

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

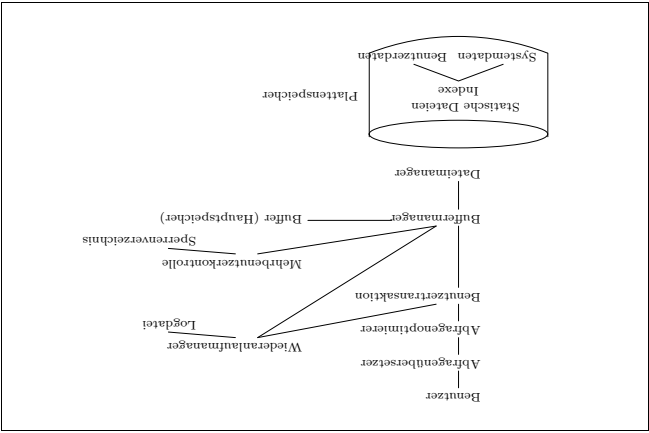
Die Programme arbeiten auf "logischen Daten"

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



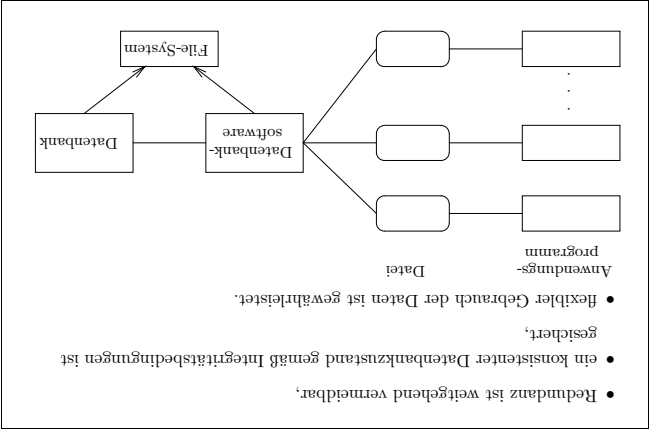
Transaktionen

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



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

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

Wiederanlauf

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

- $Dom(SV_NUMMER) = Menge\ aller\ zehnstelligen\ Zahlen,\ die\ letzten\ sechs:\ Datumformal.$
- $Dom(NAME) = Menge\ aller\ Namen.$
- $Dom(GEB_DATUM) = Menge\ der\ Datumsschtrge.$
- $Dom(TELEFON) = Menge\ aller\ vberstlligen\ Dezimalzahlen.$
- $Dom(GEHALT) = \{k \mid 500 \leq k \leq 10.000\}$
- *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 $t(GEHALT) = 2.600.$*

Datenschutz

- Nicht alle Benutzer sind gleich
- Klassifikation in Benutzergruppen
- Zugriffsrechte (ndern, lesen, ausfhren)
- Granularitt der Objekte, fr die Zugriffsrechte definiert werden knnen:
- Datenbank, Relation, Attribut
- Weitergabe und Rcknahme von Rechten

- *Der NAME-Wert von t ist t(NAME) = Huber.*
- *Die Einnennung von t auf $\{NAME, GEB_DATUM\}$ ist t' mit
 $t(GEB_DATUM) = 01.01.1963,$
 $t(TELEFON) = 4144$ und $t(GEHALT) = 2.600.$*

Beispiel 2 $t(SV_NUMMER) = 2424010163,$ $t(NAME) = Huber,$
 $t(GEB_DATUM) = 01.01.1963,$ $t(TELEFON) = 4144$ und $t(GEHALT) =$

Einnennung $t'(X)$ eines Tupels t auf die Attributmengenge X : Tupel t' , das jedem $A \in X$ dieselben Werte aus $Dom(A)$ zuordnet wie t .

Ad-hoc Abfragen

Nicht alle Abfragen sind a-priori bekannt

Es ist nicht notwendig, fr allgemeine Abfragen spezielle Anwendungsprogramme zu schreiben

Direkte Abfragen an die Datenbank:

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

Schlssel (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.

Oberschlssel (superkey) K : R enthlt einen Schlssel.

Beispiel 3 *Schlsselkandidaten:* $\{FLUGNR\}, \{VON, NACH\}, Zeitinvarianz \Rightarrow$

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

Hnge (FLUGNR VON NACH ABFLUG ANKUNFT)

Das Relationenmodell

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

ausgezeichnete Schlüssel (**designated keys**): Schlüssel, die explizit zu einem Relationenschema angeführt sind

Primärschlüssel: ausgezeichneter Schlüssel, der im Relationenmodell unterstrichen wird

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

DBVO

3.10

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

$$r \cap s = \frac{(A \quad B \quad C) \quad a_1 \quad b_2 \quad c_1}{(A \quad B \quad C) \quad a_1 \quad b_1 \quad c_1}$$
$$r - s = \frac{(A \quad B \quad C) \quad a_1 \quad b_1 \quad c_1}{(A \quad B \quad C) \quad a_2 \quad b_2 \quad c_2}$$

TU Wien

3.13

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

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

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

Operationen auf Relationen

Relationale Algebra (Codd 1972):

- Mengenoperationen
- Selektion
- Projektion
- Division
- Verbundoperation

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

DBVO

3.11

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

Auswahl von Zeilen, die einem bestimmten Kriterium entsprechen.

Die Selektion

Die Selektion ist kommutativ bezüglich der Zusammensetzung:
$$\sigma_{A_1=a}(r) = \{t \in r \mid t(A) = a\}$$
$$\sigma_{A_1=a}(r) = \sigma_{A_2=b}(\sigma_{A_1=a}(r)).$$

TU Wien

3.14

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

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

TU Wien

3.15

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

Die Mengenoperationen

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

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

Beispiel 5

$$r \quad s \quad \frac{(A \quad B \quad C) \quad a_1 \quad b_1 \quad c_1 \quad a_2 \quad b_2 \quad c_2}{(A \quad B \quad C) \quad a_1 \quad b_2 \quad c_1 \quad a_2 \quad b_2 \quad c_2}$$

DBVO

3.12

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

Beispiel 6

$$\sigma_{VON='Frankfurt'}(flüge) =$$

flüge	(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	

TU Wien

3.15

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

Die Projektion

Auswahl von Spalten einer Tabelle: Projektion nach einer Teilmenge der Attribute.
$$\pi_X(r) = \{t(X) \mid t \in r\} \text{ für } X \subseteq R$$
Der Projektionsoperator ist kommutativ bezüglich der Selektion, wenn die Attribute, auf denen selektiert wird, in den projizierten Attributen enthalten sind:
$$\pi_{A_1, A_2}(\sigma_{A_1=s}(r)) = \sigma_{A_1=s}(\pi_{A_1, A_2}(r))$$

TU Wien

3.15

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

$$\pi_{ABFLUG,ANKUNFT}(flüge) =$$

(ABFLUG ANKUNFT)	
3:00p	5:50p
9:50p	2:52a
10:05p	12:43a
11:43a	12:45p
2:20p	3:12p

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

DBVO

Weitere Eigenschaften des Verbundoperators

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

Beispiel 12

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

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

DBVO

Der Natürliche Verbund (natural join)

Der Verbundoperator verknüpft zwei Relationen über ihre gemeinsamen Attribute:

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

Der Verbundoperator ist kommutativ.

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

DBVO

Das Kartesische Produkt

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

Beispiel 11: $r \quad \begin{array}{c} \frac{a_1}{(A \ B)} \\ \frac{a_1 \ b_1}{(C \ D)} \end{array} \quad s \quad \begin{array}{c} \frac{a_2}{(C \ D)} \\ \frac{c_1 \ d_1}{(C \ D)} \end{array}$

$$r \times s = r \bowtie s = \begin{array}{c} \frac{a_1 \ b_1 \ c_1 \ d_1}{(A \ B \ C \ D)} \\ \frac{a_1 \ b_1 \ c_2 \ d_1}{a_1 \ b_1 \ c_2 \ d_2} \\ \frac{a_2 \ b_1 \ c_1 \ d_1}{a_2 \ b_1 \ c_2 \ d_2} \\ \frac{a_2 \ b_1 \ c_2 \ d_1}{a_2 \ b_1 \ c_2 \ d_2} \end{array}$$

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

DBVO

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 $r(R)$ $= \pi_R(q)$ und $s(S) = \pi_S(q)$.

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

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

DBVO

Beispiel 10

benutzbar (FLUGNR MASCHINE)	83	727	707	747
	83	747	727	747
zugehören (PILOT MASCHINE)	707	727	747	747
	727	747	747	747

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

DBVO

Beispiel 13 $R = AB, S = BC$

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

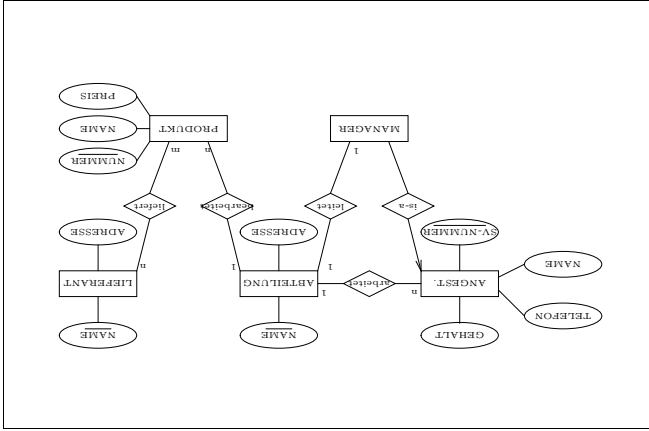
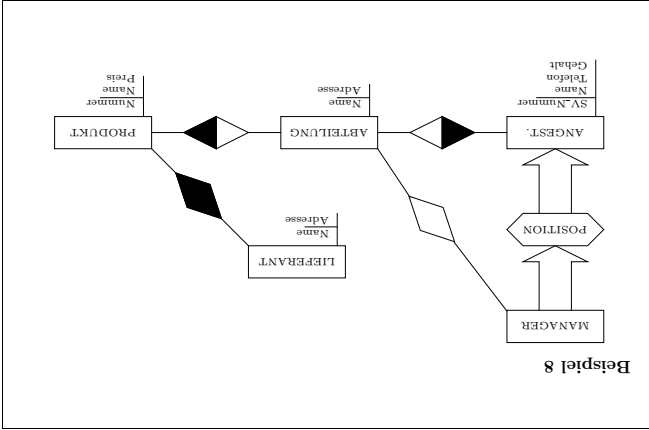
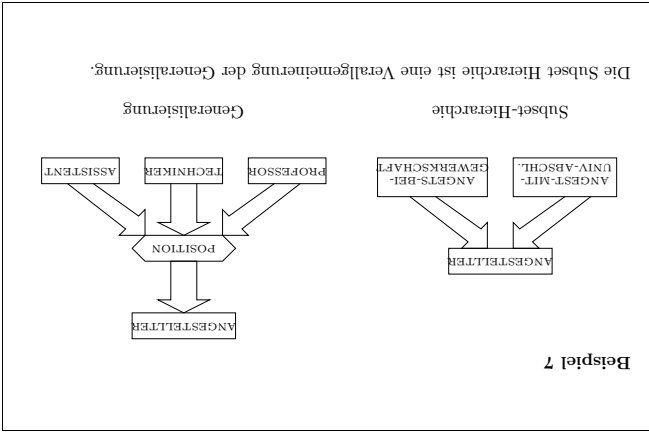
Gerald Pfeifer
http://gerald.pfeifer.com
TU Wien

DBVO

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 beschreib 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.



