



Skript zur Vorlesung  
**Datenbanksysteme I**  
Wintersemester 2012/2013

# Kapitel 4: Relationen-Kalkül

Vorlesung: Prof. Dr. Christian Böhm  
Übungen: Sebastian Goebel, Nina Hubig  
Skript © 2005 Christian Böhm

[http://www.dbs.ifi.lmu.de/cms/Datenbanksysteme\\_I](http://www.dbs.ifi.lmu.de/cms/Datenbanksysteme_I)



# Begriff

**Kalkül** *das, auch der; -s, -e* <unter Einfluss von gleichbed. fr. calcul aus lat. calculus «Steinchen, Rechen-, Spielstein; Berechnung», Verkleinerungsform von lat. calx «(Spiel)stein; Kalk»>: etwas im Voraus abschätzende, einschätzende Berechnung, Überlegung.

Quelle: DUDEN - Das große Fremdwörterbuch

...**das Kalkül**

**der Kalkül** ...

**Kalkül** *der; -s, -e* <zu <sup>1</sup>Kalkül>: durch ein System von Regeln festgelegte Methode, mit deren Hilfe bestimmte mathematische Probleme systematisch behandelt u. automatisch gelöst werden können (Math.).

Quelle: DUDEN - Das große Fremdwörterbuch



# Begriff

- Mathematik: Prädikatenkalkül
  - Formeln wie  $\{x \mid x \in \mathbb{N} \wedge x^3 > 0 \wedge x^3 < 1000\}$
- Anwendung solcher Formeln für DB-Anfragen
  - Bezugnahme auf DB-Relationen im Bedingungsteil:  
 $(x_1, y_1, z_1) \in \text{Mitarbeiter}, t_1 \in \text{Abteilungen}$
  - Terme werden gebildet aus Variablen, Konstanten usw.
  - Atomare Formeln aus Prädikaten der Datentypen:  
 $=, <, >, \leq, \text{ usw.}$
  - Atomare Formeln können mit logischen Operatoren zu komplexen Formeln zusammengefasst werden:  
 $F_1 \wedge F_2, F_1 \vee F_2, \neg F_1, \exists x: F_1, \forall x: F_1$
- Bsp: Finde alle Großstädte in Bayern:  
 $\{t \mid \text{Städte}(t) \wedge t[\text{Land}] = \text{Bayern} \wedge t[\text{SEinw}] \geq 500.000\}$   
Hinweis: Städte(t) gleichbedeutend mit  $t \in \text{Städte}$



# Unterschied zur Rel. Algebra

- Relationale Algebra ist **prozedurale** Sprache:
  - Ausdruck gibt an, unter Benutzung welcher Operationen das Ergebnis berechnet werden soll
  - WIE
- Relationen-Kalkül ist **deklarative** Sprache:
  - Ausdruck beschreibt, welche Eigenschaften die Tupel der Ergebnisrelation haben müssen ohne eine Berechnungsprozedur dafür anzugeben
  - WAS
- Es gibt zwei verschiedene Ansätze:
  - **Tupelkalkül**: Variablen sind vom Typ *Tupel*
  - **Bereichskalkül**: Variablen haben *einfachen* Typ



# Der Tupelkalkül

- Man arbeitet mit
  - Tupelvariablen:  $t$
  - Formeln:  $\psi(t)$
  - Ausdrücken:  $\{t \mid \psi(t)\}$
- Idee: Ein Ausdruck beschreibt die Menge aller Tupel, die die Formel  $\psi$  **erfüllen** (wahr machen)
- Ein Kalkül besteht immer aus
  - Syntax: Wie sind Ausdrücke aufgebaut?
  - Semantik: Was bedeuten die Ausdrücke?



# Tupelvariablen

- Tupelvariablen haben ein definiertes Schema:
  - $\text{Schema}(t) = (A_1: D_1, A_2: D_2, \dots)$
  - $\text{Schema}(t) = R_1$  ( $t$  hat dasselbe Schema wie Relation)
- Für Zugriff auf die Komponenten
  - $t[A]$  oder  $t.A$  für einen Attributnamen  $A \in \text{Schema}(t)$
  - oder auch  $t[1]$ ,  $t[2]$  usw.
- Tupelvariable kann in einer Formel  $\psi$  **frei** oder **gebunden** auftreten (s. unten)



# Atome

- Es gibt drei Arten von Atomen:
  - $R(t)$        $R$  ist Relationenname,  $t$  Tupelvariable  
**lies:  $t$  ist ein Tupel von  $R$**
  - $t.A \Theta s.B$        $t$  bzw.  $s$  sind zwei Tupelvariablen mit  
passenden Attributen  
**lies:  $t.A$  steht in Beziehung  $\Theta$  zu ...**
  - $t.A \Theta c$        $t$  ist Tupelvariable und  $c$  eine passende  
Konstante

---

$\Theta$  Vergleichsoperator:  $\Theta \in \{ =, <, \leq, >, \geq, \neq \}$



# Formeln

Der Aufbau von Formeln  $\psi$  ist rekursiv definiert:

- **Atome:** Jedes Atom ist eine Formel  
Alle vorkommenden Variablen sind **frei**
- **Verknüpfungen:** Sind  $\psi_1$  und  $\psi_2$  Formeln, dann auch:
  - $\neg \psi_1$  *nicht*
  - $(\psi_1 \wedge \psi_2)$  *und*
  - $(\psi_1 \vee \psi_2)$  *oder*Alle Variablen behalten ihren Status.
- **Quantoren:** Ist  $\psi$  eine Formel, in der  $t$  als **freie Variable** auftritt, sind auch Formeln...
  - $(\exists t)(\psi)$  *es gibt ein  $t$ , für das  $\psi$*
  - $(\forall t)(\psi)$  *für alle  $t$  gilt  $\psi$*die Variable  $t$  wird **gebunden**.





