Vorlesung
Logik und Datenbanken

Nicole Schweikardt

Johann Wolfgang Goethe-Universität Frankfurt am Main

Sommersemester 2008

Konjunktive Anfragen (I)

2.1 Deskriptiver Ansatz: regelbasiert, graphisch und logikbasiert
2.2 Auswertungskomplexität
2.3 Algebraischer Ansatz: SPC-Algebra und SPJR-Algebra

Konjunktive Anfragen (I)

Regelbasierte Konjunktive Anfragen — Informell

Beispiel-Anfrage:

(4) Welche Kinos (Name & Adresse) spielen einen Film mit “George Clooney”?

Andere Formulierung:

Wenn es in *Filme* ein Tupel $\langle x_{Titel}, x_{Regie}, \text{“George Clooney”}\rangle$ und
in *Programm* ein Tupel $\langle x_{Kino}, x_{Titel}, x_{Zeit}\rangle$ und
in *Orte* ein Tupel $\langle x_{Kino}, x_{Adr}, x_{Tel}\rangle$ gibt,
dann nimm das Tupel $\langle x_{Kino}, x_{Adr}\rangle$ in die Antwort auf

Als regelbasierte konjunktive Anfrage:

$$\begin{aligned} Ans(x_{Kino}, x_{Adr}) \leftarrow\ & Filme(x_{Titel}, x_{Regie}, \text{“George Clooney”}), \\ & Programm(x_{Kino}, x_{Titel}, x_{Zeit}), \\ & Orte(x_{Kino}, x_{Adr}, x_{Tel}) \end{aligned}$$

Regelbasierte Konjunktive Anfragen — Präzise

Definition 2.1

- **var** sei eine abzählbar unendliche Menge von Variablen(symbolen), die disjunkt zu den Mengen **att**, **dom**, **relname** ist.
  (Einzelne Variablen bezeichnen wir i.d.R. mit x, y, x_1, x_2, ...)
- Ein Term ist ein Element aus $\mathbf{var} \cup \mathbf{dom}$.
- Ein freies Tupel der Stelligkeit k ist ein Element aus $(\mathbf{var} \cup \mathbf{dom})^k$.

Definition 2.2

Sei **R** ein Datenbankschema.
Eine regelbasierte konjunktive Anfrage über **R** ist ein Ausdruck der Form

$$Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$$

wobei $\ell \geqslant 0$, $R_1, \ldots, R_\ell \in \mathbf{R}$, $Ans \in \mathbf{relname} \backslash \mathbf{R}$ und $u, u_1, \ldots, u_\ell$ freie Tupel der Stelligkeiten $arity(Ans)$, $arity(R_1), \ldots, arity(R_\ell)$, so dass jede Variable, die in u vorkommt, auch in mindestens einem der Tupel $u_1, \ldots, u_\ell$ vorkommt.

Semantik regelbasierter konjunktiver Anfragen

Sei Q eine regelbasierte konjunktive Anfrage (über einem DB-Schema **R**) der Form

$$Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$$

- Mit $Var(Q)$ bezeichnen wir die Menge aller Variablen, die in Q vorkommen.
- Eine Belegung für Q ist eine Abbildung $\beta : Var(Q) \to \mathbf{dom}$.

  Wir setzen β auf natürliche Weise fort zu einer Abbildung von $Var(Q) \cup \mathbf{dom}$ nach **dom**, so dass $\beta|_{\mathbf{dom}} = \mathrm{id}$. Für ein freies Tupel $u = \langle e_1, .., e_k \rangle$ setzen wir $\beta(u) := \langle \beta(e_1), .., \beta(e_k) \rangle$.
- Der Anfrage Q ordnen wir die folgende Anfragefunktion $[\![Q]\!]$ zu:

$$[\![Q]\!](\mathbf{I}) := \left\{ \beta(u) \;:\; \begin{array}{l} \beta \text{ ist eine Belegung für } Q, \text{ so dass} \\ \beta(u_i) \in \mathbf{I}(R_i), \text{ f.a. } i \in \{1, .., \ell\} \end{array} \right\}$$

  für alle Datenbanken $\mathbf{I} \in inst(\mathbf{R})$.

Beispiele (1/2)

- Die Anfrage (6) Welche (je 2) Regisseure haben schon in Filmen gespielt, in denen der jeweils andere Regie geführt hat?

  lässt sich durch folgende regelbasierte konjunktive Anfrage ausdrücken:

$$\mathit{Antworten}(x, y) \leftarrow \mathit{Filme}(z_1, x, y), \mathit{Filme}(z_2, y, x)$$

- Die Anfrage (5) Läuft zur Zeit ein “Detlev Buck” Film?

  lässt sich durch folgende regelbasierte konjunktive Anfrage ausdrücken:

$$\mathit{Ans}() \leftarrow \mathit{Filme}(x, \text{“Detlev Buck”}, y),\ \mathit{Programm}(z, x, w)$$

Ans ist hier also ein Relations-Name der Stelligkeit 0.
Erinnern Sie sich an unsere Konvention, dass die Ausgabe “$\emptyset$” der Antwort “nein” entspricht, und die Ausgabe der Menge $\{\langle\rangle\}$, die aus dem “Tupel der Stelligkeit 0” besteht, der Antwort “ja” entspricht.

Beispiele (2/2)

Betrachte die Datenbank $\mathbf{I} := \{\mathbf{I}(R), \mathbf{I}(S)\}$ mit
$\mathbf{I}(R) := \{\langle a, a\rangle, \langle a, b\rangle, \langle b, c\rangle, \langle c, b\rangle\}$ und $\mathbf{I}(S) := \{\langle d, a, b\rangle, \langle a, c, e\rangle, \langle b, a, c\rangle\}$.

- Die Anfrage $Q_1 :=$

$$Ans_1(x_1, x_2, x_3) \leftarrow R(x_1, y), S(y, x_2, x_3)$$

liefert auf $\mathbf{I}$ das Ergebnis $[\![Q_1]\!](\mathbf{I}) = \{\langle a, c, e\rangle, \langle a, a, c\rangle, \langle c, a, c\rangle\}$.

- Die Anfrage $Q_2 :=$

$$Ans_2(x, y) \leftarrow R(x, z_1),\ S(z_1, a, z_2),\ R(y, z_2)$$

liefert auf $\mathbf{I}$ das Ergebnis $[\![Q_2]\!](\mathbf{I}) = \{\langle a, b\rangle, \langle c, b\rangle\}$.

Bezeichnungen

Oft sagen wir kurz Regel, um eine regelbasierte konjunktive Anfrage zu bezeichnen.