DBVO 2.18

Erweiterung der Modellierungsmöglichkeiten der Relationships

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

binäre und ternäre Relationships sind Spezialfälle.

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

DBVO 2.21

Übersetzung des EER in das Relationenmodell

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

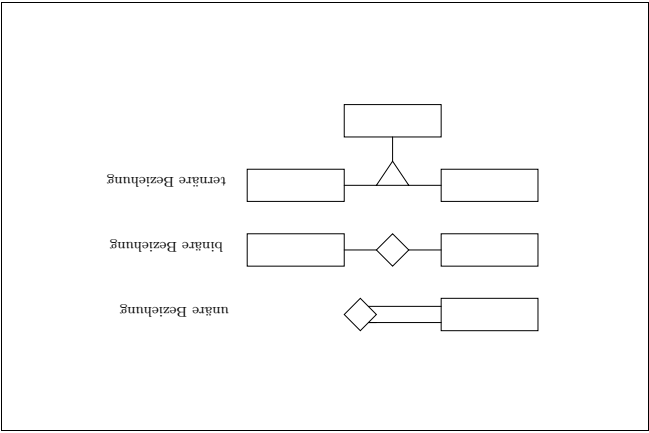
Alle Sachverhalte werden einheitlich durch Relationen (Beziehungen) zwischen Attributen dargestellt.

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 Vorkommen *genau eines* aus den der Entites E_1, \dots, E_n ist.

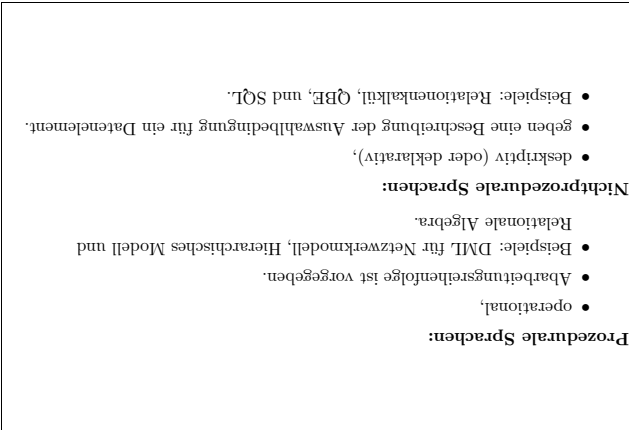
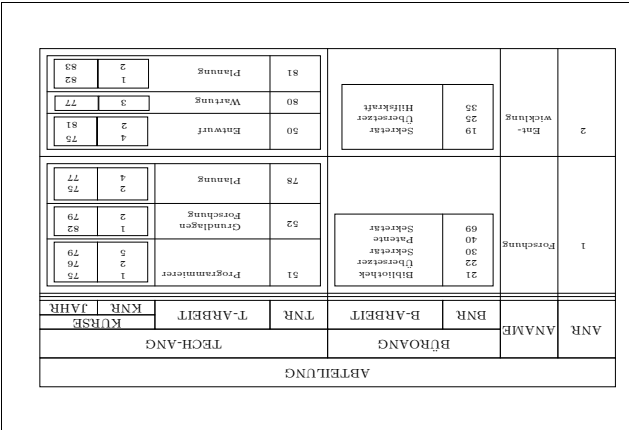
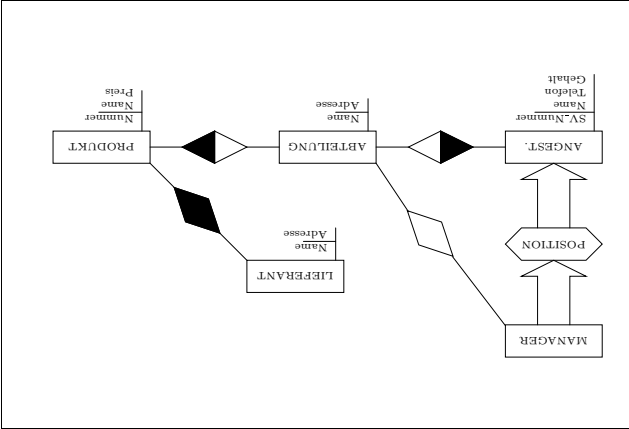
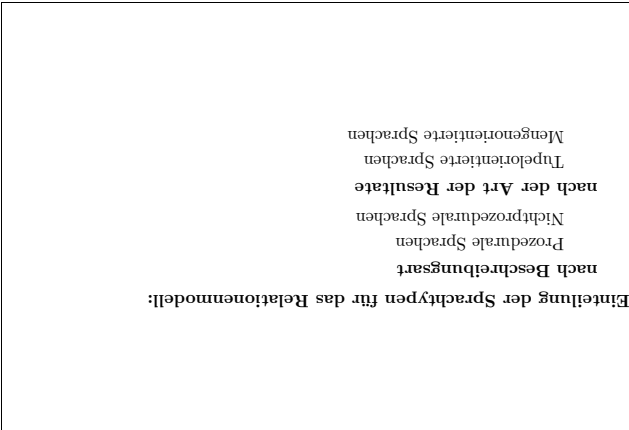
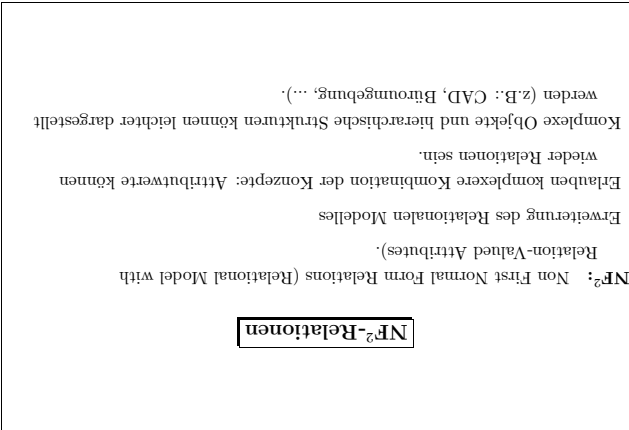
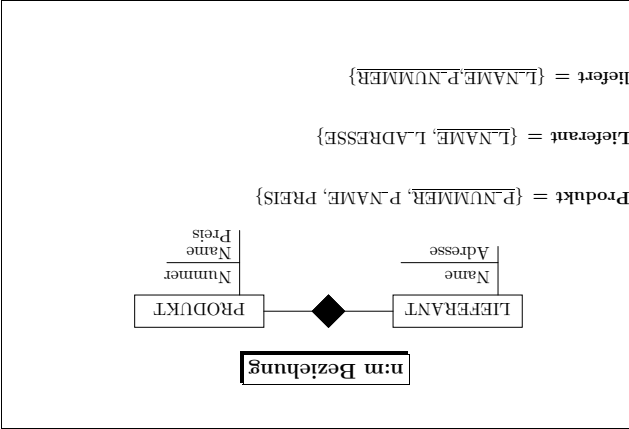
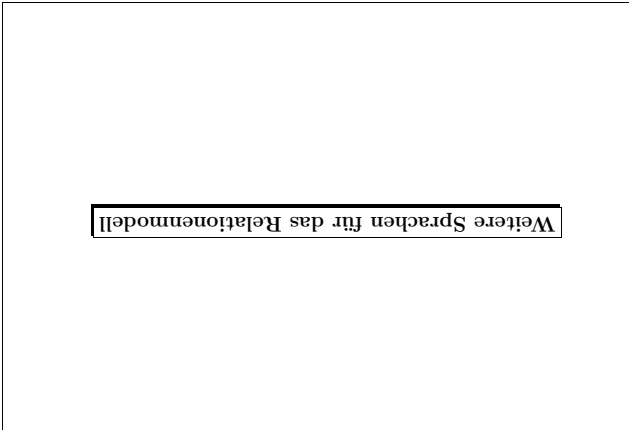
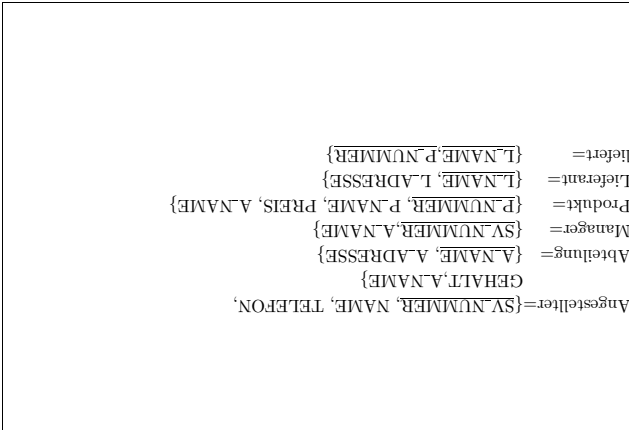
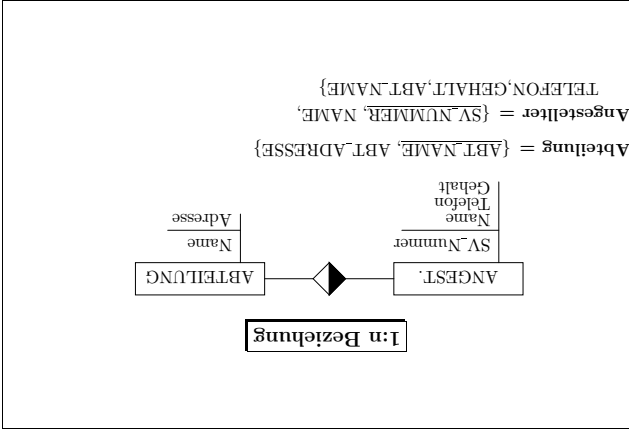


DBVO 2.22

1:1 Beziehung

Abteilung = {A_NAME, A_ADRESSE}

Manager = {SV_NUMMER, A_NAME}



ODER Verknüpfungen: mehrere Zeilen im Eingabeschema

Beispiel 7 Welche Teile gibt es in Wien oder London?

lager	TEIL	LAGERORT	MENGE
		Wien	
		London	
	P. 100		
	P. 101		
211			
2114			
2116			

Frankfurt-teile	TEILNAME	MENGE
	Boizen	
	P. 50	

Frankfurt-teile	TEILNAME	MENGE
	Boizen	
	P. 50	

Frankfurt-teile	TEILNAME	MENGE
	Sitz	106
	Sitzbezug	6

- Lehrbücher:

Date, C.J. and Darwen, H.: A Guide to the SQL Standard: A User Guide to the Standard Relational Language SQL. 3rd edition, Addison-Wesley Publishing Company, 1994.

Melton, J. and Simon, A.R.: Understanding the New SQL: A Complete Guide. Morgan Kaufmann, San Matteo, California, 1994.

