („*attributeDefinitionList*“).
- **Abstrakte Datentypen**
- **Sub- und Supertabellen** `CREATE TABLE relationName UNDER relationName`.

```
CREATE TYPE mitarbeiter (
 PRIVATE mnr INTEGER,
 PUBLIC name name,
 PUBLIC vname name,
 PUBLIC adresse adresse,
 PUBLIC schilling,
 PRIVATE geburt DATE,
 PUBLIC alter INTEGER VIRTUAL);
FUNCTION name (mitarbeiter) RETURNS name
FUNCTION name (mitarbeiter, name) RETURNS mitarbeiter
```

**Teil 4:** ANSI/ISO/IEC 9075-4-1999 Information Technology – Database Language – SQL Part 4: SQL/PSM (Persistent SQL Modules). Syntax und Semantik von Stored Procedures und benutzerdefinierten Funktionen.

Befehle: `BEGIN END`, `WHILE`, `IF THEN ELSE`, `REPEAT`, `FOR` und `LOOP` und `CASE => SQL` wird computational vollständigen Sprache.

**Teil 5:** ANSI/ISO/IEC 9075-5-1999 Information Technology – Database Language – SQL Part 5: SQL/Bindings. Methoden zum Zugriff auf DBMS aus Programmen in gängigen Programmiersprachen.

**Teil 6:** SQL/Transactions. Mechanismen zur Interaktion von Programmen mit Resource- und Transaktionsmanager bei verteilten Transaktionen.

**Teil 7:** SQL/Temporal. Beschreibung von Datentypen und Befehlen, die mit Daten, die sich über die Zeit verändern, umgehen können.

**Teil 8:** SQL/Object. Wie greift DBMS auf Datentypen aus Anwendungsprogrammen zu.

```
CREATE TABLE kunde (vorname
 VARCHAR(20),
 nachname
 VARCHAR(20),
 ...
 unterschrift BLOB,
 adresse
 address)
 UNDER person;
```

```
CREATE DOMAIN adresse AS ROW (strasse
 VARCHAR(30),
 ort
 VARCHAR(30),
 plz
 INTEGER,
 land
 VARCHAR(30));
```

```
PUBLIC FUNCTION alter (m mitarbeiter) RETURNS INTEGER
BEGIN
 DECLARE m mitarbeiter;
 DECLARE a INTEGER;
 SET a = CAST(CURRENT_DATE - geburt(m));
 RETURN a;
END;
DESTRUCTOR FUNCTION destr_mitarbeiter (m mitarbeiter)
 RETURNS mitarbeiter
BEGIN
 DESTROY m;
 RETURN m;
```

- Fertigstellung:**
- SQL/CLI und SQL/PSM haben bei der Entwicklung als Zusätze zu SQL-2 höchste Priorität.
  - SQL/CLI ... 1995,
  - SQL/PSM ... 1997
  - Der gesamte SQL-3 Standard ... 1999.
  - Vollständige Implementierungen ... ?

## Abstrakte Datentypen

- Integration objektorientierter Konzepte im Relationalen Modell (Objekt-Relationales Modell)
- Objektivität, Vererbung und Polymorphismus. Datenkapselung
- Spezifikation der Attribute und Methoden

„**CREATE TYPE *typeName* (*attributeList*)**“ Methoden und Attribute sind `PUBLIC`, `PRIVATE` `PROTECTED`

**Virtuelle Attribute:** dehnt durch Funktion oder Prozedur

**Spezielle Methoden der ADTs:** Konstruktor, Destruktor

**Spezielle Funktionen der Attribute:** Observer und Iterator

```
CONSTRUCTOR FUNCTION mitarbeiter (r INTEGER, n name, v name)
 RETURNS mitarbeiter
BEGIN
 NEW m;
 SET m.mnr = r;
 SET m.name = n;
 SET m.vname = v;
 SET m.adresse = NULL;
 SET m.gehalt = 10000;
 SET m.prov = NULL;
 SET m.geburt = NULL;
 RETURN m;
END;
END FUNCTION
```

**Rekursion in SQL-3**

- SQL-2: Relationale Algebra, Relationenkalikül und QBE sind relational vollständig.
- *rekursive* Abfragen können im allgemeinen nicht formuliert werden.
- „Klassische“ Probleme wie:
  - „Finde alle Teile eines bestimmten Teils“ (bill of material),
  - „Traveling Salesman“

„WITH RECURSIVE *relationName* AS (*initialQuery* UNION ALL *recursiveQuery*) *query*“

**Duplikate:** ALL weglassen oder in *recursiveQuery* DISTINCT verwenden.

**Arbeitsübung:** SEARCH-Klausel: DEPTH FIRST BY, BREADTH FIRST BY  
*attributList* SET *ordnungAttribut*

**Zyklen:** CYCLE-Klausel zur Erkennung von Zyklen.

**Problem:** Verwendung der Negation in der Rekursion.

- Negation in *recursiveQuery* nur in vollständig bekannten Relationen
- INTERSECT und EXCEPT nur in vollständig bekannten Relationen
- Aggregatfunktionen nur in vollständig bekannten Relationen

Der folgende SQL Befehl erzeugt einen Trigger, der bei allen Mitarbeitern, die eine Gehaltserhöhung von mehr als 5% bekommen, die Provision auf 0 setzt.

```
CREATE TRIGGER provision
AFTER UPDATE OF gehalt ON mitarbeiter
REFERENCING OLD as gehalt_alt
NEW as gehalt_neu
FOR EACH ROW
WHEN gehalt_alt * 1.05 < gehalt_neu
UPDATE mitarbeiter
SET prov = 0;
```

Wir definieren einen Trigger, der, wenn ein Mitarbeiter eines Restaurants mehr als der Durchschnitt aller Mitarbeiter dieses Restaurants verdient, das Gehalt des Mitarbeiters um 200 ATS erhöht.

**Lieferung**

| von            | bis        | lnr | kosten |
|----------------|------------|-----|--------|
| Wien           | Wien       | 1   | 30     |
| Wien           | Wien       | 1   | 20     |
| Wien           | St. Pölten | 1   | 60     |
| Wien           | Linz       | 1   | 100    |
| St. Pölten     | Krems      | 2   | 70     |
| St. Pölten     | Meik       | 2   | 60     |
| St. Pölten     | Linz       | 2   | 80     |
| Klosterneuburg | Linz       | 3   | 20     |
| Tulln          | Krems      | 3   | 30     |
| Tulln          | Wien       | 4   | 50     |

```
WITH RECURSIVE alle_wege (von, bis, kosten) AS
(SELECT von, bis, kosten
FROM Lieferung
UNION ALL
SELECT a.von l.bis, a.kosten + l.kosten
FROM alle_wege a, Lieferung l
WHERE a.bis=l.von)
SEARCH BREADTH FIRST BY von SET ordnung
CYCLE bis SET zyklus TO '1' DEFAULT '0' USING zyklus
SELECT * FROM alle_wege
ORDER BY ordnung;
```

**Nullwerte und unvollständige Information**

Wir sind an einer Abfrage interessiert, die angibt, wieviel eine Lieferung in beliebig vielen Etappen von einem Ort zu einem anderen kostet.

Relation *alle\_wege*, die alle Tupel  $\langle X, Y \rangle$  enthält, für die es einen Weg von *X* nach *Y* gibt.

$$\text{Ein\_stop}(\text{start}, \text{ziel}) := \pi_{1,5}(\text{Lieferung} \mid \text{bis} = \text{von} \mid \text{Ein\_stop})$$

$$\text{Zwei\_stop}(\text{start}, \text{ziel}) := \pi_{1,5}(\text{Lieferung} \mid \text{bis} = \text{von} \mid \text{Ein\_stop}) \cup \dots$$

$$\text{Alle\_wege}(\text{start}, \text{ziel}) := \pi_{1,2}(\text{Lieferung} \cup \text{Ein\_stop} \cup \text{Zwei\_stop} \cup \dots)$$

**Trigger in SQL-3**

- Setzen von Aktionen auf der DB nach Änderungen derselben. (Sicherstellung der Datenintegrität)
- „Event-Condition-Action“ Modell: Definiere Trigger, die nach einem Event (UPDATE, INSERT oder DELETE) abhängig von bestimmten Bedingungen eine bestimmte Aktion durchführen
- Achtung beim Entwurf auf zyklische Triggerstrukture.

„CREATE TRIGGER *triggerName* BEFORE/AFTER *event* ON *table* WHEN *triggerCondition* *triggerAction*“

**Granularität:** FOR EACH ROW bzw. FOR EACH STATEMENT

**Datenzugriff:** REFERENCEING OLD *alteDaten* NEW *neueDaten* auf die Daten vor dem Event und danach.

- Modellierung unvollständiger Information.
- Adresse einer Person ist unbekannt (aber existent)
- Attribute sind eventuell nicht anwendbar, z.B. kein Telefon vorhanden.
- Theoretischer Hintergrund zu Nullwerten.
- Betrachten ein einfaches Modell, in dem nur ein Typ von Nullwerten auftritt.

Darstellung:

- ?
- $\omega$  (Omega)
- NULL

**Beispiel 201**

Die Relation  $r$  ist eine **Vervollständigung** (completion) der Relation  $s$ , wenn  $r$  keine Nullwerte mehr enthält und aus  $s$  dadurch erhalten wird, daß in  $s$  Nullwerte durch Werte aus dem Wertebereich ersetzt werden.

**Beispiel 202** Eine **Vervollständigung** der Relation aus Beispiel 201.

|                                                      | EHEM.  | EHEM.   | EHEM.  |
|------------------------------------------------------|--------|---------|--------|
| exjob (ANGESTELLTER GEHALT DIENSTGEBER ARBEIT GEHALT |        |         |        |
| Novak                                                | 39,500 | AUA     | 36,000 |
| Maler                                                | 24,100 | TU Wien | 22,050 |
| Müller                                               | 17,300 | TU Wien | 16,400 |
| Hüber                                                | 18,260 | TU Wien | 17,888 |

Ein Tupel  $t_i$ , das einen oder mehrere Nullwerte enthält, heißt **partiell** (partial).

Ein Tupel ohne Nullwerte heißt **total**, geschrieben  $t \uparrow$ .

Ein Attribut  $A$  heißt **bestimmt** (definite), geschrieben als  $A \downarrow$ , wenn der Wert von  $A$  nicht  $?$  ist.

Die Wahrheitstabellen sehen folgendermaßen aus:

|        |   |   |   |
|--------|---|---|---|
| $\vee$ | w | f | ? |
| w      | w | w | w |
| f      | f | f | f |
| ?      | w | f | ? |

|          |   |   |   |
|----------|---|---|---|
| $\wedge$ | w | f | ? |
| w        | w | f | f |
| f        | w | w | w |
| ?        | w | f | ? |

**Funktionale Abhängigkeiten und Nullwerte**

**Markierte Nullwerte** einer Relation  $r$  bzgl.  $F$ :

- Jeder in  $r$  auftretende Nullwert wird mit einer anderen Ziffer markiert.
- Mittels der Abhängigkeiten aus  $F$  wird festgestellt, ob Nullwerte identisch sein müssen, oder durch bekannte Werte ersetzt werden können.
- Das Ersetzen wird mittels der Füllregel für markierte Nullwerte durchgeführt (s.u.).

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 4.209

**Beispiel 203**

|                                                      | EHEM.  | EHEM.   | EHEM.   |
|------------------------------------------------------|--------|---------|---------|
| exjob (ANGESTELLTER GEHALT DIENSTGEBER ARBEIT GEHALT |        |         |         |
| Novak                                                | 39,500 | AUA     | 36,000  |
| Maler                                                | 24,100 | ?       | ?       |
| Müller                                               | 17,300 | TU Wien | Beamter |
| Hüber                                                | 18,260 | TU Wien | ?       |

Frage: gibt es einen Angestellten in exjob, dessen ehemaliger Dienstgeber die TU Wien ist und der dort Beamter war?

$f1: \exists x(R) \in \text{exjob} (x(\text{EHEM.DIENSTGEBER}) = \text{"TU Wien"})$   
 $x(\text{EHEM.ARBEIT}) = \text{"Beamter"}$

*Ergebnis: wahr*

Eine **schwere Verletzung** von  $X \rightarrow A$ : es gibt zwei Tupel  $u$  und  $v$ , die in ihren  $X$ -Werten übereinstimmen, deren  $A$ -Werte verschiedene bekannte Werte sind.

$u(x) = v(x)$  und  $u(A) \neq v(A)$ ,  $u(A) \uparrow, v(A) \uparrow$

Eine **leichte Verletzung (soft violation)** von  $X \rightarrow A$ : es gibt zwei Tupel  $u$  und  $v$ , die in ihren  $X$ -Werten übereinstimmen, aber nicht in ihren  $A$ -Werten übereinstimmen, und zumindest ein  $A$ -Wert ist ein markierter Nullwert.

$u(x) = v(x)$  und  $u(A) \neq v(A)$ ,  $(-u(A) \uparrow \text{ oder } -v(A) \uparrow)$

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 4.210

**Nullsubstitutionsprinzip**

- $\phi \dots$  (wohlgeformte) Formel des Relationen- bzw. Typalkalküls über  $r$ .
- Gesucht: Interpretationsfunktion**  $I$ , die  $\phi$  in wahr, falsch,  $?$  abbildet.
- $I(\phi) \dots$  übliche Interpretationsfunktion, bildet  $\phi$  über totalen Relationen in  $\{w, f\}$  ab.
- $I$  muß  $I$  erweitern:  $I/\text{totale Relationen} = J$ .

$$I(\phi) = \begin{cases} \text{wahr, wenn } J(\phi) \text{ für jede Vervollständigung von } r \text{ wahr ist.} \\ \text{falsch, wenn } J(\phi) \text{ für jede Vervollständigung von } r \text{ falsch ist.} \\ \text{?; sonst.} \end{cases}$$

**Füllregel für markierte Nullwerte**

Verursachen die Tupel  $u$  und  $v$  eine leichte Verletzung von  $X \rightarrow A$  in  $r$ , so ersetze

- den Nullwert mit dem höheren Index durch den mit dem niederen, falls sowohl  $u(A)$  als auch  $v(A)$  ein Nullwert ist.
- den Nullwert von  $u(A)$  bzw.  $v(A)$  durch den bekannten Wert  $v(A)$  bzw.  $u(A)$ , wenn einer der Werte bekannt ist.

Wiederholte Anwendung der Füllregel:  $r$  geht in endlich vielen Schritten in  $r'$  über, so daß  $r'$  ohne leichte Verletzung ist.

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 4.211



**Logische Semantik von Datalog**

R... Datalog-Regel der Form  $L_0 :- L_1, L_2, \dots, L_n$ ;  $L_i$  Literal der Form  $p_i(t_1, \dots, t_{n_i})$

$x_1, x_2, \dots, x_k$  alle Variablen, die in R vorkommen.

$R^* = \exists x_1 \exists x_2 \dots \exists x_n ((L_1 \vee L_2 \vee \dots \vee L_n) \Rightarrow L_0)$

$P^* = R_1^1 \wedge R_2^2 \vee \dots \vee R_n^n$

$P \dots$  Datalog-Programm mit den Regeln  $R_1, R_2, \dots, R_n$

Es wird also jedem Datalog-Programm  $P$  in eindeutiger Weise eine Formel  $P^*$  zugeordnet.

DBVO 5.13

Gerald Pfeifer TU Wien http://gerald.pfeifer.com

**Beispiel 2 Datenbank DB mit den Relationen**  $\text{Frau}(\text{NAME}), \text{mann}(\text{NAME}), \text{eltern}(\text{EL-TEIL, KIND}), \text{Grote}(\text{NAME}), \text{Hans}(\text{NAME}), \text{Karl}(\text{NAME}), \text{Linda}(\text{NAME}), \text{Gerti}(\text{NAME}), \text{Michael}(\text{NAME}), \text{Karl}(\text{NAME}), \text{Linda}(\text{NAME}), \text{Gerti}(\text{NAME}), \text{Michael}(\text{NAME})$

$\text{groBvater}(X, Y) :- \text{mann}(X), \text{eltern}(X, Z), \text{eltern}(Z, Y)$ .

**Einschränkungen zur Syntax von Datalog**

- <RelationenId>: Name einer bereits bekannten, in der Datenbank abgespeicherten Relation (z.B.  $\text{eltern}$ ),
- Name einer neuen Relation, die durch das Datalog-Programm definiert wird (z.B.  $\text{vorfahre}$ )
- Vergleichsprädikate wie =, <, >, <=, >, formal behandelt wie bekannte Datenbankrelationen.
- Namen bereits existierender Relationen dürfen nur in Regelköpfen vorkommen.

DBVO 5.14

Gerald Pfeifer TU Wien http://gerald.pfeifer.com

*Ein Fakt ist z.B.  $\text{eltern}(\text{Linda}, \text{Gerti})$*

$DB^* = \text{frau}(\text{grote}) \wedge \text{frau}(\text{linda}) \wedge \text{frau}(\text{gerti}) \wedge \text{mann}(\text{hans}) \wedge \text{mann}(\text{karl}) \wedge \text{mann}(\text{michael}) \wedge \text{eltern}(\text{hans}, \text{linda}) \wedge \text{eltern}(\text{grote}, \text{linda}) \wedge \text{eltern}(\text{karl}, \text{michael}) \wedge \text{eltern}(\text{linda}, \text{michael}) \wedge \text{eltern}(\text{karl}, \text{gerti}) \wedge \text{eltern}(\text{linda}, \text{gerti})$

$P^* = \text{A} \vee \text{X} \vee \text{Y} \vee \text{A} \vee \text{Z} : ((\text{mann}(\text{X}) \vee \text{eltern}(\text{X}, \text{Z}) \vee \text{eltern}(\text{Z}, \text{Y})) \Rightarrow \text{groBvater}(\text{X}, \text{Y}))$

Neue Fakten in  $\text{cons}(P^* \wedge DB^*)$  :  $\text{groBvater}(\text{hans}, \text{michael}), \text{groBvater}(\text{hans}, \text{gerti})$ .

- Jede Variable, die im Kopf einer Datalog-Regel vorkommt, muß auch im Rumpf der selben Regel vorkommen.
- Variablen, die als Argumente spezieller Vergleichsprädikate vorkommen, müssen im selben Regelrumpf auch in Literalen ohne Vergleichsprädikate vorkommen.
- Jedes verwendete <RelationId> tritt immer mit der selben Anzahl von Argumenten auf.
- Datalog-Abfragen werden auch "Datalog-Programme" genannt.

DBVO 5.15

Gerald Pfeifer TU Wien http://gerald.pfeifer.com

*Das Datalog-Programm P definiert eine neue Relation groBvater, mit der aktuellen Instanz*

$\text{groBvater}(X, Y)$

$M[P] : DB \rightarrow \text{cons}(P^* \wedge DB^*)$

**Operationale Semantik von Datalog**

- Datalog-Regeln als einfache Inferenzregeln,

- Ein Fakt, der im Kopf der Regel auftritt, kann abgeleitet werden, wenn die Fakten, die im Rumpf auftreten, ableitbar sind.

**Beispiel 3** Aus den Fakten  $\text{eltern}(\text{linda}, \text{michael}), \text{eltern}(\text{linda}, \text{michael}), \text{eltern}(\text{linda}, \text{gerth})$  und der Regel

$\text{geschwister}(\text{michael}, \text{gerth}) :-$

$\text{eltern}(\text{linda}, \text{michael}), \text{eltern}(\text{linda}, \text{gerth}).$

kann der Fakt

$\text{geschwister}(\text{michael}, \text{gerth})$

gewonnen werden.

**Idee:** Anwendung eines Datalog-Programms  $P$  auf eine Datenbank  $DB$ :

iteriertes Ableiten von Fakten bis zur Sättigung.

**Formalisierung:**

- definierte Operator  $T_P(DB)$ : erweitert  $DB$  um alle jene Fakten, die in einem Schritt durch die Anwendung der Regeln aus  $P$  erhalten werden können.