Sei Q eine Regel der Form $Ans(u) \leftarrow R_1(u_1), \ldots, R_\ell(u_\ell)$

- $Ans(u)$ wird als Kopf der Regel bezeichnet.
- $R_1(u_1), \ldots, R_\ell(u_\ell)$ wird als Rumpf der Regel bezeichnet.
- Die Relations-Namen aus **R** werden als extensionale Datenbankprädikate (kurz: edb-Prädikate) bezeichnet.
  Wir schreiben $edb(Q)$, um die Menge aller edb-Prädikate zu bezeichnen, die in Q vorkommen.
- Der Relations-Namen, der im Kopf von Q vorkommt, wird als intensionales Datenbankprädikat (kurz: idb-Prädikat) bezeichnet.
- Mit $adom(Q)$ bezeichnen wir die Menge aller Konstanten (also Elemente aus **dom**), die in Q vorkommen. ("*adom*" steht für "active domain")

Der “Active Domain” einer Datenbank

Definition 2.3

Sei **R** ein Datenbankschema und sei **I** eine Datenbank vom Schema **R**.

Der Active Domain von **I**, kurz: $adom(\mathbf{I})$, ist die Menge aller Elemente aus **dom**, die in **I** vorkommen. D.h.:

$$adom(\mathbf{I}) \;=\; \bigcup_{R \in \mathbf{R}} adom\big(\mathbf{I}(R)\big)$$

wobei für jedes R aus **R** gilt: $adom\big(\mathbf{I}(R)\big)$ ist die kleinste Teilmenge von **dom**, so dass jedes Tupel $t \in \mathbf{I}(R)$ eine Funktion von $sort(R)$ nach $adom\big(\mathbf{I}(R)\big)$ ist.

Ist Q eine Anfrage und **I** eine Datenbank, so setzen wir

$$adom(Q, \mathbf{I}) \;:=\; adom(Q) \cup adom(\mathbf{I})$$

Proposition 2.4

*Für jede regelbasierte konjunktive Anfrage Q und jede Datenbank **I** (vom passenden DB-Schema) gilt:* $adom\big([\![Q]\!](\mathbf{I})\big) \;\subseteq\; adom(Q, \mathbf{I})$.

Beweis: siehe Tafel.

Monotonie und Erfüllbarkeit

Sind **I** und **J** zwei Datenbanken vom gleichen Schema **R**, so sagen wir "**J** ist eine Erweiterung von **I**" und schreiben kurz "$\mathbf{I} \subseteq \mathbf{J}$", falls für alle $R \in \mathbf{R}$ gilt: $\mathbf{I}(R) \subseteq \mathbf{J}(R)$ (d.h. jedes Tupel, das in einer Relation von **I** vorkommt, kommt auch in der entsprechenden Relation von **J** vor).

Definition 2.5

Sei **R** ein DB-Schema und sei q eine Anfragefunktion über **R**.

(a) q heißt monoton, falls für alle Datenbanken **I** und **J** über **R** gilt: Falls $\mathbf{I} \subseteq \mathbf{J}$, so ist $q(\mathbf{I}) \subseteq q(\mathbf{J})$.

(b) q heißt erfüllbar, falls es eine Datenbank **I** gibt mit $q(\mathbf{I}) \neq \emptyset$.

Satz 2.6

Jede regelbasierte konjunktive Anfrage ist monoton und erfüllbar.

Beweis: siehe Tafel.

Anwendung von Satz 2.6

Satz 2.6 liefert ein einfaches Kriterium, um zu zeigen, dass bestimmte Anfragefunktionen nicht durch eine regelbasierte konjunktive Anfrage beschrieben werden können:

> Wenn eine Anfragefunktion q nicht monoton ist, dann kann sie auch nicht durch eine regelbasierte konjunktive Aussage beschrieben werden.

Beispiel: Die Anfrage

(15) Welche Filme laufen nur zu 1 Uhrzeit?

ist nicht monoton, kann also nicht durch eine regelbasierte konjunktive Anfrage beschrieben werden.

Vorsicht: Dies heißt nicht, dass jede monotone Anfragefunktion durch eine Regel beschrieben werden kann!

“Graphische” Variante: Tableau-Anfragen

Beispiel-Anfrage:

(4) Welche Kinos (Name & Adresse) spielen einen Film mit “George Clooney”?

Darstellung als Tableau **T** (ähnlich zu QBE):

Filme

Titel	Regie	Schauspieler
x_{Titel}	x_{Regie}	“George Clooney”

Programm

Kino	Titel	Zeit
x_{Kino}	x_{Titel}	x_{Zeit}

Orte

Kino	Adresse	Telefon
x_{Kino}	x_{Adr}	x_{Tel}

Zugehörige Tableau-Anfrage: $\left(\mathbf{T},\ \langle x_{Kino},\ x_{Adr}\rangle\right)$

Tableaus — Präzise

Definition 2.7

Sei **R** ein Datenbankschema und R ein Relationsschema.

- Ein Tableau über R (auch: Einzel-Tableau) ist eine endliche Menge von freien Tupeln (also Tupeln über $\mathbf{dom} \cup \mathbf{var}$) der Stelligkeit $arity(R)$.

  (D.h. ein Tableau über R ist eine "Relation vom Schema R, die als Einträge nicht nur Elemente aus **dom**, sondern auch Variablen aus **var** haben kann".)

- Ein Tableau **T** über **R** ist eine Abbildung, die jedem $R \in \mathbf{R}$ ein Tableau über **R** zuordnet.

  (D.h. ein Tableau über **R** ist eine "Datenbank vom Schema **R**, die als Einträge auch Variablen enthalten kann".)

- Eine Tableau-Anfrage über **R** (bzw. R) ist von der Form $(\mathbf{T}, u)$, wobei **T** ein Tableau über **R** (bzw. R) und u ein freies Tupel ist, so dass jede Variable, die in u vorkommt, auch in $adom(\mathbf{T})$ vorkommt.

  u heißt Zusammenfassung der Anfrage $(\mathbf{T}, u)$.

Semantik von Tableau-Anfragen

Sei $Q = (\mathbf{T}, u)$ eine Tableau-Anfrage.

- $Var(Q)$ bezeichnet die Menge aller Variablen, die in u oder $\mathbf{T}$ vorkommen.
  $adom(Q)$ bezeichnet die Menge aller Konstanten, die in u oder $\mathbf{T}$ vorkommen.

- Eine Belegung für Q ist eine Abbildung $\beta : Var(Q) \to \mathbf{dom}$.

- Sei $\mathbf{I}$ eine Datenbank vom Schema $\mathbf{R}$.

  Eine Belegung β für Q heißt Einbettung von $\mathbf{T}$ in $\mathbf{I}$, falls "$\beta(\mathbf{T}) \subseteq \mathbf{I}$", d.h. f.a. $R \in \mathbf{R}$ gilt:

$$\beta\big(\mathbf{T}(R)\big) \;:=\; \{\beta(t) : t \in \mathbf{T}(R)\} \;\subseteq\; \mathbf{I}(R).$$

- Der Tableau-Anfrage Q ordnen wir die folgende Anfragefunktion $[\![Q]\!]$ zu:

$$[\![Q]\!](\mathbf{I}) \;:=\; \{\beta(u) : \beta \text{ ist eine Einbettung von } \mathbf{T} \text{ in } \mathbf{I}\}$$

  für alle Datenbanken $\mathbf{I} \in inst(\mathbf{R})$.