UND Verknüpfungen: Verwendung derselben Variablen in verschiedenen Zeilen

Beispiel 8 Welche Teile gibt es in Wien und London?

lager	TEIL	LAGERORT	MENGE
		Wien	
		London	
	P. 100		
	100		
211			
2116			

Beispiel 10 Welche Teile sind Bestandteile desselben Teils?

teile_info	TEIL	BESTANDTEIL_VON	TEILNAME
	100	200	
	101	200	
			BEINGUNGEN
			200 ≠ 0
			100 < 101

teile	TEIL1	TEIL2
	2114	2116
	21163	21164
	2061	2066
	2061	2068
	2066	2068

Allgemeines

Bezeichnung: Relationen ... Tabellen (tables);
Attribut ... Spalten (columns)
Tupel ... Zeilen (rows)

SQL-DDL (Data Definition Language):

Befehle zum Erzeugen, Entfernen einer Relation
Befehle zum Ändern des Relationschemas einer Relation

SQL-DML (Data Manipulation Language):

Befehle zum Abfragen, Einfügen, Ändern und Löschen von Tupeln in Relationen

Beziehungen zwischen Tupeln in verschiedenen Relationen:
Verwendung derselben Variablen in verschiedenen Relationenschemata.
Ergebnisschema mit einem neuen Namen muß vorgeben werden

Beispiel 9 Gesucht sind die Namen und Anzahl der Teile, die am Frankfurter Flughafen gelagert sind.

teile_info	TEIL	BESTANDTEIL_VON	TEILNAME
			Boizen
lager	TEIL	LAGERORT	MENGE
		Frankfurt	
	100		
			50

Interaktives SQL

- 1974 von IBM entwickelt für das experimentelle DBMS System R als SEQUEL (Structured English Query Language).
- von ANSI (American National Standard Institute) und ISO (International Standard Organization) als Sprachschnittstelle für relationale Datenbanksysteme genannt.
- Hier: SQL-2-Standard von 1992
- interaktiv bzw. in Wirtsprache (host language) z. B. C, eingebettet.

Datentypen

- CHAR(n) (Zeichenkette der Länge n),
- VARCHAR(n) (Zeichenkette variabler Länge, die kleiner oder gleich n ist; n < 255),
- INTEGER,
- REAL,
- DATE,
- Auf Zeichenketten, Zahlen und Datumsangaben sind =, <, >, <=, >=, <> definiert.
- Auf Zahlen können +, −, *, / angewandt werden.

DBVO	Gerald Pfeifer	http://gerald.pfeifer.com	TU Wien
4.31	<ul style="list-style-type: none">• 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 positionsweise (wahr, wenn der Vergleich für jede Position der Liste wahr liefert).• Ein einzelner Ausdruck statt Ausdruckliste \Rightarrow die Abfrage darf nur ein Attribut selektieren.		
4.34	Gerald Pfeifer	http://gerald.pfeifer.com	TU Wien
<div>Division</div> <p>Welcher Pilot fliegt(pilot, typ) kann alle Maschinen in Maschine(typ) fliegen?</p> $\text{fliegt} \div \text{Maschine}$ <pre>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));</pre>			
4.37	Gerald Pfeifer	http://gerald.pfeifer.com	TU Wien
<div>Erzeugen von Relationenschemata</div> <ul style="list-style-type: none">• Wertebereich zu jedem Attribut,• Integritätsbedingungen zu jedem Attribut,– PRIMARY KEY Primärschlüssel,– NOT NULL kein Nullwert, default für Schlüssel,– UNIQUE eindeutig, default für Schlüssel,– CHECK (<i>condition</i>) Wertebereichseinschränkung,– "DEFAULT <i>value</i>" Defaultwert.			

DBVO	Gerald Pfeifer	http://gerald.pfeifer.com	TU Wien
4.32	<div>Spezielle Formen von Teilbedingungen</div> <ul style="list-style-type: none">• "EXISTS <i>query</i>" liefert wahr, wenn die Abfrage <i>query</i> mindestens ein Tupel selektiert.• "<i>expressionList</i> <i>comparisonOp</i> ANY <i>query</i>" liefert wahr, wenn der Vergleich der Ausdruckliste <i>expressionList</i> mit irgendeinem von der Abfrage <i>query</i> erzeugten Tupel wahr liefert.• "<i>expressionList</i> <i>comparisonOp</i> ALL <i>query</i>" liefert wahr, wenn der Vergleich der Ausdruckliste <i>expressionList</i> mit allen von der Abfrage <i>query</i> erzeugten Tupel wahr liefert.• "<i>expressionList</i> IN <i>query</i>", wenn der Vergleich der Ausdruckliste <i>expressionList</i> mit einem Tupel der Abfrage <i>query</i> identisch ist.• "<i>expressionList</i> NOT IN <i>query</i>", wenn der Vergleich der Ausdruckliste <i>expressionList</i> mit keinem Tupel der Abfrage <i>query</i> identisch ist.		
4.35	Gerald Pfeifer	http://gerald.pfeifer.com	TU Wien
<div>Mengenoperationen</div> <ul style="list-style-type: none">• UNION (Mengenvereinigung),• UNION ALL (Multimengenvereinigung, d. h. gleiche Tupel werden nicht eliminiert),• INTERSECT (Durchschnitt),• EXCEPT (Mengendifferenz) <p>für typkompatible Relationen</p> <p>Beispiel 17 <i>Selektiere die Vornamen aller Personen und Kinder.</i></p> <pre>(SELECT vorname FROM personen) UNION (SELECT vorname FROM kinder);</pre>			
4.38	Gerald Pfeifer	http://gerald.pfeifer.com	TU Wien
<div>Beispiel 18 <i>Erzeuge ein Relationenschema mit den Attributen für Personen aus Beispiel 11 und dazu eine leere Relation mit dem Namen personen.</i></div> <pre>CREATE TABLE personen (pnr INTEGER PRIMARY KEY, vorname VARCHAR(15) NOT NULL, nachname VARCHAR(15) NOT NULL, geschlecht CHAR(1) CHECK (geschlecht IN ('w','m')), gehalt REAL DEFAULT 2000);</pre>			

DBVO	Gerald Pfeifer	http://gerald.pfeifer.com	TU Wien
4.33	<div>Beispiel 16 <i>Selektiere die Personennummern jener Personen, die ein Kind haben, dessen Vorname identisch mit dem Vornamen eines Einzelkindes einer anderen Person ist.</i></div> <pre>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);</pre>		
4.36	Gerald Pfeifer	http://gerald.pfeifer.com	TU Wien
<div>DDL (Data Definition Language)</div> <ul style="list-style-type: none">• Erzeugen von Relationen,• Festlegung von globalen Integritätsbedingungen,• Ändern von Relationen,• Entfernen von Relationen.			
4.39	Gerald Pfeifer	http://gerald.pfeifer.com	TU Wien
<div>Integritätsbedingungen zur Relation</div> <ul style="list-style-type: none">• CONSTRAINT-Klauseln mit eigenem Namen,• REFERENCES-Klausel: Fremdschlüsselbedingung,• "ON UPDATE <i>action</i>" und "ON DELETE <i>action</i>"; Aktion, was beim Ändern oder Löschen des referenzierten Schlüssels geschehen soll.• <i>action</i>: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).			

Views (Benutzersichten)

- Logische Datenunabhängigkeit in SQL durch die Definition von Views (Sichtrelationen, Benutzersichten)
- 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.

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.

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: Rechte zurücknehmen

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

```
REVOKE privilege ON relationName FROM authID
```

rekursiv, auch für alle weitergegebenen Rechte.

Beispiel 31 Benutzer Huber darf kleinverdiener REVOKE UPDATE(pnr, vorname) ON kleinverdiener FROM Huber;

Updates 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.

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

```
WITH CHECK OPTION
```

verhindert solche Änderungen.

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.

Views (Benutzersichten)

- Logische Datenunabhängigkeit in SQL durch die Definition von Views (Sichtrelationen, Benutzersichten)
- 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.

Updates über Views / Löschen von Views

- USER verweist auf den augenblicklichen Benutzer.

Beispiel 27 Die View persoentl enthält für jeden Benutzer seine Personaldaten.

```
CREATE VIEW persoentl AS
SELECT *
FROM personen p
WHERE p.zuname = USER;
```

Löschen: "DROP VIEW relationName"

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: Rechte zurücknehmen

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

```
REVOKE privilege ON relationName FROM authID
```

rekursiv, auch für alle weitergegebenen Rechte.

Beispiel 31 Benutzer Huber darf kleinverdiener

```
REVOKE UPDATE(pnr, vorname)
```

```
ON kleinverdiener
```

```
FROM Huber;
```

Updates 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.

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

```
WITH CHECK OPTION
```

verhindert solche Änderungen.

- 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.

Wieviel verdienen die Mitarbeiter an Gehalt, Provision und insgesamt?

SELECT vname, name, gehalt, prov, AS sum

FROM mitarbeiter;

VNAME	NNAME	GEHALT	PROV	SUM
Gerti Müller	20000	1000	21000	
Anton Eder	19000	500	19500	
Max Muster	13000			
Edith Huber	15000			
Rita Muster	18000			
Lise Hofner	17000	900	17900	
Rudi Mayer	21000			
Theo Novak	20000			
Karin Meini	25000	1200	26200	

Wir wollen nun wissen, wieviel Gehalt und Provision die Mitarbeiter in jedem Restaurant bekommen und wieviele Mitarbeiter es gibt.

SELECT rnr, COUNT (gehalt+prov), SUM(gehalt+prov)

FROM mitarbeiter

GROUP BY rnr;

RNR	COUNT(GEHALT+PROV)	SUM(GEHALT+PROV)
1	2	40500
2	0	
3	2	44100

Beispiel 31 Betrachte abt und ang

abt	(ABT#	NAME)
2	Buchhaltung	
3	Verkauf	
4	Einkauf	

ang	* abt	(ABT#	NAME
1	Buchhaltung		
2	Buchhaltung		
3	Verkauf		
4	Einkauf		

ang	* abt	(ABT#	NAME	PERS#	GEHALT)
1	Buchhaltung			20,000	
2	Buchhaltung			20,000	
3	Verkauf			10,000	
4	Einkauf			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 Meini	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 rehnamel [FULL | LEFT | RIGHT] OUTER JOIN rehnamel2 ON joinCondition WHERE condition"

Beispiel 32 Selektiere die Personennummern 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;

Outer Join

Seien $r(R), s(S)$ Relationen mit $R \cap S = X \neq \{\}$, und $?(X)$ das Tupel über X , bestehend nur aus Nullwerten.

$q = r \bowtie s, \quad r' = r \setminus \pi_R(q), \quad s' = s \setminus \pi_S(q).$

Der Äußere Verbund von r und s (outer join of r and s) ist definiert als

$r * s = q \cup (r' \times ?(S \setminus X)) \cup (s' \times ?(R \setminus X)).$

Der Äußere Verbund ist im Gegensatz zum Verbund nicht assoziativ!

turing-vollständige Sprache für die Definition und Wartung von persistenzen, komplexen Objekten.

