Ludwig Maximilians Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Skript zur Vorlesung

Datenbanksysteme I

Wintersemester 2013/2014

Kapitel 4: Relationen-Kalkül

Vorlesung: Prof. Dr. Christian Böhm
Übungen: Sebastian Goebl

http://www.dbs.ifi.lmu.de/Lehre/DBS

Begriff

Kal|kül *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 ...

Kal|kül *der; -s, -e* <zu [1]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 $\{x \mid x \in I\!N \wedge x^3 > 0 \wedge x^3 < 1000\}$
- Anwendung solcher Formeln für DB-Anfragen
  - Bezugnahme auf DB-Relationen im Bedingungsteil: $(x_1, y_1, z_1) \in$ Mitarbeiter, $t_1 \in$ Abteilungen
  - Terme werden gebildet aus Variablen, Konstanten usw.
  - Atomare Formeln aus Prädikaten der Datentypen: $=, <, >, \leq$, 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:
  $\{t \mid$ Städte(t) $\wedge\ t$[Land] = Bayern $\wedge\ t$[SEinw] $\geq 500.000\}$
  Hinweis: Städte(t) gleichbedeutend mit t $\in$ 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 ψ erfüllen (wahr machen)
- Ein Kalkül besteht immer aus
  - Syntax: Wie sind Ausdrücke aufgebaut?
  - Semantik: Was bedeuten die Ausdrücke?

Tupelvariablen

- Tupelvariablen haben ein definiertes Schema:
  - Schema(t) = (A_1: D_1, A_2: D_2, ...)
  - Schema(t) = R_1 (t hat dasselbe Schema wie Relation)
- Für Zugriff auf die Komponenten
  - $t[A]$ oder $t.A$ für einen Attributnamen $A \in$ Schema(t)
  - oder auch $t[1]$, $t[2]$ usw.
- Tupelvariable kann in einer Formel ψ 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 Θ zu ...
  - $t.A\ \Theta\ c$ — t ist Tupelvariable und c eine passende Konstante

---

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

Formeln

Der Aufbau von Formeln ψ ist rekursiv definiert:

- **Atome**: Jedes Atom ist eine Formel
  Alle vorkommenden Variablen sind frei
- **Verknüpfungen**: Sind ψ_1 und ψ_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 ψ eine Formel, in der *t* als freie Variable auftritt, sind auch Formeln...
  - $(\exists t)(\psi)$ es gibt ein *t*, für das ψ
  - $(\forall t)(\psi)$ für alle *t* gilt ψ

  die Variable *t* wird gebunden.

Formeln

- Gebräuchliche vereinfachende Schreibweisen:
  - $\psi_1 \Rightarrow \psi_2$ für $(\neg\, \psi_1) \vee \psi_2$ (Implikation)
  - $\exists t_1,...,t_k\colon \psi(t_1,...t_k)$ für $(\exists t_1)\,(...((\exists t_k)\,(\psi(t_1,...t_k)))...)$
  - $(\exists t \in R)\,(\psi(t))$ 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:
  - $(\forall s)\,(s.\mathrm{A} \leq u.\mathrm{B} \vee (\exists u)(\mathrm{R}(u) \wedge u.\mathrm{C} > t.\mathrm{D}))$
  - t ist frei
  - s ist gebunden
  - u ist frei beim ersten Auftreten und dann gebunden

Ausdruck (Anfrage)