Logikbasierte Variante: Konjunktiver Kalkül

Beispiel-Anfrage:

(4) Welche Kinos (Name & Adresse) spielen einen Film mit “George Clooney”?

Formulierung als Anfrage des konjunktiven Kalküls:

$$\left\{ \langle x_{Kino}, x_{Adr} \rangle \quad : \quad \exists x_{Titel}\, \exists x_{Regie}\, \exists x_{Zeit}\, \exists x_{Tel}\, \Big(\begin{array}{l} \mathit{Filme}(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \\ \mathit{Programm}(x_{Kino}, x_{Titel}, x_{Zeit}) \wedge \\ \mathit{Orte}(x_{Kino}, x_{Adr}, x_{Tel}) \Big) \end{array} \right\}$$

Erinnerung an “Diskrete Modellierung”:

Ähnlichkeit zu Formeln der Logik erster Stufe (d.h. Prädikatenlogik).

Hier: eingeschränkte Variante, in der es nur $\exists$-Quantoren und Konjunktionen ($\wedge$) gibt.

Konjunktiver Kalkül (CQ) — Präzise

Definition 2.8

Sei **R** ein Datenbankschema.
Die Menge CQ[**R**] aller Formeln des konjunktiven Kalküls über **R** ist induktiv wie folgt definiert: (CQ steht für "conjunctive queries")

(A) $R(v_1, \ldots, v_r)$ gehört zu CQ[**R**],
für alle $R \in \mathbf{R}$, $r := \mathit{arity}(R)$ und $v_1, \ldots, v_r \in \mathbf{var} \cup \mathbf{dom}$.

(K) $(\varphi \wedge \psi)$ gehört zu CQ[**R**],
für alle $\varphi \in$ CQ[**R**] und $\psi \in$ CQ[**R**].

(E) $\exists x\, \varphi$ gehört zu CQ[**R**],
für alle $\varphi \in$ CQ[**R**] und $x \in \mathbf{var}$.

Insbesondere: Jede Formel in CQ[**R**] ist eine Formel der Logik erster Stufe über der Signatur $\mathbf{R} \cup \mathbf{dom}$ (wobei jedes Element aus **dom** als "Konstanten-Symbol" aufgefasst wird, das stets "mit sich selbst" interpretiert wird.

Semantik von CQ[**R**]

Sei φ eine CQ[**R**]-Formel.

- $adom(\varphi)$ bezeichnet die Menge aller Konstanten (also Elemente aus **dom**), die in φ vorkommen. $Var(\varphi)$ bezeichnet die Menge aller Variablen (also Elemente aus **var**), die in φ vorkommen.
- $frei(\varphi)$ bezeichnet die Menge aller Variablen, die frei in φ vorkommen.
  D.h.: $frei(R(v_1,..,v_r)) = \{v_1,..,v_r\} \cap \mathbf{var}$; $frei((\varphi \wedge \psi)) = frei(\varphi) \cup frei(\psi)$;
  $frei(\exists x \varphi) = frei(\varphi) \setminus \{x\}$.
- Eine Belegung für φ ist eine Abbildung $\beta : frei(\varphi) \to \mathbf{dom}$.
- Einer Datenbank **I** vom Schema **R** ordnen wir die logische Struktur

$$\mathcal{A}_{\mathbf{I}} := \left(\mathbf{dom},\ (\mathbf{I}(R))_{R\in\mathbf{R}},\ (c)_{c\in\mathbf{dom}}\right)$$

  zu. (Insbesondere ist $\mathcal{A}_{\mathbf{I}}$ eine σ-Struktur über der Signatur $\sigma := \mathbf{R} \cup \mathbf{dom}$.)
- Ist **I** eine Datenbank vom Schema **R** und β eine Belegung für φ, so sagen wir "**I** erfüllt φ unter β" und schreiben $\mathbf{I} \models \varphi[\beta]$ bzw. $(\mathbf{I}, \beta) \models \varphi$, um auszudrücken, dass $(\mathcal{A}_{\mathbf{I}}, \beta) \models \varphi$.

Notation

- Mit CQ bezeichnen wir die Klasse aller CQ[**R**]-Formeln für alle Datenbankschemata **R**.
- Manchmal schreiben wir $\{x_1/a_1, \ldots, x_r/a_r\}$ um die Belegung $\beta : \{x_1, .., x_r\} \to \mathbf{dom}$ zu bezeichnen mit $\beta(x_i) = a_i$, f.a. $i \in \{1, .., r\}$.
- Für eine CQ-Formel φ schreiben wir oft $\varphi(x_1, .., x_r)$, um anzudeuten, dass $\mathit{frei}(\varphi) = \{x_1, .., x_r\}$.
- Ist $a_1, .., a_r \in \mathbf{dom}$, so schreiben wir vereinfachend $\mathbf{I} \models \varphi[a_1, .., a_r]$ an Stelle von $\mathbf{I} \models \varphi[\{x_1/a_1, \ldots, x_r/a_r\}]$.
- Ist $y \in \mathbf{dom} \cup \mathbf{var}$, so bezeichnet $\varphi(x_1/y, x_2, .., x_r)$ die Formel, die aus φ entsteht, indem jedes Vorkommen der Variablen x_1 durch y ersetzt wird.
- Beim Schreiben von Formeln lassen wir Klammern "(", ")" oft weg und schreiben $\exists x_1, .., x_n$ als Abkürzung für $\exists x_1 \, \exists x_2 \, \cdots \exists x_n$.
- Zwei CQ[**R**]-Formeln φ und ψ heißen äquivalent, falls $\mathit{frei}(\varphi) = \mathit{frei}(\psi)$ und für jede Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{R})$ und jede Belegung β für φ (und ψ) gilt: $\mathbf{I} \models \varphi[\beta] \iff \mathbf{I} \models \psi[\beta]$.

Konjunktiver Kalkül: Syntax und Semantik

Definition 2.9

Sei **R** ein Datenbankschema.

Eine Anfrage des konjunktiven Kalküls ist von der Form

$$\{\langle e_1, \ldots, e_r \rangle \ : \ \varphi\}$$

wobei $\varphi \in \mathrm{CQ}[\mathbf{R}]$, $r \geqslant 0$ und $\langle e_1, \ldots, e_r \rangle$ ein freies Tupel ist, so dass $\mathit{frei}(\varphi) = \{e_1, .., e_r\} \cap \mathbf{var}$.

Semantik:

Einer Anfrage Q der Form $\{\langle e_1, .., e_r \rangle : \varphi\}$ ordnen wir die folgende Anfragefunktion $[\![Q]\!]$ zu:

$$[\![Q]\!](\mathbf{I}) \ := \ \left\{ \beta(\langle e_1, .., e_r \rangle) \ : \ \begin{array}{l} \beta \text{ ist eine Belegung für } \varphi \text{ mit} \\ \mathbf{I} \models \varphi[\beta] \end{array} \right\}$$

für alle Datenbanken $\mathbf{I} \in \mathit{inst}(\mathbf{R})$.