- $T_P^i(DB) = T_P(T_P^{i-1}(DB))$  drückt die wiederholte Anwendung von  $T_P$  aus.

$$T_P^i(DB) = DB \cup \bigcup_{R \in P} \{L_0 \mid L_0 : -L_1, \dots, L_n \in \text{Ground}(R; P, DB), L_1, \dots, L_n \in DB\}$$

**Algorithmus INFER**

INPUP: Datalog-Programm  $P$ , Datenbank  $DB$   
 OUTPUP:  $T_P^i(DB)$  ( $= \text{cons}(P^* \wedge DB^*)$ )

Schritt 1.  $GP := \bigcup_{R \in P} \text{Ground}(R; P, DB)$ , (\*  $GP$  ist die Menge aller Grundinstanzen \*)

Schritt 2.  $OLD := \{\}$ ;  $NEW := DB$ ;

Schritt 3. while  $NEW \neq OLD$  do begin

$OLD := NEW$ ;  $NEW := TP(OLD)$ ;

end;

Schritt 4. output  $OLD$ .

- Regel  $R$  mit Variablen: Repräsentation aller variablenfreien Regeln, die durch Ersetzung der Variablen mit Konstantensymbolen gewonnen werden.
- Die Konstante kommen im Programm  $P$  oder in der Datenbank  $DB$  vor.
- **Grundinstanz** von  $R$  bzgl.  $P$  und  $DB$ : eine solche Ersetzung.
- $\text{Ground}(R; P, DB) \dots$  Menge aller Grundinstanzen von  $R$ .

$$\begin{aligned} T_P^0(DB) &= DB \\ T_P^1(DB) &= T_P(T_P^0(DB)) = T_P(DB) \\ T_P^2(DB) &= T_P(T_P^1(DB)) = T_P(T_P(DB)) \\ &\dots \\ T_P^i(DB) &= T_P(T_P^{i-1}(DB)) = T_P(\dots T_P(DB)) \end{aligned}$$

**Subroutine TP**

INPUP: Fakttenmenge  $OLD$

OUTPUP:  $T_P(OLD)$

Schritt 1.  $F := OLD$ ;

Schritt 2. for each rule  $L_0 : -L_1, \dots, L_n$  in  $GP$  do  
 if  $L_1, \dots, L_n \in OLD$  then  $F := F \cup \{L_0\}$ ;

Schritt 3. return  $F$ ;

**Beispiel 4** Alle Geschwisterbeziehungen:

$\text{geschwister}(Y, Z) :- \text{eltern}(X, Y), \text{eltern}(X, Z), Y \neq Z.$

Die 6<sup>3</sup> Grundinstanzen dieser Regel bezüglich  $P$  und  $DB$  aus Beispiel 2 sind (Konstantensymbole = grete, linda, gerth, hans, michael, karl):

$\text{geschwister}(\text{grete}, \text{grete}), \text{eltern}(\text{grete}, \text{grete}), \text{grete} \langle \rangle \langle \rangle \text{grete}$  ( $X = Y = Z = \text{grete}$ ),  
 $\text{geschwister}(\text{grete}, \text{linda}), \text{eltern}(\text{grete}, \text{linda}), \text{grete} \langle \rangle \langle \rangle \text{linda}$  ( $X = Y = Z = \text{grete}, Z = \text{linda}$ ),  
 $\dots$

$\text{geschwister}(\text{karl}, \text{karl}), \text{eltern}(\text{karl}, \text{karl}), \text{karl} \langle \rangle \langle \rangle \text{karl}$  ( $X = Y = Z = \text{karl}$ )

gewonnen werden.

- Die Faktenmenge wächst monoton d.h. es gilt  $T_P^i(DB) \subseteq T_P^{i+1}(DB)$ .

Die Folge  $(T_P^i(DB))$  konvergiert endlich:

es gibt ein  $n$  so daß  $T_P^n(DB) = T_P^{n+1}(DB)$  für alle  $m \geq n$ .

- $T_P^i(DB) \dots$  Faktenmenge, zu der  $(T_P^i(DB))$  konvergiert

Resultat der Anwendung von  $P$  auf  $DB$ .

- Die operationale Semantik eines Datalog-Programms  $P$  ordnet jeder Datenbank  $DB$  die Faktenmenge  $T_P^i(DB)$  zu:

$$O[P] : DB \rightarrow T_P^i(DB).$$

- Es gilt:

$$M[P](DB) = \text{cons}(P^* \wedge DB^*) = O[P](DB) = T_P^i(DB).$$

**Beispiel 5** Wende Programm  $P$

$\text{vorfahre}(X, Y) :- \text{eltern}(X, Y).$

$\text{vorfahre}(X, Z) :- \text{eltern}(X, Y), \text{vorfahre}(Y, Z).$

zur Berechnung der Vorfahren auf die Datenbank  $DB$  aus Beispiel 2 an.

Schritt 1. von INFER: Bildung von  $GP$

$GP = \{\text{vorfahre}(\text{grete}, \text{grete}) :- \text{eltern}(\text{grete}, \text{grete}), \text{vorfahre}(\text{grete}, \text{linda}), \text{vorfahre}(\text{grete}, \text{grete}) :- \text{eltern}(\text{grete}, \text{grete}), \text{vorfahre}(\text{grete}, \text{grete}) :- \text{eltern}(\text{grete}, \text{linda}), \dots\}$ .

$\text{vorfahre}(\text{grete}, \text{grete}), \text{vorfahre}(\text{grete}, \text{grete}), \text{vorfahre}(\text{grete}, \text{linda}), \text{vorfahre}(\text{grete}, \text{grete}), \dots$ .

(Insgesamt gibt es  $6^2 + 6^3 = 252$  Grundinstanzen.)

Schritt 2.  $OLD := \{ \}$ ,  $NEW := DB$ ;

Schritt 3.  $OLD \neq NEW$

Erster Durchlauf:  $OLD := DB, NEW := TP(OLD) = TP(DB)$   
 $TP(OLD) = OLD \cup \{ \text{vorfahre}(A, B) \mid \text{eltern}(A, B) \in DB \}$ ;

Zweiter Durchlauf:  
 $OLD := TP(DB), NEW := TP(OLD) = TP(TP(DB))$   
 $TP(OLD) = TP(OLD) \cup \{ \text{vorfahre}(\text{hans}, \text{michael}), \text{vorfahre}(\text{hans}, \text{gertr}), \text{vorfahre}(\text{grete}, \text{michael}), \text{vorfahre}(\text{grete}, \text{gertr}) \}$ ;

Dritter Durchlauf:  $TP(OLD) = OLD$ , es können keine neuen Fakten mehr abgeleitet werden.

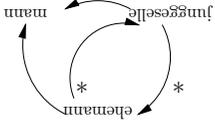
Schritt 4. Ausgabe von  $OLD$ . Das Resultat entspricht der Erweiterung der Datenbank  $DB$  um die neue Tabelle *vorfahre*.

### Graphendarstellung

- $P$  ... ein erweitertes Datalog-Programm, mit negierten Literalen im Regelrumpf.
- $DEP(P)$  ... gerichteter Graph, mit:
  - Knoten  $\dots, R \rightarrow S$ , wenn es eine Regel in  $P$  gibt, deren Kopfrädikant  $r$  – Kanthen  $\dots, R \rightarrow S$ , wenn es eine Regel in  $P$  gibt, deren Kopfrädikant  $r$  ist, und die im Rumpf das Prädikat  $s$  enthält.
  - Markiere jede Kante  $R \rightarrow S$  von  $DEP(P)$  mit einem Stern “\*”, wenn das Prädikat  $s$  im Rumpf negiert ist.

Beispiel 7 Datalogprogramm mit den Regeln:

$\text{ehemann}(X) :- \text{mann}(X), \text{non\_jungeselle}(X)$ .  
 $\text{jungeselle}(X) :- \text{mann}(X), \text{non\_ehemann}(X)$ .



Dieses Programm ist nicht stratifiziert.

| eltern | (EL-TEIL | KIND) | vorfahre | (AHNE   | NAME) |
|--------|----------|-------|----------|---------|-------|
| Hans   | Linda    |       | Hans     | Linda   |       |
| Grete  | Linda    |       | Grete    | Linda   |       |
| Karl   | Michael  |       | Karl     | Michael |       |
| Linda  | Michael  |       | Linda    | Michael |       |
| Karl   | Gerti    |       | Karl     | Gerti   |       |
| Linda  | Gerti    |       | Linda    | Gerti   |       |
| Hans   | Gerti    |       | Hans     | Gerti   |       |
| Grete  | Michael  |       | Grete    | Michael |       |
| Grete  | Gerti    |       | Grete    | Gerti   |       |

- Ein um Negationen erweitertes Datalog-Programm  $P$  ist zulässig  $\Leftrightarrow$  im Graph  $DEP(P)$  kein gerichteter Kreis vorkommt, der eine durch “\*” markierte Kante enthält.
- Ein solches Programm wird auch **stratifiziert** genannt, da es bezüglich der Negation in eine Hierarchie von Schichten (Strata) zerlegt werden kann.

### Die Schichtendarstellung

Eine Schicht besteht aus der größtmöglichen Menge von Prädikaten, für die gilt:

1. Kommt ein Prädikat  $p$  im Kopf einer Regel vor, die im Rumpf ein negiertes Prädikat  $q$  enthält, so ist  $p$  in einer höheren Schicht als  $q$ .
2. Kommt ein Prädikat  $p$  im Kopf einer Regel vor, die im Rumpf ein nicht negiertes Prädikat  $q$  enthält, so ist  $p$  in einer mindestens so hohen Schicht wie  $q$ .

Schritt 2.  $OLD := \{ \}$ ,  $NEW := DB$ ;

Schritt 3.  $OLD \neq NEW$

Erster Durchlauf:  $OLD := DB, NEW := TP(OLD) = TP(DB)$   
 $TP(OLD) = OLD \cup \{ \text{vorfahre}(A, B) \mid \text{eltern}(A, B) \in DB \}$ ;

Zweiter Durchlauf:  
 $OLD := TP(DB), NEW := TP(OLD) = TP(TP(DB))$   
 $TP(OLD) = TP(OLD) \cup \{ \text{vorfahre}(\text{hans}, \text{michael}), \text{vorfahre}(\text{hans}, \text{gertr}), \text{vorfahre}(\text{grete}, \text{michael}), \text{vorfahre}(\text{grete}, \text{gertr}) \}$ ;

Dritter Durchlauf:  $TP(OLD) = OLD$ , es können keine neuen Fakten mehr abgeleitet werden.

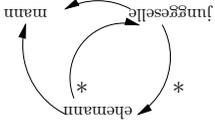
Schritt 4. Ausgabe von  $OLD$ . Das Resultat entspricht der Erweiterung der Datenbank  $DB$  um die neue Tabelle *vorfahre*.

### Graphendarstellung

- $P$  ... ein erweitertes Datalog-Programm, mit negierten Literalen im Regelrumpf.
- $DEP(P)$  ... gerichteter Graph, mit:
  - Knoten  $\dots, R \rightarrow S$ , wenn es eine Regel in  $P$  gibt, deren Kopfrädikant  $r$  – Kanthen  $\dots, R \rightarrow S$ , wenn es eine Regel in  $P$  gibt, deren Kopfrädikant  $r$  ist, und die im Rumpf das Prädikat  $s$  enthält.
  - Markiere jede Kante  $R \rightarrow S$  von  $DEP(P)$  mit einem Stern “\*”, wenn das Prädikat  $s$  im Rumpf negiert ist.

Beispiel 7 Datalogprogramm mit den Regeln:

$\text{ehemann}(X) :- \text{mann}(X), \text{non\_jungeselle}(X)$ .  
 $\text{jungeselle}(X) :- \text{mann}(X), \text{non\_ehemann}(X)$ .



Dieses Programm ist nicht stratifiziert.

### Erweitertes Datalog mit Negation

• Datalog ist nicht relational vollständig (Differenzen zwischen Relationen).

- Durch Einführung des Negationszeichens (**non**) in Regelrumpfen kann dieser Mangel behoben werden.
- Einschränkung bei der Verwendung der Negation: Keine Relation darf aufgrund ihrer eigenen Negation definiert werden.

• Überprüfung dieser Bedingung: durch graphentheoretische Hilfsmittel.

### Beispiel 6 Datalogprogramm mit den Regeln:

$\text{ehemann}(X) :- \text{mann}(X), \text{verheiratet}(X)$ .  
 $\text{jungeselle}(X) :- \text{mann}(X), \text{non\_ehemann}(X)$ .

*Dieses Programm ist stratifiziert.*

### INPT: Eine Menge von Datalog-Regeln.

**OUTPUT:** Eine Entscheidung, ob das Programm stratifiziert ist, und wenn ja, eine Einteilung der Prädikate in Schichten.

**Methode:** 1. Setze Schicht für alle Prädikate auf 1.  
 2. Wiederhole für jede Regel  $R$  mit Prädikat  $p$  im Kopf:  
 • hat  $R$  im Rumpf ein negiertes Prädikat  $q$ ,  $p$  aus Schicht  $i$ ,  $q$  aus Schicht  $j$ , und  $i \leq j \Rightarrow i := j + 1$ .  
 • hat  $R$  im Rumpf ein nicht negiertes Prädikat  $q$ ,  $p$  aus Schicht  $i$ ,  $q$  aus Schicht  $j$ , und  $i < j \Rightarrow i := j$ .  
 bis gilt:  
 • Zustand stabil  $\Rightarrow$  Programm stratifiziert.  
 • Schicht  $n \geq \#$  Prädikate  $\Rightarrow$  nicht stratifiziert.



$X, Y$  Teilmengen von  $R$ .  
 Relation  $r(R)$  erfüllt die **funktionale Abhängigkeit (FD)**  $X \rightarrow Y$ , wenn  $\forall u, v \in r(R)$  gilt:

$$u(X) = v(X) \Rightarrow u(Y) = v(Y).$$

$X \dots$  linke Seite (left hand side, LHS),

$Y \dots$  rechte Seite (right hand side, RHS) von FD.

$$X \rightarrow Y \Leftrightarrow \forall u, v \in r(R) \quad u(X) = v(X) \Rightarrow u(Y) = v(Y)$$

**Ableitungsregeln für funktionale Abhängigkeiten**

**Beispiel 2**  $F = \{\text{PERSONALNR} \rightarrow \text{ARTEILUNG}, \text{ARTEILUNG} \rightarrow \text{ADRESSE}\}$  eine Menge von *funktionalen Abhängigkeiten*.

$\Rightarrow$  PERSONALNR  $\rightarrow$  ADRESSE.

**Axiomatisierung:** ein System von Ableitungsregeln, mit deren Hilfe *alle* durch  $F$  implizierten funktionalen Abhängigkeiten  $f$  abgeleitet werden können:

$$F \models f \Leftrightarrow F \vdash f.$$

**Anmerkung:**

- $\models$  ... Gültigkeitsbegriff, modelltheoretisch/semantisch
- $\vdash$  ... Ableitbarkeitsbegriff, beweistheoretisch/syntaktisch

$$F^+ = \{f \mid F \models f\}$$

Die Menge aller funktionalen Abhängigkeiten, die von  $F$  auf  $R$  impliziert werden, heißt die **Hülle** (closure) von  $F$  auf  $R$ ,  $F^+_R$ .

Eine funktionale Abhängigkeit  $X \rightarrow Y$  heißt **trivial**, wenn  $Y$  eine Teilmenge von  $X$  ist.

$$X \rightarrow Y \text{ trivial} \Leftrightarrow Y \subseteq X.$$

Unter der **Hülle einer Menge von Attributen**  $X \subseteq R$ , geschrieben als  $X^+$ , bezüglich  $F$  verstehen wir die Menge aller Attribute  $A$ , sodaß  $X \rightarrow A$  aus  $F^+$ .

**Beispiel 1**  $F = \{\text{PERSONALNR} \rightarrow \text{ARTEILUNG}, \text{ARTEILUNG} \rightarrow \text{ADRESSE}\}$  Menge funktionaler Abhängigkeiten

| PERSONALNR | ARTEILUNG   | ADRESSE   |
|------------|-------------|-----------|
| 12         | Buchhaltung | 1010 Wien |
| 13         | Verkauf     | 1030 Wien |
| 11         | Filiale     | 8010 Graz |
| 14         | Buchhaltung | 1010 Wien |

FDs erfüllt

| PERSONALNR | ARTEILUNG   | ADRESSE       |
|------------|-------------|---------------|
| 12         | Buchhaltung | 1010 Wien     |
| 13         | Verkauf     | 1030 Wien     |
| 11         | Filiale     | 8010 Graz     |
| 14         | Buchhaltung | 5010 Salzburg |

ARTEILUNG  $\rightarrow$  ADRESSE nicht erfüllt

**Ableitungsregeln (inference axioms)**

- Armstrong (1974)
  - sind **korrekt (sound)**, d.h. sie leiten von  $F$  nur wirklich **gültige** FDs ab ( $F \vdash f \Rightarrow F \models f$ ).
  - sind auch **vollständig (complete)**, d.h. sie erzeugen bei wiederholter Anwendung **alle** von  $F$  implizierten Abhängigkeiten ( $F \models f \Rightarrow F \vdash f$ ).
- Beweis: [Ullman, 1988 Bd. I bzw. Maier, 1983]

**Beispiel 3** Sei  $G$  die folgende Menge funktionaler Abhängigkeiten:

- $g1$  { PERSONALNR, ABT  $\rightarrow$  ORT
- $g2$  PERSONALNR  $\rightarrow$  ABT
- $g3$  ABT  $\rightarrow$  ORT }

$G^+ = G \cup$  alle trivialen Abhängigkeiten  $\cup$

*Transitivität*  $g2, g3$

*Vereinigung*  $g2, g4$

*Erweiterung*  $g5$

*Erweiterung*  $g2$

*Zerlegung*  $g7$

$R$  Relationenschema.

Eine Teilmenge  $X$  von Attributen in  $R$  ist ein **Oberschlüssel** für  $R$ , genau dann wenn  $X$  die funktionale Abhängigkeit  $X \rightarrow R$  erfüllt.

$$X \subseteq R \text{ Oberschlüssel für } R \Leftrightarrow X \rightarrow R.$$

Ist die Teilmenge  $X$  minimal, dh. gilt für jede Teilmenge  $Y \subset X$ , daß sie die funktionale Abhängigkeit  $Y \rightarrow R$  nicht erfüllt, so ist  $X$  ein **Schlüssel**.

$$X \text{ Schlüssel für } R \Leftrightarrow X \rightarrow R, \text{ und } \forall A \in X (X \setminus A \not\rightarrow R).$$

$X \setminus A \dots$  Mengendifferenz  $X - A$ .

F1.: Reflexivität (reflexivity)

$$Y \subseteq X \vdash X \rightarrow Y$$

F2.: Erweiterung (augmentation)

$$X \rightarrow Y \vdash XZ \rightarrow YZ$$

F3.: Vereinigung (additivity)

$$X \rightarrow Y, X \rightarrow Z \vdash X \rightarrow YZ$$

F4.: Zerlegung (projectivity)

$$X \rightarrow YZ \vdash X \rightarrow Y, X \rightarrow Z$$

F5.: Transitivität (transitivity)

$$X \rightarrow Y, Y \rightarrow Z \vdash X \rightarrow Z$$

F6.: Pseudotransitivität (pseudotransitivity)

$$X \rightarrow Y, YW \rightarrow Z \vdash XW \rightarrow Z$$

**Das Membership-Problem**

Gegeben eine Menge von funktionalen Abhängigkeiten  $F$ .

**Frage:** welche zusätzlichen Abhängigkeiten werden durch  $F$  impliziert?

**Algorithmen:**

**CLOSURE:** berechnet die Hülle einer Menge von Attributen.

**MEMBER:** stellt fest, ob eine funktionale Abhängigkeit in der Hülle enthalten ist.

## Algorithmus CLOSURE

INPUT:  $X$  Attributmengge,  $F$  Menge von FDsOUTPUT:  $X^+$  $X^+ := X$ while  $\exists (Y \rightarrow Z) \in F$  mit  $Y \subseteq X^+$  und  $Z \not\subseteq X^+$  do $X^+ := X^+ \cup Z$  endreturn  $(X^+)$ 

## Äquivalenz von Systemen funktionaler Abhängigkeiten

**Problem:** gesucht ist eine möglichst knappe Darstellung für funktionale Abhängigkeiten. $F, G$  zwei Mengen funktionaler Abhängigkeiten. $G$  und  $F$  sind äquivalent,  $G \equiv F$ , wenn sie dieselbe Hülle besitzen.  $F$  heißtdann Überdeckung (cover) von  $G$ .Wann gilt  $F \equiv G$ ?1. Prüfe für alle  $g \in G$ , ob  $F \models g$ ,2. prüfe für alle  $f \in F$ , ob  $G \models f$ .Eine Menge funktionaler Abhängigkeiten  $F$  heißt linksreduziert, wenn sie keine FD  $X$  mit überflüssigen Attributen auf der linken Seite enthält.Eine Menge funktionaler Abhängigkeiten  $F$  heißt minimal, wenn sie

kanonisch, nichtredundant und linksreduziert ist.