# Formeln

- Gebräuchliche vereinfachende Schreibweisen:
  - $\psi_1 \Rightarrow \psi_2$  für  $(\neg \psi_1) \vee \psi_2$  (**Implikation**)
  - $\exists t_1, \dots, t_k: \psi(t_1, \dots, t_k)$  für  $(\exists t_1) (\dots ((\exists t_k) (\psi(t_1, \dots, t_k))) \dots)$
  - $(\exists t \in R) (\psi(t))$  für  $(\exists t) (R(t) \wedge \psi(t))$
  - $(\forall t \in R) (\psi(t))$  für  $(\forall t) (R(t) \Rightarrow \psi(t))$
  - Bei Eindeutigkeit können Klammern weggelassen werden
- Beispiel:
  - $(\forall s) (s.A \leq u.B \vee (\exists u)(R(u) \wedge u.C > t.D))$
  - $t$  ist **frei**
  - $s$  ist **gebunden**
  - $u$  ist **frei beim ersten Auftreten und dann gebunden**



# Ausdruck (Anfrage)

- Ein Ausdruck des Tupelkalküls hat die Form
$$\{t \mid \psi(t)\}$$
- In Formel  $\psi$  ist  $t$  die einzige freie Variable



# Semantik

Bedeutung, die einem korrekt gebildeten Ausdruck durch eine Interpretation zugeordnet wird:

Syntax  $\xrightarrow{\text{Interpretation}}$  Semantik

Tupelvariablen  $\longrightarrow$  konkrete Tupel

Formeln  $\longrightarrow$  true, false

Ausdrücke  $\longrightarrow$  Relationen



# Belegung von Variablen

- Gegeben:
  - eine Tupelvariable  $t$  mit  $\text{Schema}(t) = (D_1, D_2, \dots)$
  - eine Formel  $\psi(t)$ , in der  $t$  frei vorkommt
  - ein beliebiges konkretes Tupel  $r$  (d.h. mit Werten).  
Es muß nicht zu einer Relation der Datenbank gehören
- Bei der Belegung wird jedes freie Vorkommen von  $t$  durch  $r$  ersetzt. Insbesondere wird  $t.A$  durch den Attributwert von  $r.A$  ersetzt.
- Man schreibt:  $\psi(r \mid t)$



# Beispiel

Gegeben sei folgendes Relationenschema:

Städte (SName: String, SEinw: Integer, Land: String)

Länder (LName: String, LEinw: Integer, Partei\*: String)

\* bei Koalitionsregierungen: jeweils eigenes Tupel pro Partei

- 
- $\psi(t) = (t.Land=Bayern \wedge t.SEinw \geq 500.000)$   
mit  $Schema(t) = Schema(Städte)$ 
    - $r_1 = (Passau, 49.800, Bayern):$   
 $\psi(r_1 | t) = (Bayern = Bayern \wedge 49.800 \geq 500.000)$
    - $r_2 = (Bremen, 535.058, Bremen):$   
 $\psi(r_2 | t) = (Bremen = Bayern \wedge 535.058 \geq 500.000)$



# Interpretation von Formeln

Interpretation  $I(\psi)$  analog zu syntaktischem Aufbau

– Anm: Alle Variablen sind durch konkrete Tupel belegt

• Atome:

–  $R(r)$ :  $I(R(r)) = \mathbf{true} \Leftrightarrow r$  ist in  $R$  enthalten

–  $c_i \Theta c_j$ :  $I(c_i \Theta c_j) = \mathbf{true} \Leftrightarrow$  der Vergleich ist erfüllt

• Logische Operatoren:

–  $\neg\psi$ :  $I(\neg\psi) = \mathbf{true} \Leftrightarrow I(\psi) = \mathbf{false}$

–  $\psi_1 \wedge \psi_2$ :  $I(\psi_1 \wedge \psi_2) = \mathbf{true} \Leftrightarrow I(\psi_1) = \mathbf{true}$  und  $I(\psi_2) = \mathbf{true}$

–  $\psi_1 \vee \psi_2$ :  $I(\psi_1 \vee \psi_2) = \mathbf{true} \Leftrightarrow I(\psi_1) = \mathbf{true}$  oder  $I(\psi_2) = \mathbf{true}$



# Beispiele

- Atome:
  - $I(\text{Städte}(\text{Passau}, 49.800, \text{Bayern})) = \text{true}$
  - $I(49.800 \geq 500.000) = \text{false}$
- Logische Operatoren:
  - $I(\neg 49.800 \geq 500.000) = \text{true}$
  - $I(\text{Städte}(\text{Passau}, 49.800, \text{Bayern}) \vee 49.800 \geq 500.000) = \text{true}$
  - $I(\text{Städte}(\text{Passau}, 49.800, \text{Bayern}) \wedge 49.800 \geq 500.000) = \text{false}$