ANSI und ISO

- Rekursive Abfragen
- Objektorientiertes Datenmanagement (object identifiers, Objekthierarchien, Vererbung (inheritance), Datenkapselung,...)
- benutzerspezifische Datentypen, abstrakte Datentypen (ADT's)
- Triggers und Assertions
- Unterstützung für wissensbasierte Systeme

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

• SQL-2, Relationale Algebra, Relationenalkül 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“

Rekursion in SQL-3

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Lieferung

von	bis	
Wien	Wt. Neustadt	1
Wien	Klosterneuburg	1
Wien	St. Pölten	1
Wien	Linz	1
Wien	Krems	2
St. Pölten	Krems	2
St. Pölten	Melk	2
St. Pölten	Linz	2
Tulln	Krems	3
Tulln	Klosterneuburg	3
Tulln	Wt. Neustadt	4
30	kosten	50

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

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

Abarbeitung: 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

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

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_old
NEW as gehalt_new
FOR EACH ROW
WHEN gehalt_old * 1.05 < gehalt_new
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.

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

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;

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Nullwerte und unvollständige Information

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

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.
 $\text{Ein_stop}(\text{start}, \text{ziel}) := \pi_{1,5}(\text{Lieferung}[\text{bis} = \text{von}][\text{Lieferung}])$
 $\text{Zwei_stop}(\text{start}, \text{ziel}) := \pi_{1,5}(\text{Lieferung}[\text{bis} = \text{von}][\text{Ein_stop}])$
 $\text{Alle_wege}(\text{start}, \text{ziel}) := \pi_{1,2}(\text{Lieferung} \cup \text{Ein_stop} \cup \text{Zwei_stop} \cup \dots)$

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

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 Triggeraufrufe.

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

Granularität: FOR EACH ROW bzw. FOR EACH STATEMENT

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

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Darstellung:

- 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.

- ?
- ω (Omega)
- NULL

Beispiel 201

exjob

(ANGESTELLTER

GEHALT

Dienstgeber

ARBEIT

GEHALT

EHEM.

EHEM.

Novak

39,500

AUA

36,000

Pilot

36,000

GEHALT

Maier

24,100

?

?

?

Beamter

?

?

17,888

Huber

18,260

TU Wien

?

?

17,888

Ein Tupel t , 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 $A \uparrow$, wenn der Wert von A nicht $?$ ist.

Die Wahrheitstabellen sehen folgendermaßen aus:

\vee

\wedge

\neg

w

w

w

f

f

f

w

f

f

f

w

w

f

f

f

$?$

$?$

$?$

w

w

w

f

f

f

w

f

f

f

w

w

$?$

$?$

$?$

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

4.209

Markierte Nullwerte einer Relation r bzg. 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.).

Funktionale Abhängigkeiten und Nullwerte

http://gerald.pfeifer.com

TU Wien

DBVO

4.207

DBVO

4.204

Beispiel 202 Eine Vervollständigung der Relation aus Beispiel 201.

exjob

(ANGESTELLTER

GEHALT

Dienstgeber

ARBEIT

GEHALT

EHEM.

EHEM.

Novak

39,500

AUA

36,000

Pilot

36,000

GEHALT

Maier

24,100

TU Wien

Dozent

22,050

Beamter

16,400

Assistent

17,888

Huber

18,260

TU Wien

Assistent

17,888

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.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

4.207

Beispiel 203

exjob

(ANGESTELLTER

GEHALT

Dienstgeber

ARBEIT

GEHALT

EHEM.

EHEM.

Novak

39,500

AUA

36,000

Pilot

36,000

GEHALT

Maier

24,100

?

?

?

Beamter

?

?

17,888

Huber

18,260

TU Wien

?

?

17,888

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

Ergebnis: wahr

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

4.210

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, aber nicht in ihren A -Werten übereinstimmen, und zumindest ein A -Wert ist ein markierter Nullwert.

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

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

4.211

DBVO

4.205

Nullsubstitutionsprinzip

• \dots (wohlgeformte) Formel des Relationen- bzw. Typalkalküls über r .

Gesucht: Interpretationsfunktion I , die ϕ in wahr, falsch, $?$ abbildet.

• $J(\phi) \dots$ übliche Interpretationsfunktion, bildet ϕ über totalen Relationen in $\{w, f\}$ ab.

• I muß J erweitern: $I/\text{totale Relationen} = J$.

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

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

4.208

Gibt es einen Angestellten namens Huber, dessen ehemaliges Gehalt höher ist, als das aller anderen?

$f2: \exists x(R) \in \text{exjob} (x(\text{ANGESTELLTER}) = \text{"Huber"}) \wedge \forall y(R) \in \text{exjob} (x(\text{EHEM.GEHALT}) > y(\text{EHEM.GEHALT}))$

Ergebnis: falsch

Gibt es einen Angestellten namens Novak, dessen ehemaliges Gehalt höher ist, als das aller anderen?

$f3: \exists x(R) \in \text{exjob} (x(\text{ANGESTELLTER}) = \text{"Novak"}) \wedge \forall y(R) \in \text{exjob} (x(\text{EHEM.GEHALT}) > y(\text{EHEM.GEHALT}))$

Ergebnis: ?

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

4.211

Füllregel für markierte Nullwerte

Versuchen 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

Gerald Pfeifer

DBVO

4.212

http://gerald.pfeifer.com

TU Wien

Eine partielle Relation $r(R)$ ist **zulässig** bzgl. einer Menge funktionaler Abhängigkeiten F , wenn es eine Vervollständigung von r gibt, die F erfüllt.

Theorem: r ist zulässig \Leftrightarrow wenn in r' keine schwere Verletzung auftritt \Leftrightarrow wenn in keinem r_i eine schwere Verletzung auftritt. (r' ist im allgemeinen nicht zulässig.)

Gerald Pfeifer

DBVO

4.213

http://gerald.pfeifer.com

TU Wien

Beispiel 204 Betrachte r mit $FDs\ F = \{A \rightarrow C, C \rightarrow D\}$.

r

$(A\ B\ C\ D)$

$a\ b\ c\ d$

$a\ b\ c\ d$

r'

$(A\ B\ C\ D)$

$a\ b\ c\ d$

$a\ b\ c\ d$

$Erg\text{bnis } r'$

$a\ b\ c\ d$

$a\ b\ c\ d$

$a\ b\ c\ d$

$a\ b\ c\ d$

$a\ b\ c\ d$

$a\ b\ c\ d$

$a\ b\ c\ d$

DBVO 5.4

Gerald Pfeifer

DBVO

5.4

http://gerald.pfeifer.com

TU Wien

Datalog

Gerald Pfeifer

DBVO

5.4

http://gerald.pfeifer.com

TU Wien

Lösung: Zuhilfenahme einer Programmiersprache, in die eine relational vollständige Abfragesprache eingebettet ist.

begin
 $result := \{\}$
 $newtuples := Eltern;$
while $newtuples \neq \{\}$ **do**
 $result := result \cup newtuples;$
 $newtuples := \pi_{1,4}(newtuples[2 = 1]Eltern);$
end;
 $vorfahre := result$
end.

Methode: prozedural, erfordert Programmierekenntnisse.

DBVO 5.5

Gerald Pfeifer

DBVO

5.5

http://gerald.pfeifer.com

TU Wien

- SQL, Relationale Algebra, Relationenkalcul, QBF sind relational vollständig.
- Das bedeutet allerdings nicht, daß alle überhaupt möglichen Abfragen in diesen Sprachen formuliert werden können.
- **rekursive** Abfragen können in diesen Sprachen **nicht** formuliert werden.

Gerald Pfeifer

DBVO

5.5

http://gerald.pfeifer.com

TU Wien

Datalog: Prolog-ähnliche logische Abfragesprache, rekursive Anfragen an eine Datenbank direkt (nichtprozedural)

$vorfahre(X, Y) :- Eltern(X, Y).$
 $vorfahre(X, Z) :- Eltern(X, Y), vorfahre(Y, Z).$

Nachkommen einer bestimmten Person (z.B. Hans):
 $?vorfahre(hans, X)$

besser:
 $hans_successor(X) :- Eltern(hans, X).$
 $hans_successor(Y), Eltern(Y, X).$

DBVO 4.214

Gerald Pfeifer

DBVO

4.214

http://gerald.pfeifer.com

TU Wien

Beispiel 205 Betrachte s mit $FDs\ F = \{A \rightarrow B, C \rightarrow B\}$.

s

$(A\ B\ C)$

$1\ 2\ 1$

$1\ 2\ 1$

s'

$(A\ B\ C)$

$1\ 2\ 1$

$1\ 2\ 1$

$1\ 2\ 1$

$1\ 2\ 1$

$1\ 2\ 1$

$1\ 2\ 1$

$1\ 2\ 1$

$1\ 2\ 1$

$1\ 2\ 1$

$1\ 2\ 1$

Schwere Verletzung $\Rightarrow s$ nicht zulässig bzgl. F .

DBVO 5.3

Gerald Pfeifer

DBVO

5.3

http://gerald.pfeifer.com

TU Wien

Beispiel 1 Relation $Eltern(ELTERNTEIL, KIND)$, die die Elternteil-Kind-Beziehung über einer bestimmten Personengruppe widerspiegelt.

- Relation **VORFAHRE** mit $\langle X, Y \rangle, X, Y \in ELTERN, X$ ein Vorfahre von Y .
- Erzeugung jeder Generation: ein Join und eine Projektion

$enkel(GROSSELTERNTEIL, ENKELKIND) :=$
 $\pi_{1,4}(Eltern[KIND = ELTERNTEIL]Eltern)$
 $urenkel(URGROSSELTERNTEIL, URENKELKIND) :=$
 $\pi_{1,4}(Eltern[KIND = GROSSELTERNTEIL]enkel)$

- Das Ergebnis der Abfrage: Vereinigung aller Teilergebnisse für jede Generationsstufe:

$vorfahre(AHNE, NAME) := Eltern \cup enkel \cup urenkel \cup \dots$

Die Syntax von Datalog

$\langle Datalog_Programm \rangle ::= \langle Datalog_Regel \rangle \mid \langle Datalog_Programm \rangle \mid \langle Datalog_Regel \rangle$
 $\langle Datalog_Regel \rangle ::= \langle Literal \rangle$
 $\langle Literal \rangle ::= \langle Literal \rangle \mid \langle Regelkumpf \rangle$
 $\langle Regelkopf \rangle ::= \langle Literal \rangle$
 $\langle Datalog_Regel \rangle ::= \langle Regelkopf \rangle :- \langle Regelkumpf \rangle.$

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* $\text{eltern}(\text{linda}, \text{michael})$, $\text{eltern}(\text{linda}, \text{geert})$ und der *Regel*

$$\text{geschwister}(\text{michael}, \text{geert}) :- \text{eltern}(\text{linda}, \text{michael}), \text{eltern}(\text{linda}, \text{geert}).$$

kann der Fakt

$$\text{geschwister}(\text{michael}, \text{geert})$$
gewonnen werden.

Idee: Anwendung eines Datalog-Programms P auf eine Datenbank DB : iteriertes Ableiten von Fakten bis zur Sättigung.

Formalisierung:

- definiere Operator $T_P(DB)$: erweitere 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.

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

Algorithmus INFER

INPUT: Datalog-Programm P , Datenbank DB

OUTPUT: $T_P^*(DB)$ ($= \text{cons}(P^* \wedge DB^*)$)

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 .