Wertebereich von Anfragen des konjunktiven Kalküls

Für eine Anfrage Q der Form $\{\langle e_1, \ldots, e_r \rangle : \varphi\}$ setzen wir

$$adom(Q) \;:=\; adom(\varphi) \cup \big(\{e_1, \ldots, e_r\} \cap \mathbf{dom}\big).$$

Ist Q eine Anfrage und $\mathbf{I}$ eine Datenbank, so setzen wir – wie üblich –

$$adom(Q, \mathbf{I}) \;:=\; adom(Q) \cup adom(\mathbf{I})$$

Analog zu Proposition 2.4 gilt auch für Anfragen des konjunktiven Kalküls:

Proposition 2.10

Für jede Anfrage Q des konjunktiven Kalküls und jede Datenbank $\mathbf{I}$ (vom passenden DB-Schema) gilt: $adom\big([\![Q]\!](\mathbf{I})\big) \;\subseteq\; adom(Q, \mathbf{I})$.

Beweis: Einfache Induktion über den Formelaufbau. Details: Übung.

Beispiel

Die Anfrage

> Gibt es einen Schauspieler, der sowohl in einem Film von “Detlev Buck” als auch in einem Film von “Stephen Spielberg” mitgespielt hat?

wird durch die folgende Anfrage des konjunktiven Kalküls beschrieben:

$$\left\{ \langle\rangle \;:\; \exists x_{Schauspieler} \left(\exists x_{Titel1}\, Filme(x_{Titel1}, \text{“Detlev Buck”}, x_{Schauspieler}) \wedge \exists x_{Titel2}\, Filme(x_{Titel2}, \text{“Stephen Spielberg”}, x_{Schauspieler}) \right) \right\}$$

und durch die dazu äquivalente Anfrage

$$\left\{ \langle\rangle \;:\; \exists x_{Schauspieler}\, \exists x_{Titel1}\, \exists x_{Titel2} \left(Filme(x_{Titel1}, \text{“Detlev Buck”}, x_{Schauspieler}) \wedge Filme(x_{Titel2}, \text{“Stephen Spielberg”}, x_{Schauspieler}) \right) \right\}$$

Eine Normalform für CQ

Definition 2.11

Eine CQ[**R**]-Formel $\varphi(x_1, \ldots, x_r)$ ist in Normalform, falls sie von der Form

$$\exists x_{r+1} \cdots \exists x_{r+s} \left(R_1(u_1) \wedge \cdots \wedge R_\ell(u_\ell) \right)$$

ist, mit $r, s, \ell \geqslant 0$, $R_1, \ldots, R_\ell \in \mathbf{R}$ und $u_1, \ldots, u_\ell$ freie Tupel über $\{x_1, \ldots, x_{r+s}\} \cup \mathbf{dom}$.

Analog ist eine Anfrage Q des konjunktiven Kalküls in Normalform, falls sie von der Form $\{\langle e_1, \ldots, e_r \rangle : \varphi\}$ ist, wobei φ eine CQ-Formel in Normalform ist.

Lemma 2.12

Jede CQ*-Formel ist äquivalent zu einer* CQ*-Formel in Normalform.*

Beweis: Übung.

Äquivalenz von Anfragesprachen

Definition 2.13

Seien $\mathcal{Q}_1$ und $\mathcal{Q}_2$ zwei Anfragesprachen. Wir schreiben

- $\mathcal{Q}_1 \leqslant \mathcal{Q}_2$ (bzw. $\mathcal{Q}_2 \geqslant \mathcal{Q}_1$; "$\mathcal{Q}_2$ ist mindestens so ausdrucksstark wie $\mathcal{Q}_1$"), falls jede Anfragefunktion, die durch eine Anfrage in $\mathcal{Q}_1$ ausgedrückt werden kann, auch durch eine Anfrage in $\mathcal{Q}_2$ ausgedrückt werden kann.
- $\mathcal{Q}_1 \equiv \mathcal{Q}_2$ ("$\mathcal{Q}_1$ und $\mathcal{Q}_2$ haben dieselbe Ausdrucksstärke") falls $\mathcal{Q}_1 \leqslant \mathcal{Q}_2$ und $\mathcal{Q}_2 \leqslant \mathcal{Q}_1$
- $\mathcal{Q}_1 < \mathcal{Q}_2$ (bzw. $\mathcal{Q}_2 > \mathcal{Q}_1$; "$\mathcal{Q}_2$ ist ausdrucksstärker als $\mathcal{Q}_1$") falls $\mathcal{Q}_1 \leqslant \mathcal{Q}_2$ und nicht $\mathcal{Q}_2 \leqslant \mathcal{Q}_1$.

Äquivalenz der bisher eingeführten Anfragesprachen

Lemma 2.14

Die Klassen der regelbasierten konjunktiven Anfragen, der Tableau-Anfragen und der Anfragen des konjunktiven Kalküls haben dieselbe Ausdrucksstärke.

Es gilt sogar: Jede Anfrage aus einer dieser drei Anfragesprachen kann in polynomieller Zeit in äquivalente Anfragen der beiden anderen Anfragesprachen übersetzt werden.

Beweis: siehe Tafel.

Einige Erweiterungen der Anfragesprachen

(1) Test auf "Gleichheit" von Variablen zulassen

(2) Hintereinanderausführung (Komposition) mehrerer Anfrage zulassen

Zu (1):

Beispiel-Anfrage (6): Welche (je 2) Schauspieler haben schon in Filmen gespielt, in denen der jeweils andere Regie geführt hat?

ausdrücken durch

$$Ans(y_1, y_2) \leftarrow Filme(x_1, y_1, z_1),\ Filme(x_2, y_2, z_2),\ y_1{=}z_2,\ y_2{=}z_1$$

aber auch, äquivalent, durch

$$Ans(y_1, y_2) \leftarrow Filme(x_1, y_1, y_2),\ Filme(x_2, y_2, y_1)$$

Regelbasierte konjunktive Anfragen mit "="

Prinzipiell:

Im Rumpf von Regeln auch Atome der Form "$x{=}y$" und "$x{=}c$" zulassen, für beliebige Variablen $x, y \in \mathbf{var}$ und Konstante $c \in \mathbf{dom}$.

$\rightsquigarrow$ regelbasierte konjunktive Anfragen mit "="

$\rightsquigarrow$ Konjunktiver Kalkül mit "="

Aber Vorsicht: Die Regel

$$Q := \quad Ans(x, y) \leftarrow R(x),\ y{=}z$$

ausgewertet in einer Datenbank $\mathbf{I}$ mit $\mathbf{I}(R) = \{a\}$, liefert als Ergebnis

$$[\![Q]\!](\mathbf{I}) \ = \ \{\langle a, d\rangle : d \in \mathbf{dom}\} \ = \ \{a\} \times \mathbf{dom}$$

Da **dom** unendlich viele Elemente hat, ist $[\![Q]\!](\mathbf{I})$ also eine "unendliche Relation"; per Definition (siehe Folie 21), sind als Relationen aber nur endliche Mengen erlaubt.