# Interpretation von Quantoren

- Interpretation  $I((\exists s)(\psi))$  bzw.  $I((\forall s)(\psi))$ :
  - In  $\psi$  darf **nur**  $s$  als freie Variable auftreten.
  - $I((\exists s)(\psi)) = \mathbf{true} \Leftrightarrow$  ein Tupel  $r \in D_1 \times D_2 \times \dots$  existiert, daß bei Belegung der Variablen  $s$  die Formel  $\psi$  gilt:
$$I(\psi(r \mid s)) = \mathbf{true}$$
  - $I((\forall s)(\psi)) = \mathbf{true} \Leftrightarrow$  für alle Tupel  $r \in D_1 \times D_2 \times \dots$  gilt die Formel  $\psi$ .
- Beispiele:
  - $I((\exists s)(\text{Städte}(s) \wedge s.\text{Land} = \text{Bayern})) = \mathbf{true}$
  - $I((\forall s)(s.\text{Name} = \text{Passau})) = \mathbf{false}$





# Interpretation von Ausdrücken

- Interpretation von Ausdruck  $I(\{t|\psi(t)\})$  stützt sich
  - auf Belegung von Variablen
  - und Interpretation von Formeln
- Gegeben:
  - $E = \{t \mid \psi(t)\}$
  - $t$  die einzige freie Variable in  $\psi(t)$
  - $\text{Schema}(t) = D_1 \times D_2 \times \dots$
- Dann ist der Wert von  $E$  die Menge aller\* (denkbaren) Tupel  $r \in D_1 \times D_2 \times \dots$  für die gilt:

$$I(\psi(r \mid t)) = \mathbf{true}$$

---

\*Grundmenge sind hier **nicht** nur die gespeicherten Tupel aus der DB



# Beispiel-Anfragen

Gegeben sei folgendes Relationenschema:

Städte (SName: String, SEinw: Integer, Land: String)  
Länder (LName: String, LEinw: Integer, Partei\*: String)

\* bei Koalitionsregierungen: jeweils eigenes Tupel pro Partei

- Finde alle Großstädte (SName, SEinw, Land) in Bayern:

$\text{Schema}(t) = \text{Schema}(\text{Städte})$

$\{t \mid \text{Städte}(t) \wedge t.\text{Land} = \text{Bayern} \wedge t.\text{SEinw} \geq 500.000\}$

- In welchem Land liegt Passau?

$\text{Schema}(t) = (\text{Land}:\text{String})$

$\{t \mid (\exists u \in \text{Städte})(u.\text{Sname} = \text{Passau} \wedge u.\text{Land} = t.\text{Land})\}$

- Finde alle Städte in CDU-regierten Ländern:

$\text{Schema}(t) = \text{Schema}(\text{Städte})$

$\{t \mid \text{Städte}(t) \wedge (\exists u \in \text{Länder})(u.\text{Lname} = t.\text{Land} \wedge u.\text{Partei} = \text{CDU})\}$



# Beispiel-Anfragen

Gegeben sei folgendes Relationenschema:

Städte (SName: String, SEinw: Integer, Land: String)  
Länder (LName: String, LEinw: Integer, Partei\*: String)

\* bei Koalitionsregierungen: jeweils eigenes Tupel pro Partei

- Welche Länder werden von der SPD allein regiert?

$\text{Schema}(t) = \text{Schema}(\text{Länder})$

$\{t \mid \text{Länder}(t) \wedge (\forall u \in \text{Länder})(u.\text{LName} = t.\text{LName} \Rightarrow u.\text{Partei} = \text{SPD})\}$

- Gleichbedeutend mit:

$\text{Schema}(t) = \text{Schema}(\text{Länder})$

$\{t \mid \text{Länder}(t) \wedge (\forall u \in \text{Länder}) \neg (u.\text{LName} = t.\text{LName} \wedge u.\text{Partei} \neq \text{SPD})\}$



# Beispiel Bundesländer

Länder:	LName	LEinw	Partei
	Baden-Württemberg	10.745.000	Grüne
	Baden-Württemberg	10.745.000	SPD
	Bayern	12.510.000	CSU
	Bayern	12.510.000	FDP
	Berlin	3.443.000	SPD
	Berlin	3.443.000	Linke
	Brandenburg	2.512.000	SPD
	Brandenburg	2.512.000	Linke
	Bremen	662.000	SPD
	Bremen	662.000	Grüne
	Hamburg	1.774.000	SPD
	...	...	...



# Sichere Ausdrücke

- Mit den bisherigen Definitionen ist es möglich, unendliche Relationen zu beschreiben:
  - $\text{Schema}(t) = \{\text{String}, \text{String}\}$
  - $\{t \mid t.1 = t.2\}$
  - Ergebnis:  $\{(A,A), (B,B), \dots, (AA,AA), (AB,AB), \dots\}$
- Probleme:
  - Ergebnis kann nicht gespeichert werden
  - Ergebnis kann nicht in endlicher Zeit berechnet werden
- Definition:

Ein Ausdruck heißt *sicher*, wenn jede Tupelvariable nur Werte einer gespeicherten Relation annehmen kann, also positiv in einem Atom  $R(t)$  vorkommt.