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

Subroutine T_P

INPUT: Faktenmenge OLD

OUTPUT: $T_P(OLD)$

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 6³ Grundinstanzen dieser Regel bezüglich P und DB aus Beispiel 2 sind

(Konstantensymbole = grete , linda , geert , hans , michael , karl):

$$\begin{aligned} &\text{geschwister}(\text{grete}, \text{grete}), \text{eltern}(\text{grete}, \text{grete}), \\ &\text{geschwister}(\text{grete}, \text{linda}), \text{eltern}(\text{grete}, \text{linda}), \\ &\text{grete} \neq \text{linda} \quad (X = Y = Z = \text{grete}), \\ &\text{grete} \neq \text{geert} \quad (X = Y = Z = \text{grete}), \\ &\text{eltern}(\text{grete}, \text{grete}), \text{eltern}(\text{grete}, \text{grete}), \\ &\text{eltern}(\text{grete}, \text{linda}), \text{eltern}(\text{grete}, \text{linda}), \\ &\text{eltern}(\text{grete}, \text{geert}), \text{eltern}(\text{grete}, \text{geert}), \\ &\text{eltern}(\text{grete}, \text{hans}), \text{eltern}(\text{grete}, \text{hans}), \\ &\text{eltern}(\text{grete}, \text{michael}), \text{eltern}(\text{grete}, \text{michael}), \\ &\text{eltern}(\text{grete}, \text{karl}), \text{eltern}(\text{grete}, \text{karl}), \\ &\text{eltern}(\text{linda}, \text{karl}), \text{eltern}(\text{linda}, \text{karl}), \\ &\text{eltern}(\text{geert}, \text{karl}), \text{eltern}(\text{geert}, \text{karl}), \\ &\text{eltern}(\text{hans}, \text{karl}), \text{eltern}(\text{hans}, \text{karl}), \\ &\text{eltern}(\text{michael}, \text{karl}), \text{eltern}(\text{michael}, \text{karl}), \\ &\text{eltern}(\text{karl}, \text{karl}), \text{eltern}(\text{karl}, \text{karl}), \\ &\text{karl} \neq \text{karl} \quad (X = Y = Z = \text{karl}). \end{aligned}$$

- 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 $T_P^m(DB) = T_P^n(DB)$ für alle $m \geq n$.
- $T_P^n(DB) \dots$ Faktenmenge, zu der $(T_P^i(DB))$ konvergiert
- Die operationale Semantik eines Datalog-Programms P ordnet jeder Datenbank DB die Faktenmenge $T_P^n(DB)$ zu:

$$O[P] : DB \longrightarrow T_P^n(DB).$$

- Es gilt:

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

Beispiel 5 Wende Programm P

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

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

Schritt 1. von *INFER*: Bildung von GP

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

(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:
 $OLD := TP(DB), NEW := TP(OLD) = TP(TP(DB))$

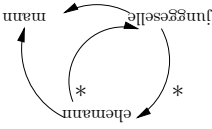
$OLD \cup \{ \text{vorfahre}(\text{hans}, \text{michael}), \text{vorfahre}(\text{hans}, \text{gertr}), \text{vorfahre}(\text{grete}, \text{michael}), \text{vorfahre}(\text{grete}, \text{gertr}) \}$.

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*.

- $P \dots$ ein erweitertes Datalog-Programm, mit negierten Literalen im Regelrumpf.
- $DEP(P) \dots$ gerichteter Graph, mit:
 - Knoten $\dots R_i$, eine in P vorkommende Relation,
 - Kanten $\dots R \rightarrow S$, wenn es eine Regel in P gibt, deren Kopfprädikat 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.

Graphendarstellung



ehemann(X) :- mann(X), non ehemann(X).
junggeselle(X) :- mann(X), non ehemann(X).

Beispiel 7 Datalogprogramm mit den Regeln:

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	Michael	Hans	Michael
Gerte	Michael	Gerte	Michael

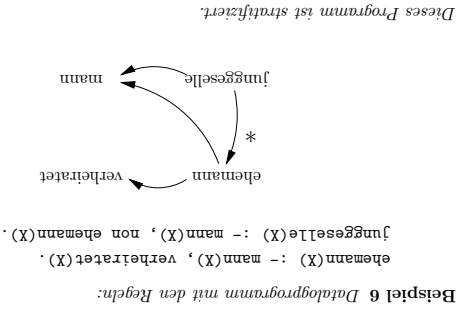
- Ein um Negationen erweitertes Datalog-Programm P ist zulässig \Leftrightarrow im Graph $DEP(P)$ kein gerichteter Kreis vorkommt, der eine durch “*” markierte Kante enthält.
- Ein solches Programm wird auch **stratifiziert** genannt, da es bezüglich der Negation in eine Hierarchie von Schichten (Strata)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

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 negiertes Prädikat q enthält, so ist p in einer mindestens so hohen Schicht wie q .

Die Schichtendarstellung

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.



ehemann(X) :- mann(X), verheiratet(X).
junggeselle(X) :- mann(X), non ehemann(X).

Beispiel 6 Datalogprogramm mit den Regeln:

Dieses Programm ist stratifiziert.

INPUT: 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.

Algorithmus *CLOSURE*

INPUT: X Attributmengen, F Menge von FDs

OUTPUT: X^+

$X^+ := X$

while $\exists (Y \rightarrow Z) \in F$ mit $Y \subseteq X^+$ und $Z \not\subseteq X^+$ **do**

$X^+ := X^+ \cup Z$

return (X^+)

Aquivalenz 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 heißt dann **Überdeckung** (cover) von G .

$F \equiv G \Leftrightarrow F^+ = G^+$

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.

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

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])

Algorithmus *MEMBER*

INPUT: $X \rightarrow Y$ FD, F Menge von FDs

OUTPUT: true, if $F \models X \rightarrow Y$
false, if $F \not\models X \rightarrow Y$

return $(Y \subseteq \text{CLOSURE}(X, F))$

Beispiel 4 $F \equiv G$?

$F = \{ \text{PERSONALNR, ABT} \rightarrow \text{ORT}$
 $\text{PERSONALNR} \rightarrow \text{ABT}$
 $\text{ABT} \rightarrow \text{ORT}$
 $\text{PERSONALNR} \rightarrow \text{ORT} \}$
 $G = \{ \text{PERSONALNR, ABT} \rightarrow \text{ORT}$
 $\text{PERSONALNR} \rightarrow \text{ABT}$
 $\text{ABT} \rightarrow \text{ORT} \}$
 g_2
 g_3

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 der Transitivität 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^+ neu aufgenommenen wurde.

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.

$G' = \{ \text{PERSONALNR} \rightarrow \text{ABT}$
 $\text{PERSONALNR} \rightarrow \text{ORT}$
 $\text{PERSONALNR, ABT} \rightarrow \text{NAME} \}$
 g_1
 g_2
 g_3
 g_4

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.

$G' = \{ \text{PERSONALNR} \rightarrow \text{ABT}$
 $\text{PERSONALNR} \rightarrow \text{ORT}$
 $\text{PERSONALNR, ABT} \rightarrow \text{NAME} \}$
 g_1
 g_2
 g_3
 g_4

2. Berechnung von G'' linksreduziert.

$G'' = \{ PERSONALNR \rightarrow ABT$
 $PERSONALNR \rightarrow ORT$
 $ABT \rightarrow ORT$
 $PERSONALNR \rightarrow NAME \}$
 $g1$
 $g2$
 $g3$

3. Berechnung von G''' nichtredundant.

$G''' = \{ PERSONALNR \rightarrow ABT$
 $ABT \rightarrow ORT$
 $PERSONALNR \rightarrow NAME \}$
 $g5$

4. Ausgabe der minimalen Überdeckung G'''

$PERSONALNR \rightarrow NAME$
 $ABT \rightarrow ORT$

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

6.26

$PERSONEN = \{PNR, ZUENAME, VORNAME, GEGALT, GESCHLECHT\}$

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

$kinder[PNR] \subseteq personen[PNR] \dots Inklusionsabhängigkeit$

PNR ist für $PERSONEN$ Schlüssel $\Leftrightarrow PNR$ ist Fremdschlüssel für $KINDER$.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

6.29

Beispiel 7 Händler verkaufen Produkte für Firmen, dargestellt im Schema $R = \{HÄNDLER, FIRMA, PRODUKT\}$

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: $(*[HÄNDLER\ FIRMA\ PRODUKT, HÄNDLER\ PRODUKT])$

kann zerlegt werden in

$R_1 = \{HÄNDLER, FIRMA\}$

$R_2 = \{FIRMA, PRODUKT\}$

DBVO

6.24

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 $\pi_X(r) \subseteq \pi_Y(s)$ gilt.

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

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

6.27

Ableitungsregeln für Inklusionsabhängigkeiten

Casanova et. al. 1984

II: Reflexivität
 $X \subseteq R \vdash R[X] \subseteq R[X]$

12: Projektion und Permutation
 $R[X_1 \dots X_n] \subseteq S[Y_1 \dots Y_n] \vdash R[X_k \dots X_m] \subseteq S[Y_k \dots Y_m]$, wobei $k \dots m$ Folgen von Indizes aus dem Bereich $1 \dots n$ sind

13: Transitivität
 $R[X] \subseteq S[Y], S[Y] \subseteq T[Z] \vdash R[X] \subseteq T[Z]$.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

6.30

Beispielaufgabe

Geben ist das Relationenschema PQIRSTU und die Menge F von funktionalen Abhängigkeiten. Bestimmen Sie eine minimale Überdeckung.

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

DBVO

6.25

Inklusionsabhängigkeiten

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

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

6.28

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

6.28

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 $*[XY, XZ]$.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

7.1

Normalformen

Ein FD $X \rightarrow Y$ heißt **volle funktionale** Abhängigkeit, wenn für keine **echte** Teilmenge $X' \subset X$ gilt, daß $X' \rightarrow Y$.

$X \rightarrow Y$ heißt dann **voll funktional** abhängig von X .

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

Einfache Bedingung für die Minimalität in der Definition eines Schlüssels:

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

http://gerald.pfeifer.com

TU Wien

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

A ist prim $\Leftrightarrow \exists X. X$ 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 Attributmenge $Y \subseteq R$ gibt, sodas $X \rightarrow Y \in F^+$ und $Y \rightarrow A \in F^+$ sind und $Y \rightarrow X \notin F^+$, $A \notin XY$ gilt.

$X \rightarrow A$ transitiv $\Leftrightarrow \exists Y, Y \subseteq R. A \notin XY, X \rightarrow Y \rightarrow A, Y \neq X$

http://gerald.pfeifer.com

TU Wien

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.

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

http://gerald.pfeifer.com

TU Wien

Beispiel 5

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

$Relationenschema\ R = (ÜBUNG, STUDENT, TUTOR)$

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

$Zerlegung\ des\ Schemas\ R\ in\ R_1 = (STUDENT, TUTOR),\ R_2 = (TUTOR, ÜBUNG)$

$Die\ Abhängigkeit\ TUTOR \rightarrow ÜBUNG\ verletzt\ die\ BCNF\ (aber\ nicht\ die\ 3.NF).$

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

http://gerald.pfeifer.com

TU Wien

Sei $R = (S, F)$ ein Relationenschema, und $R_1 = (S_1, F_1), S_1 \subseteq S, R_1$ Subschema.

Ein 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 .