Daher syntaktische Einschränkung aus bereichsbeschränkte Anfragen, um zu garantieren, dass das Ergebnis einer Anfrage stets eine endliche Relation ist ...

Bereichsbeschränkte konjunktive Anfragen mit "="

Präzise:

Jede Variable x, die im Rumpf der Regel vorkommt, muss auch in einem im Rumpf der Regel stehenden Atom der Form $R(u)$ oder $x{=}c$, für ein $c \in \mathbf{dom}$, vorkommen — oder es muss eine Kette von Gleichheits-Atomen der Form $x{=}y_1, y_1{=}y_2, \ldots, y_j{=}z$ und ein Atom der Form $z{=}c$ für eine Konstante $c \in \mathbf{dom}$ geben oder ein Atom der Form $R(u)$ im Rumpf der Regel geben, so dass die Variable z im freien Tupel u vorkommt.

$\rightsquigarrow$ bereichsbeschränkte regelbasierte konjunktive Anfragen mit "="

Analog wird die Klasse $\mathrm{CQ}^{=}$ aller bereichsbeschränkten Formeln des konjunktiven Kalküls mit "=" definiert.

Beobachtung 2.15

Jede $\mathrm{CQ}^{=}$-Formel ist entweder unerfüllbar oder äquivalent zu einer CQ-Formel. (Details siehe Übung.)

Beispiel für eine $\mathrm{CQ}^{=}$-Anfrage, die nicht erfüllbar ist:

$$\{ \langle x \rangle : \big(R(x) \wedge x{=}a \wedge x{=}b \big) \}$$

wobei a und b zwei verschiedene Elemente aus **dom** sind.

Einige Erweiterungen der Anfragesprachen

(1) Test auf “Gleichheit” von Variablen zulassen

(2) Hintereinanderausführung (Komposition) mehrerer Anfrage zulassen

Zu (2):

Wende Anfrage auf das Resultat einer (oder mehrerer Anfragen an) . . .

Regelbasierte konjunktive Programme

Definition 2.16

Sei **R** ein Datenbankschema.
Ein regelbasiertes konjunktives Programm über **R** (mit oder ohne "=") hat die Form

$$\begin{array}{lcl} S_1(u_1) & \leftarrow & \text{Rumpf}_1 \\ S_2(u_2) & \leftarrow & \text{Rumpf}_2 \\ \vdots & \vdots & \vdots \\ S_m(u_m) & \leftarrow & \text{Rumpf}_m \end{array}$$

wobei $m \geqslant 1$, $S_1, \ldots, S_m$ sind paarweise verschiedene Relations-Namen aus **relname** $\setminus$ **R** und für jedes $i \in \{1, \ldots, m\}$ gilt:

$$Q_i \;:=\; S_i(u_i) \leftarrow \text{Rumpf}_i$$

ist eine regelbasierte konjunktive Anfrage über $\big(\mathbf{R} \cup \{S_j : 1 \leqslant j < i\}\big)$ (mit oder ohne "=").

Die Relations-Namen aus **R**, die im Programm vorkommen, heißen extensionale Prädikate (edb-Prädikate). Die Relations-Namen $S_1, \ldots, S_m$ heißen intensionale Prädikate (idb-Prädikate).

Semantik regelbasierter konjunktiver Programme

Ausgewertet über einer Datenbank $\mathbf{I} \in \mathit{inst}(\mathbf{R})$ beschreibt obiges Programm P der vorigen Folie die Relationen

$$[\![P(S_1)]\!](\mathbf{I}), \quad \ldots, \quad [\![P(S_m)]\!](\mathbf{I}),$$

die induktiv für alle $i \in \{1, \ldots, m\}$ folgendermaßen definiert sind:

$$[\![P(S_i)]\!](\mathbf{I}) \;:=\; [\![Q_i]\!](\mathbf{J}_{i-1})$$

wobei $\mathbf{J}_{i-1}$ die Erweiterung der Datenbank $\mathbf{I}$ um die Relationen $\mathbf{J}(S_j) := [\![P(S_j)]\!](\mathbf{I})$, für alle j mit $1 \leqslant j < i$ ist.

Äquivalenz von Regeln und Programmen

Beobachtung 2.17

Für jedes regelbasierte konjunktive Programm P über einem Relationsschema $\mathbf{R}$ (mit oder ohne "=") und jedes idb-Prädikat S von P gibt es eine regelbasierte konjunktive Anfrage Q (mit "=") über $\mathbf{R}$, so dass

$$[\![Q]\!](\mathbf{I}) \;=\; [\![P(S)]\!](\mathbf{I})$$

für alle $\mathbf{I} \in inst(\mathbf{R})$.

Beispiel: Sei $\mathbf{R} = \{Q, R\}$. Betrachte folgendes regelbasierte konjunktive Programm P:

$$\begin{aligned} S_1(x,z) &\leftarrow Q(x,y),\ R(y,z,w) \\ S_2(x,y,z) &\leftarrow S_1(x,w),\ R(w,y,v),\ S_1(v,z) \end{aligned}$$

Die von P durch S_2 definierte Anfrage ist äquivalent zu

$$\begin{aligned} S_2(x,y,z) \leftarrow\ & x{=}x',\ w{=}z',\ Q(x',y'),\ R(y',z',w'), \\ & R(w,y,v), \\ & v{=}x'',\ z{=}z'',\ Q(x'',y''),\ R(y'',z'',w'') \end{aligned}$$

Konjunktive Anfragen (I)

Auswertungskomplexität konjunktiver Anfragen

AUSWERTUNGSPROBLEM FÜR CQ: (kombinierte Komplexität)

Eingabe: Anfrage Q des konjunktiven Kalküls,
Datenbank $\mathbf{I}$ (von einem zu Q passenden Schema $\mathbf{R}$)

Aufgabe: Berechne $[\![Q]\!](\mathbf{I})$

Schön wäre: Algorithmus, der zur Lösung dieses Problems mit Zeit polynomiell in "Größe der Eingabe + Größe der Ausgabe" auskommt.

Frage: Gibt es einen solchen Algorithmus?

Größe der Eingabe: $k + n$, wobei

- $k := ||Q||$ die Länge der Anfrage
  (betrachtet als Wort über dem Alphabet $\mathbf{dom} \cup \mathbf{var} \cup \mathbf{R} \cup \{\exists, \wedge, (,), \langle, \rangle, \{, :, \}\}$)
- $n := ||\mathbf{I}||$ die Größe der Datenbank, also

$$n := ||\mathbf{I}|| := \sum_{R \in \mathbf{R}} ||\mathbf{I}(R)|| \quad \text{wobei} \quad ||\mathbf{I}(R)|| := \underbrace{arity(R) \cdot |\mathbf{I}(R)|}_{\text{Anzahl Tupel in } \mathbf{I}(R)}$$

Größe der Ausgabe: $||\ [\![Q]\!](\mathbf{I})\ ||$:= "Stelligkeit" · "Anzahl Tupel im Ergebnis"

Auswertung konjunktiver Anfragen

Proposition 2.18

Das Auswertungsproblem für CQ *lässt sich in Zeit* $\mathcal{O}\big((k+n)^k\big)$ *lösen.*

Beweis: siehe Tafel.