**Theorem:** Zu jeder Menge funktionaler Abhängigkeiten  $F$  gibt es eine minimale Überdeckung. (Beweis in [Ullman 1988, Bd I])

$$B \text{ ist überflüssig in } XBY \rightarrow A \text{ bzgl. } F \\ \Leftrightarrow F \setminus \{XBY \rightarrow A\} \cup \{XY \rightarrow A\} \equiv F.$$

## Algorithmus MEMBER

INPUT:  $X \rightarrow Y$  FD,  $F$  Menge von FDsOUTPUT: true, if  $F \models X \rightarrow Y$ false, if  $F \not\models X \rightarrow Y$ return  $(Y \subseteq \text{CLOSURE}(X,F))$ Beispiel 4  $F \equiv G$ ? $F = \{ \text{PERSONALNR ABT} \rightarrow \text{ORT}$  $\text{PERSONALNR} \rightarrow \text{ABT}$  $\text{ABT} \rightarrow \text{ORT}$  $\text{PERSONALNR} \rightarrow \text{ORT} \}$  $G = \{ \text{PERSONALNR ABT} \rightarrow \text{ORT}$  $\text{PERSONALNR} \rightarrow \text{ABT}$  $\text{ABT} \rightarrow \text{ORT} \}$ 1. Prüfe für alle  $g \in G$ , ob  $F \models g$ . Antwort: ja, trivialerweise ( $G \subseteq F$ ).2. Prüfe für alle  $f \in F$ , ob  $G \models f$ .Frage: gilt  $G \models \{ \text{PERSONALNR} \rightarrow \text{ORT} \}$  ? Antwort: ja, folgt aus derTransitivität von  $g_2$  und  $g_3$ . $F \equiv G$ .

## Algorithmus zur Berechnung der minimalen Überdeckung:

1. Zerlege alle FDs in kanonische.

2. Linksreduktion: entferne alle überflüssigen Attribute:

Für jede FD  $AXB \rightarrow Y \in F$ : $X \rightarrow Y \in F^+ \Rightarrow$  entferne  $B$  aus  $XB \rightarrow Y$ .3. Entferne aller redundanten Abhängigkeiten  $X \rightarrow Y$ :Für jede FD  $X \rightarrow Y \in F$ : $X \rightarrow Y \in \{ F \setminus \{ X \rightarrow Y \} \}^+ \Rightarrow$  entferne  $X \rightarrow Y$  aus  $F$ .

## Effiziente Implementierung von CLOSURE (Grundidee):

• für jedes Attribut: Referenzliste, die angibt, in welchen FDs aus  $F$ 

dieses Attribut auf der linken Seite vorkommt (ATTRLIST)

• zu jeder FD: Zähler, der angibt, wieviele Attribute der linken Seite noch nicht in  $X^+$  sind (ZÄHLER)

• Wird der ZÄHLER für eine FD null

 $\Rightarrow$  die linke Seite von FD ist im augenblicklichen  $X^+$  enthalten $\Rightarrow$  die Attribute der rechten Seite von FD werden zu  $X^+$  hinzugefügt.Für jedes neue  $A$  in  $X^+$  muß dessen ATTRLIST – die angibt, in welchen FDs  $A$  in der linken Seite vorkommt – durchgegangen werden.In diesen FDs muß ZÄHLER um 1 erniedrigt werden, da  $A$  in  $X^+$ Eine Menge funktionaler Abhängigkeiten  $F$  heißt kanonisch, wenn für jedeFD  $X \in F$  die rechte Seite aus nur einem Attribut besteht.Regel F4  $\Rightarrow$  Zerlegung trivial.Eine Menge funktionaler Abhängigkeiten  $F$  heißt nichtredundant, wenn eskeine FD  $f$  in  $F$  gibt, so daß  $F \setminus \{f\}$  äquivalent zu  $F$  ist.

$$(X \rightarrow A) \text{ redundant in } F \Leftrightarrow F \setminus \{X \rightarrow A\} \equiv F.$$

## Beispiel 5

 $G = \{ \text{PERSONALNR} \rightarrow \text{ABT ORT}$  $\text{ABT} \rightarrow \text{ORT}$  $\text{PERSONALNR ABT} \rightarrow \text{NAME} \}$ 1. Berechnung von  $G'$  kanonisch. $G' = \{ \text{PERSONALNR} \rightarrow \text{ABT}$  $\text{PERSONALNR} \rightarrow \text{ORT}$  $\text{ABT} \rightarrow \text{ORT} \}$  $\text{PERSONALNR ABT} \rightarrow \text{NAME} \}$

2. Berechnung von  $G''$  linksreduziert.

$$G'' = \{ \text{PERSONALNR} \rightarrow \text{ABT} \quad g1$$

$$\text{PERSONALNR} \rightarrow \text{ORT} \quad g2$$

$$\text{ABT} \rightarrow \text{ORT} \quad g3$$

$$\text{PERSONALNR} \rightarrow \text{NAME} \} \quad g5$$

3. Berechnung von  $G'''$  nichtreduzant.

$$G''' = \{ \text{PERSONALNR} \rightarrow \text{ABT}$$

$$\text{ABT} \rightarrow \text{ORT}$$

$$\text{PERSONALNR} \rightarrow \text{NAME} \}$$

4. Ausgabe der minimalen Überdeckung  $G'''$

**Beispiel 6**

$$\text{PERSONN} = \{ \text{PNR}, \text{ZUENAME}, \text{VORNAME}, \text{GEMALT}, \text{GESCHLECHT} \}$$

$$\text{KINDER} = \{ \text{PNR}, \text{VORNAME}, \text{GEBDAT}, \text{GESCHLECHT} \}$$

$$\text{KINDER}[\text{PNR}] \subseteq \text{PERSONN}[\text{PNR}] \dots \text{Inklusionsabhängigkeit}$$

$\text{PNR}$  ist für  $\text{PERSONN}$  Schlüssel  $\Leftrightarrow \text{PNR}$  ist Fremdschlüssel für  $\text{KINDER}$ .

**Beispiel 7** Händler verkaufen Produkte für Firmen, dargestellt im Schema

$$R = \{ \text{HÄNDLER}, \text{FIRMA}, \text{PRODUKT} \}$$

Es gilt folgende Wettbewerbsregel: Ein Händler, der ein bestimmtes Produkt verkauft und eine Firma repräsentiert, die dieses Produkt vertreibt, verkauft dieses Produkt für diese Firma.

Formal:  $( * [\text{HÄNDLER FIRMA FIRMA PRODUKT}, \text{HÄNDLER PRODUKT}] )$

kann zerlegt werden in

$$R_1 = \{ \text{HÄNDLER}, \text{FIRMA} \}$$

$$R_2 = \{ \text{FIRMA}, \text{PRODUKT} \}$$

**Beispielauflage**

Gegeben ist das Relationenschema  $R[\text{QRSTU}]$  und die Menge  $F$  von funktionalen Abhängigkeiten. Bestimmen Sie eine minimale Überdeckung.

$$F = \{ T \rightarrow U, TS \rightarrow Q, Q \rightarrow TS, PQ \rightarrow R, Q \rightarrow S, U \rightarrow SU \}$$

**Inklusionsabhängigkeiten**

- eine weitere besondere Form von statischen Integritätsbedingungen.
- Dienen zur Gewährleistung der referentiellen Integrität.

$R, S$  zwei Relationenschemata,

$X, Y$  Folgen von Attributen von  $R$  bzw.  $S, |X| = |Y|$

$r(R)$  und  $s(S)$  erfüllen die **Inklusionsabhängigkeit**  $R[X] \subseteq S[Y]$ , wenn

$$\pi_X(r) \subseteq \pi_Y(s) \text{ gilt.}$$

$$R[X] \subseteq S[Y] \Leftrightarrow \pi_X(r) \subseteq \pi_Y(s)$$

**Ableitungsregeln für Inklusionsabhängigkeiten**

Casanova et. al. 1984

II: Reflexivität

$$X \subseteq R \rightarrow R[X] \subseteq R[X]$$

12: Projektion und Permutation

$$R[X_1 \dots X_n] \subseteq S[Y_1 \dots Y_n] \rightarrow R[X_{\pi(1)} \dots X_{\pi(n)}] \subseteq S[Y_{\pi(1)} \dots Y_{\pi(n)}]$$

wobei  $k \dots m$  Folgen von Indizes aus dem Bereich  $1 \dots n$  sind

13: Transitivität

$$R[X] \subseteq S[Y], S[Y] \subseteq T[Z] \rightarrow R[X] \subseteq T[Z]$$

**Verbundabhängigkeiten**

- statische Integritätsbedingung

Eine Relation  $r(R)$  erfüllt die **Verbundabhängigkeit (join dependency)**  $*[R_1, R_2, \dots, R_n]$ , wenn  $r(R)$  verlustfrei in  $R_1, \dots, R_n$  zerlegbar ist.

trivial, wenn ein  $R_i = R$

Sonderfall von Verbundabhängigkeiten: mehrwertige Abhängigkeiten.

Eine **mehrwertige Abhängigkeit** hat die Form  $X \twoheadrightarrow Y \mid Z$  und entspricht der Verbundabhängigkeit  $*[XY, XZ]$ .

**Normalformen**

**Normalformen**

**Inklusionsabhängigkeiten**

- Eine Inklusionsabhängigkeit  $R[X] \subseteq S[Y]$  heißt **schlüsselbasierter**, wenn  $Y$  ein Schlüssel für  $S$  ist.
- Schlüsselbasierter Inklusionsabhängigkeiten: Gewährleistung der referentiellen Integrität.
- $R[X] \subseteq S[Y]$  schlüsselbasierter, so heißt  $X$  **Fremdschlüssel** und die Inklusionsabhängigkeit **Fremdschlüsselbedingung**.

(siehe SQL Teil)

**Inklusionsabhängigkeiten**

**Beispielauflage**

**Beispielauflage**

**Verbundabhängigkeiten**

**Verbundabhängigkeiten**

**Normalformen**

**Normalformen**

**Kriterien eines guten Datenbankentwurfs:**

- geringe Redundanz,
- Vermeidung von Einfüge-, Lösch-, Änderungsanomalien.

**Normalformen für Datenbankschemata:**

**Erste Normalform**

Ein Relationenschema  $R$  ist in 1. Normalform (1NF), wenn die Wertebereiche aller Attribute von  $R$  atomar sind.

$$1NF \Leftrightarrow \text{Wertebereiche atomar}$$

*Änderungsanomalie*

| vorrat |       |       |                   |
|--------|-------|-------|-------------------|
| TEIL   | LAGER | MENGE | ADRESSE           |
| 101    | 1     | 25    | Pantlg. 16        |
| 102    | 3     | 410   | Kruggerstr. 42    |
| 102    | 1     | 300   | Waagg. 10         |
| 112    | 4     | 10    | Brünnner Str. 105 |

| vorrat         |              |                |                           |
|----------------|--------------|----------------|---------------------------|
| TEIL           | LAGER        | MENGE          | ADRESSE                   |
| 101            | 1            | 25             | Pantlg. 16                |
| <del>102</del> | <del>3</del> | <del>410</del> | <del>Kruggerstr. 42</del> |
| 102            | 1            | 300            | Waagg. 10                 |
| 112            | 4            | 10             | Brünnner Str. 105         |

**Beispiel 4** Relation personal(PERSNR, ABTEILUNG, GEBÄUDE),

Annahme: jede Abteilung befindet sich in genau einem Gebäude  $\Rightarrow F = \{PERSNR \rightarrow ABTEILUNG \text{ GEBÄUDE, ABTEILUNG} \rightarrow GEBÄUDE\}$ .

personal PERSNR ABTEILUNG GEBÄUDE

|      |        |     |
|------|--------|-----|
| 1215 | Buchh. | A12 |
| 2412 | PR     | B8  |
| 809  | Buchh. | A12 |

**Beispiel 1** Von den beiden folgenden Tabellen ist die erste nicht in 1NF, die zweite schon.

| vorrat |       |       |         |
|--------|-------|-------|---------|
| TEIL   | LAGER | MENGE | ADRESSE |
| 101    | 1     | 1     | 101     |
| 101    | 3     | 3     | 101     |
| 102    | 1     | 1     | 102     |
| 102    | 2     | 2     | 102     |
| 102    | 4     | 4     | 102     |
| 112    | 4     | 4     | 112     |

| vorrat |       |       |         |
|--------|-------|-------|---------|
| TEIL   | LAGER | MENGE | ADRESSE |
| 101    | 1     | 1     | {1,3}   |
| 102    | 1     | 1     | {1,2,4} |
| 112    | 4     | 4     | {4}     |

**Beispiel 3**

LAGER  $\rightarrow$  ADRESSE.

TEIL LAGER  $\rightarrow$  MENGE und

Aufspalten des Schemas in zwei Schemata nach den FDs

| vorrat |       |       |                   |
|--------|-------|-------|-------------------|
| TEIL   | LAGER | MENGE | ADRESSE           |
| 101    | 1     | 25    | Waagg. 10         |
| 102    | 3     | 410   | Kruggerstr. 42    |
| 102    | 1     | 300   | Brünnner Str. 105 |

*Anomalien.*

Vertegung der Buchhaltungsabteilung oder Entfernen des Angestellten 2/12  $\Rightarrow$

| personal |           |         |  |
|----------|-----------|---------|--|
| PERSNR   | ABTEILUNG | GEBÄUDE |  |
| 1215     | Buchh.    | A07     |  |
| 2410     | Vertrieb  | A14     |  |
| 2412     | PR        | B8      |  |
| 809      | Buchh.    | A12     |  |
| 1215     | Buchh.    | A12     |  |
| 2410     | Vertrieb  | A14     |  |
| 2412     | PR        | B8      |  |
| 809      | Buchh.    | A12     |  |

| personal |           |         |  |
|----------|-----------|---------|--|
| PERSNR   | ABTEILUNG | GEBÄUDE |  |
| 1215     | Buchh.    | A12     |  |
| 2410     | Vertrieb  | A14     |  |
| 2412     | PR        | B8      |  |
| 809      | Buchh.    | A12     |  |
| 2412     | PR        | B8      |  |
| 809      | Buchh.    | A12     |  |

**Die dritte Normalform**

Veränderung der Adresse in Zeile 1  
Löschen von Teil 102 aus Lager 3

| vorrat |       |       |                   |
|--------|-------|-------|-------------------|
| TEIL   | LAGER | MENGE | ADRESSE           |
| 101    | 1     | 25    | Waagg. 10         |
| 102    | 3     | 410   | Kruggerstr. 42    |
| 102    | 1     | 300   | Waagg. 10         |
| 112    | 4     | 10    | Brünnner Str. 105 |

**Beispiel 2**

*Keine Änderungsanomalie*

| vorrat |       |       |                   |
|--------|-------|-------|-------------------|
| TEIL   | LAGER | MENGE | ADRESSE           |
| 101    | 1     | 25    | Pantlg. 16        |
| 102    | 3     | 410   | Kruggerstr. 42    |
| 102    | 1     | 300   | Brünnner Str. 105 |
| 112    | 4     | 10    | Brünnner Str. 105 |

*Keine Löschanomalie*

| vorrat         |              |                |                           |
|----------------|--------------|----------------|---------------------------|
| TEIL           | LAGER        | MENGE          | ADRESSE                   |
| 101            | 1            | 25             | Waagg. 10                 |
| <del>102</del> | <del>3</del> | <del>410</del> | <del>Kruggerstr. 42</del> |
| 102            | 1            | 300            | Brünnner Str. 105         |
| 112            | 4            | 10             | Brünnner Str. 105         |

**Lösung:** Zerlegung in zwei Relationen.

| personal |           |         |  |
|----------|-----------|---------|--|
| PERSNR   | ABTEILUNG | GEBÄUDE |  |
| 1215     | Buchh.    | A12     |  |
| 2410     | Vertrieb  | A14     |  |
| 2412     | PR        | B8      |  |
| 809      | Buchh.    | A12     |  |

Eine FD  $X \rightarrow Y$  heißt **volle funktionale** Abhängigkeit, wenn für keine **echte** Teilmenge  $X' \subset X$  gilt, daß  $X' \rightarrow Y$ .  
 $Y$  heißt dann **voll funktional** abhängig von  $X$ .

$$X \rightarrow Y \text{ voll funktional} \Leftrightarrow \exists X' \subset X \mid X' \rightarrow Y$$

Einfache Bedingung für die Minimalität in der Definition eines Schlüssels:

$$X \text{ ist Schlüssel von } R \Leftrightarrow X \rightarrow R \text{ voll funktional.}$$

### Die Boyce-Codd Normalform

Ein Relationenschema  $R$  in 1.NF ist in **Boyce-Codd Normalform** (BCNF) bezüglich einer Menge funktionaler Abhängigkeiten  $F$ , wenn kein Attribut transitiv von einem Schlüssel abhängt.  
Analogie zu Theorem für die 3. Normalform:  
ein Relationenschema  $R$  in erster Normalform ist in BCNF bezüglich der FDs  $F$ , wenn für alle nichttrivialen funktionalen Abhängigkeiten  $X \rightarrow A$  auf  $R$  gilt:  $X$  ist ein Oberschlüssel.

$$BCNF \Leftrightarrow X \rightarrow A \Leftrightarrow X \rightarrow R \vee A \in X$$

Sei  $R = (S, F)$  ein Relationenschema, und  $R_1 = (S_1, F_1), S_1 \subseteq S, R_1$  Subschema.  
Eine FD  $X \rightarrow Y \in F$  ist **eingebettet** in  $R_1$ , wenn  $XY \subseteq S_1$ ,  
 $F[S_1] \dots$  die Menge aller in  $S_1$  eingebetteten FDs aus  $F$ .

$$F[S_1] = \{X \rightarrow Y \in F \mid XY \subseteq S_1\}$$

$$F^+[S_1] = \{X \rightarrow Y \in F^+ \mid XY \subseteq S_1\}$$

Ein Attribut  $A$  heißt **prim** in  $R$ , wenn es in **mindestens einem** Schlüssel von  $R$  enthalten ist, sonst heißt es **nichtprim**.

$$A \text{ ist prim} \Leftrightarrow \exists X. X \text{ ist Schlüssel, } A \in X$$

$F$  Menge funktionaler Abhängigkeiten über  $R, X \subseteq R, A \in R$ .

$A$  ist **transitiv abhängig** von  $X$ , wenn es eine Attributmengne  $Y \subseteq R$  gibt, sodas  $X \rightarrow Y \in F^+$  und  $Y \rightarrow A \in F^+$  sind und  $Y \rightarrow X \notin F^+, A \notin XY$  gilt.

$$X \rightarrow A \text{ transitiv} \Leftrightarrow \exists Y, Y \subseteq R, A \notin XY, X \rightarrow Y \rightarrow A, Y \neq X$$

Die strengste Normalform für FDs ist die Boyce-Codd-Normalform.

### Beispiel 5

Relationenschema  $R = (\text{ÜBUNG, STUDENT, TUTOR})$

$FDs: F = \{\text{STUDENT ÜBUNG} \rightarrow \text{TUTOR, TUTOR} \rightarrow \text{ÜBUNG}\}$

Die Abhängigkeit  $TUTOR \rightarrow ÜBUNG$  verletzt die BCNF (aber nicht die 3.NF).  
Zerlegung des Schemas  $R$  in  $R_1 = (\text{STUDENT TUTOR}), R_2 = (\text{TUTOR ÜBUNG})$

$\Rightarrow$  zwei neue Schemata  $R_1, R_2$ , die in BCNF sind.