# Der Bereichskalkül

- Tupelkalkül:           Tupelvariablen  $t$  (ganze Tupel)
- Bereichskalkül:       Bereichsvariablen  $x_1:D_1, x_2:D_2, \dots$   
für einzelne Attribute  
(Bereich=Wertebereich=Domäne)

Ein **Ausdruck** hat die Form:

$$\{x_1, x_2, \dots \mid \psi(x_1, x_2, \dots)\}$$

**Atome** haben die Form:

- $R_1(x_1, x_2, \dots)$ : Tupel  $(x_1, x_2, \dots)$  tritt in Relation  $R_1$  auf
- $x \Theta y$ :  $x, y$  Bereichsvariablen bzw. Konstanten  
 $\Theta \in \{=, <, \leq, >, \geq, \neq\}$

**Formeln** analog zum Tupelkalkül



# Beispiel-Anfragen

Städte (SName: String, SEinw: Integer, Land: String)

Länder (LName: String, LEinw: Integer, Partei\*: String)

\*bei Koalitionsregierungen: jeweils eigenes Tupel pro Partei

- In welchem Land liegt Passau?

$\{x_3 \mid \exists x_1, x_2: (\text{Städte}(x_1, x_2, x_3) \wedge x_1 = \text{Passau}) \}$

oder auch

$\{x_3 \mid \exists x_2: (\text{Städte}(\text{Passau}, x_2, x_3) ) \}$

- Finde alle Städte in CDU-regierten Ländern:

$\{x_1 \mid \exists x_2, x_3, y_2: (\text{Städte}(x_1, x_2, x_3) \wedge \text{Länder}(x_3, y_2, \text{CDU})) \}$

- Welche Länder werden von der SPD allein regiert?

$\{x_1 \mid \exists x_2: (\text{Länder}(x_1, x_2, \text{SPD}) \wedge \neg \exists y_3: (\text{Länder}(x_1, x_2, y_3) \wedge y_3 \neq \text{SPD})) \}$



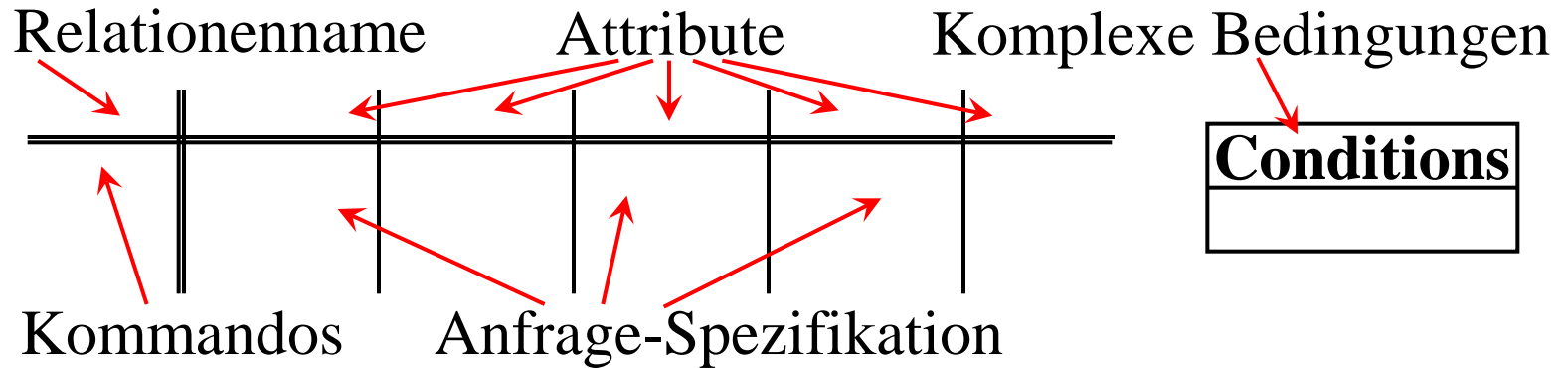
# Query By Example (QBE)

- Beruht auf dem Bereichskalkül
- Ausdrücke nicht wie in SQL als Text
- Dem Benutzer wird am Bildschirm ein Tabellen-Gerüst angeboten, das mit Spezial-Editor bearbeitet werden kann
- Nach Eintrag von Werten in das Tabellengerüst (Anfrage) füllt das System die Tabelle
- Zielgruppe: Gelegentliche Benutzer





# Query By Example (QBE)



Sprachelemente:

- Kommandos, z.B. **P.** (print), **I.** (insert), **D.** (delete) ...
- Bereichsvariablen (beginnen mit ‘\_’): **\_x**, **\_y**
- Konstanten (Huber, Milch)
- Vergleichsoperatoren und arithmetische Operatoren
- Condition-Box: Zusätzlicher Kasten zum Eintragen einer Liste von Bedingungen (**AND**, **OR**, kein **NOT**)