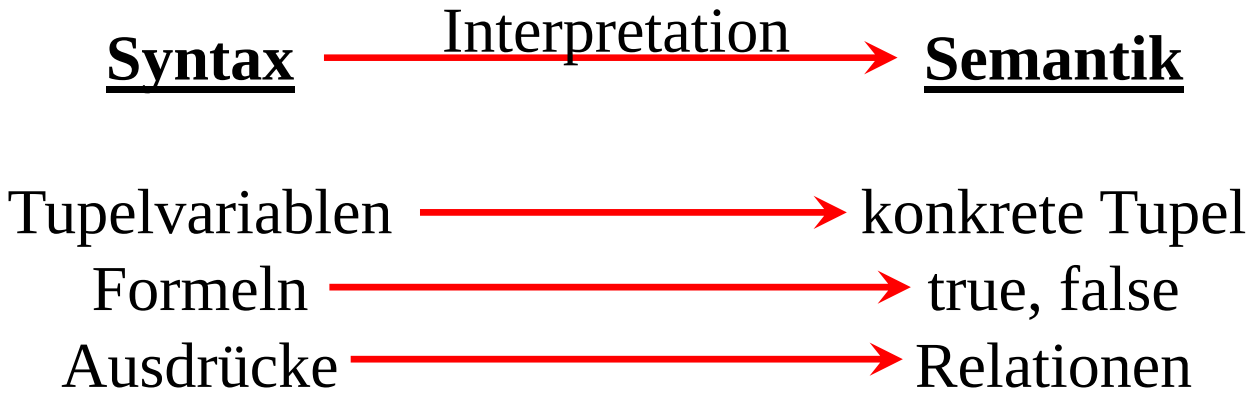
- Ein Ausdruck des Tupelkalküls hat die Form

$$\{t \mid \psi(t)\}$$

- In Formel ψ ist t die einzige freie Variable

Semantik

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



Belegung von Variablen

- Gegeben:
  - eine Tupelvariable t mit Schema(t) = (D_1, D_2, ...)
  - 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

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

Interpretation von Formeln

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

- Anm: Alle Variablen sind durch konkrete Tupel belegt

- Atome:
  - $R(r)$: $I(R(r))$ = **true** $\Leftrightarrow$ r ist in R enthalten
  - $c_i \Theta c_j$: $I(c_i \Theta c_j)$ = **true** $\Leftrightarrow$ der Vergleich ist erfüllt
- Logische Operatoren:
  - $\neg\psi$: $I(\neg\psi)$ = **true** $\Leftrightarrow$ $I(\psi)$ = **false**
  - $\psi_1 \wedge \psi_2$: $I(\psi_1 \wedge \psi_2)$=**true** $\Leftrightarrow$ $I(\psi_1)$=**true** und $I(\psi_2)$=**true**
  - $\psi_1 \vee \psi_2$: $I(\psi_1 \vee \psi_2)$=**true** $\Leftrightarrow$ $I(\psi_1)$=**true** oder $I(\psi_2)$=**true**

Beispiele

- Atome:
  - I(Städte (Passau, 49.800, Bayern)) = **true**
  - I(49.800 ≥ 500.000) = **false**

- Logische Operatoren:
  - $I(\neg$ 49.800 ≥ 500.000) = **true**
  - I(Städte (Passau, 49.800, Bayern) $\vee$ $\vee$ 49.800 ≥ 500.000) = **true**
  - I(Städte (Passau, 49.800, Bayern) $\wedge$ $\wedge$ 49.800 ≥ 500.000) = **false**

Interpretation von Quantoren

- Interpretation $I((\exists s)(\psi))$ bzw. $I((\forall s)(\psi))$:
  - In ψ darf nur s als freie Variable auftreten.
  - $I((\exists s)(\psi))$ = **true** $\Leftrightarrow$ ein Tupel $r \in D_1 \times D_2 \times ...$ existiert, daß bei Belegung der Variablen s die Formel ψ gilt:

    $$I(\psi(r \mid s)\ ) = \textbf{true}$$

  - $I((\forall s)(\psi))$ = **true** $\Leftrightarrow$ für alle Tupel $r \in D_1 \times D_2 \times ...$ gilt die Formel ψ.
- Beispiele:
  - $I((\exists s)(\text{Städte}(s) \wedge s.\text{Land} = \text{Bayern}))$ = **true**
  - $I((\forall s)(s.\text{Name} = \text{Passau}))$ = **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 $\psi(t)$
  - Schema(t) = $D_1 \times D_2 \times ...$
- Dann ist der Wert von E die Menge aller* (denkbaren) Tupel $r \in D_1 \times D_2 \times ...$ für die gilt:

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

*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:

  Schema(t) = Schema(Städte)

  $\{t \mid \text{Städte}(t) \wedge t.\text{Land} = \text{Bayern} \wedge t.\text{SEinw} \geq 500.000\}$
- In welchem Land liegt Passau?