Bemerkung: Das ist exponentiell in der Länge der Anfrage.

Frage: Geht das effizienter?

Boolesche Anfragen

- Zur Erinnerung:

  Boolesche Anfragen sind “ja / nein”-Anfragen,
  d.h. Anfragen, deren Ergebnis die Stelligkeit 0 hat.

- Klar:

  Falls wir zeigen können, dass das Auswertungsproblem für Boolesche Anfragen des konjunktiven Kalküls schwierig ist, so ist es auch für allgemeine Anfragen des konjunktiven Kalküls schwierig.

- Wir werden sehen, dass umgekehrt aber auch gilt:

  Falls wir einen Algorithmus haben, der das Auswertungsproblem für *Boolesche* Anfragen des konjunktiven Kalküls löst, so können wir diesen Algorithmus verwenden, um das Auswertungsproblem für *beliebige* Anfragen des konjunktiven Kalküls zu lösen.

Algorithmen mit Verzögerung $f(k, n)$

Definition 2.19

Sei $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, sei $\mathcal{A}$ eine Datenbankanfragesprache und sei $\mathbb{A}$ irgendein Algorithmus, der bei allen Eingaben terminiert.

Wir sagen:

> das Auswertungsproblem für $\mathcal{A}$ kann unter Rückgriff auf $\mathbb{A}$ mit Verzögerung $f(k, n)$ gelöst werden,

falls es einen Algorithmus $\mathbb{B}$ gibt, der bei Eingabe einer Anfrage Q aus $\mathcal{A}$ und einer Datenbank $\mathbf{I}$ nach und nach genau die Tupel aus $[\![Q]\!](\mathbf{I})$ ausgibt und

- vor der Ausgabe des ersten Tupels,
- zwischen der Ausgabe von je zwei aufeinanderfolgenden Tupeln,
- nach der Ausgabe des letzten Tupels

je höchstens $f(||Q||, ||\mathbf{I}||)$ viele Elementarschritte oder Schritte, in denen der Algorithmus $\mathbb{A}$ aufgerufen wird, macht.

Boolesche Anfragen $\rightsquigarrow$ beliebige Anfragen

Theorem 2.20

Sei $\mathbb{A}$ ein Algorithmus, der das Auswertungsproblem für Boolesche Anfragen des konjunktiven Kalküls löst.

Dann gibt es einen Algorithmus $\mathbb{B}$, der das Auswertungsproblem für (beliebige) Anfragen des konjunktiven Kalküls unter Rückgriff auf $\mathbb{A}$ mit Verzögerung $\mathcal{O}(k^3 \cdot n)$ löst.

Beweis: siehe Tafel.

Folgerung: Falls wir das Auswertungsproblem für *Boolesche* Anfragen des konjunktiven Kalküls effizient lösen können, dann können wir es auch für *beliebige* Anfragen des konjunktiven Kalküls effizient lösen.

Die Komplexitätsklasse NP

Zur Erinnerung:

- Komplexitätsklassen bestehen aus Entscheidungsproblemen, d.h. Problemen mit "ja/nein"-Ausgabe
  - $\rightsquigarrow$ das Auswertungsproblem für Boolesche Anfragen passt gut zum Konzept der klassischen Komplexitätstheorie;
  - $\rightsquigarrow$ das Auswertungsproblem für beliebige Anfragen nicht

- Ein Entscheidungsproblem B gehört zur Klasse NP, falls es eine nichtdeterministische Turingmaschine T und eine Konstante c gibt, so dass für jede Zahl N und jede zum Problem B passende Eingabe w der Größe N gilt:

  (1) Jeder Lauf von T bei Eingabe w hat die Länge $\leqslant N^c$ und endet mit der Ausgabe "ja" oder "nein".

  (2) Ist w eine "ja"-Instanz für B, so gibt es mindestens einen Lauf von T auf w, der mit der Ausgabe "ja" endet.

  (3) Ist w eine "nein"-Instanz für B, so endet jeder Lauf von T auf w mit der Ausgabe "nein".

NP-Vollständigkeit

Zur Erinnerung:

- Ein Entscheidungsproblem B heißt NP-vollständig (bzgl. Polynomialzeit-Reduktionen), falls gilt:

  (1) $B \in \text{NP}$ und

  (2) B ist NP-hart, d.h. für jedes Problem $A \in \text{NP}$ gibt es eine Polynomialzeit-Reduktion f von A auf B (kurz: $f : A \leqslant_p B$)

- Eine Polynomialzeit-Reduktion f von A auf B ist eine (deterministisch) in polynomieller Zeit berechenbare Funktion, die jede zum Problem A passende Eingabe w auf eine zum Problem B passende Eingaben $f(w)$ abbildet, so dass gilt

  $$w \text{ ist eine “ja”-Instanz für } A \iff f(w) \text{ ist eine “ja”-Instanz für } B.$$

  Anschaulich bedeutet $A \leqslant_p B$, dass A “höchstens so schwer” wie B ist. NP-vollständige Probleme sind also “die schwierigsten Probleme” in NP.

- Der Begriff Polynomialzeit-Reduktion ist so definiert, dass folgendes gilt (wobei P die Klasse aller Entscheidungsprobleme bezeichnet, die deterministisch in Polynomialzeit lösbar sind)
  - $A \leqslant_p B$ und $B \in \text{P} \implies A \in \text{P}$
  - $A \notin \text{P}$ und $A \leqslant_p B \implies B \notin \text{P}$
  - A NP-hart und $A \leqslant_p B \implies B$ NP-hart.

Auswertungskomplexität konjunktiver Anfragen

Theorem 2.21 (Chandra, Merlin, 1977)
Das Auswertungsproblem für Boolesche regelbasierte konjunktive Anfragen ist NP*-vollständig.* *(kombinierte Komplexität)*

Beweis: siehe Tafel . . .

Zum Nachweis der NP-Härte benutzen wir das folgende Resultat, das Sie bereits aus der Veranstaltung Algorithmentheorie kennen:

Theorem: CLIQUE *ist* NP*-vollständig.*

Das Problem CLIQUE ist dabei folgendermaßen definiert:

CLIQUE

Eingabe: Ein endlicher ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$

Frage: Enthält G eine k-Clique?
(D.h. gibt es k verschiedene Knoten in G, von denen jeder mit jedem durch eine Kante verbunden ist?)

endlicher ungerichteter Graph:
jede Kante in E ist eine 2-elementige Teilmenge von V.

Folgerung aus der NP-Vollständigkeit

Folgerung:

Falls P $\neq$ NP, so gibt es keinen Algorithmus, der das Auswertungsproblem für Boolesche Anfragen des konjunktiven Kalküls deterministisch in Zeit $(k+n)^{\mathcal{O}(1)}$ löst.