# Beispiel-Dialog

- Beginn: leeres Tabellengerüst



- *Benutzer* gibt interessierende Relation und **P.** ein  
**Kunde P.**



- *System* trägt Attributnamen der Relation ein  
Kunde    KName    KAdr    Kto

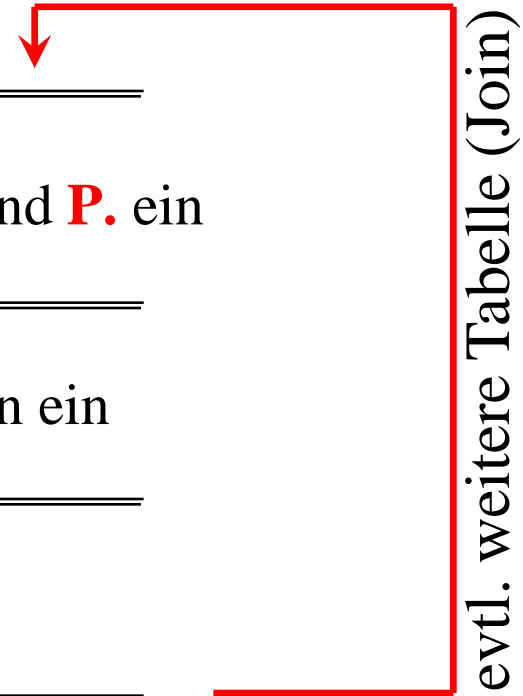
Kunde	KName	KAdr	Kto

- *Benutzer* stellt Anfrage

Kunde	KName	KAdr	Kto
	<b>P.</b>	<b>P.</b>	<b>&lt;0</b>

- *System* füllt Tabelle mit Ergebnis-Werten

Kunde	KName	KAdr
	Huber	Innsbruck
	Maier	München

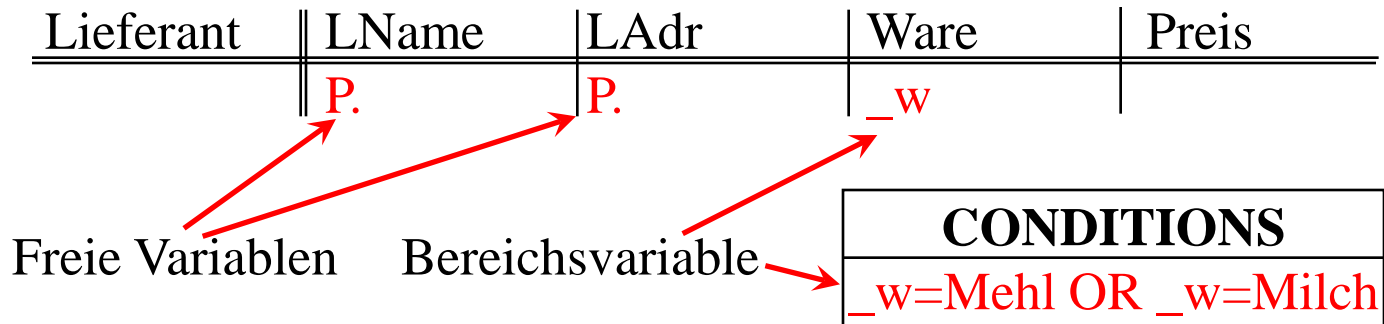


evtl. weitere Tabelle (Join)



# Anfragen mit Bedingungen

- Welche Lieferanten liefern Mehl oder Milch?



- Bedeutung:  
 $\{x_1, x_2 \mid \exists w, x_4: \text{Lieferant}(x_1, x_2, w, x_4) \wedge (w = \text{Mehl} \vee w = \text{Milch})\}$
- Kommando **P.** für print bzw. auch für die Projektion



# Anfragen mit Bedingungen

- Welche Lieferanten liefern Brie und Perrier, wobei Gesamtpreis 7,00 € nicht übersteigt?

Lieferant	LName	LAdr	Ware	Preis
	P. _L		Brie	_y
	_L		Perrier	_z

<b>CONDITIONS</b>
$\_y + \_z \leq 7.00$

- Bedeutung:  
 $\{l \mid \exists x_1, x_2, y, z: \text{Lieferant}(l, x_1, \text{Brie}, y) \wedge \text{Lieferant}(l, x_2, \text{Perrier}, z) \wedge y + z \leq 7.00\}$



# Join-Anfragen

- Welcher Lieferant liefert etwas das Huber bestellt hat?

Lieferant	LName	LAdr	Ware	Preis
	P.		_w	

Auftrag	KName	Ware	Menge
	Huber	_w	

- Bedeutung:

$$\{x_1 \mid \exists x_2, w, x_4, y_3: \text{Lieferant}(x_1, x_2, w, x_4) \wedge \text{Auftrag}(\text{Huber}, w, y_3)\}$$

- Beachte:

Automatische Duplikat-Elimination in QBE



# Join-Anfragen

Meist ist für Ergebnis neues Tabellengerüst nötig:

- Beispiel: Bestellungen mit Kontostand des Kunden
- Falsch (leider nicht möglich):

<del>Kunde</del>	<del>KName</del>	<del>KAdr</del>	<del>Kto</del>
	<del>P. _n</del>		<del>P.</del>
<del>Auftrag</del>	<del>KName</del>	<del>Ware</del>	<del>Menge</del>
	<del>_n</del>	<del>P.</del>	<del>P.</del>

- Richtig:

Kunde	KName	KAdr	Kto
	_n		_k
Auftrag	KName	Ware	Menge
	_n	_w	_m

Bestellung	Name	Was	Wieviel	Kontostand
P.	P. _n	P. _w	P. _m	P. _k

Abkürzung!