Eine **Zerlegung eines Relationenschemas**  $R = (S, F)$  ist eine Menge von Relationenschemata

$$\{R_1 = (S_1, F_1), R_2 = (S_2, F_2), \dots, R_n = (S_n, F_n)\}$$

wobei folgendes gilt:

$$1. S = S_1 \cup S_2 \cup \dots \cup S_n$$

$$2. F_i \equiv F^+[S_i]$$

$$3. (F_1 \cup F_2 \cup \dots \cup F_n)^+ \subseteq F^+.$$

Ein Relationenschema  $R$  in 1.NF ist in **3. Normalform** (3.NF), wenn kein nichtprim.es Attribut von einem Schlüssel in  $R$  transitiv abhängig ist.

$$3.NF \Leftrightarrow A \text{ nichtprim} \Rightarrow \text{Schlüssel} \rightarrow A \text{ nicht transitiv.}$$

**Theorem:** Ein Relationenschema  $R$  in erster Normalform ist in 3. Normalform bezüglich der FDs  $F$  genau dann, wenn für alle nichttrivialen funktionalen Abhängigkeiten  $X \rightarrow A$  auf  $R$  gilt:  $X$  ist ein Oberschlüssel oder  $A$  ist ein prim.es Attribut.

$$3.NF \Leftrightarrow X \rightarrow A \Leftrightarrow X \rightarrow R \vee A \in X \vee A \text{ prim.}$$

### Zerlegungen von Relationenschemata

Ein Relationenschema  $R$  ist ein Paar  $(S, F)$  mit den Komponenten

- $S$ : eine Menge von Attributen
- $F$ : eine Menge funktionaler Abhängigkeiten über  $S$ .

Ein **Relationales Datenbankschema**  $R$  ist eine Menge von Relationenschemata:  $R = \{R_1, R_2, \dots, R_n\}$ .

Dieses Datenbankschema  $R$  ist in 3.NF bzw. BCNF, wenn jedes Relationenschema in  $R$  in 3.NF bzw. BCNF ist.

Eine Zerlegung ist **abhängigkeitstreu** (dependency preserving, keine

Abhängigkeit geht verloren), wenn:

$$(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$$

Die Zerlegung ist **verbundtreu** (verlustfrei, lossless), wenn für jede zulässige

$$r(S_1) \bowtie r(S_2) \bowtie \dots \bowtie r(S_n) = r(S)$$

**Beispiel 6** Sei  $R(S, F)$  ein Relationenschema mit  $S = (ABCDE)$  und

$$F = \{A \rightarrow BCD, CD \rightarrow E, EC \rightarrow B\}.$$

Eine mögliche Zerlegung von  $R$  in  $m$   $R_1 = (S_1, F_1), R_2 = (S_2, F_2)$  mit

$$S_1 = \{A, B, C, D\}, F_1 = \{A \rightarrow BCD, CD \rightarrow B\}$$

$$S_2 = \{C, D, E\}, F_2 = \{CD \rightarrow E\}.$$

Die Abhängigkeit  $EC \rightarrow B$  geht verloren  $\Rightarrow$  die Zerlegung ist nicht

*abhängigkeitsstreu!*

ist.

$F' = \cup_{i=1}^m F'_i$  die Hülle eines der generierten Relationenschemata  $S'_i$  gleich  $S$   
 $R_2 = (S_2, F_2), \dots, R_n = (S_n, F_n)$  } ist verbunden, wenn bezüglich  
**Theorem:** Eine Zerlegung von  $R = (S, F)$  in  $\{R_1 = (S_1, F_1),$

die Zerlegung verbunden.

Wenn das Join-Atribut in einer der beiden Teilrelationen Schlüssel ist, so ist

$$\exists S'_i \mid S'_i = S \text{ bzgl. } F' = \cup_{i=1}^m F'_i$$

$$\Leftrightarrow$$

$$\text{Zerlegung von } S \text{ in } S_1, S_2, \dots, S_n \text{ ist verbunden}$$

4. Berechne Überdeckungen  $F_1$  und  $F_2$  für  $R_1$  und  $R_2$ , aus denen sich alle in  $R_1$  bzw.  $R_2$  gültigen funktionalen Abhängigkeiten ableiten lassen. D.h. berechne  $F_1$  und  $F_2$  über  $S_1$  bzw.  $S_2$ , mit:  $F_1^+ = F_1^+ [S_1]$  und  $F_2^+ = F_2^+ [S_2]$ .

5. Das Verfahren wird nun rekursiv auf  $R_1 = (S_1, F_1)$  und  $R_2 = (S_2, F_2)$  angewendet. (Amm:  $R_2$  ist in 3.NF bzw. BCNF)

6. Die Menge aller Subchemata in 3.NF bzw. BCNF, die im Laufe des Verfahrens erzeugt werden, ist eine Zerlegung in 3.NF bzw. BCNF des ursprünglichen Schemas  $R$ .

**Beispiel 7** Die Zerlegung in Beispiel 5 ist verbunden, aber nicht abhängigkeitsstreu: die Abhängigkeit STUDENT ÜBUNG  $\rightarrow$  TUTOR geht verloren.

• Zerlegung in BCNF ist nicht immer verbind- und abhängigkeitsstreu

möglich  $\Leftrightarrow$  Nachteil der BCNF.

Verzichte auf die Abhängigkeitsstreue, nicht auf die Verbundtreue; denn:

Die Abhängigkeitsstreue stellt Metainformation dar;

die Verbundtreue aber Information.

• Das Entscheidungsproblem, ob es eine abhängigkeitsstreue (und

verbundtreue) Zerlegung eines Datenbankschemas in BCNF gibt, ist NP-vollständig.

• Der Test, ob ein Datenbankschema in BCNF ist, ist hingegen polynomial.

**Verbundtreue Zerlegung in 3.NF bzw. BCNF**

Im Folgenden beschreiben wir ein Verfahren zur verbundenen Zerlegung in 3.NF bzw. BCNF.

**Algorithmus DECOMP**

**INPUT:** Relationenschema  $R = (S, F)$

**OUTPUT:** verbundtreue Zerlegung  $\{R_1 = (S_1, F_1), \dots, R_n = (S_n, F_n)$  von  $R$  in 3.NF bzw. BCNF.

**METHODE (Prinzip):**

1. Ersetze  $F$  durch minimale Überdeckung, diese heiÙe weiterhin  $F$ .

• Das Verfahren terminiert immer: Kardinalität der erzeugten Schemata nimmt in jedem Schritt ab.

• Die Zerlegung ist verbunden: erhalte  $R$  aus  $S_1 = S - A$  und  $S_2 = XA$  durch  $R_1 \bowtie R_2$ , da  $X$  Schlüssel für  $R_2$  ist.

• Die Korrektheit des Verfahrens ist gegeben, wenn Schritt 2 (Überprüfung der Normalform) korrekt ist.

Mitße alle Abhängigkeiten  $X \rightarrow A \in F^+$  und nicht nur in  $F$  überprüfen. Aber:  $R = (S, F)$  ist nicht in 3.NF bzw. BCNF  $\Leftrightarrow F$  enthält börsartige FD (wichtig:  $F$  ist eine minimale Überdeckung).

**Beispiel 8** Sei  $R$  das Relationenschema  $R = (\{ABC\}, \{A \rightarrow B, C \rightarrow B\})$ . Zerlege  $R$  in  $R_1 = (\{AB\}, F_1)$  und  $R_2 = (\{BC\}, F_2)$ . Für die erste der beiden nachfolgenden Instanzen  $r(R)$  erhalten wir eine verbundtreue Zerlegung, für die zweite nicht.

|     |    |    |    |                   |    |    |           |    |    |      |    |    |    |
|-----|----|----|----|-------------------|----|----|-----------|----|----|------|----|----|----|
| $r$ | A  | B  | C  | $r_1$             | A  | B  | $r_2$     | B  | C  | $r'$ | A  | B  | C  |
|     | a1 | b1 | c1 | $\Leftrightarrow$ | a1 | b1 | $\bowtie$ | b1 | c1 | =    | a1 | b1 | c1 |
|     | a2 | b2 | c2 |                   | a2 | b2 |           | b2 | c2 |      | a2 | b2 | c2 |

**3.NF:** eine FD der Form  $X \rightarrow A$ , wobei  $X$  kein Schlüssel ist und  $A$  nichtprim ist. Sie verletzt die 3.NF-Bedingungen für  $R = (S, F)$ .  
**BCNF:** eine FD der Form  $X \rightarrow A$ , wobei  $X$  kein Schlüssel von  $R$  ist. Enthält  $F$  keine börsartige Abhängigkeit, dann ist  $R = (S, F)$  bereits in 3.NF bzw. im BCNF. Fertig.

3. Gibt es eine börsartige FD  $X \rightarrow A$  in  $F$ , dann spalte  $R$  in zwei Teile  $R_1 = (S_1, F_1)$  und  $R_2 = (S_2, F_2)$  mit:

$$S_1 = S - A \text{ und}$$

$$S_2 = XA, \text{ wobei } X \text{ der Schlüssel für die neue Relation } R_2 \text{ wird.}$$

• Das schwierigste Teilproblem: Berechnung von  $F_1$  und  $F_2$  aus  $F$  und  $R_1$  bzw.  $R_2$ . (später: Algorithmus HBR).  
• Gibt es in Schritt 2/3 mehrere börsartige Abhängigkeiten  $\Rightarrow$  das Verfahren führt je nach Auswahl zu verschiedenen Zerlegungen.  
• Die Relationenschemata können als Baum mit Wurzel  $R = (S, F)$  dargestellt werden (siehe auch das folgende Beispiel).

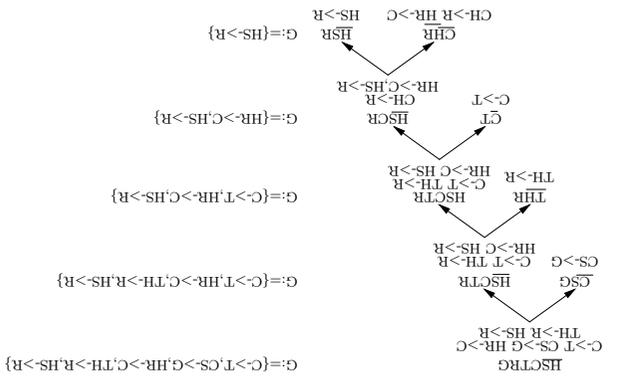


3.  $G := \{C \rightarrow T, HR \rightarrow C, HS \rightarrow R\}$   
 $\Rightarrow$  Zerlegung nach  $C \rightarrow T$   
 $HS \rightarrow R$  gutartig,  
 $C \rightarrow T$  *erhaltend*
4.  $G := \{HR \rightarrow C, HS \rightarrow R\}$   
 $\Rightarrow$  Zerlegung nach  $HR \rightarrow C$   
 $HS \rightarrow R$  gutartig,  
 $HR \rightarrow C$  *erhaltend*
5.  $G := \{HS \rightarrow R\}$   
 $G$  enthält keine "böswartige" Abhängigkeiten  $\Rightarrow$  fertig

$f$ ... eine Abhängigkeit  $XAY \rightarrow Z$   $g$ ... eine Abhängigkeit  $V \rightarrow A$ ,  
 $X, Y, \dots$  möglicherweise leere Attributmengen  
 $h : XYV \rightarrow Z$  heißt **A-Resolvente** von  $f$  und  $g$ .  
 Jede A-Resolvente von  $f$  und  $g$  folgt aus  $\{f, g\}$  durch Pseudotransitivität.  
**Beispiel 11**  $AC \rightarrow D$  und  $DEF \rightarrow B$  haben als D-Resolvente  $ACEF \rightarrow B$ .  
 $RES(f, A)$  bezeichnet die Menge aller nichttrivialen A-Resolventen, die  
 unmittelbar aus  $F$  erzeugt werden können.

**Resolventen**

Gottlob, 1987 (modifizierte Version)  
**Algorithmus RBR (Reduction By Resolution)**  
**INPUT** Schema  $R = (S, F)$ ,  $S_1 \subset S$   
**OUTPUT** Überdeckung  $G$  mit  $G^+ = F^+[S_1]$



$F$ ... eine Menge von funktionalen Abhängigkeiten.  
**unfold**( $F$ ) bezeichnet die Menge jener Abhängigkeiten, die aus Elementen  
 von  $F$  durch Aufspalten der rechten Seite in Einzelattribute gewonnen wird  
 (d.h.  $F$  wird in eine kanonische Darstellung gebracht).  
**fold**( $F$ ) bezeichnet jene Überdeckung von  $F$ , die durch Zusammenfassen aller  
 Abhängigkeiten mit gleicher linker Seite gewonnen wird.

**begin**  
 $G := \text{unfold}(F)$ ;  
 $X := S \setminus S_1$ ;  
**while**  $X \neq \emptyset$  **do**  
 wähle ein Attribut  $A$  aus  $X$   
 $X := X \setminus A$ ;  
 $G := (G \cup \text{RES}(G, A))$ ;  
 entferne alle Abhängigkeiten, die  $A$  enthalten, aus  $G$ ;  
**end**  
 $G := \text{fold}(G)$ ;  
**return**  $G$   
**end**

**Beispiel 12** Berechnen wir *unfold* und *fold* von  
 $F = \{AB \rightarrow CD, AB \rightarrow E, AF \rightarrow EGH\}$ .  
 $\text{unfold}(F) = \{AB \rightarrow C, AB \rightarrow D, AB \rightarrow E, AF \rightarrow E, AF \rightarrow G, AF \rightarrow H\}$   
 $\text{fold}(F) = \{AB \rightarrow CDE, AF \rightarrow EGH\}$

$\text{unfold}(F)$  und  $\text{fold}(F)$  sind immer Überdeckungen von  $F$ ; im Allgemeinen gilt:  $\text{fold}(\text{unfold}(F)) \neq F$ .

**begin**  
 $G := F$ ; da  $F$  bereits in kanonischer Darstellung  
 $X := \{T\}$ ;  
**while**  $X \neq \emptyset$  **do**  
 wähle  $T$  aus  $X$   
 $X := \{\}$ ;  
 $G := (G \cup \{CH \rightarrow R\})$ ;  
 $G := \{CH \rightarrow R, HR \rightarrow C, HS \rightarrow R\}$   
**end**  
 $G := \{CH \rightarrow R, HR \rightarrow C, HS \rightarrow R\}$   
**return**  $\{CH \rightarrow R, HR \rightarrow C, HS \rightarrow R\}$   
**end**

**Überdeckungen für eingebaute Abhängigkeiten**

$R = (S, F)$ ,  $S_1$  Subschema von  $S$ .  
**Gesucht:** eine Überdeckung  $G$  der aus  $F$  auf  $S_1$  gültigen FDs, sodaß  $G$  alle  
 auf  $S_1$  gültigen Abhängigkeiten repräsentiert.

**Annahme:**  $F$  ist minimal und enthält keine überflüssigen Attribute.

**Formal:** Finde  $G$  mit  $G^+ = F^+[S_1]$ .

**Triviale Lösung:**  $G = F^+[S_1]$ .

Schlecht, da

1. Berechne  $F^+$  (z.B. mittels der Armstrong Axiome) und projiziere auf  $S_1 \Rightarrow$  Aufwand exponentiell in der Anzahl der Attribute.

2.  $F^+[S]$  ist von maximaler Größe  $\Rightarrow$  viel Speicherplatz (unhandlich).

**Bottom-up Auswertung des Operatorbaumes:**

- ungünstig, wenn die Join-Operatoren nahe bei den Blättern stehen.
- Zeit- und Platzaufwand binärer Operationen steigt mit # der Tupel und der # der Attribute in den Argumentrelationen.
- Grundprinzip: unäre Operatoren in Richtung der Blätter des Baumes verschieben
- durch Zusammenfassung unärer Operationen werden diese in einem Schritt durchgeführt.
- Selektion vor der Projektion wegen der Entfernung doppelter Tupel.

**Regeln für Join und kartesisches Produkt**

Der Join und das kartesische Produkt sind kommutativ und assoziativ.

**1. Kommutativität:**

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

$$E_1 \times E_2 \equiv E_2 \times E_1$$

**2. Assoziativität:**

$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$

$$(E_1 \times E_2) \times E_3 \equiv E_1 \times (E_2 \times E_3)$$

**Grundlegende Aspekte**

- Aufwendigste Operationen: kartesisches Produkt und Join.
- $n \dots$  Tupel von  $A$ ,  $m \dots$  Tupel von  $B$
- Aufwand zur Berechnung des Joins oder kartesischen Produktes  $O(mn)$ .
- Projektion ist aufwendig: Duplikate müssen entfernt werden.
- Selektionen so früh wie möglich durchführen: kleinere Zwischenergebnisdate.
- Unäre Operationen: je einen Durchlauf aller Tupel  $\Rightarrow$  mehrere zusammenziehen.
- Gemeinsame Teilausdrücke: nur einmal auswerten.
- Zeitaufwand zur Untersuchung der Möglichkeiten: geringer als Durchführung einer ineffizienten Query.

**Zusammenfassen gleicher Teilausdrücke**

- Schrittweise von unten nach oben.
  - Algebraische Transformationen können die Existenz gleicher Teilausdrücke verschleiern.
- Beispiel 2** *Phase gleiche Teilausdrücke im Baum aus Beispiel 1 zusammen.*

**Regeln für Selektion und Projektion**

**3. Zusammenfassung von Projektionen:** nur möglich, wenn die Menge der Attribute der ersten Projektion eine Teilmenge der Attribute der zweiten Projektion ist.

falls  $\{A_i \mid i = 1, \dots, n\} \subseteq \{B_j \mid j = 1, \dots, m\}$  so gilt:

$$\pi_{A_1, \dots, A_n}(\pi_{B_1, \dots, B_m}(E)) \equiv \pi_{A_1, \dots, A_n}(E)$$

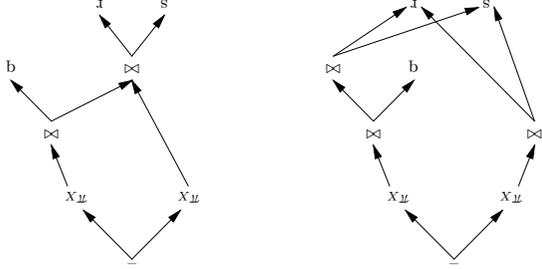
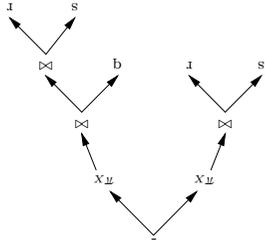
$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_1 \wedge F_2}(E)$$

$$\sigma_{F_1}(\sigma_{F_2}(E)) \equiv \sigma_{F_2}(\sigma_{F_1}(E))$$

**Algebraische Optimierung**

Bei der algebraischen Optimierung verwenden wir zur Darstellung von Queryes einen Operatorbaum.