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

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

http://gerald.pfeifer.com

TU Wien

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:

- $S = S_1 \cup S_2 \cup \dots \cup S_n$
- $F_i \equiv F^+[S_i]$
- $(F_1 \cup F_2 \cup \dots \cup F_n)^+ \subseteq F^+$.

http://gerald.pfeifer.com

TU Wien

Ein FD $X \rightarrow Y$ heißt **prim** in R , wenn es in **mindestens einem** Schlüssel von R enthalten ist, sonst heißt es **nichtprim**.

A ist prim $\Leftrightarrow \exists X. X$ 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 Attributmenge $Y \subseteq R$ gibt, sodas $X \rightarrow Y \in F^+$ und $Y \rightarrow A \in F^+$ sind und $Y \rightarrow X \notin F^+$, $A \notin XY$ gilt.

$X \rightarrow A$ transitiv $\Leftrightarrow \exists Y, Y \subseteq R. A \notin XY, X \rightarrow Y \rightarrow A, Y \neq X$

http://gerald.pfeifer.com

TU Wien

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.

http://gerald.pfeifer.com

TU Wien

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.

$3.NF \Leftrightarrow A$ nichtprim \Rightarrow Schlüssel $\rightarrow A$ 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.

$3.NF \Leftrightarrow X \rightarrow A \Rightarrow X \rightarrow R \vee A \in X \vee A$ prim.

http://gerald.pfeifer.com

TU Wien

Ein **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:

- $S = S_1 \cup S_2 \cup \dots \cup S_n$
- $F_i \equiv F^+[S_i]$
- $(F_1 \cup F_2 \cup \dots \cup F_n)^+ \subseteq F^+$.

Die Zerlegung ist **verbundtreu** (verlustfrei, lossless), wenn für jede zulässige Abhängigkeit geht verloren), wenn:

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

Die Zerlegung ist **abhängigkeitstreu** (dependency preserving, keine Abhängigkeit geht verloren), wenn:

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

Relation $r(S)$ gilt:

http://gerald.pfeifer.com

TU Wien

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

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 $R_1 = (S_1, F_1)$, $R_2 = (S_2, F_2)$ mit

$$S_1 = \{A, B, C, D\}, \quad F_1 = \{A \rightarrow BCD, CD \rightarrow B\}$$
$$S_2 = \{C, D, E\}, \quad F_2 = \{CD \rightarrow E\}.$$

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

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

7.23

Wenn das Join-Attribut in einer der beiden Teilrelationen Schlüssel ist, so ist die Zerlegung verbundenre.

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

$$\exists S_i \mid S_i^+ = S \text{ bzgl. } F' = \cup_{i=1}^n F_i$$
$$\Rightarrow$$

Zerlegung von S in S_1, S_2, \dots, S_n ist verbundenre

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

7.26

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: $F_1^+ = F^+[S_1]$ und $F_2^+ = F^+[S_2]$.

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

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

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

Beispiel 7 Die Zerlegung in Beispiel 5 ist verbundenre, aber nicht abhängigkeitsreul: die Abhängigkeit STUDENT ÜBUNG \rightarrow TUTOR geht verloren.

- Zerlegung in BCNF ist nicht immer verbund- und abhängigkeitsreul möglich \Rightarrow Nachteil der BCNF.
- Verzichte auf die Abhängigkeitsreul, nicht auf die Verbundreul, denn: Die Abhängigkeitsreul stellt Metainformation dar, die Verbundreul aber Information.
- Das Entscheidungsproblem, ob es eine abhängigkeitsreul (und verbundenreul) Zerlegung eines Datenbankschemas in BCNF gibt, ist NP-vollständig.
- Der Test, ob ein Datenbankschema in BCNF ist, ist hingegen polynomial.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

7.24

Verbundreul Zerlegung in 3.NF bzw. BCNF

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

Algorithmus DECOMP

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

OUTPUT: verbundreul 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 heiße weiterhin F .

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

7.27

- Das Verfahren terminiert immer: Kardinalität der erzeugten Schemata nimmt in jedem Schritt ab.
- Die Zerlegung ist verbundreul: 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 böserige FD (wichtig: F ist eine minimale Überdeckung).

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

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 verbundreul Zerlegung, für die zweite nicht.

r	A	B	C	r_1	A	B	r_2	B	C	r'	A	B	C
a_1	b_1	c_1	\Rightarrow	a_1	b_1	a_1	b_1	b_1	c_1	$=$	a_1	b_1	c_1
a_2	b_2	c_2	\Rightarrow	a_2	b_2	a_2	b_2	b_2	c_2	$=$	a_2	b_2	c_2

r	A	B	C	r_1	A	B	r_2	B	C	r'	A	B	C
a_1	b_1	c_1	\Rightarrow	a_1	b_1	a_1	b_1	b_1	c_1	$=$	a_1	b_1	c_1
a_2	b_1	c_2	\Rightarrow	a_2	b_1	a_2	b_1	b_1	c_2	$=$	a_2	b_1	c_2

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

7.25

2. Prüfe, ob F eine "böserige" Abhängigkeit enthält, d.h.

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 böserige Abhängigkeit, dann ist $R = (S, F)$ bereits in 3.NF bzw. im BCNF. Fertig.

3. Gibt es eine böserige 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:

$$S_1 = S - A \text{ und } S_2 = XA, \text{ wobei } X \text{ der Schlüssel für die neue Relation } R_2 \text{ wird.}$$

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

7.28

- 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 böserige 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).

abhängigkeitsreul

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

Logische Abfrageoptimierung

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 $O(nm)$.
- Projektion ist aufwendig: Duplikate müssen entfernt werden.
- Selektionen so früh wie möglich durchführen: kleinere Zwischenergebnisate.
- \cup äre 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.

http://gerald.pfeifer.com

http://gerald.pfeifer.com

- Bottom-up Auswertung des Operatorbaumes:**
- ungünstig, wenn die Join-Operatoren nahe bei den Blättern stehen.
 - Zeit- und Plataufwand binärer Operationen steigt mit # der Tupel und der # der Attribute in den Argumentrelationen.
 - Grundprinzip: \cup äre Operatoren in Richtung der Blätter des Baumes verschieben
 - durch Zusammenfassung \cup ärer Operationen werden diese in einem Schritt durchgeführt.
 - Selektion **vor** der Projektion wegen der Entfernung doppelter Tupel.

http://gerald.pfeifer.com

Zusammenfassen gleicher Teilausdrücke

- Schrittweise von unten nach oben.
- Algebraische Transformationen können die Existenz gleicher Teilausdrücke verschleiern.

Beispiel 2 Fasse gleiche Teilausdrücke im Baum aus Beispiel 1 zusammen.

http://gerald.pfeifer.com

http://gerald.pfeifer.com

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

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

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

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

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

- 4. Zusammenlegung/Kommutativität von Selektionen:**

http://gerald.pfeifer.com

Der Join und das kartesische Produkt sind kommutativ und assoziativ.

- 1. Kommutativität:**

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

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

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

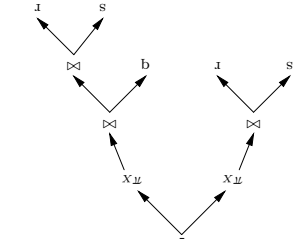
http://gerald.pfeifer.com

Regeln für Join und kartesisches Produkt

Algebraische Optimierung

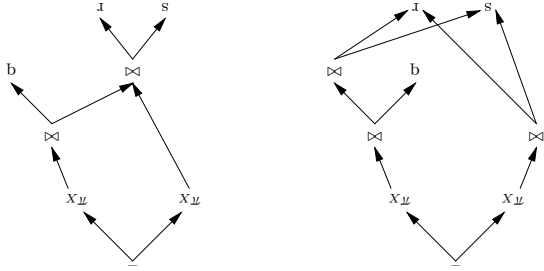
Bei der algebraischen Optimierung verwenden wir zur Darstellung von Queries einen Operatorbaum.

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



http://gerald.pfeifer.com

http://gerald.pfeifer.com



http://gerald.pfeifer.com

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:

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

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 $\{A_i \mid i = 1, \dots, n\} = \{B_i \mid i = 1, \dots, m\} \cup \{C_i \mid i = 1, \dots, k\}$, so gilt:

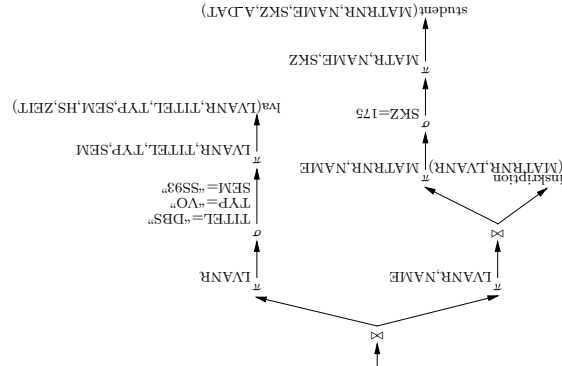
$$\pi_{A_1, \dots, A_n}(E_1 \times E_2) \equiv \pi_{B_1, \dots, B_m}(E_1) \times \pi_{C_1, \dots, C_k}(E_2)$$

Beispiel 3 Relationenschemata:

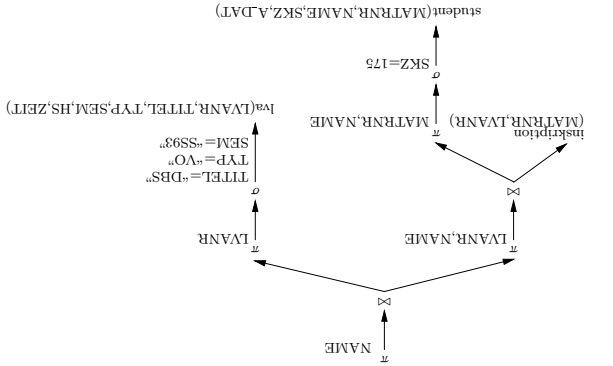
student(MATRNR, NAME, SKZ, ANFANGSDATUM)
iva(LVANR, TITEL, TYP, SEM, HS, ZEIT)
inskrption(MATRNR, LVANR)
Formuliere eine Query in relationaler Algebra, die die Namen aller Wirtschaftsinformatiker (SKZ="175") ermittelt, die im SS93 die Lehrveranstaltung mit dem Titel "DBS" (TYP="VO") besucht haben:

$\pi_{NAME}[\sigma_{SKZ=175, TITEL='DBS', TYP='VO', SEM='SS93'}(student \bowtie iva \bowtie inskrption)].$

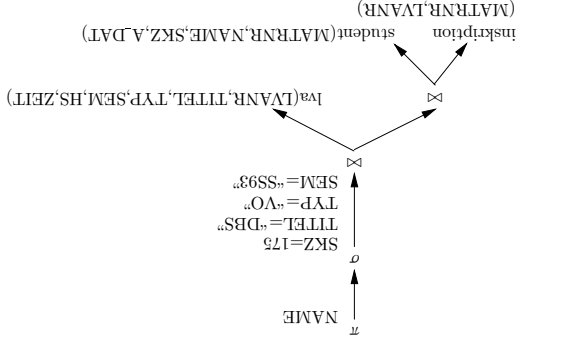
Schritt 3:



Schritt 4:



Schritt 1:



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 $A_k = B_l$:

$$R_1 \bowtie R_2 = \pi_{C_m \setminus B_l}(R_1 \times R_2)$$

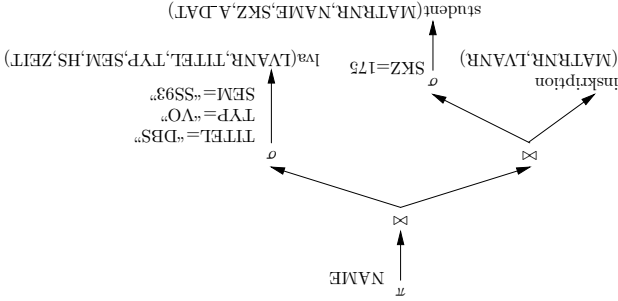
\Rightarrow Regeln für den Join folgen aus den Regeln oben.