Bemerkung 2.22 (hier ohne Beweis)

Unter Verwendung einer stärkeren Annahme aus der Parametrischen Komplexitätstheorie lässt sich folgendes zeigen:

Theorem (Papadimitriou, Yannakakis, 1997)

Falls FPT $\neq$ W[1]*, so gibt es keinen Algorithmus, der das Auswertungsproblem für Boolesche Anfragen des konjunktiven Kalküls deterministisch in Zeit* $f(k)\cdot n^c$ *löst*

(wobei f *irgendeine berechenbare Funktion und* c *irgendeine Konstante ist).*

FPT und W[1] sind Komplexitätsklassen, die in der parametrischen Komplexitätstheorie Rollen spielen, die in etwa mit den Rollen von P und NP in der klassischen Komplexitätstheorie vergleichbar sind.

Konjunktive Anfragen (I)

Algebraische Anfragen — Informell

Beispiel-Anfrage:

(4) Welche Kinos (Name & Adresse) spielen einen Film mit “George Clooney”?

Als Anfrage in der SPJR-Algebra:

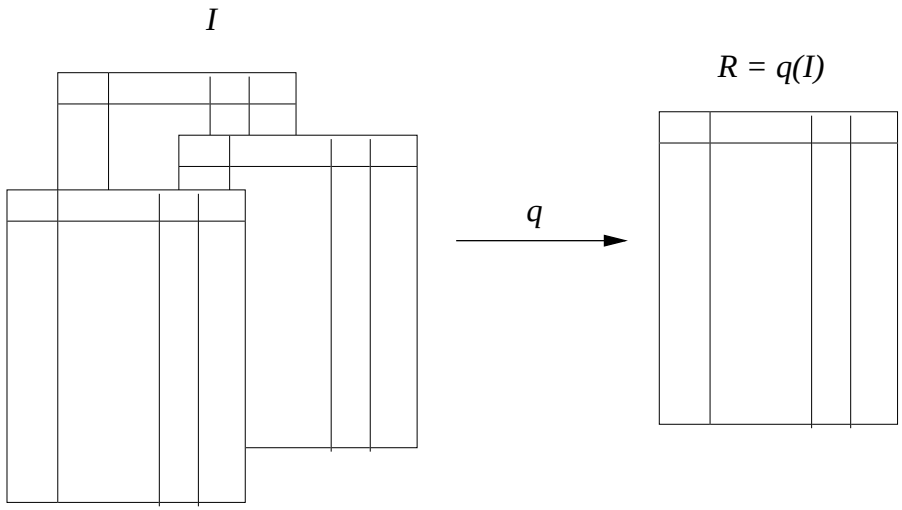
$$\pi_{Kino,Adresse}\left(\sigma_{\substack{Schauspieler = \\ \text{“George Clooney”}}}\left(Filme \bowtie Programm \bowtie Orte\right)\right)$$

Als Anfrage in der SPC-Algebra:

$$\pi_{7,8}\left(\sigma_{4=7}\left(\sigma_{1=5}\left(\sigma_{3 = \text{“George Clooney”}}\left(Filme \times Programm \times Orte\right)\right)\right)\right)$$

SPC-Algebra bzw. SPJR-Algebra

Zur Erinnerung:



- Mathematik:

  Algebraische Struktur $\stackrel{\triangle}{=}$ Grundmenge + Operationen

- Hier:
  - Operationen auf (endlichen) Relationen
  - Speziell: Projektion, Selektion,
    Cartesisches Produkt bzw. Join und Umbenennung

Unbenannte Perspektive: Die SPC-Algebra (1/4)

Operationen: Selektion σ, Projektion π, Cartesisches Produkt $\times$

Selektion: Zwei Varianten:

- Operator $\sigma_{j=a}$, für eine Konstante $a \in$ **dom** und eine natürliche Zahl $j \geqslant 1$.

  Dieser Operator kann angewendet werden auf Relationen I der Stelligkeit $\geqslant j$ und liefert als Ausgabe die folgende Teilmenge der Relation I:

$$\sigma_{j=a}(I) := \{t \in I : \text{in der } j\text{-ten Komponente von } t \text{ steht ein } a\}$$

- Operator $\sigma_{j=k}$, für zwei natürliche Zahlen $j, k \geqslant 1$.

  Dieser Operator kann angewendet werden auf Relationen I der Stelligkeit $\geqslant \max(j,k)$ und liefert als Ausgabe die folgende Teilmenge der Relation I:

$$\sigma_{j=k}(I) := \left\{ t \in I \ : \begin{array}{l} \text{in der } j\text{-ten Komponente von } t \text{ steht derselbe Eintrag} \\ \text{wie in der } k\text{-ten Komponente von } t \end{array} \right\}$$

Selektion ist eine “horizontale” Operation: sie wählt einzelne Tabellen-Zeilen aus.

“$j=a$” und “$j=k$” werden Selektionsbedingungen genannt.

Unbenannte Perspektive: Die SPC-Algebra (2/4)

Operationen: Selektion σ, Projektion π, Cartesisches Produkt $\times$

Projektion:

Operator $\pi_{j_1,\ldots,j_k}$, für (nicht notwendigerweise paarweise verschiedene) natürliche Zahlen $j_1,\ldots,j_k \geqslant 1$ (und $k \geqslant 0$).

Dieser Operator kann angewendet werden auf Relationen I der Stelligkeit $\geqslant \max\{j_1,\ldots,j_k\}$ und liefert als Ausgabe die folgende Relation der Stelligkeit k:

$$\pi_{j_1,\ldots,j_k}(I) \;:=\; \left\{ \left\langle t(j_1),\ldots,t(j_k)\right\rangle \;:\; t \in I \right\}$$

Projektion ist eine “vertikale” Operation: sie wählt einzelne Tabellen-Spalten aus und arrangiert sie in möglicherweise anderer Reihenfolge

Unbenannte Perspektive: Die SPC-Algebra (3/4)

Operationen: Selektion σ, Projektion π, Cartesisches Produkt $\times$

Cartesisches Produkt:

Operator $\times$

Dieser Operator kann angewendet werden auf Relationen I und J beliebiger Stelligkeiten m und n und liefert als Ausgabe die folgende Relation der Stelligkeit $m+n$:

$$I \times J := \left\{ \left\langle t(1), \ldots, t(m), s(1), \ldots, s(n) \right\rangle : t \in I \text{ und } s \in J \right\}$$

Wir benutzen den Operator manchmal auch für einzelne Tupel:
Sind t und s Tupel der Stelligkeiten m und n, so schreiben wir $t \times s$, um das Tupel $\langle t(1), \ldots, t(m), s(1), \ldots, s(n) \rangle$ zu bezeichnen.