**Beispiel 1** *Operatorbaum von  $\pi_X(s \bowtie r) - \pi_X(q \bowtie r \bowtie s)$ :*



**5. Kommutativität Selektion-Projektion:**

nur möglich, wenn sich  $F$  nur auf Attribute  $A_i$  bezieht:

$$\pi_{A_1, \dots, A_n}(\sigma_F(E)) \equiv \sigma_F(\pi_{A_1, \dots, A_n}(E));$$

wenn sich  $F$  auf alle Attribute  $B_j$  und möglicherweise auf Attribute  $A_i$  bezieht:

$$\pi_{A_1, \dots, A_n}(\sigma_F(E)) \equiv \pi_{A_1, \dots, A_n}(\sigma_F(\pi_{A_1, \dots, A_n, B_1, \dots, B_m}(E)))$$

**6. Kommutativität Selektion-Kartesisches Produkt:**

nur möglich, wenn sich  $F_i$  nur auf Attribute von  $E_i$  beziehen

$$(F = F_1 \cup F_2):$$

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_1}(E_1) \times \sigma_{F_2}(E_2);$$

wenn  $F_i$  sich nur auf Attribute von  $E_1$ ,  $F_2$  aber auf Attribute von  $E_1$  und  $E_2$  bezieht:

$$\sigma_F(E_1 \times E_2) \equiv \sigma_{F_2}(\sigma_{F_1}(E_1) \times E_2)$$

- 7. Kommutativität Selektion-Vereinigung:**  
 $\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$
- 8. Kommutativität Selektion-Mengendifferenz:**  
 $\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$
- 9. Kommutativität Projektion-Kartesisches Produkt:**  
 Seien  $B_i$  Attribute von  $E_1$ ,  $C_i$  Attribute von  $E_2$  und  $\{A_i \mid i = 1, \dots, n\} = \{B_i, i = 1, \dots, m\} \cup \{C_i \mid i = 1, \dots, k\}$ , so gilt:  
 $\pi_{A_1, \dots, A_n}(E_1 \times E_2) \equiv \pi_{B_1, \dots, B_m}(E_1) \times \pi_{C_1, \dots, C_k}(E_2)$

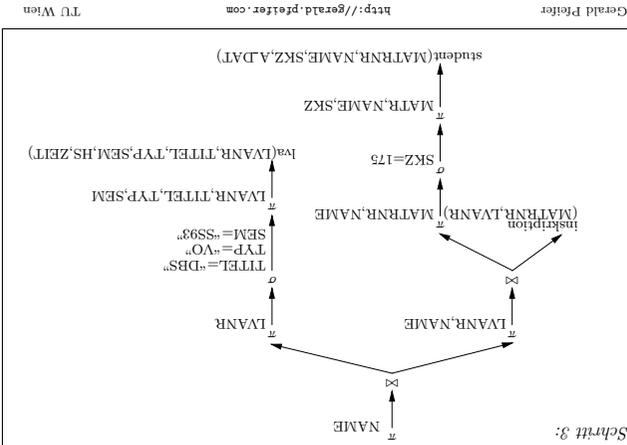
**Beispiel 3** Relationenschemata:

```

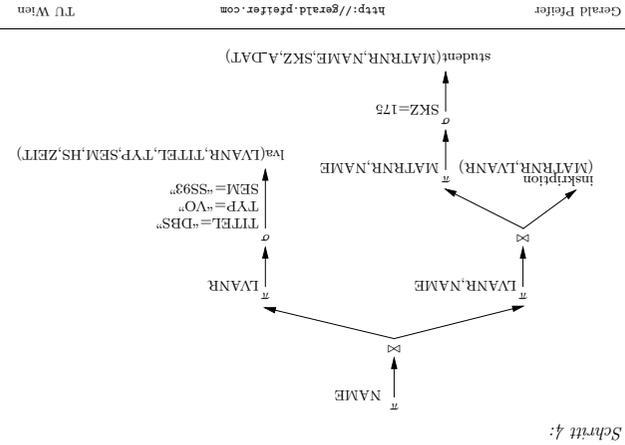
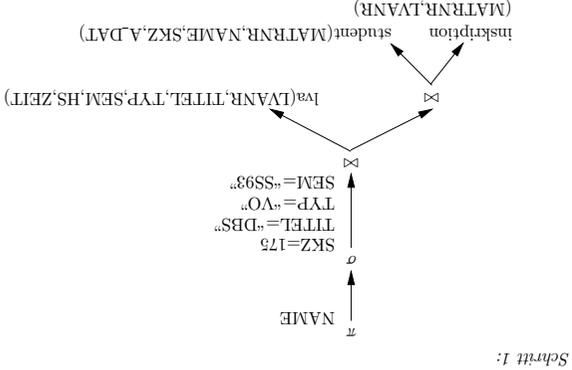
student (MATRNR, NAME, SKZ, ANFANGSDATUM)
lva (LVANR, TITEL, TYP, SEM, HS, ZEIT)
inskription (MATRNR, LVANR)

```

Formuliere eine Query in relationaler Algebra, die die Namen aller Wirtschaftsinformatiker (SKZ="175") ermittelt, die im SS93 die Lehrveranstaltung mit dem Titel "DBS" (TYP="VO") besucht haben:

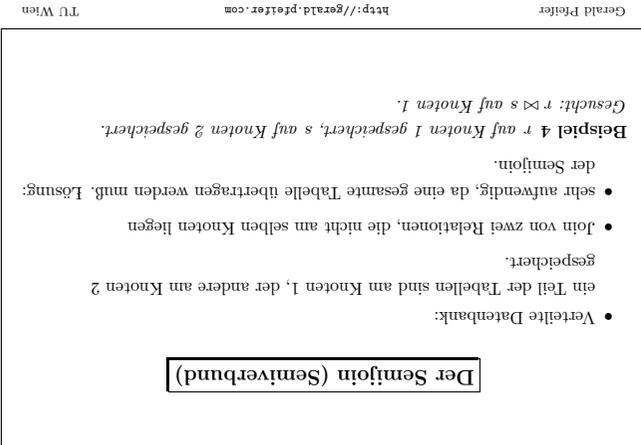
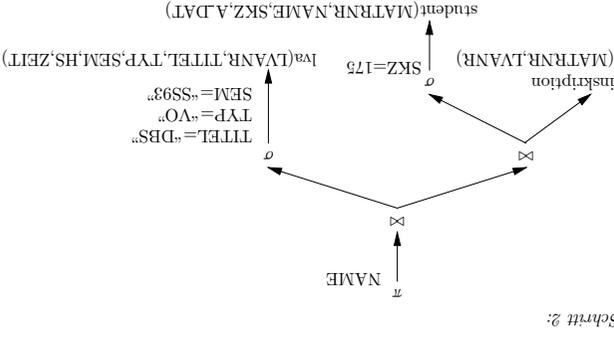
$$\pi_{NAME}(\sigma_{SKZ=175, TITEL='DBS', TYP='VO', SEM='SS93'}(\text{student} \bowtie \text{lva} \bowtie \text{inskription}))$$


- 10. Kommutativität Projektion-Vereinigung:**  
 $\pi_{A_1, \dots, A_n}(E_1 \cup E_2) \equiv \pi_{A_1, \dots, A_n}(E_1) \cup \pi_{A_1, \dots, A_n}(E_2)$
- Join:** darstellbar als Kombination von kartesischem Produkt, Projektion und Selektion.
- $A_i \dots$  Attribute von  $R_1$ ,  $B_j \dots$  Attribute von  $R_2$ ,  $A_i \cup B_j = C_m$ , Join über  $A_k = B_l$ :
- $$R_1 \bowtie R_2 = \pi_{C_m \setminus B_l}(\sigma_{A_k=B_l}(R_1 \times R_2))$$
- $\Rightarrow$  Regeln für den Join folgen aus den Regeln oben.
- Es gilt keine Kommutativität zwischen Mengendifferenz und Projektion!**



- Zerlege Selektionen der Art  $\sigma_{F_1 \wedge \dots \wedge F_n}(E)$  nach Regel 4 in  $\sigma_{F_1}(\dots(\sigma_{F_n}(E))\dots)$ . Regeln 4 – 8.
- Schiebe jede Selektion soweit wie möglich in Richtung Blätter mit den Regeln 3, 5, 9 und 10.
- Schiebe jede direkt aufeinanderfolgenden Selektionen und Projektionen zu einer einzigen Selektion, einer einzigen Projektion oder einer Selektion gefolgt von einer Projektion mit den Regeln 3-5 zusammen.

**Ein einfacher Optimierungsalgorithmus**



Knoten 1  $r$  (A B)  $s$  (B C d)  $r'$  (B)  $s'$  (B C D) Knoten 2 Knoten 1

|   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |   |   |    |    |    |   |   |   |   |   |   |
|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|----|----|----|---|---|---|---|---|---|
| 1 | 4 | 4 | 13 | 16 | 4  | 4  | 13 | 16 | 4  | 14 | 16 | 5  | 5  | 4  | 13 | 16 | 7  | 13 | 17 | 1  | 6 | 2 | 4 | 2 | 4 | 10 | 14 | 16 | 7 | 7 | 6 | 8 | 9 |   |
| 2 | 3 | 7 | 10 | 15 | 17 | 10 | 15 | 17 | 11 | 15 | 16 | 10 | 15 | 17 | 11 | 15 | 16 | 11 | 15 | 16 | 3 | 8 | 3 | 7 | 3 | 7  | 3  | 8  | 3 | 8 | 3 | 8 | 3 | 9 |

Bei einer Berechnung des joins ohne Verwendung des Semijoins müßten wir von Knoten 1 nach 2 und zurück übertragen.

$3 * 8 = 24$  Attributwerte von  $s$  auf dem Knoten 1 übertragen.

### Begriffsklärungen

- Eine Datenbank ist **konsistent**, wenn sie eine Menge vorgegebener logischer Bedingungen (Integritätsbedingungen) erfüllt.
- Verwendung der Datenbank: logisch zusammengehörende Folgen von Zugriffsbefehlen (Lese- und Schreib-Befehle) werden auf der Datenbank ausgeführt.
- Eine **Transaktion** ist Befehlsfolge. Sie führt die Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand über.
- Führt eine Transaktion nur Lesezugriffe durch, dann wird sie als Lese-Transaktion bezeichnet, sonst als Schreib-Transaktion.

DBVO 10.5

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

### Atomarität (atomicity)

Eine Transaktion wird entweder ganz oder gar nicht ausgeführt. Können nicht alle Operationen einer Transaktion ausgeführt werden => Abbruch und Änderungen rückgängig machen.

**Beispiel 2** Sicherstellung, daß beide Aktionen durchgeführt werden, einerseits die Abbuchung vom Lohnkonto, andererseits die Buchung auf das Sparbuch. *Bricht die Transaktion nach der Abbuchung und vor der Buchung ab, so wäre das eine Verletzung der Atomarität.*

**Lösung:** 1. Wir schicken zuerst  $r' = \pi_B(r)$  von 1 nach 2.  
 2. Wir bilden  $s' = s \bowtie r'$  beim Knoten 2.  
 3. Wir schicken  $s'$  von 2 nach 1.  
 4. Wir bilden  $r \bowtie s'$ .

$s' = s \times r = \pi_S(s \bowtie r)$

**Beispiel 5** In Beispiel 4 werden beim Semjoin  $6 + 3 * 3 = 15$  Attributwerte von Knoten 1 nach 2 und zurück übertragen.

DBVO 10.3

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

**Beispiel 1** Transaktion: bucht 500 Euro von Konto 123 ab und auf das Sparbuch 333 auf.

Relationen: sparbuch (SPNR, EINLAGE, BESITZER) und konto (KONTNR, KSTAND).

```

begin transaction
update KONTNO
 set KSTAND=KSTAND-500
 where KONTNR=123;
update SPARBUCH
 set EINLAGE=EINLAGE+500
 where SPNR=333;
end transaction

```

DBVO 10.6

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

### Dauerhaftigkeit (durability)

Die Ergebnisse einer erfolgreich durchgeführten Transaktion bleiben erhalten.

**Beispiel 3** Nach der Durchführung der Umbuchung muß der neue Kontostand sowohl auf dem Lohnkonto als auch auf dem Sparbuch aufschreiben und kann nicht mehr etwa aus Systemgrundrücken rückgängig gemacht werden.

Atomarität und Dauerhaftigkeit wird von einer **Recovery** Komponente gewährleistet (nächstes Kapitel)

### Mehrbenutzerkontrolle (Concurrency Control)

DBVO 10.4

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

### ACID Eigenschaften von Transaktionen

**Erwartung:** Die Transaktion läuft wie geplant ab.

**Problem:** Bei der Ausführung von einer bzw. mehreren Transaktionen kann es zu Störungen bzw. Fehlern kommen.

**Lösung:** Verlangte gewisse Eigenschaften von den Transaktionen. Diese sind: **ACID-Eigenschaften:** Atomarität, Konsistenzzerhaltung, Isolation und Dauerhaftigkeit.

DBVO 10.7

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

### Konsistenzzerhaltung (consistency)

Das Gesamtergebnis einer parallelen Ausführung von Transaktionen entspricht **irgendeiner** Hintereinanderanführung dieser Transaktionen. (Eine Transaktion ist konsistenzzerhaltend => eine Hintereinanderanführung konsistenzzerhaltend.)

Konsistenzzerhaltung wird durch eine Komponente für **Mehrbenutzerkontrolle** (Concurrency Control) sichergestellt.

**Beispiel 4** Neben der Umbuchungstransaktion aus Beispiel 1 findet eine Transaktion statt, die eine Bankomathebung vom Lohnkonto abbucht. Der Gesamteffekt ist entweder:

1. vom Lohnkonto abbuchen und am Sparbuch aufbuchen,
2. vom verringerten Lohnkontostand den am Bankomat abgehobenen Betrag abbuchen.

oder

1. Lohnkontostand um den beim Bankomat abgehobenen Betrag verringern,
2. vom verringerten Lohnkontostand abbuchen und am Sparbuch aufbuchen.

*T<sub>1</sub>: Überweisung von einem Konto auf ein Sparbuch*  
*T<sub>2</sub>: Überweisung einer Telefongrechnung*

|       |                                                                                                                            |
|-------|----------------------------------------------------------------------------------------------------------------------------|
| $T_1$ | READ Konto<br>Konto: = Konto - 10.000<br>WRITE Konto<br>READ Sparbuch<br>Sparbuch: = Sparbuch + 10.000                     |
| $T_2$ | READ Konto<br>Konto: = Konto - 2.000<br>WRITE Konto<br>READ TEL-Konto<br>TEL-Konto: = TEL-Konto + 2.000<br>WRITE TEL-Konto |

DBVO Gerald Pfeifer TU Wien

$I(k) \dots k$ -ter Ausführungsschritt.

Schreibweise:  $I(k) = (T_j, a_i, e_i)$  der  $i$ -te Transaktionsschritt der Transaktion  $T_j$  ist der  $k$ -te Ausführungsschritt in  $I$ .

Eine Ausführung  $I$  von  $\mathbf{T}$  ist **seriell**, wenn für jede Transaktion  $T_j$  aus  $\mathbf{T}$  gilt, daß alle ihre Ausführungsschritte in  $I$  unmittelbar hintereinander liegen.

Intuitiv:  $I = T_{\pi_1}, T_{\pi_2}, \dots, T_{\pi_n}, \dots, T_{\pi_n}, \dots, T_{\pi_1}$  Permutation von  $1, \dots, n$ .

**Isolation**

Teilergebnisse von Transaktionen bleiben bis zum Ende der Transaktion für alle anderen Transaktionen unsichtbar.

**Beispiel 5** Bei der Umbuchungstransaktion wird die Abbuchung vom Lohnkonto erst sichtbar, nachdem auch die Aufbuchung auf das Sparbuch durchgeführt worden ist.

**Ausführung mehrerer Transaktionen**

Eine **Transaktion**  $T$  ist eine Folge von Paaren  $(a_1, e_1), \dots, (a_i, e_i), \dots, (a_n, e_n)$ ,  $i = 1 \dots n$ , von Operationen  $a_i$  auf Objekten  $e_i$ .

$|T|$ : Anzahl der Schritte  $n$ .

Ein Paar  $(a_i, e_i)$  nennt man einen **Transaktionsschritt**.

Ein Transaktionsschritt zusammen mit der Angabe der ihm ausführenden Transaktion  $T$  wird als **Ausführungsschritt** bezeichnet:  $(T, a_i, e_i)$

DBVO Gerald Pfeifer TU Wien

**Beispiel 7** ad Beispiel 6:

Ein Transaktionsschritt ist z.B. das Paar (READ, Konto).

Ein Ausführungsschritt das Typel (T2, WRITE, TEL-Konto).

Eine Ausführung von  $\mathbf{T} = \{T_1, T_2\}$  ist z.B.

$I(1) = (T_1, \text{READ}, \text{Konto})$   
 $I(2) = (T_1, \text{WRITE}, \text{Konto})$   
 $I(3) = (T_1, \text{READ}, \text{Sparbuch})$   
 $I(4) = (T_1, \text{WRITE}, \text{Sparbuch})$   
 $I(5) = (T_2, \text{READ}, \text{Konto})$   
 $I(6) = (T_2, \text{WRITE}, \text{Konto})$   
 $I(7) = (T_2, \text{READ}, \text{TEL-Konto})$   
 $I(8) = (T_2, \text{WRITE}, \text{TEL-Konto})$

Diese Ausführung ist **seriell**.

DBVO Gerald Pfeifer TU Wien

**Konfliktrelation einer Ausführung**

- Bei der Ausführung von mehreren Operationen auf einem Objekt spielt die Reihenfolge der einzelnen Operationen im allgemeinen eine Rolle. Z.B. wesentlich bei zwei Schreiboperationen, die auf ein Objekt die Werte 100 bzw. 200 schreiben.
- Operationen  $a_i, a_j$  auf einem Objekt, deren Reihenfolge eine Rolle spielt, heißen **Konfliktoperationen**
- gehören  $a_i$  und  $a_j$  zu verschiedenen Transaktionen, gibt es einen Konflikt.
- die Paare (WRITE, WRITE), (WRITE, READ) und (READ, WRITE) stellen Konfliktoperationen dar,
- das Paar (READ, READ) hingegen nicht.

DBVO Gerald Pfeifer TU Wien

**Parallele Ausführung von Transaktionen**

vereinfachtes Datenbankmodell:

- besteht aus atomaren Objekten, die einen Namen und einen Wert haben.
- Zugriff: mittels der atomaren Operationen READ und WRITE.

DBVO Gerald Pfeifer TU Wien

Die **Ausführung**  $I$  einer Transaktionsmenge  $\mathbf{T} = \{T_1, T_2, \dots, T_n\}$  ist eine beliebige Folge von Ausführungsschritten

$$I(k) = (T_j, a_i, e_i), \quad k = 1, \dots, m$$

die folgende Bedingungen erfüllt:

1.  $m = \sum_{j=1}^n |T_j|$ ,  $j \in \{1, \dots, n\}$
2.  $i \in \{1, \dots, |T_j|\}$ ,  $j \in \{1, \dots, n\}$
3. Für jede Transaktion  $T_j$  aus  $\mathbf{T}$  sind alle Transaktionsschritte  $(a_i, e_i) \in T_j$  in  $I$  in derselben Reihenfolge enthalten wie in  $T_j$ .

Ein Tupel  $(T_i, e, T_j)$  gehört zur "Konfliktrelation", wenn die folgenden Bedingungen erfüllt sind:

- Im Schritt  $k$  einer Ausführung  $I$  wird durch eine Transaktion  $T_i$  eine Operation  $a_m$  auf einem Objekt  $e$  ausgeführt.
- In einem späteren Schritt  $l$  der Ausführung  $I$  wird durch eine andere Transaktion  $T_j$  eine Operation  $a_{jm}$  auf demselben Objekt  $e$  ausgeführt.
- Mindestens eine der beiden Operationen ist eine Schreib-Operation.
- In den Ausführungsschritten dazwischen wird keine Schreib-Operation auf  $e$  ausgeführt.

Formal:

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 10.20

- Schwächerer Begriff der Serialisierbarkeit: eine Ausführung heißt effektiserialisierbar (view-serializable), wenn sie denselben Output und dieselben Effekte auf eine Datenbank hat, wie eine beliebige serielle Ausführung.
- Es gilt: jede in unserem Sinn serialisierbare Ausführung ist effektiserialisierbar dh.
- Serialisierbarkeit ist eine hinreichende, aber keine notwendige Bedingung dafür, daß eine gegebene parallele Ausführung von Transaktionen effektiserialisierbar ist.

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 10.23

$I_2$  ist *seriell*,

$$D(I_1) = \{(T_1, a, T_2), (T_1, b, T_2)\}$$

$$D(I_2) = \{(T_1, a, T_2), (T_1, b, T_2)\}$$

$$D(I_3) = \{(T_1, a, T_2), (T_2, b, T_1)\}$$

$D(I_1) = D(I_2) \Rightarrow I_1$  *serialisierbar*.

Sei  $D(I_i) = \{(T_2, a, T_1), (T_2, b, T_1)\}$  die Konfliktrelation der zweiten möglichen *seriellen Ausführung*.

$$D(I_3) \neq D(I_2) \Rightarrow I_3 \text{ nicht serialisierbar.}$$

$I \dots$  Ausführung einer Menge  $\mathbf{T}$  von Transaktionen auf einer Menge von Objekten  $E$ .  $D(I) \subseteq \mathbf{T} \times E \times \mathbf{T}$

$(T_i, e, T_j) \in D(I) \Leftrightarrow$  es gibt  $k$  und  $l$ ,  $1 \leq k < l \leq |I|$ ,  $I(k) = (T_i, a_m, e)$ ,  $I(l) = (T_j, a_{jm}, e)$ , so daß gilt:

- $T_i \neq T_j$  und  $(a_m = \text{write oder } a_{jm} = \text{write})$
- Es gibt kein  $h$  für  $k > h > l$ , so daß  $I(h) = (T_j, a_{go}, e)$  wobei  $(T_j \neq T_i$  und  $T_j \neq T_j)$  und  $a_{go} = \text{write}$

$D(I)$  heißt **Konfliktrelation**.

Zwei Ausführungen  $I_1$  und  $I_2$  über denselben Transaktionsmenge  $\mathbf{T}$  heißen **(konflikt-)äquivalent** genau dann, wenn ihre Konfliktrelationen  $D(I_1)$  und  $D(I_2)$  identisch sind.