Es gilt keine Kommutativität zwischen Mengendifferenz und Projektion!

Ein einfacher Optimierungsalgorithmus

1. Zerlege Selektionen der Art $\sigma_{F_1 \wedge \dots \wedge F_n}(E)$ nach Regel 4 in $\sigma_{F_1}(\dots(\sigma_{F_n}(E))\dots)$.
2. Schiebe jede Selektion soweit wie möglich in Richtung Blätter mit den Regeln 4 – 8.
3. Schiebe jede Projektion soweit wie möglich in Richtung Blätter mit den Regeln 3, 5, 9 und 10.
4. Fasse alle 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.

Schritt 2:



Der Semijoin (Semiverbund)

- Verteilte Datenbank: ein Teil der Tabellen sind am Knoten 1, der andere am Knoten 2 gespeichert.
 - Join von zwei Relationen, die nicht am selben Knoten liegen sehr aufwendig, da eine gesamte Tabelle übertragen werden muß. Lösung: der Semijoin.
- Beispiel 4** r auf Knoten 1 gespeichert, s auf Knoten 2 gespeichert. Gesucht: $r \bowtie s$ auf Knoten 1.

Knoten 1

$r \begin{pmatrix} A & B \end{pmatrix}$

Knoten 2

$s \begin{pmatrix} B & C & d \end{pmatrix}$

Knoten 1

$r' \begin{pmatrix} B \end{pmatrix}$

Knoten 2

$s' \begin{pmatrix} B & C & D \end{pmatrix}$

1	4	4	13	16	4	14	16	4	13	17	1	6	2	4	2	3	7	3	8	3	9
10	14	16	10	15	16	10	15	16	11	15	16	11	15	16	12	15	16	11	15	16	12

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

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.*

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

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.

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

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 $r \bowtie s'$.

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

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

Bei einer Berechnung des Joins ohne Verwendung des Semijoins müßten wir $3 * 8 = 24$ Attributwerte von s auf den Knoten 1 übertragen.

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

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

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

```
begin transaction
update KONTOKonto
set KSTAND=KSTAND-500
where KONTONR=123;
update SPARBUCH
set EINLAGE=EINLAGE+500
where SPNR=333;
end transaction
```

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

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 aufscheinen und kann nicht mehr etwa aus Systemgründen rückgängig gemacht werden.

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

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

Konsistenzzerhaltung (consistency)

Das Gesamtergebnis einer parallelen Ausführung von Transaktionen entspricht **irgendeiner** Hintereinanderausführung dieser Transaktionen. (Eine Transaktion ist konsistenzzerhaltend ⇒ eine Hintereinanderausführung konsistenzzerhaltend.)

Mehrbenutzerkontrolle (Concurrency Control) sichergestellt.

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

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.

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

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.*

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

Mehrbenutzerkontrolle (Concurrency Control)

Das Gesamtergebnis einer parallelen Ausführung von Transaktionen entspricht **irgendeiner** Hintereinanderausführung dieser Transaktionen. (Eine Transaktion ist konsistenzzerhaltend ⇒ eine Hintereinanderausführung konsistenzzerhaltend.)

Mehrbenutzerkontrolle (Concurrency Control) sichergestellt.

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

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.*

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

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 aufscheinen und kann nicht mehr etwa aus Systemgründen rückgängig gemacht werden.

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

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

Konsistenzzerhaltung (consistency)

Das Gesamtergebnis einer parallelen Ausführung von Transaktionen entspricht **irgendeiner** Hintereinanderausführung dieser Transaktionen. (Eine Transaktion ist konsistenzzerhaltend ⇒ eine Hintereinanderausführung konsistenzzerhaltend.)

Mehrbenutzerkontrolle (Concurrency Control) sichergestellt.

TU Wien

http://gerald.pfeifer.com

Gerald Pfeifer

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

Formal:

- 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_{j_m} 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.

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

$$D(I_1) = \{(T_1, a, T_2), (T_1, b, T_2)\}$$
$$D(I_2) = \{(T_1, a, T_2), (T_1, b, T_2)\}$$
$$D(I_3) = \{(T_1, a, T_2), (T_2, b, T_1)\}$$

I_2 ist serial, I_3 ist serialisierbar.

$Sei D(I_1) = \{(T_2, a, T_1), (T_2, b, T_1)\}$ die Konfliktrelation der zweiten möglichen serialen Ausführung.

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

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

$I \dots$ 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 , $1 \leq k < l \leq |I|$, $I(k) = (T_i, a_k, e)$, $I(l) = (T_j, a_{j_m}, e)$, so daß gilt:
1. $T_i \neq T_j$ und $(a_i = \text{write oder } a_{j_m} = \text{write})$
2. Es gibt kein h für $k < h < l$, sodaß $I(h) = (T_g, a_{g_o}, e)$ wobei $(T_g \neq T_i$ und $T_g \neq T_j)$ und $a_{g_o} = \text{write}$
 $D(I)$ 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
TU Wien
http://gerald.pfeifer.com

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.

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

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 serialen 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
TU Wien
http://gerald.pfeifer.com

- 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 Effektserialisierbarkeit auch dann vorliegen, wenn es keine äquivalente serielle Ausführung gibt.
- Annahme : die CWA gilt.

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

- Schwächerer Begriff der Serialisierbarkeit: eine Ausführung heißt effektserialisierbar (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 effektserialisierbar d.h.
- Serialisierbarkeit ist eine hinreichende, aber keine notwendige Bedingung dafür, daß eine gegebene parallele Ausführung von Transaktionen effektserialisierbar ist.

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

Test auf Serialisierbarkeit

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.

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

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

I_1	I_2	I_3
READ A WRITE A READ B WRITE B READ B WRITE B	READ A WRITE A READ B WRITE B READ B WRITE B	READ A WRITE A READ B WRITE B READ B WRITE B
T_1	T_2	T_1 T_2

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

Überprüfung der Serialisierbarkeit von Ausführungen mit graphentheoretischen Mitteln:

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

T_1

T_2

$G(I_1) = G(I_2)$

a, b

T_1

T_2

$G(I_3)$

a

b

$G(I_3)$

Anmerkung: i.A. keine eindeutige Ordnung.

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

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

10.29

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. UNLOCK A		4. WRITE B
	5. WRITE B		6. UNLOCK A
	6. UNLOCK A		7. WRITE B
	7. WRITE B		8. UNLOCK B

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

10.32

Wohlgeformte Transaktionen

Eine Transaktion ist **wohlgeformt**, 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.

DBVO

10.27

Sperrprotokolle

- Dynamisches Sperren und Freigeben von Datenanobjekten: 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.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

10.30

Eine gültige Ausführung I_1 dieser beiden Transaktionen ist:

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

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

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

10.33

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

DBVO

10.28

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.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

10.31

T_1

T_2

$LOCK A$

$WRITE A$

$LOCK B$

$WRITE B$

$UNLOCK A$

$WRITE B$

$UNLOCK B$

$1.$

$2.$

$3.$

$4.$

$5.$

$6.$

$7.$

$8.$

$9.$

$10.$

$11.$

$12.$

$13.$

$14.$

$LOCK A$

$WRITE A$

$LOCK B$

$WRITE B$

$UNLOCK A$

$WRITE B$

$UNLOCK B$

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

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

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

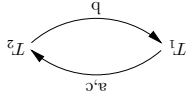
DBVO

10.34

Beispiel 11

T_1	T_2
LOCK a	
LOCK c	
WRITE a	
WRITE c	
UNLOCK a	
	LOCK a
	WRITE a
	LOCK b
	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:

Sperr-anforderung	S-LOCK	+
	X-LOCK	–
Bestehende Sperre	S-LOCK	+
	X-LOCK	–

+ ... Verträglich, – ... Konflikt

Das 2-Phasen-Sperverfahren (2-Phase-Locking)

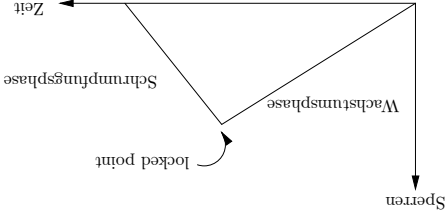
- Eine Transaktion $T = \langle (a_i, c_i) \rangle$, $i = 1, \dots, n$ ist eine 2-Phasen-Transaktion, wenn für ein $j < n$ und für alle $i = 1, \dots, n$ gilt:
- $i < j \Rightarrow a_i < > \text{UNLOCK}$ Sperrphase (Wachstumsphase)
 - $i = j \Rightarrow a_i = \text{UNLOCK}$ Sperrpunkt
 - $i > j \Rightarrow a_i < > \text{LOCK}$ Schrumpfungsphase

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

<div><div>LOCK a</div><div>LOCK b</div><div>UNLOCK a</div><div>LOCK c</div><div>UNLOCK c</div><div>UNLOCK b</div></div> <div><div>T₁</div><div>T₂</div></div>	→ Nicht 2-Phasen	<div><div>LOCK a</div><div>LOCK b</div><div>UNLOCK c</div><div>UNLOCK a</div><div>UNLOCK b</div></div> <div><div>T₁</div><div>T₂</div></div>
---	------------------	--

Das Baumprotokoll

- Ansammlung von Objekten (Datenbanken, Menge von Relationen, Relationen, oder Tupel) gespeichert als Knoten eines Baumes (z.B. B*-Baum).
- Verwendung von exklusiven 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.



Deadlock

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.

LOCK a

T₁

T₂

LOCK b

(LOCK b) → T₁ wartet auf b

(LOCK a) → T₂ wartet auf a

ACHTUNG: hinreichend ≠ notwendig!

Theorem: Wenn alle Transaktionen einer Transaktionsmenge T wohlgeformt sind und das Baumprotokoll erfüllen, dann ist jede gültige Ausführung I von **T** serialisierbar.

Hinreichende Bedingung für Serialisierbarkeit:

DBVO10.44

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Beispiel 13

T_1, T_2 und T_3 erfüllen das Baumprotokoll, die **T** ist wohlgeformt, I ist gültig $\Rightarrow T$ serialisierbar.

DBVO10.47

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Eine hinreichende Bedingung für Serialisierbarkeit ist:

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

Kompatibilitätsrelation:

Sper-	I-Lock	+	—
anforderung	X-Lock	—	—

Bestehende Sperre

— I-Lock
+ I-Lock
— X-Lock
— X-Lock

+, . . . , Verträglich, — . . . , Konflikt

DBVO10.50

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Konsistenz wird sichergestellt:

- Eine Transaktion mit Zeitstempel t_1 darf ein Objekt mit Schreibstempel t_w nicht lesen, wenn $t_w > t_1$ 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 Lese-Zeitstempel 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.