# Anfragen mit Ungleichung

- Wer liefert Milch zu Preis zw. 0,50 € und 0,60 €?
- Variante mit zwei Zeilen:

Lieferant	LName	LAdr	Ware	Preis
P.	<u>L</u>		Milch	$\geq 0.5$
	<u>L</u>		Milch	$\leq 0.6$

- Variante mit Condition-Box

Lieferant	LName	LAdr	Ware	Preis
P.			Milch	<u>p</u>

CONDITIONS
$\underline{p} \geq 0.5 \text{ AND } \underline{p} \leq 0.6$



# Anfragen mit Negation

- Finde für jede Ware den billigsten Lieferanten

Lieferant	LName	LAdr	Ware	Preis
P.			$\_w$	$\_p$
$\neg$			$\_w$	$< \_p$

- Das Symbol  $\neg$  in der ersten Spalte bedeutet:  
Es gibt kein solches Tupel

- Bedeutung:

$$\{x_1, x_2, w, p \mid \neg \exists y_1, y_2, y_3: \text{Lieferant}(x_1, x_2, w, p) \wedge \text{Lieferant}(y_1, y_2, w, y_3) \wedge y_3 < p\}$$





# Einfügen

- Einfügen von einzelnen Tupeln
  - Kommando **I.** für INSERT

Kunde	KName	KAdr	Kto
<b>I.</b>	<b>Schulz</b>	<b>Wien</b>	<b>0</b>

- Einfügen von Tupeln aus einem Anfrageergebnis
  - Beispiel: Alle Lieferanten in Kundentabelle übernehmen

Kunde	KName	KAdr	Kto
<b>I.</b>	<b>_n</b>	<b>_a</b>	<b>0</b>

Lieferant	LName	LAdr	Ware	Preis
	<b>_n</b>	<b>_a</b>		



# Löschen und Ändern

- Löschen aller Kunden mit negativem Kontostand

Kunde	KName	KAdr	Kto
<b>D.</b>			<b>&lt; 0</b>

- Ändern eines Tupels (**U.** für UPDATE)

Kunde	KName	KAdr	Kto
	<b>Schulz</b>	<b>Wien</b>	<b>U. 100</b>

- oder auch:

Kunde	KName	KAdr	Kto
	<b>Meier</b>	<b>_a</b>	<b>_k</b>
<b>U.</b>	<b>Meier</b>	<b>_a</b>	<b>_k - 110</b>

- oder auch mit Condition-Box



# Vergleich

QBE	Bereichskalkül
Konstanten	Konstanten
Bereichsvariablen	Bereichsvariablen
leere Spalten	paarweise verschiedene Bereichsvariablen, $\exists$ -quantifiziert
Spalten mit <b>P.</b>	freie Variablen
Spalten ohne <b>P.</b>	$\exists$ -quantifizierte Variablen

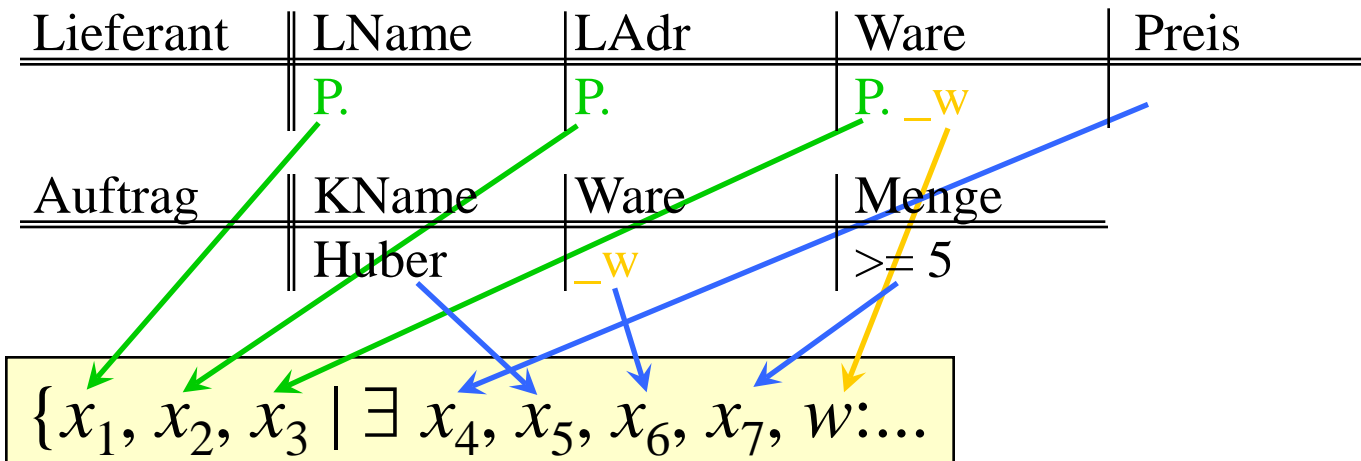
Anmerkung: QBE ist relational vollständig, jedoch ist für manche Anfragen der relationalen Algebra eine Folge von QBE-Anfragen nötig