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 10.21

- Sie ist notwendig nur dann, wenn die Constraint Write Assumption (CWA) gilt.
- CWA: alle Transaktionen müssen vor einem Schreibzugriff auf ein Objekt dieses auch lesen.
- Wenn Transaktionen Objekte ohne vorheriges Lesen überschreiben können, so kann Effektiserialisierbarkeit auch dann vorliegen, wenn es keine äquivalente serielle Ausführung gibt.
- Annahme : die CWA gilt.

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 10.24

Der **Prädezenzgraph** für eine Ausführung  $I$  über eine Transaktionsmenge  $\mathbf{T}$  ist ein gerichteter Graph  $G(T) = (N, E)$ , dessen Knoten  $N$  die Transaktionen im  $\mathbf{T}$  sind, und in dem eine Kante von  $T_i$  nach  $T_j$  führt, genau dann wenn  $(T_i, e, T_j) \in D(I)$ , d.h.

- $T_j$  liest oder schreibt ein Objekt, das zuletzt von  $T_i$  geschrieben wurde, oder
- $T_i$  liest ein Objekt, auf das  $T_j$  danach als erste Transaktion schreibt.

Markiere Kante  $T_i \rightarrow T_j$  in  $G$  mit der Liste jener Objekte  $e$ , für die  $(T_i, e, T_j) \in D(I)$  gilt.

**Serialisierbarkeit**

Eine Ausführung  $I_1$  heißt **serialisierbar** genau dann, wenn es eine zu  $I_1$  äquivalente serielle Ausführung  $I_2$  gibt.

Serialisierbarkeit besagt, daß eine Ausführung  $I$  einer seriellen Ausführung  $I'$  entspricht  $\Rightarrow$  Konsistenz-Eigenschaft von Transaktionen ist gegeben.

**Theorem:** Eine Ausführung  $I$  über einer Transaktionsmenge  $\mathbf{T}$  ist genau dann serialisierbar, wenn es eine lineare Ordnung " $<$ " auf  $\mathbf{T}$  gibt, so daß für jedes Tripel  $I(k) = (T_i, a, e)$ ,  $I(l) = (T_j, a', e')$ , wo  $T_i \neq T_j$  und  $a, a'$  Konfliktoperationen sind, gilt:  $k < l \Leftrightarrow T_i < T_j$ .

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 10.22

**Beispiel 8** Betrachten wir drei verschiedene parallele Ausführungen von zwei Transaktionen  $T_1$  und  $T_2$ :

|         |         |         |         |         |         |
|---------|---------|---------|---------|---------|---------|
| $T_1$   | $T_2$   | $T_1$   | $T_2$   | $T_1$   | $T_2$   |
| READ A  |
| WRITE A |
| READ B  |
| WRITE B |
| $T_2$   | $T_1$   | $T_2$   | $T_1$   | $T_2$   | $T_1$   |

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien DBVO 10.25

Überprüfung der Serialisierbarkeit von Ausführungen mit graphentheoretischen Mitteln:

**Theorem:** Eine Ausführung  $I$  ist serialisierbar, genau dann wenn der Prädezenzgraph  $G(I)$  zyklentfrei ist.

Durch topologisches Sortieren: serielle Ausführung.

Der Prädezenzgraph gibt die lineare Ordnung  $<$  der Transaktionen von  $\mathbf{T}$ .

**Beispiel 9** Präzedenzgraph für  $I_k$  aus Beispiel 8

Anmerkung: i.A. keine eindeutige Ordnung.

$G(I_1)$  ist azylisch  $\Rightarrow$  serialisierbar mit der linearen Ordnung  $T_1 < T_2$ .

$G(I_1) = G(I_2)$   
 $G(I_3)$

**Beispiel 10** Betrachten wir die beiden Transaktionen  $T_1$  und  $T_2$ .

|       |             |       |             |
|-------|-------------|-------|-------------|
| $T_1$ | 1. LOCK A   | $T_2$ | 1. LOCK A   |
|       | 2. WRITE A  |       | 2. WRITE A  |
|       | 3. LOCK B   |       | 3. LOCK B   |
|       | 4. WRITE A  |       | 4. WRITE A  |
|       | 5. WRITE B  |       | 5. WRITE B  |
|       | 6. UNLOCK A |       | 6. UNLOCK A |
|       | 7. WRITE B  |       | 7. WRITE B  |
|       | 8. UNLOCK B |       | 8. UNLOCK B |

**Wohligeformte Transaktionen**

Eine Transaktion ist **wohligeformt**, wenn sie folgende Bedingungen erfüllt:

1. Eine Transaktion greift auf ein Objekt nur zu, wenn sie dieses zuvor gesperrt hat.
2. Eine Transaktion sperrt niemals ein Objekt, das sie bereits selbst gesperrt hat.
3. Eine Transaktion versucht niemals, ein Objekt freizugeben, das sie nicht zuvor gesperrt hat.
4. Vor Beendigung einer Transaktion werden die Sperren auf alle Objekte, die von ihr gesperrt wurden, wieder freigegeben.

**Sperreprotokolle**

- Dynamisches Sperren und Freigeben von Datenbankobjekten: am weitesten verbreitete Methode, Serialisierbarkeit von Ausführungen zu erlangen.
- Sperren: Aktion auf einem Objekt, die eine Transaktion durchführt, um das Objekt vor Zugriffen durch andere Transaktionen zu schützen, während es in inkonsistentem Zustand ist.
- Sperren werden benutzt, um die relative Abfolge von im Konflikt stehenden Operationen zu kontrollieren.

$D(I) = \{(T_1, A, T_2), (T_1, B, T_2)\}$

Eine gültige Ausführung  $I_1$  dieser beiden Transaktionen ist:

|       |              |       |              |
|-------|--------------|-------|--------------|
| $T_1$ | 1. LOCK A    | $T_2$ | 1. LOCK A    |
|       | 2. WRITE A   |       | 2. LOCK B    |
|       | 3. LOCK B    |       | 3. UNLOCK A  |
|       | 4. UNLOCK A  |       | 4. WRITE A   |
|       | 5. LOCK A    |       | 5. LOCK B    |
|       | 6. WRITE A   |       | 6. WRITE B   |
|       | 7. WRITE B   |       | 7. UNLOCK B  |
|       | 8. UNLOCK B  |       | 8. LOCK B    |
|       | 9. LOCK B    |       | 9. WRITE B   |
|       | 10. WRITE B  |       | 10. WRITE A  |
|       | 11. WRITE A  |       | 11. UNLOCK A |
|       | 12. UNLOCK A |       | 12. WRITE B  |
|       | 13. WRITE B  |       | 13. UNLOCK B |
|       | 14. UNLOCK B |       | 14. UNLOCK B |

- Sperren alleine ist nicht hinreichend, damit eine Transaktion im allgemeinen Fall serialisierbar ist.
- Die Ausführung im Beispiel 11 ist nicht serialisierbar, obwohl jedes Objekt vor dem Schreiben gesperrt wurde.
- Der Präzedenzgraph enthält einen Zyklus.

**Gültige Ausführungen**

Eine Ausführung ist **gültig**, wenn für jeden Ausführungsschritt gilt: wenn ein Objekt von einer Transaktion gesperrt ist, dann ist es durch keine weitere Transaktion gesperrt.

Für die Gültigkeit einer Ausführung ist es belanglos, ob auf die gesperrten Objekte auch tatsächlich zugegriffen wird.

$D(I_2) = \{(T_2, A, T_1), (T_2, B, T_1)\}$

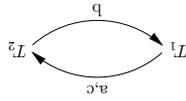
Ein weiteres Beispiel einer gültigen Ausführung  $I_2$  der Transaktionen  $T_1$  und  $T_2$  ist:

|       |              |       |              |
|-------|--------------|-------|--------------|
| $T_1$ | 1. LOCK A    | $T_2$ | 1. LOCK A    |
|       | 2. WRITE A   |       | 2. LOCK B    |
|       | 3. LOCK B    |       | 3. WRITE B   |
|       | 4. WRITE B   |       | 4. UNLOCK B  |
|       | 5. WRITE A   |       | 5. UNLOCK A  |
|       | 6. UNLOCK A  |       | 6. LOCK A    |
|       | 7. LOCK A    |       | 7. WRITE A   |
|       | 8. WRITE A   |       | 8. LOCK B    |
|       | 9. WRITE B   |       | 9. UNLOCK B  |
|       | 10. UNLOCK B |       | 10. LOCK B   |
|       | 11. LOCK B   |       | 11. UNLOCK A |
|       | 12. UNLOCK A |       | 12. WRITE B  |
|       | 13. WRITE B  |       | 13. UNLOCK B |
|       | 14. UNLOCK B |       | 14. UNLOCK B |

**Beispiel 11**

|       |          |       |          |
|-------|----------|-------|----------|
| $T_1$ | LOCK a   | $T_2$ | LOCK a   |
|       | LOCK c   |       | WRITE a  |
|       | WRITE a  |       | LOCK b   |
|       | WRITE c  |       | UNLOCK a |
|       | UNLOCK a |       | WRITE b  |
|       |          |       | UNLOCK b |
|       |          |       | UNLOCK c |
|       |          |       | LOCK c   |
|       |          |       | WRITE c  |
|       |          |       | UNLOCK c |

Der dazugehörige Präzedenzgraph sieht folgendermaßen aus:



Hinreichende Bedingung für Serialisierbarkeit:  
**Theorem:** Wenn alle Transaktionen einer Transaktionsmenge **T** wohlgeformt und 2-Phasen-Transaktionen sind, dann ist jede gültige Ausführung von **T** serialisierbar.  
**Serialisierungsreihenfolge:** eine äquivalente serielle Ausführung ist nach der Reihenfolge der Sperrpunkte der einzelnen Transaktionen aus **T** möglich.  
**Praxis:** UNLOCK-Befehle werden bis zum Ende der Transaktion verzögert und/oder zu einem Befehl (COMMIT) zusammengefaßt, der das Ende der Transaktion darstellt.

### Sperrtypen

Zur Verminderung von Deadlocks: zwei verschiedene Sperrtypen  
**S-LOCK** (shared lock), wenn ein Objekt nur zum Lesen gebraucht wird,  
**X-LOCK** (exclusive lock), wenn das Objekt zum Lesen und Schreiben verwendet werden soll.  
Kompatibilitätsrelationen:

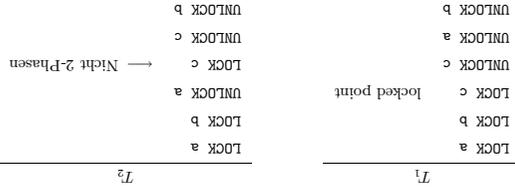
|                   |        |        |        |
|-------------------|--------|--------|--------|
| Sperranforderung  | S-LOCK | S-LOCK | X-LOCK |
|                   | X-LOCK | +      | -      |
| Bestehende Sperre | S-LOCK | S-LOCK | X-LOCK |
|                   | X-LOCK | -      | -      |

+ ... Verträglich, - ... Konflikt

Eine Transaktion  $T = \langle (a_i, e_i) \rangle$ ,  $i = 1, \dots, n$  ist eine 2-Phasen-Transaktion, wenn für ein  $j < n$  und für alle  $i = 1, \dots, n$  gilt:

- $i < j \Rightarrow a_i << UNLOCK$  Sperrphase (Wachstumsphase)
- $i = j \Rightarrow a_i = UNLOCK$  Sperrpunkt
- $i > j \Rightarrow a_i <> LOCK$  Schrumpfungsphase

### Das 2-Phasen-Sperrverfahren (2-Phase-Locking)

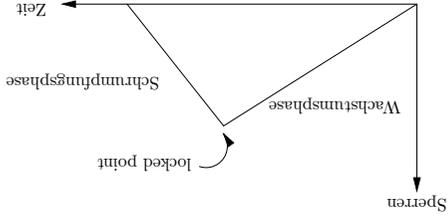


Beispiel 12 Die folgenden beiden Tabellen zeigen zwei Transaktionen, von denen die erste das 2-Phasen-Sperrprotokoll erfüllt, die zweite nicht.

### Das Baumprotokoll

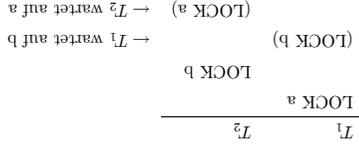
Ansammlung von Objekten (Datenbanken, Menge von Relationen, Relationen, oder Typel) gespeichert als Knoten eines Baumes (z.B. B+-Baum).  
Verwendung von exekutiven Sperren.  
Eine Transaktion  $T$  erfüllt das Baumprotokoll, wenn gilt:

- Ein Objekt  $O$  wird nur dann von  $T$  gesperrt, wenn  $T$  bereits eine Sperre auf den Vater von  $O$  hält.
- Regel 1. gilt nicht für das erste Objekt im Baum, das von  $T$  gesperrt wird.
- Ein Objekt  $O$ , das von einer Transaktion  $T$  gesperrt und wieder freigegeben wurde, kann von  $T$  anschließend nicht mehr gesperrt werden.



Dynamisches Sperren von Objekten und 2-Phasen Sperrprotokolle führen manchmal zum Problem der Verklemmung (deadlock). Deadlocks müssen entweder von vornherein vermieden oder erkannt und aufgelöst werden.

### Deadlock



Hinreichende Bedingung für Serialisierbarkeit:

**Theorem:** Wenn alle Transaktionen einer Transaktionsmenge  $T$  wohlgeformt sind und das Baumprotokoll erfüllen, dann ist jede gültige Ausführung  $I$  von  $T$  serialisierbar.  
**ACHTUNG:** hinreichend  $\neq$  notwendig!

**Beispiel 13**

$T_1, T_2$  und  $T_3$  erfüllen das Baumprotokoll, die  $\mathbf{T}$  ist wohlgeformt,  $T$  ist gültig  $\Rightarrow T$  serialisierbar.

|     |          |          |          |          |          |          |          |
|-----|----------|----------|----------|----------|----------|----------|----------|
| 1.  | LOCK A   | UNLOCK B | LOCK C   | UNLOCK A | LOCK E   | UNLOCK G | UNLOCK E |
| 2.  | LOCK B   | UNLOCK B | LOCK C   | UNLOCK A | LOCK E   | UNLOCK G | UNLOCK E |
| 3.  | UNLOCK B | LOCK B   | LOCK C   | UNLOCK A | LOCK E   | UNLOCK G | UNLOCK E |
| 4.  | LOCK B   | UNLOCK B | LOCK C   | UNLOCK A | LOCK E   | UNLOCK G | UNLOCK E |
| 5.  | LOCK C   | UNLOCK B | LOCK C   | UNLOCK A | LOCK E   | UNLOCK G | UNLOCK E |
| 6.  | UNLOCK B | UNLOCK B | LOCK C   | UNLOCK A | LOCK E   | UNLOCK G | UNLOCK E |
| 7.  | UNLOCK B | UNLOCK B | UNLOCK C | UNLOCK A | LOCK E   | UNLOCK G | UNLOCK E |
| 8.  | UNLOCK B | UNLOCK B | UNLOCK C | UNLOCK A | UNLOCK E | UNLOCK G | UNLOCK E |
| 9.  | UNLOCK C | UNLOCK B | UNLOCK C | UNLOCK A | UNLOCK E | UNLOCK G | UNLOCK E |
| 10. | UNLOCK C | UNLOCK B | UNLOCK C | UNLOCK A | UNLOCK E | UNLOCK G | UNLOCK E |
| 11. | UNLOCK C | UNLOCK B | LOCK E   | UNLOCK A | UNLOCK E | UNLOCK G | UNLOCK E |
| 12. | UNLOCK C | UNLOCK B | LOCK E   | UNLOCK A | LOCK E   | UNLOCK G | UNLOCK E |
| 13. | UNLOCK C | UNLOCK B | UNLOCK E | UNLOCK A | LOCK E   | UNLOCK G | UNLOCK E |
| 14. | UNLOCK C | UNLOCK B | UNLOCK E | UNLOCK A | UNLOCK E | UNLOCK G | UNLOCK E |

Eine hinreichende Bedingung für Serialisierbarkeit ist:

**Theorem:** wenn alle Transaktionen einer Transaktionsmenge  $T$  wohlgeformt sind und das Hierarchische Sperriprotokoll erfüllen, dann ist jede gültige Ausführung von  $T$  serialisierbar.

**Kompatibilitätsrelation:**

|                  |        |        |
|------------------|--------|--------|
| Sperranforderung | I-Lock | X-Lock |
| I-Lock           | +      | -      |
| X-Lock           | -      | -      |

Bestehende Sperre

+ ... Verträglich, - ... Konflikt

DBVO 10.50

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

Konsistenz wird sichergestellt:

- Eine Transaktion mit Zeitstempel  $t_1$  darf ein Objekt mit Schreibstempel  $t_m$  nicht lesen, wenn  $t_m > t_1$  gilt, also das Objekt bereits von einer jüngeren Transaktion verändert wurde. Die Transaktion muß in diesem Fall abgebrochen werden.
- Eine Transaktion mit Zeitstempel  $t_1$  darf ein Objekt mit Lesestempel  $t_r$  nicht verändern, wenn  $t_r > t_1$  gilt, also das Objekt bereits von einer jüngeren Transaktion gelesen wurde. Die Transaktion muß in diesem Fall abgebrochen werden.

**Abbruch einer Transaktion:** sie wird neu gestartet und erhält die dem Zeitpunkt des Neustarts entsprechende Zeitmarke.

**Serialisierungsreihenfolge:** entspricht der Reihenfolge der Zeitstempel, also der Startzeitpunkte.

**Das Hierarchische Sperriprotokoll**

Zwei Arten von Sperren:

das Exklusive Lock (X-Sperre) ("Absichtserklärung", daß später ein Subobjekt (Teilbaum) gesperrt wird),

**Beispiel 14** I-Sperre auf Relation  $r$ : ein Tupel in  $r$  wird später bearbeitet und exklusiv gesperrt;

I-Sperre auf  $r$  verhindert, daß eine andere Transaktion die ganze Relation  $r$  exklusiv sperrt;

eine andere Transaktion kann gleichzeitig eine I-Sperre auf  $r$  legen und später ein anderes Tupel exklusiv sperren.

**Beispiel 15**

$T_1, T_2$  und  $T_3$  erfüllen das hierarchische Sperriprotokoll,  $T$  ist wohlgeformt und  $I$  ist gültig  $\Rightarrow I$  serialisierbar.

|     |          |             |          |          |             |                   |
|-----|----------|-------------|----------|----------|-------------|-------------------|
| 1.  | I-LOCK A | I-LOCK A, B | I-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 2.  | I-LOCK A | I-LOCK A, B | I-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 3.  | I-LOCK A | I-LOCK A, B | I-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 4.  | I-LOCK A | I-LOCK A, B | I-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 5.  | I-LOCK A | I-LOCK A, B | X-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 6.  | I-LOCK A | I-LOCK A, B | X-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 7.  | I-LOCK A | I-LOCK A, B | X-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 8.  | I-LOCK A | I-LOCK A, B | X-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 9.  | I-LOCK A | I-LOCK A, B | X-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 10. | I-LOCK A | I-LOCK A, B | X-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 11. | I-LOCK A | I-LOCK A, B | X-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 12. | I-LOCK A | I-LOCK A, B | X-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 13. | I-LOCK A | I-LOCK A, B | X-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |
| 14. | I-LOCK A | I-LOCK A, B | X-LOCK B | UNLOCK D | UNLOCK B, A | UNLOCK B, F, C, A |

DBVO 10.51

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

**Achtung:** zwei Leszugriffe: kein Konflikt.

zwei Schreibzugriffe ohne einen Leszugriff dazwischen: nur der jüngere geschrieben wird (siehe Fall 3 im Algorithmus).

**Realisierung:** 2 Phasen, um Transaktionen erfolgreich rückgängig zu machen.

**Phase 1:** die geänderten Werte eines Objekts werden in einen zu dem Objekt gehörenden privaten Arbeitsbereich geschrieben;

**Phase 2:** bei erfolgreicher Beendigung der Transaktion werden die geänderten Werte in die Datenbank übernommen.

Eine wohlgeformte 2-Phasen-Transaktion  $T$  gehorcht dem Hierarchischen Sperriprotokoll, wenn gilt:

- $T$  greift auf die Knoten des Baumes in hierarchischer Reihenfolge von der Wurzel ausgehend zu.
- $T$  sperrt einen Knoten  $K$  nur dann, wenn sie eine I-Sperre auf alle Vorgänger von  $K$  besitzt.
- N.B. Es ist nicht nötig einen Knoten mit einer X-Sperre zu belegen, wenn sein Vorgängerknoten schon eine X-Sperre besitzt.