DBVO10.45

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Das Hierarchische Sperprotokoll

Zwei Arten von Sperren:

das Exklusive Lock (X-Sperre)

das Intention Lock (I-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.

DBVO10.48

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Beispiel 15

T_1, T_2 und T_3 erfüllen das Hierarchische Sperprotokoll, **T** ist wohlgeformt und I ist gültig $\Rightarrow I$ serialisierbar.

DBVO10.51

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Achtung:

zwei Lesezugriffe: kein Konflikt.

zwei Schreibzugriffe ohne einen Lesezugriff dazwischen: nur der jüngere bleibt erhalten, da der ältere Wert überschreiben bzw. gar nicht 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.

DBVO10.46

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Eine wohlgeformte 2-Phasen-Transaktion T gehorcht dem Hierarchischen Sperprotokoll, 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.
- T gibt einen Knoten K erst dann frei, wenn sie alle Nachfolger von K freigegeben hat.

DBVO10.49

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

Zeitstempelverfahren (Time Stamping)

Zuordnung von Zeitstempeln (timestamps) zu Transaktionen und Datenbankobjekten.

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

DBVO10.52

Gerald PfeiferTU Wienhttp://gerald.pfeifer.com

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.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

- 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.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO 11.6

Beispiel 2 T_1 , T_2 und T_3 sind beim Systemabsturz abgeschlossen \Rightarrow *Dauerhaftigkeit* muß gewahrt sein. Für *Atomarität*: T_4 und T_5 müssen rückgängig gemacht werden, da sie nicht abgeschlossen sind.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO 11.9

Beispiel 4 \Rightarrow **Zurücksetzung von T_3**

Da x bereits freigegeben und von T_1 und T_2 gelesen wurde, müssen T_1 und T_2 wegen *Atomarität* zurückgesetzt werden (*Cascading Abort*). T_1 und T_2 waren erfolgreich abgeschlossen \Rightarrow *Verteilung der Dauerhaftigkeit*.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO 11.4

Beispiel 1 *Bankenszenario (Umbuchung)*

Datenstruktur:
SPARBUCH (SPNR, EINLAGE, BESITZER)
KONTO (KNR, KSTAND)
SPAREINLAGEN_GESAMT

Integritätsbedingung:
KSTAND >= 0
EINLAGE > 0
SPAREINLAGEN_GESAMT = Summe EINLAGE

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO 11.7

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.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO 11.10

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 *Widerherstellbarkeit* ist garantiert.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO 11.5

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

```
begin transaction
update KONTO
set KSTAND=KSTAND-50
where KNR=123;
update SPARBUCH
set EINLAGE=EINLAGE+50
where SPNR=321
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.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO 11.8

Beispiel 3

T_i kein T_k schreibt auf X

T_j read(X)

T_i write(X)

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO 11.11

ACA Ausführung

Problem: kaskadierendes Zurücksetzen

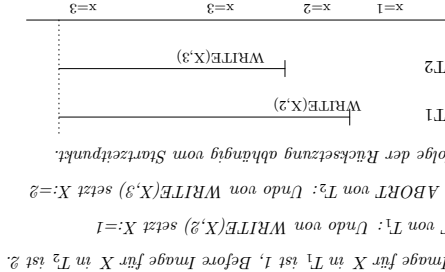
Beispiel 5 T_2 muß bei einem COMMIT-Request abbrechen.

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 .

- Typische Implementierungsmöglichkeiten:
 - Merken 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.

Zurücksetzen einer Transaktion



Pufferverwaltung

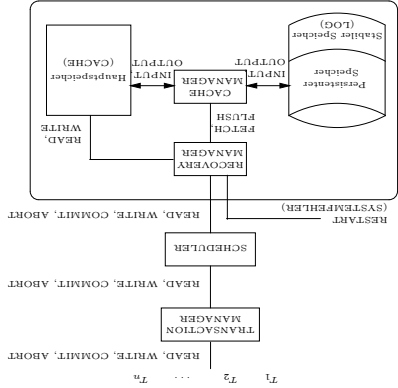
- Drei Arten von Speicher:
- Hauptspeicher (volatile storage)
 - Hintergrundspeicher (non-volatile storage): z.B.: Festplatte
 - Stabiler Speicher (stable storage): Information geht nicht verloren. (Annahme!)

- read(X, x_i)** weist den Wert des Datenobjekts X der Programmvariable x_i zu.
1. Ist der Block, auf dem sich X befindet, noch nicht im Hauptspeicher, dann führe **input(X)** aus.
 2. Weise x_i den Wert von X im Pufferblock zu.
- write(X, x_i)** weist den Wert der Programmvariable x_i 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_i .

Programmuzugriff auf DB

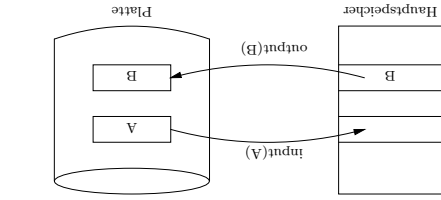
- Eine Ausführung I von Transaktionen ist **strikt (ST)**, wenn für jede Transaktion gilt:
- T_j liest und schreibt nur Objekte X , die zuletzt von erfolgreich abgeschlossenen Transaktionen T_j geschrieben wurden.
- Praxis:** (2-Phasen Sperrverfahren garantiert Strictness)
- Alle **WRITE**-Befehle werden bis zum locked point verzögert
 - **UNLOCK**-Befehle werden bis Transaktionsende verzögert
 - alle **UNLOCKS** werden zu einem Befehl (**COMMIT**) zusammengefasst
 - **COMMIT** = Ende der Transaktion.

Strikte Ausführung



- 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 Pufferverwalter auf die Festplatte wenn 1. Puffer wird zu klein, 2. vom Wiederanlaufverwalter verlangt (**forced write**).

Puffer als Cache



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 10

Zeit

Checkpoint

Absturz

T1

T2

T3

T4

T5

REDO

REDO

UNDO

UNDO

UNDO

- *UNDO* T₄, T₅
- *REDO* T₂, T₃
- *RESTART* T₄, T₅

Beispiel 12

Zeit

Checkpoint

Absturz

T1

T2

T3

T4

T5

REDO

REDO

UNDO

UNDO

UNDO

- *DB:=Backup*
- *REDO* T₂
- *RESTART* T₃

DBVO 11.45

Seiten auf der Platte

Seiten-tabelle

1

2

3

4

5

...

n

1

2

3

4

5

6

7

8

Beispiel 11 Achtung: nicht strikte Ausführung

Zeit

Checkpoint

Absturz

T1

T2

T3

T4

T5

REDO

REDO

UNDO

UNDO

UNDO

- *UNDO* von *WRITE*(X) in T₄ macht ein *REDO* von *WRITE*(X) in T₂ notwendig.

Schattenkopieverfahren (shadow paging)

Vermeidung eines Logs durch Führen einer zweiten Datenbank-Kopie

Schattenkopie: enthält den Datenbankzustand vor dem Start der Transaktion, repräsentiert den nach der letzten erfolgreich abgeschlossenen Transaktion gültigen Datenbankzustand.

Arbeitskopie: auf ihr werden alle Änderungen der Transaktion durchgeführt. Initialisierung mit Schattenkopie.

Bei Transaktionsabbruch: Arbeitskopie verwerfen.

Bei Transaktions-Commit: Arbeitskopie wird neue Schattenkopie.

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

Schattenkopie-Realisierung

- Schatten-speicher-verfahren: verwende zwei Seiten-tabellen.
- Seiten-, die in der Schatten-kopie und in der Arbeits-kopie gleich sind, werden nur einmal gespeichert und von beiden Tabellen referenziert.
- Eine fixe Speicher-adresse auf der Platte enthält einen Zeiger auf die Schatten-seiten-tabelle.

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

DBVO 11.41

Wiederanlauf nach einem Plattenfehler

Voraussetzungen:

- Sicherungs-kopie (Backup) und Log-protokoll ab dem Zeitpunkt der Sicherungs-kopie vorhanden
- keine Transaktion während der Sicherung aktiv.

Algorithmus

1. Ersetze die Datenbank durch die Sicherungs-kopie.
2. begin transaction T_i im Log ⇒ T_i in RedoListe
commit transaction T_i im Log ⇒ T_i in RedoListe und T_i aus RedoListe
3. Lies das Log vom Anfang bis zum Ende und übernimmt für Änderungen von Transaktionen in der RedoListe die neuen Werte in die Datenbank.

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

Annahme für Realisierung

- Aufteilung der Datenbank in Seiten (pages).
- Eine Seite umfaßt eine vorgegebene Anzahl Blöcke fixer Länge.
- Die Seiten sind nicht notwendigerweise sequentiell gespeichert.
- Seiten-tabelle (page table) ordnet jeder Seite, identifiziert durch die Seitennummer, ihre Adresse auf der Platte zu.

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

DBVO 11.47

SST

AST

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

Schattenkopie - Schreiben

Zum Transaktionsbeginn wird die Schattenseitenentabelle in die Arbeitsseitenentabelle kopiert.

Eine Schribooperation $write(X, x_j)$ auf ein Objekt X , das auf der i -ten Seite liegt, wird wie folgt behandelt:

- Ist die i -te Seite nicht im Puffer, dann übertrage sie mit $input(X)$.
- 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 Arbeitsseitenentabelle zu.
- Weise den Wert x_i dem Objekt X im Puffer zu.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

99.4

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:

- Schreibe alle "schmutzigen" Seiten im Puffer auf die Platte.
- Schreibe die Arbeitsseitenentabelle auf die Platte.
- Schreibe die Adresse der Arbeitsseitenentabelle auf jene fixe Speicheradresse, die die Adresse der Schattenseitenentabelle enthält.

Das Wiederanlaufverfahren nach einem Systemfehler oder Transaktionsfehler ist einfach: setze die Arbeit mit der Schattenkopie fort.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

99.5

1. Selektieren Sie die Namen aller Fahrer, die bei der Tour de France die meisten Etappensiege feierten.

Schatten-speicher-verfahren

Vorteile:

- Keine Verwaltung eines Logprotokolls.
- Wiederanlauf beträchtlich schneller.

Nachteile:

- Datenfragementierung
- Freispeicherverwaltung
- Schwerer als Verfahren mit Logprotokoll auf Mehrbenutzerbetrieb erweiterbar
- Indirekter Datenzugang, langsamer Zugriff

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

99.6

3. Schreiben Sie eine Abfrage in Relationaler Algebra, die jene Fahrer selektiert, die in dieser Saison mindestens eine Schlußetappe gewonnen haben.

DBVO

99.9

immer paarweise (d.h., verbinden zwei Bauteile). Auf einer Platine können auch unverbundene Bauteile existieren.

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien

DBVO

99.8

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). (Anm.: 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

DBVO

99.7

$$\pi_{f_name} \sigma_{platz=1} (wertung_{[w.e.nr=e.e.nr,w.r.name=e.r.name]} (etappe_{[e.r.name=r.name,bis=ziel]rennen}))$$

Gerald Pfeifer

http://gerald.pfeifer.com

TU Wien