  Schema(t) = (Land: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:

  Schema(t) = Schema(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?

  Schema(t) = Schema(Länder)

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

- Gleichbedeutend mit:

  Schema(t) = Schema(Länder)

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

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:
  - Schema(t) = {String, String}
  - $\{t \mid t.1 = t.2\}$
  - Ergebnis: {(A,A),(B,B),...,(AA,AA),(AB,AB),...}
- 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 $x_1{:}D_1$, $x_2{:}D_2$, ... für einzelne Attribute
  (Bereich=Wertebereich=Domäne)

Ein **Ausdruck** hat die Form:

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

Atome haben die Form:

- $R_1\ (x_1, x_2, \ldots)$: Tupel $(x_1, x_2, \ldots)$ 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?

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

  oder auch

  $\{x_3 \mid \exists\, x_2\colon (\text{Städte}(\text{Passau}, x_2, x_3)\ )\ \}$
- Finde alle Städte in CDU-regierten Ländern:

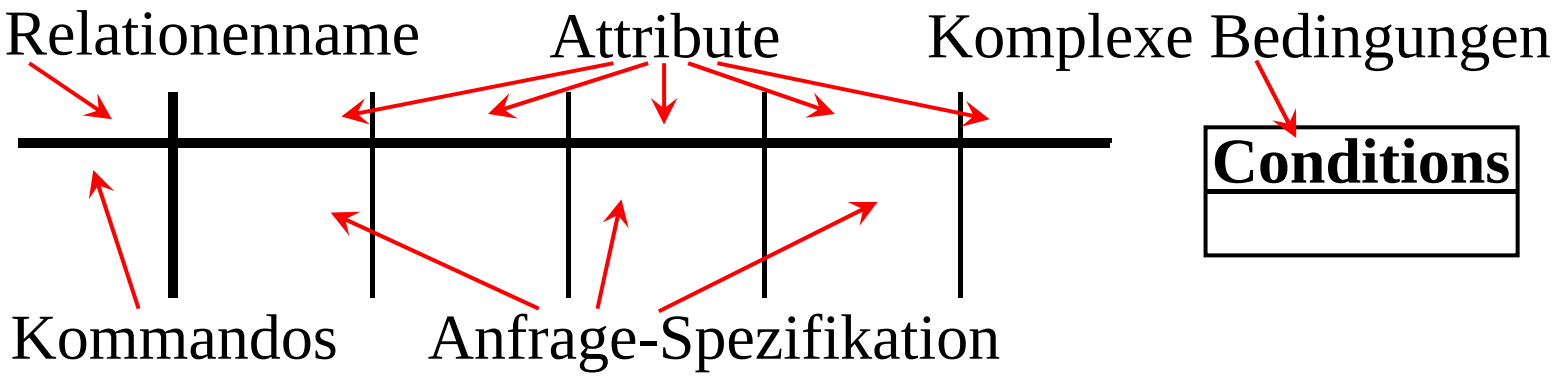
  $\{x_1 \mid \exists\, x_2\,, x_3\,, y_2\,\colon (\text{Städte}(x_1,x_2,x_3) \wedge \text{Länder}\ (x_3,y_2,\text{CDU}))\}$
- Welche Länder werden von der SPD allein regiert?

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

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 Attributsnamen der Relation ein

Kunde	KName	KAdr	Kto

- *Benutzer* stellt Anfrage

Kunde	KName	KAdr	Kto
	P.	**P.**	**<0**

- *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?