$T$  gibt einen Knoten  $K$  erst dann frei, wenn sie alle Nachfolger von  $K$  freigegeben hat.

DBVO 10.49

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

**Zeitstempelverfahren (Time Stamping)**

Zurordnung von Zeitstempeln (time stamps) zu Transaktionen und Datenbankobjekten.

- Jede Transaktion erhält als eindeutigen Zeitstempel den Zeitpunkt des Transaktionsbeginns.
- Jedes Datenbankobjekt übernimmt sowohl den Zeitstempel der letzten zugegriffenen Schreib-Transaktion, als auch den der letzten Les-Transaktion.

DBVO 10.52

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

**Algorithmus:**

verändert Zeitstempel und bricht gegebenenfalls Transaktionen ab.

$T$  .. Transaktion, die die Operation  $X$  auf dem Datenbankobjekt  $O$  ausführt.

$t$  ... Zeitpunkt des Beginns von  $T$ ,

$t_r$  ... Lesestempel von  $O$ ,

$t_w$  ... Schreibstempel von  $O$ .

Nach den Werten dieser Stempel wird genau eine der Aktionen (1) bis (4) ausgeführt.

**Beispiel 16**

|       |        |        |       |          |          |
|-------|--------|--------|-------|----------|----------|
| $T_1$ | 200    | READ B | $T_1$ | 175      | WRITE A  |
| $T_2$ | 150    | READ A | $T_2$ | 150      | WRITE B  |
| $T_3$ | 175    | READ C | $T_3$ | 200      | WRITE C  |
| A     | RT = 0 | WT = 0 | A     | RT = 150 | WT = 200 |
| B     | RT = 0 | WT = 0 | B     | RT = 200 | WT = 200 |
| C     | RT = 0 | WT = 0 | C     | RT = 175 | WT = 200 |

$T_2$  wird abgebrochen und bekommt einen neuen Zeitstempel, der größer als 200 ist, z.B. 250.  $\Rightarrow T_3 < T_1 < T_2$

**Beispiel 17** Durch zyklisches Abbrechen von  $T_1$  und  $T_2$  tritt ein Livelock (unbeschränkte Verzögerung) auf.

|       |        |         |       |        |         |       |        |         |       |        |          |
|-------|--------|---------|-------|--------|---------|-------|--------|---------|-------|--------|----------|
| $T_1$ | 100    | WRITE B | $T_1$ | 110    | WRITE A | $T_1$ | 120    | WRITE B | $T_1$ | 130    | WRITE A  |
| $T_2$ | 110    | WRITE A | $T_2$ | 120    | WRITE B | $T_2$ | 130    | WRITE A | $T_2$ | 140    | WRITE B  |
| A     | RT = 0 | WT = 0  | A     | RT = 0 | WT = 0  | A     | RT = 0 | WT = 0  | A     | RT = 0 | WT = 100 |
| B     | RT = 0 | WT = 0  | B     | RT = 0 | WT = 0  | B     | RT = 0 | WT = 0  | B     | RT = 0 | WT = 120 |

DBVO 10.59

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

**SQL-92:** verschiedene Isolationsstufen (Isolation levels).

|                  |               |                    |               |
|------------------|---------------|--------------------|---------------|
| Stufe            | Dirty Read    | Nonrepeatable Read | Phantom Read  |
| READ UNCOMMITTED | möglich       | möglich            | möglich       |
| READ COMMITTED   | nicht möglich | möglich            | möglich       |
| REPEATABLE READ  | nicht möglich | nicht möglich      | möglich       |
| SERIALIZABLE     | nicht möglich | nicht möglich      | nicht möglich |

SET TRANSACTION ISOLATION LEVEL  
 %B, SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

**Transaktionen in SQL**

- Kein spezieller Befehl zum Starten einer Transaktion in SQL.
- Die meisten SQL-Statements können nur innerhalb einer Transaktion ablaufen (z.B. SELECT, INSERT, CREATE TABLE, nicht aber z. B. CONNECT).
- Transaktionen werden vom System gestartet, wenn ein solcher Befehl auftritt.
- Beendigung mit dem Befehl COMMIT  $\Rightarrow$  Speicherung.
- Abbruch mit dem Befehl ROLLBACK  $\Rightarrow$  Zurücknahme aller Änderungen.
- Setzen von Attributen mit SET TRANSACTION:

```

SET TRANSACTION READ ONLY
SET TRANSACTION READ WRITE

```

DBVO 11.1

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

**Wiederanlauf (Recovery)**

**Deadlock:** gibt es beim Zeitstempelverfahren nicht (Abbruch).  
**Livelock:** entsteht durch eine unendliche Folge von Abbrüchen, (Indefinite postponement)

DBVO 11.2

Gerald Pfeifer <http://gerald.pfeifer.com> TU Wien

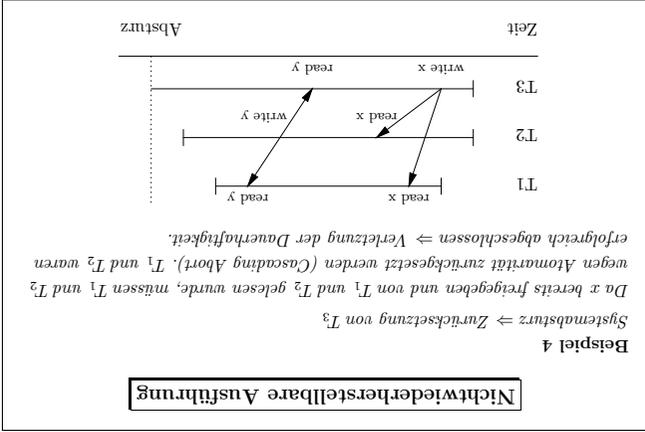
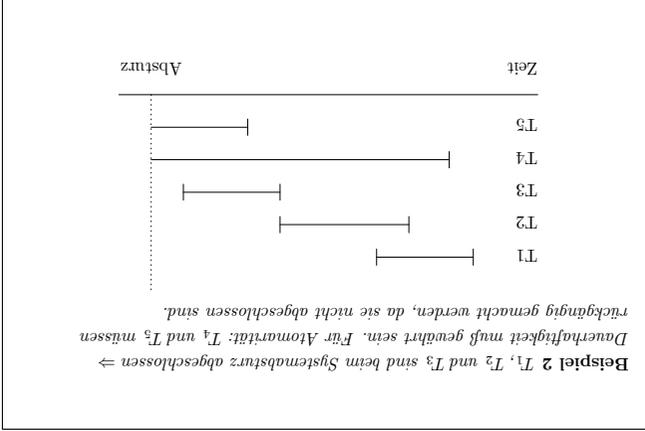
**Ziele**

**Wiederherstellung** eines konsistenten Datenbankszustandes nach einem Fehler:

**Fehler:**

- Transaktionsabbruch:** eine Transaktion muß nach einem logischen Fehler (z.B. „Konto nicht gedeckt“) abgebrochen werden.
- Systemfehler:** alle Änderungen, die nur im Hauptspeicher, aber noch nicht auf der Festplatte durchgeführt wurden, gehen verloren.
- Plattenfehler:** alle Änderungen, die seit der letzten Sicherung auf der Festplatte durchgeführt wurden, gehen verloren.

- Korrektes Wiederanlaufverfahren garantiert **Dauerhaftigkeit** und **Atomarität**.
- Abbruch von Transaktionen:
  - selbst (z.B. durch Programmierer oder durch Laufzeitfehler im Anwendungsprogramm) oder
  - durch DBMS (z.B. durch Concurrency Control oder bei Auftreten von Deadlocks).
- Teilweise durchgeführte Transaktionen sollen keinerlei Effekt auf die Datenbank haben. (Rückwärts-Recovery, Rollback)
- Synchronisationspunkte: Punkte innerhalb einer Transaktion, Rückgängigmachen nur bis zu diesem Punkt nötig.



**Nichtwiederherstellbare Ausführung**

**Beispiel 1** *Bankenszenario (Umbuchung)*

**Datenstruktur:**  
 SPARBUCH (SPNR, EINLAGE, BESETZER)  
 KONTO (KMR, KSTAND)  
 SPAREINLAGEN\_GESAMT

**Integritätsbedingung:**  
 KSTAND >= 0  
 EINLAGE > 0  
 SPAREINLAGEN\_GESAMT = Summe EINLAGE

**Reihenfolge von Ausführungen**

Eine Transaktion  $T_j$  liest  $X$  vor einer Transaktion  $T_i$  falls:

1.  $T_j$  liest  $X$  zum Zeitpunkt  $t_j$ ,  $T_i$  schreibt  $X$  zum Zeitpunkt  $t_i$ , und  $t_i < t_j$ .
2.  $T_i$  wird nicht vor  $t_j$  abgebrochen.
3. jede Transaktion  $T_k$ , die  $X$  zum Zeitpunkt  $t_k$ ,  $t_i < t_k < t_j$  schrieb, wurde vor  $t_j$  abgebrochen.

**Problem:** Zurücksetzen von Transaktionen:  
 Bereits erfolgreich abgeschlossene Transaktionen werden kaskadenartig zurückgesetzt  $\Rightarrow$  Dauerhaftigkeit wird verletzt.

**Lösung:** Transaktion gibt Änderungen erst frei, wenn sichergestellt ist, daß die Transaktion ordnungsgemäß beendet werden kann (**Isolation**).

**Praxis:** Einführung der Isolation um Widerspruch zwischen Dauerhaftigkeit und Zurücksetzen zu verhindern  $\Rightarrow$  *Wiederherstellbarkeit* ist garantiert.

**Transaktion:** *Umbuchung von 50 € von Konto 123 auf das Sparbuch 321*

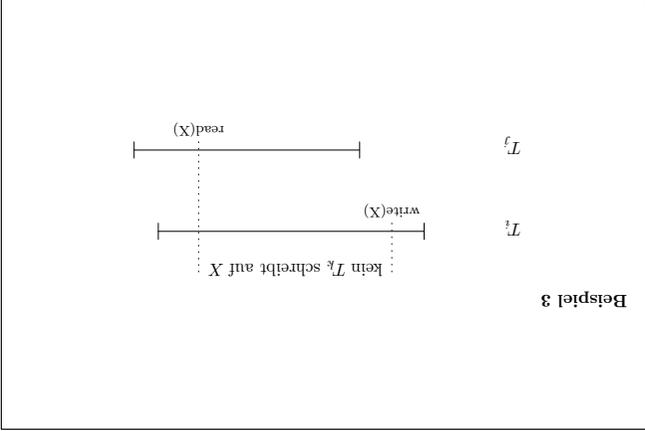
```

begin transaction
update KONTO
where KMR=123;
set KSTAND=KSTAND-50
update SPARBUCH
where SPNR=321
set EINLAGE=EINLAGE+50
SPAREINLAGEN_GESAMT := SPAREINLAGEN_GESAMT+50
end transaction

```

**Systemabsturz** während einer Umbuchung: einseitige Buchung möglich.

**Systemanlauf:** konsistenter DB-Zustand (d.h. Buchungen werden vollständig durchgeführt oder gar nicht) muß hergestellt werden.



**Problem:** kaskadierendes Zurücksetzen

**Beispiel 5**  $T_2$  muß bei einem *COMMIT-Request* abbrechen.

|    |          |       |
|----|----------|-------|
| 1. | WRITE(X) | $T_1$ |
| 2. | READ(X)  | $T_2$ |
| 3. | ABORT    |       |

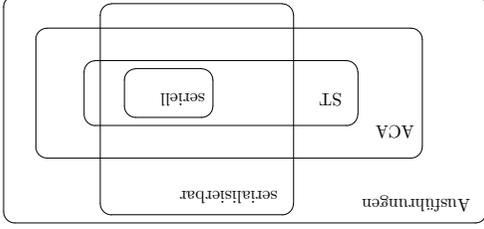
Ausführung  $I$  verhindert kaskadierendes Zurücksetzen (avoids cascading abort, ACA), wenn für jedes  $T_i$  und  $T_j$ ,  $i \neq j$  gilt:  
 wenn  $T_i$   $X$  von  $T_j$  liest, dann ist  $T_j$  bereits erfolgreich abgeschlossen, i.e. Commit von  $T_j$  kommt vor  $READ_i(X)$  von  $T_i$  in  $I$ .

**Zurücksetzen einer Transaktion**

- Typische Implementierungsmöglichkeiten:
  - Markieren des alten Wertes bei WRITE (Before Image)
  - Restaurieren des alten Wertes nach dem Before Image
- nur korrekt, wenn Transaktionen Objekte nur dann schreiben, wenn sie zuletzt von erfolgreich abgeschlossenen Transaktionen geschrieben wurden.

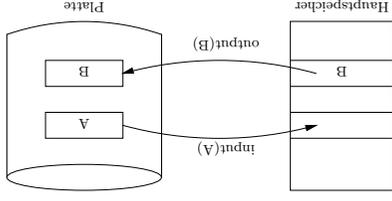
Es gilt folgende Beziehung:

$$\text{serial} \subset \text{ST} \subset \text{ACA}$$



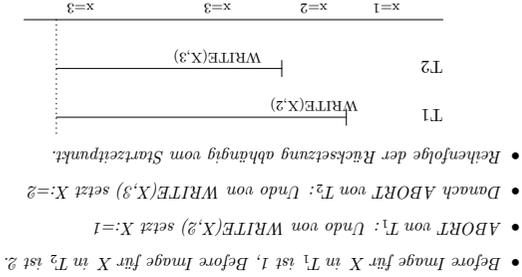
Ab hier wird immer Strictness angenommen.

**Zugriff auf Blöcke des Hintergrundspeichers**



**input(X)** überträgt den physischen Block, der das Datenobjekt X enthält, in den Hauptspeicher (Puffer).  
**output(X)** überträgt den Pufferblock, der das Datenobjekt X enthält, auf die Festplatte und ersetzt dort den entsprechenden physischen Block.

**Beispiel 6**



- Before Image für X in T<sub>1</sub> ist 1, Before Image für X in T<sub>2</sub> ist 2.
- ABORT von T<sub>1</sub>: Undo von WRITE(X,2) setzt X:=1
- Danach ABORT von T<sub>2</sub>: Undo von WRITE(X,3) setzt X:=2
- Reihenfolge der Rücksetzung abhängig vom Startzeitpunkt.

**Pufferverwaltung**

- Drei Arten von Speichern:
- Hauptspeicher (volatile storage)
  - Hintergrundspeicher (non-volatile storage): z.B.: Festplatte
  - Stabiler Speicher (stable storage): Information geht nicht verloren. (Annahme!)

**Programmzugriff auf DB**

- read(X,x<sub>i</sub>)** weist den Wert des Datenobjekts X der Programmvariable x<sub>i</sub> zu.  
 1. Ist der Block, auf dem sich X befindet, noch nicht im Hauptspeicher, dann führe **input(X)** aus.  
 2. Weise x<sub>i</sub> den Wert von X im Pufferblock zu.
- write(X,x<sub>i</sub>)** weist den Wert der Programmvariable x<sub>i</sub> dem Datenobjekt X im Pufferblock zu.  
 1. Ist der Block, auf dem sich X befindet, noch nicht im Hauptspeicher, dann führe **input(X)** aus.  
 2. Ersetze den Wert von X im Pufferblock durch den Wert von x<sub>i</sub>.

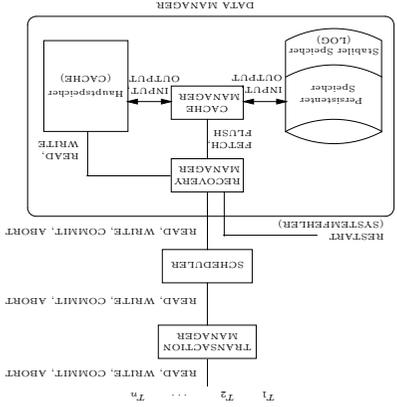
**Strikte Ausführung**

Eine Ausführung I von Transaktionen ist strikt (ST), wenn für jede Transaktion gilt:

T<sub>j</sub> liest und schreibt nur Objekte X, die zuletzt von erfolgreich abgeschlossenen Transaktionen T<sub>j</sub> geschrieben wurden.

**Praxis:** (2-Phasen Sperrverfahren garantiert Strictness)

- Alle WRITE-Befehle werden bis zum locked point verzögert
- UNLOCK-Befehle werden bis Transaktionsende verzögert
- alle UNLOCKS werden zu einem Befehl (COMMIT) zusammengesfaßt
- COMMIT = Ende der Transaktion.



**Puffer als Cache**

- Änderungen im Puffer werden nicht immer sofort auf Hintergrundspeicher übertragen.
- Effizienz: Datenobjekt, auf das oft zugegriffen wird, länger im Puffer behalten.
- Übertragung des Pufferblocks vom Pufferwähler auf die Festplatte wenn 1. Puffer wird zu klein.
- 2. vom Wiederanlaufwähler verlangt (**forced write**).

**Absturz**

Mögliche inkonsistente Datenbankzustände:

- Änderungen abgeschlossener Transaktionen nur im Puffer, aber noch nicht auf der Festplatte.
- Änderungen noch nicht abgeschlossener Transaktionen bereits auf der Festplatte.

**Privater Arbeitsbereich**

- Pufferverwaltung: separater Puffer für jede Transaktion.
- Effekte einer Transaktion  $T$ :
  - $T$  wird privater Arbeitsbereich als Puffer für Datenbankobjekte zugeordnet.
  - Read Object  $X$ :** Suche in privatem Arbeitsbereich von  $T$  nach  $X$ .  $X$  nicht gefunden  $\Rightarrow$  Übertrag von  $X$  von Festplatte in Arbeitsbereich.

**Wiederanlaufverfahren mit Logprotokoll**

- Private Speicherverwaltung kann kompliziert sein.
- Sofortige Eintragung aller Änderungen und aller für korrekten Wiederanlauf nötigen Informationen ins Logprotokoll.
- Kein privater Speicher mehr notwendig, alle Daten im globalen Cache. Logprotokoll enthält Logeinträge als Vermerk von
- Start (**begin transaction**),
- Ende (**commit transaction**),
- Abbruch (**abort transaction**) einer Transaktion,
- Standard-Logeinträge, registrieren alle durchgeführten Änderungen.

**Beispiel 7**

```
T: read(A, a1)
 a1 = a1 - 50
 write(A, a1)
 read(B, b1)
 b1 = b1 + 50
 write(B, b1)
```

**Write Object  $X$ :**  $X$  im Arbeitsbereich gefunden  $\Rightarrow$  überschreibe  $X$ .  
 Sonst: trage neuen Wert im Arbeitsbereich ein.

**End Transaction  $T$ :**

- Übertrage alle im Arbeitsbereich geänderten oder neu geschaffenen Werte von Objekten auf die Festplatte.
- spezielles Protokoll garantiert, daß entweder alle oder keine Werte übertragen werden.
- Freigabe des privaten Arbeitsbereichs.

**Standardeintrag im Logprotokoll**

1. ID der Transaktion
2. ID des geänderten Datensatzes
3. Art der Änderung (insert, delete, update)
4. alter Wert (für UNDO)
5. neuer Wert (für REDO)
6. Hilfsinformation (Zeiger auf vorigen Logeintrag der Transaktion, etc.)

Nach Abschluß der Transaktion: Änderungen nur im Hauptspeicher, aber nicht auf der Platte.

**Write Ahead-Protokoll**

Realisiert privaten Arbeitsbereich. Annahme: ab hier strikte und serialisierbare Ausführung.

**1. Phase:** Alle zu ändernden Werte werden aus dem privaten Arbeitsbereich in stabilen Speicherbereich (**Log**) übertragen (erfolgt von einem speziellen **Commit Record**).

**2. Phase:** Die geänderten Werte werden in die Datenbank auf der Festplatte eingetragen.

- Systemausfall während 1. Phase: die Transaktion gilt als nicht durchgeführte. Die Festplatte ist und bleibt unverändert.
- Systemausfall während 2. Phase: DBS muß beim Wiederanlauf alle Updates aus dem stabilen Speicher übernehmen.