# Umsetzung einer QBE-Anfrage

(ohne Negation)

- Erzeuge für alle Attribute  $A_i$  aller vorkommenden Tabellen-Zeilen der Anfrage eine Bereichsvariable  $x_i$
- Steht bei Attribut  $A_i$  das Kommando **P.** dann schreibe  $x_i$  zu den freien Variablen ( $\{\dots x_i, \dots \mid \dots\}$ ), sonst binde  $x_i$  mit einem  $\exists$ -Quantor ( $\{\dots \mid \exists \dots, x_i, \dots\}$ )
- Binde alle Variablen der Anfrage mit einem  $\exists$ -Quantor





# Umsetzung einer QBE-Anfrage

<b>Lieferant</b>	LName	LAdr	Ware	Preis
	P.	P.	P. _w	
<b>Auftrag</b>	KName	Ware	Menge	
	<b>Huber</b>	_w	<b>&gt;= 5</b>	

- Füge für jede vorkommende Relation  $R$  ein Atom der Form  $R(x_i, x_{i+1}, \dots)$  mit  $\wedge$  an die Formel  $\Psi$  an

$\{x_1, x_2, x_3 \mid \exists x_4, x_5, x_6, x_7, w: \text{Lieferant}(x_1, x_2, x_3, x_4) \wedge \text{Auftrag}(x_5, x_6, x_7) \dots$

- Steht bei  $A_i$  ein Zusatz der Form **Const** bzw.  $\leq$  **Const** etc., dann hänge  $x_i = \text{Const}$  bzw.  $x_i \leq \text{Const}$  mit  $\wedge$  an Formel.

$\{x_1, x_2, x_3 \mid \exists x_4, x_5, x_6, x_7, w: \text{Lieferant}(x_1, x_2, x_3, x_4) \wedge \text{Auftrag}(x_5, x_6, x_7) \wedge x_5 = \text{Huber} \wedge x_7 \geq 5$



# Umsetzung einer QBE-Anfrage

Lieferant	LName	LAdr	Ware	Preis
	P.	P.	P. $w$	
Auftrag	KName	Ware	Menge	
	Huber	$w$	$\geq 5$	

- Gleiches Vorgehen bei Zusätzen der Form  $\_Variable$  bzw.  $\leq \_Variable$  usw:

$$\{x_1, x_2, x_3 \mid \exists x_4, x_5, x_6, x_7, w: \text{Lieferant}(x_1, x_2, x_3, x_4) \wedge \text{Auftrag}(x_5, x_6, x_7) \wedge x_5 = \text{Huber} \wedge x_7 \geq 5 \wedge w = x_3 \wedge w = x_6\}$$

- Ggf. wird der Inhalt der Condition-Box mit  $\wedge$  angehängt.
- Meist lässt sich der Term noch vereinfachen:

$$\{x_1, x_2, w \mid \exists x_4, x_5, x_7: \text{Lieferant}(x_1, x_2, w, x_4) \wedge \text{Auftrag}(\text{Huber}, w, x_7) \wedge x_7 \geq 5\}$$



# Quantoren und Subqueries in SQL

- Quantoren sind Konzept des Relationenkalküls
- In relationaler Algebra nicht vorhanden
- Können zwar simuliert werden:
  - Existenzquantor implizit durch Join und Projektion:  
$$\{x \in R \mid \exists y \in S: \dots\} \equiv \pi_{R.*} (\sigma\dots (R \times S) )$$
  - Allquantor mit Hilfe des Quotienten  
$$\{x \in R \mid \forall y \in S: \dots\} \equiv (\sigma\dots (R \times S)) \div S$$
- Häufig Formulierung mit Quantoren natürlicher
- SQL: Quantifizierter Ausdruck in einer **Subquery**



# Quantoren und Subqueries in SQL

- Beispiel für eine Subquery  
**select \* from Kunde where exists (select...from...where...)**  
**Subquery**
- In Where-Klausel der Subquery auch Zugriff auf Relationen/Attribute der Hauptquery
- Eindeutigkeit ggf. durch Aliasnamen für Relationen (wie bei Self-Join):  
**select \***  
**from kunde k1**  
**where exists ( select \***  
**from Kunde k2**  
**where k1.Adr=k2.Adr and...**  
**)**





# Existenz-Quantor

- Realisiert mit dem Schlüsselwort **exists**
- Der  $\exists$ -quantifizierte Ausdruck wird in einer **Subquery** notiert.
- Term **true** gdw. Ergebnis der Subquery **nicht leer**
- Beispiel:  
KAdr der Kunden, zu denen ein Auftrag existiert:

```
select KAdr from Kunde k
where exists
  ( select * from Auftrag a
    where a.KName = k.KName
  )
```

Äquivalent  
mit Join ??