Lieferant	LName	LAdr	Ware	Preis
	P.	P.	_w	

Freie Variablen

Bereichsvariable

CONDITIONS
_w=Mehl OR _w=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

CONDITIONS
_y + _z <= 7.00

- Bedeutung:

$\{l \mid \exists\, x_1, x_2, y, z$: Lieferant $(l, x_1,$ Brie$, y) \wedge$
Lieferant $(l, x_2,$ 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:

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

Kunde	KName	KAdr	Kto
	P. _n		P.

Auftrag	KName	Ware	Menge
	_n	P.	P.

- 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.	_L		Milch	>= 0.5
	_L		Milch	<= 0.6

- Variante mit Condition-Box

Lieferant	LName	LAdr	Ware	Preis
P.			Milch	_p

CONDITIONS
_p >= 0.5 AND _p <= 0.6

Anfragen mit Negation

- Finde für jede Ware den billigsten Lieferanten

Lieferant	LName	LAdr	Ware	Preis
P.			_w	_p
¬			_w	< _p

- Das Symbol ¬ in der ersten Spalte bedeutet:
  Es gibt kein solches Tupel

- Bedeutung:

$\{x_1, x_2, w, p \mid \neg\exists\, y_1, y_2, y_3$: Lieferant $(x_1, x_2, w, p) \wedge$
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
I.	Schulz	Wien	0

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

Kunde	KName	KAdr	Kto
I.	_n	_a	0

Lieferant	LName	LAdr	Ware	Preis
	_n	_a		

Löschen und Ändern

- Löschen aller Kunden mit negativem Kontostand

Kunde	KName	KAdr	Kto
D.			< 0

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

Kunde	KName	KAdr	Kto
	Schulz	Wien	U. 100

- oder auch:

Kunde	KName	KAdr	Kto
	Meier	_a	_k
U.	Meier	_a	_k – 110

- oder auch mit Condition-Box

Vergleich

QBE	Bereichskalkül
Konstanten	Konstanten
Bereichsvariablen	Bereichsvariablen
leere Spalten	paarweise verschiedene Bereichsvariablen, ∃-quantifiziert
Spalten mit **P.**	freie Variablen
Spalten ohne **P.**	∃-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 ({... x_i, ... | ...}), sonst binde x_i mit einem ∃-Quantor ({... | ∃..., x_i, ...})
- Binde alle Variablen der Anfrage mit einem ∃-Quantor

Lieferant	LName	LAdr	Ware	Preis
	P.	P.	P. _w	

Auftrag	KName	Ware	Menge
	Huber	_w	>= 5

$\{x_1, x_2, x_3 \mid \exists\ x_4, x_5, x_6, x_7, w{:}\ldots$

Umsetzung einer QBE-Anfrage

Lieferant	LName	LAdr	Ware	Preis
	P.	P.	P. _w	

Auftrag	KName	Ware	Menge
	Huber	_w	>= 5

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

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

- Steht bei A_i ein Zusatz der Form **Const** bzw. **≤ Const** etc., dann hänge x_i = Const bzw. $x_i \leq$ 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	>= 5

- Gleiches Vorgehen bei Zusätzen der Form _Variable bzw. ≤ _Variable usw:

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

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

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

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:
    $\{x \in R \mid \exists y \in S: ...\} \equiv \pi_{R.*}(\sigma...(R \times S))$
  - Allquantor mit Hilfe des Quotienten
    $\{x \in R \mid \forall y \in S: ...\} \equiv (\sigma...(R \times S)) \div S$
- 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 ∃-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:

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

- 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 Θ all (select...from...where...) — ∀-Quantor
  - A_i Θ some (select...from...where...) — ∃-Quantor
  - A_i Θ any (select...from...where...) — ∃-Quantor

  Vergleichsoperatoren Θ ∈ { = , < , <= , > , >= , <>}
- Bedeutung:
  - A_i Θ all (Subquery) ≡ {...| ∀t ∈ Subquery: A_i Θ t}
  - ist größer als alle Werte, die sich aus Subquery ergeben
- Einschränkung bezüglich Subquery:
  - Darf nur ein Ergebnis-Attribut ermitteln — Menge nicht Relation
  - 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<=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=*…**)**