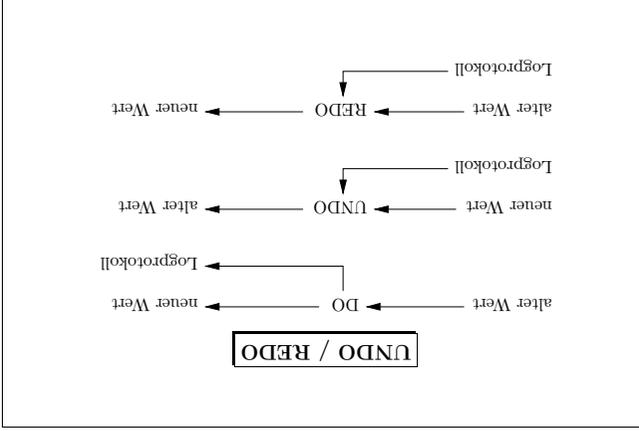
**Transaktionsrecovery vom Logprotokoll**

- um Transaktionen rückgängig zu machen (**Rollback, backward recovery**):  $UNDO(T)$
- um Transaktionen nachzuführen (**forward recovery**):  $REDO(T)$ .

Fehler auch während des Wiederanlaufs möglich

$\Rightarrow UNDO$  und  $REDO$  müssen **idempotent** sein:

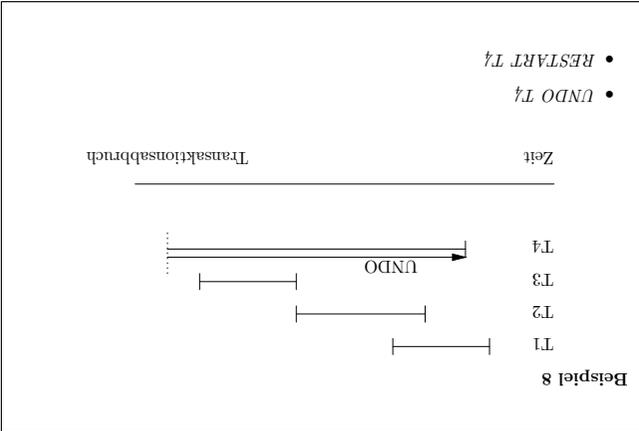
$$UNDO(UNDO(T)) = UNDO(T) \text{ und } REDO(REDO(T)) = REDO(T)$$



5. Restart  $T_i$  in **Undoliste**.  
durch.

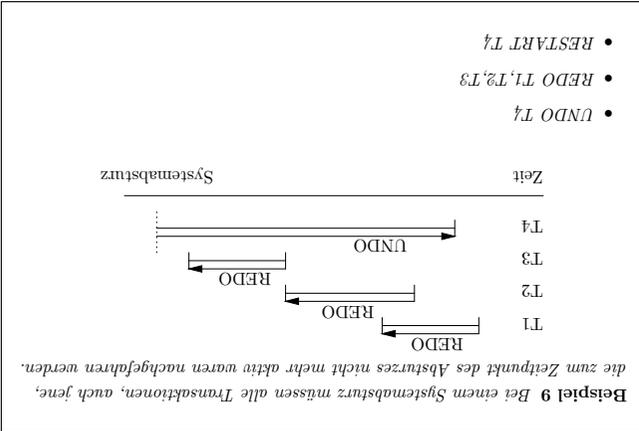
**Checkpoint:**  
Erzwingen der Übereinstimmung von Platte und Hauptspeicher für schnelleren Wiederanlauf.  
1. alle "schmutzigen" Pufferblöcke auf Platte schreiben  
2. spezieller Logeintrag, **checkpoint L<sub>i</sub>**, auf stabilen Speicher schreiben, (**L** ... Menge der zum Zeitpunkt des Checkpoints aktiven Transaktionen).  
Bis zum nächsten Checkpoint:  
• vermerke alle Änderungen im Logprotokoll.  
• Änderungen werden unter Umständen nur im Hauptspeicher und nicht in der Datenbank vorgenommen.

**Wiederanlauf nach einem Systemabsturz**  
Voraussetzung: Logprotokoll mit den alten Werten ist vorhanden.  
**Algorithmus:**  
**UNDO(T):** Lies das Logprotokoll vom letzten Eintrag in Richtung Anfang bis zum Eintrag **begin transaction T** und ersetze sämtliche Änderungen von **T** in der Datenbank durch die alten Werte.  
**RESTART(T):** Starte **T** neu.

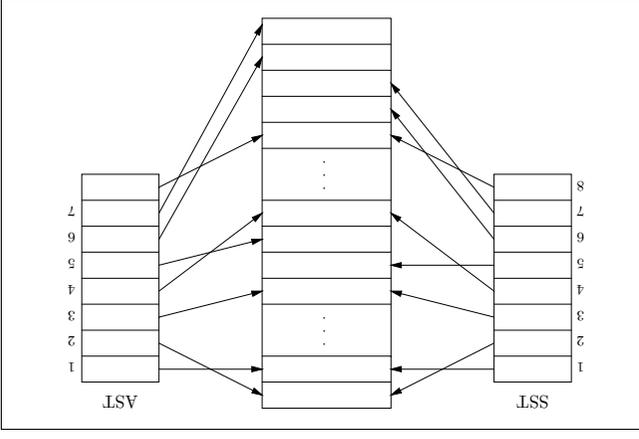
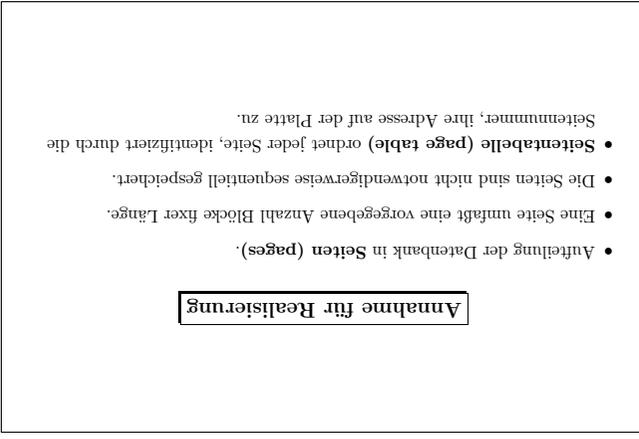
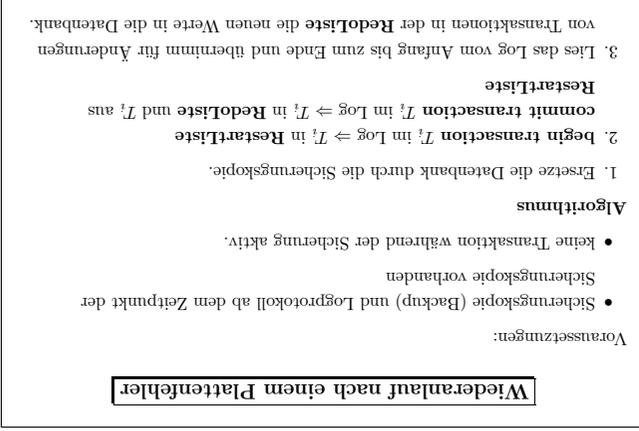
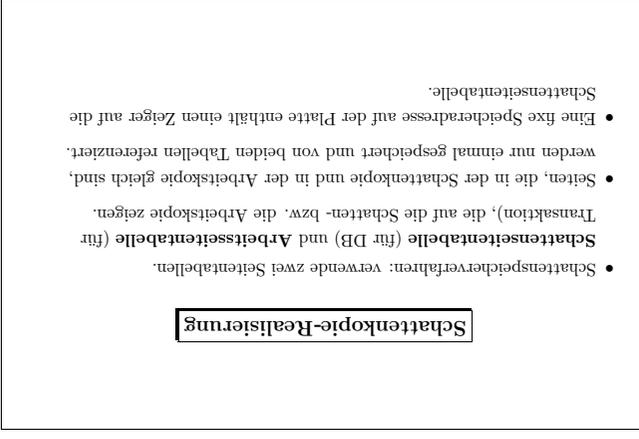
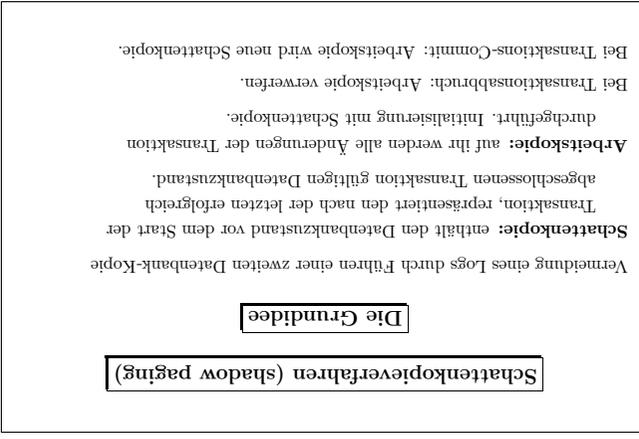
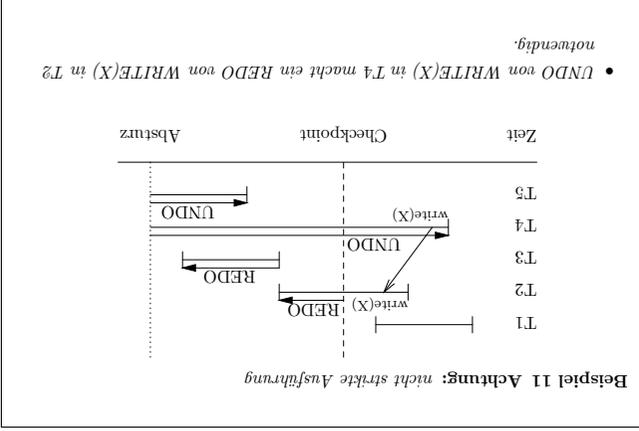
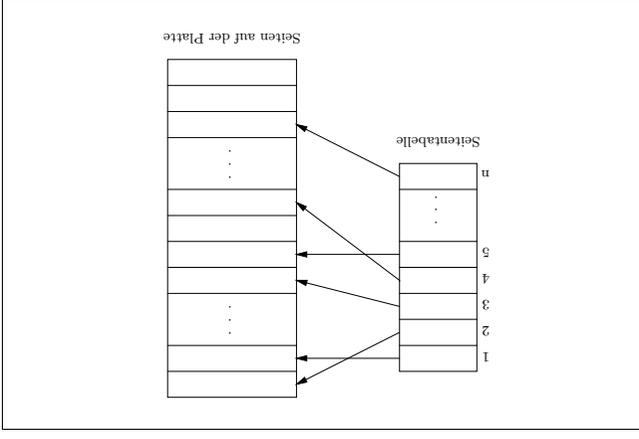
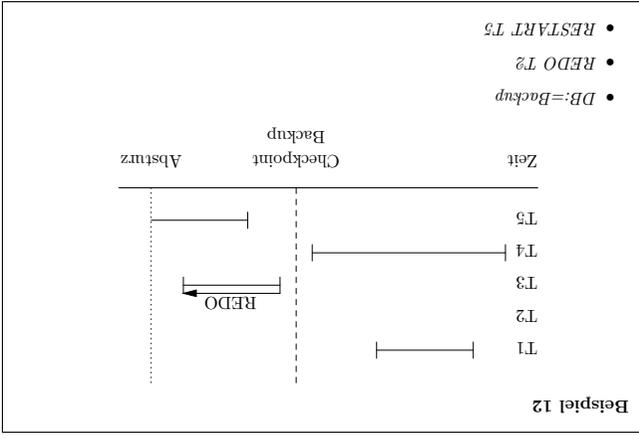
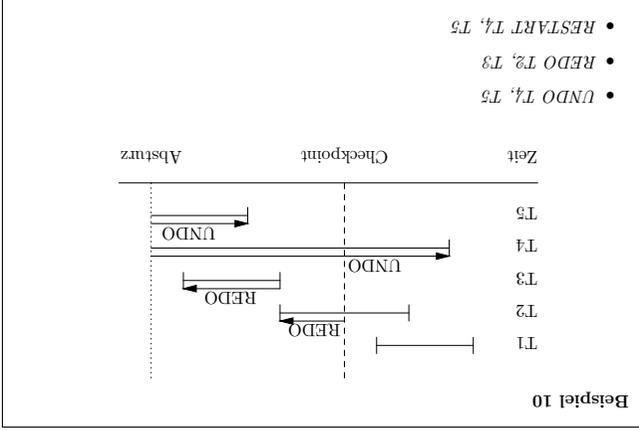


**Wiederanlauf nach einem Systemabsturz**  
Voraussetzung: Logprotokoll ist ab dem Startzeitpunkt der ältesten beim letzten Checkpoint aktiven Transaktionen auf stabilem Speicher vorhanden.  
**Algorithmus:**  
1. Baue **Undoliste** und **Redoliste** auf:  
• Suche den letzten Checkpoint-Eintrag **checkpoint L<sub>i</sub>**;  
**Undoliste** := **L<sub>i</sub>**, **Redoliste** := {}  
lies von **L** an vorwärts (in Richtung des letzten Eintrags).  
• wird ein Logeintrag **commit transaction T<sub>i</sub>** gefunden, füge **T<sub>i</sub>** zur **Redoliste** hinzu und streiche **T<sub>i</sub>** von der **Undoliste**.  
• wird ein Logeintrag **begin transaction T<sub>i</sub>** gefunden, füge **T<sub>i</sub>** zur **Undoliste** hinzu.

**Wiederanlauf nach einem Systemabsturz**  
Voraussetzung: Logprotokoll mit den alten Werten ist vorhanden.  
**Algorithmus:**  
1. Baue **Undoliste** und **Redoliste** auf:  
• Lies Logprotokoll vorwärts, bei **begin transaction T<sub>i</sub>** füge **T<sub>i</sub>** zur **Undoliste** hinzu.  
• bei **commit transaction T<sub>i</sub>** füge **T<sub>i</sub>** zur **Redoliste** hinzu und streiche **T<sub>i</sub>** von der **Undoliste**.  
2. Lies Logprotokoll rückwärts und führe das UNDO für **T<sub>i</sub>** in **Undoliste** durch.  
3. Lies Logprotokoll rückwärts, bis alle **begin transaction**-Einträge für **T<sub>i</sub>** in **Redoliste** gefunden wurden.  
4. Lies das Logprotokoll vorwärts und führe ein REDO für **T<sub>i</sub>** in **Redoliste**



**Wiederanlauf (Forts.)**  
2. Lies das Logprotokoll vom letzten Eintrag ausgehend rückwärts und mache alle Änderungen von Transaktionen in der **Undoliste** rückgängig. Fahre am Checkpoint fort, im Logprotokoll rückwärts zu lesen, bis die **begin transaction**-Einträge für alle Transaktionen in der **Undoliste** gefunden wurden.  
3. Lies das Logprotokoll ab dem Checkpoint vorwärts und schreibe für jede von einer Transaktion in der **Redoliste** durchgeführte Änderung den neuen Wert in die Datenbank.  
4. Starte alle Transaktionen in der **Undoliste** neu.



### Schattenkopie - Schreiben

Zum Transaktionsbeginn wird die Schattenseitenentabelle in die

Arbeitsentabelle kopiert.

Eine Schreiboperation  $write(X, x_j)$  auf ein Objekt  $X$ , das auf der  $i$ -ten Seite

liegt, wird wie folgt behandelt:

1. Ist die  $i$ -te Seite nicht im Puffer, dann übertrage sie mit  $input(X)$ .

2. Wird die  $i$ -te Seite zum ersten Mal verändert, dann suche eine freie Seite

auf der Platte und ordne dieser die  $i$ -te Seite in der Arbeitsentabelle

zu.

3. Weise den Wert  $x_j$  dem Objekt  $X$  im Puffer zu.

```

etappe(ear, r_name, von, bis)
fahrer(f_name, team, gewicht, alter)
rennen(r_name, start, ziel)
wertung(ear, r_name, f_name, gewicht, zeit, platz)

```

Gegeben sind folgende Relationen einer Datenbank über sämtliche in einer Saison stattgefundenen Radrennen (Schlüssel sind unterstrichen).

Zu jedem Fahrer ist sein Name, sein Gewicht, sein Alter und das Team

gespeichert, für das er in der Saison gefahren ist. Zu jedem Rennen sind der

Name, Start und Ziel gespeichert. Zu jedem Rennen gibt es die einzelnen

Etappen, von denen jeweils die Nummer, der Start und das Ziel gespeichert

sind. Zu jeder Etappe wiederum gibt es die Wertung, in der alle Fahrer

gespeichert sind, die am Start dabei waren, ihre Platzierung und die Zeit. Hat

ein Fahrer das Rennen abgebrochen, steht in der Platzierung der Wert 0.

2. Selektieren Sie die Namen und das Alter der jüngsten Teilnehmer des Giro d'Italia.

### Schattenkopie - Transaktionsende

Um eine Transaktion erfolgreich abzuschließen, werden folgende Schritte

gesetzt:

1. Schreibe alle "schmutzigen" Seiten im Puffer auf die Platte.

2. Schreibe die Arbeitsentabelle auf die Platte.

3. Schreibe die Adresse der Arbeitsentabelle auf jene fixe

Speicheradresse, die die Adresse der Schattenentabelle enthält.

Das Wiederanlaufverfahren nach einem Systemfehler oder Transaktionsfehler

ist einfach: setze die Arbeit mit der Schattenkopie fort.

1. Selektieren Sie die Namen aller Fahrer, die bei der Tour de France die meisten Etappensiege feierten.

```

select f_name, alter
from fahrer f1, wertung w1
where f1.f_name=w1.f_name and
 w1.r_name='GdI' and
 f1.alter =all (select min(alter)
 from fahrer f, wertung w
 where f.f_name=w.f_name and
 w.r_name='GdI')

```

### Schatten-speicher-Verfahren

**Vorteile:**

- Keine Verwaltung eines Logprotokolls.
- Wiederanlauf beträchtlich schneller.

**Nachteile:**

- Datenfragmentierung
- Freispeicher-Verwaltung
- Schwerer als Verfahren mit Logprotokoll auf Mehrbenutzerbetrieb erweiterbar
- Indirekter Datenzugang, langsamer Zugriff

```

select f_name
from wertung w1
where w1.r_name='TdF' and
 w1.platz=1
group by w1.f_name
having count (*) >= all (select count (*)
 from wertung w
 where w.r_name='TdF' and
 w.platz=1
 group by w.f_name)

```

3. Schreiben Sie eine Abfrage in Relationaler Algebra, die jene Fahrer selektiert, die in dieser Saison mindestens eine Schlußetappe gewonnen haben.

$$\pi_{\text{NameOfPlatz}} = 1$$

$$\left( \text{wertung} \left[ \text{w.cmr} = \text{e.mr}, \text{w.r.name} = \text{e.r.name} \right] \right)$$

$$\left( \text{etappe} \left[ \text{e.r.name} = \text{r.name}, \text{bis} = \text{zeit}[\text{r.name}] \right] \right)$$

Gerald Pfeifer  
<http://gerald.pfeifer.com>  
 TU Wien

1. Bauteile werden durch eine eindeutige Bauteilnummer (B#) identifiziert und durch einen Typ (TYP; z.B. Widerstand, Kondensator, ASIC) sowie die Kosten (Kosten) und den Hersteller (Hersteller) beschrieben.  
 2. ASICs werden außerdem durch die Angabe der Pin-Anzahl (PAnz) und der Anzahl der Transistoren (TAnz) beschrieben.  
 3. Platinen haben eine eindeutige Nummer (P#) und werden beschrieben durch Angabe ihrer Länge (L) und Breite (B) sowie der daran angebrachten Typen von Steckern (STTyp). Da an einer Platine beliebig viele Stecker angebracht sein können, auch mehrere vom selben Typ, muß außerdem für jeden Typ auf einer Platine die Anzahl gespeichert werden (STAnz). (Amm.: Platinen sind keine Bauteile).  
 4. Eine Baugruppe, identifiziert durch eine Typbezeichnung (BGTyp), besteht aus einer Platine mit darauf angebrachten Bauteilen. Auf einer Platine können beliebig viele Bauteile angebracht werden. (Annahme: pro Platine höchstens ein Bauteil von jedem Typ.) Gleichzeitig muß gespeichert werden, welche Bauteile auf der Platine durch Leiterbahnen miteinander verbunden werden. Solche Verbindungen sind

Gerald Pfeifer  
<http://gerald.pfeifer.com>  
 TU Wien

immer paarweise (d.h., verbinden zwei Bauteile). Auf einer Platine können auch unverbundene Bauteile existieren.

Gerald Pfeifer  
<http://gerald.pfeifer.com>  
 TU Wien