# Allquantor

- Keine direkte Unterstützung in SQL
- Aber leicht ausdrückbar durch die Äquivalenz:

$$\forall x: \psi(x) \Leftrightarrow \neg \exists x: \neg \psi(x)$$

- Also Notation in SQL:  
...where **not exists** (select...from...**where not**...)
- Beispiel:  
Die Länder, die von der SPD allein regiert werden  
**select \* from** Länder L1  
**where not exists**  
( **select \* from** Länder L2  
**where** L1.LName=L2.LName and **not** L2.Partei='SPD'  
)



# Direkte Subquery

- An jeder Stelle in der **select**- und **where**-Klausel, an der ein konstanter Wert stehen kann, kann auch eine Subquery (**select...from...where...**) stehen.
- Einschränkungen:
  - Subquery darf nur ein Attribut ermitteln (Projektion)
  - Subquery darf nur ein Tupel ermitteln (Selektion)
- Beispiel: Dollarkurs aus Kurstabelle

```
select  Preis,
        Preis * ( select Kurs from Devisen
                  where DName = 'US$' ) as USPreis
from Waren where ...
```
- Oft schwierig, Eindeutigkeit zu gewährleisten...



# Weitere Quantoren

- Quantoren bei Standard-Vergleichen in WHERE

- Formen:

- $A_i \Theta$  **all** (select...from...where...)  $\forall$ -Quantor
  - $A_i \Theta$  **some** (select...from...where...)
  - $A_i \Theta$  **any** (select...from...where...)
- }  $\exists$ -Quantor

---

Vergleichsoperatoren  $\Theta \in \{ =, <, <=, >, >=, <> \}$

- Bedeutung:

- $A_i \Theta$  **all** (Subquery)  $\equiv \{ \dots \mid \forall t \in \text{Subquery}: A_i \Theta t \}$
- ist größer als **alle** Werte, die sich aus Subquery ergeben

- Einschränkung bezüglich Subquery:

- Darf nur ein Ergebnis-Attribut ermitteln
  - Aber mehrere Tupel sind erlaubt
- } Menge  
nicht Relation



# Beispiel

- Ermittle den Kunden mit dem höchsten Kontostand

```
select KName, KAdr  
from Kunde  
where Kto >= all      ( select Kto  
                        from Kunde  
                        )
```

- Äquivalent zu folgendem Ausdruck mit EXISTS:

```
select KName, KAdr  
from Kunde k1  
where not exists    ( select *  
                      from Kunde k2  
                      where not k1.Kto >= k2.Kto  
                      )
```



# Subquery mit IN

- Nach dem Ausdruck  $A_i$  **[not] in ...** kann stehen:
  - Explizite Aufzählung von Werten:  $A_i$  **in** (2,3,5,7,11,13)
  - Eine Subquery:

$A_i$  **in** (**select** wert **from** Primzahlen **where** wert $\leq$ 13)

Auswertung:

- Erst Subquery auswerten
- In explizite Form (2,3,5,7,11,13) umschreiben
- Dann einsetzen
- Zuletzt Hauptquery auswerten



# Beispiele

- Gegeben:
  - MagicNumbers (Name: String, Wert: Int)
  - Primzahlen (Zahl: Int)
- Anfrage: Alle MagicNumbers, die prim sind  
**select \* from MagicNumbers where Wert in**  
**( select Zahl from Primzahlen )**
- ist äquivalent zu folgender Anfrage mit EXISTS:  
**select \* from MagicNumbers where exists**  
**( select \* from Primzahlen where Wert = Zahl )**
- und zu folgender Anfrage mit SOME/ANY/ALL:  
**select \* from MagicNumbers where**  
**Wert = some (select Zahl from Primzahlen)**



# Beispiele

- Gegeben:
  - MagicNumbers (Name: String, Wert: Int)
  - Primzahlen (Zahl: Int)
- Anfrage: Alle MagicNumbers, die **nicht** prim sind  
**select \* from MagicNumbers where  
Wert not in (select Zahl from Primzahlen)**
- ist äquivalent zu folgender Anfrage mit EXISTS:  
**select \* from MagicNumbers where  
not exists (select \* from Primzahlen where Wert = Zahl)**
- und zu folgender Anfrage mit SOME/ANY/ALL:  
**select \* from MagicNumbers where  
Wert <> all (select Zahl from Primzahlen)**  
*bzw.:* **select \* from MagicNumbers where  
not (Wert = any (select Zahl from Primzahlen))**





# Typische Form der Subquery

- Bei **exists** bzw. **not exists** ist für die Haupt-Query nur relevant, ob das Ergebnis die leere Menge ist oder nicht.
  - Deshalb muss keine Projektion durchgeführt werden:  
**select ... from ... where exists (select \* from ...)**
- Bei **some**, **any**, **all** und **in** ist das Ergebnis der Subquery eine *Menge von Werten* (d.h. ein Attribut, mehrere Tupel), die in die Hauptquery eingesetzt werden.
  - Deshalb muss in der Subquery eine Projektion auf *genau ein* Attribut durchgeführt werden:  
**select ... from ... where A <= all (select B from ...)**
- Das Ergebnis der direkten Subquery ist *genau ein* Wert.
  - Projektion auf ein Attribut, Selektion eines Tupels:  
**... where A <= (select B from ... where *Schlüssel*=...)**