Unbenannte Perspektive: Die SPC-Algebra (4/4)

Definition 2.23

Sei **R** ein Datenbankschema. Die Klasse der Anfragen der SPC-Algebra über **R** (kurz: SPC[**R**]) ist induktiv wie folgt definiert:

- Für alle Relations-Namen $R \in \mathbf{R}$ ist R eine SPC[**R**]-Anfrage der Stelligkeit $arity(R)$.
- Für alle Konstanten $c \in \mathbf{dom}$ ist $\{\langle c \rangle\}$ eine SPC[**R**]-Anfrage der Stelligkeit 1.
- Ist Q eine SPC[**R**]-Anfrage der Stelligkeit m, sind j, k natürliche Zahlen aus $\{1, \ldots, m\}$ und ist $a \in \mathbf{dom}$ eine Konstante, so sind auch $\sigma_{j=a}(Q)$ und $\sigma_{j=k}(Q)$ SPC[**R**]-Anfragen der Stelligkeit m.
- Ist Q eine SPC[**R**]-Anfrage der Stelligkeit m, ist $k \geqslant 0$ und sind $j_1, \ldots, j_k$ natürliche Zahlen aus $\{1, \ldots, m\}$, so ist $\pi_{j_1, \ldots, j_k}(Q)$ eine SPC[**R**]-Anfrage der Stelligkeit k.
- Sind Q und P zwei SPC[**R**]-Anfragen der Stelligkeiten m und n, so ist $(Q \times P)$ eine SPC[**R**]-Anfrage der Stelligkeit $m+n$.

Die Semantik $[\![Q]\!]$ von SPC[**R**]-Anfragen Q ist induktiv auf die offensichtliche Art definiert.

Verallgemeinerung des Selektions-Operators

Eine positive konjunktive Selektionsbedingung ist eine Formel F der Form

$$\gamma_1 \wedge \cdots \wedge \gamma_n$$

wobei $n \geqslant 1$ und jedes γ_i eine Selektionsbedingung der Form $j_i{=}a_i$ oder $j_i{=}k_i$ für natürliche Zahlen $j_i, k_i \geqslant 1$ und Konstanten $a_i \in \mathbf{dom}$.

Der Selektionsoperator σ_F hat dieselbe Wirkung wie die Hintereinanderausführung der Selektionsoperatoren σ_{γ_i} für alle $i \in \{1, \ldots, n\}$.

Eine Normalform für SPC-Anfragen

Sei **R** ein Relationsschema.

Definition 2.24

Eine SPC[**R**]-Anfrage ist in Normalform, falls sie von der Form

$$\pi_{j_1,\ldots,j_k}\Big(\{\langle c_1\rangle\}\times\cdots\times\{\langle c_m\rangle\}\times\sigma_F\big(R_1\times\cdots\times R_\ell\big)\Big)$$

ist, für $k, m, \ell \geqslant 0$, paarweise verschiedene Elemente $j_1,\ldots,j_k$, so dass $\{1,\ldots,m\}\subseteq\{j_1,\ldots,j_k\}$, Konstanten $c_1,\ldots,c_m \in \mathbf{dom}$, $R_1,\ldots,R_\ell \in \mathbf{R}$ und F eine positive konjunktive Selektionsbedingung.

Proposition 2.25

Für jede SPC[**R**]*-Anfrage Q gibt es eine* SPC[**R**]*-Anfrage Q' in Normalform, die dieselbe Anfragefunktion definiert.*

Beweis: Übung.

Benannte Perspektive: Die SPJR-Algebra (1/7)

Operationen: Selektion σ, Projektion π, Join $\bowtie$, Umbenennung (Renaming) δ

Attribut-Namen an Stelle von Spaltennummern !

Selektion: Zwei Varianten:

- Operator $\sigma_{A=a}$, für eine Konstante $a \in$ **dom** und einen Attribut-Namen A.

  Dieser Operator kann angewendet werden auf R-Relationen I mit $A \in sort(R)$ und liefert als Ausgabe die folgende Teilmenge der Relation I:

$$\sigma_{A=a}(I) := \{ t \in I : t(A) = a \}$$

- Operator $\sigma_{A=B}$, für zwei Attribut-Namen A und B.

  Dieser Operator kann angewendet werden auf R-Relationen I mit $A, B \in sort(R)$ und liefert als Ausgabe die folgende Teilmenge der Relation I:

$$\sigma_{A=B}(I) := \{ t \in I : t(A) = t(B) \}$$

"$A{=}a$" und "$A{=}B$" werden Selektionsbedingungen genannt.

Benannte Perspektive: Die SPJR-Algebra (2/7)

Operationen: Selektion σ, Projektion π, Join $\bowtie$, Umbenennung (Renaming) δ

Projektion:

Operator $\pi_{A_1,\ldots,A_k}$, für paarweise verschiedene Attribut-Namen $A_1, \ldots, A_k$ (und $k \geqslant 0$).

Dieser Operator kann angewendet werden auf R-Relationen I mit $sort(R) \supseteq \{A_1, \ldots, A_k\}$ und liefert als Ausgabe die folgende Relation der Sorte $\{A_1, \ldots, A_k\}$:

$$\pi_{A_1,\ldots,A_k}(I) := \left\{ \left\langle A_1 : t(A_1),\ \ldots,\ A_k : t(A_k) \right\rangle \ :\ t \in I \right\}$$

Benannte Perspektive: Die SPJR-Algebra (3/7)

Operationen: Selektion σ, Projektion π, Join $\bowtie$, Umbenennung (Renaming) δ

An Stelle des Cartesischen Produkts: Natürlicher Join

Beispiel: Wenn I und J zwei Relationen mit disjunkten Attributmengen (d.h. Spaltenbezeichnungen) sind, so bilde einfach das herkömmliche Cartesische Produkt.

I

A	B
1	a
2	b
3	c

$\bowtie$

J

C	D	E
e	f	g
h	i	j

$=$

$I \bowtie J$

A	B	C	D	E
1	a	e	f	g
1	a	h	i	j
2	b	e	f	g
2	b	h	i	j
3	c	e	f	g
3	c	h	i	j

Frage: Was soll passieren, wenn I und J gemeinsame Attribute haben?

Antwort: Zueinander passende Tupel werden verschmolzen.

Beispiel:

I

A	B
1	a
2	b
3	c

$\bowtie$

J

B	C	D
a	f	g
a	i	j
c	l	m

$=$

$I \bowtie J$

A	B	C	D
1	a	f	g
1	a	i	j
3	c	l	m

Benannte Perspektive: Die SPJR-Algebra (4/7)

Operationen: Selektion σ, Projektion π, Join $\bowtie$, Umbenennung (Renaming) δ

Natürlicher Join:

Operator $\bowtie$

Dieser Operator kann angewendet werden auf eine R-Relation I und eine S-Relation J beliebiger Sorten und liefert als Ausgabe die folgende Relation der Sorte $\Sigma := sort(R) \cup sort(S)$:

$$I \bowtie J := \Big\{ \text{Tupel } t \text{ der Sorte } \Sigma : \text{es gibt Tupel } t' \in I \text{ und } t'' \in J \text{ so dass } t|_{sort(R)} = t' \text{ und } t|_{sort(S)} = t'' \Big\}$$

Wir benutzen den Operator manchmal auch für einzelne Tupel:
Sind t' und t'' Tupel der Sorten $sort(R)$ und $sort(S)$, so schreiben wir $t' \bowtie t''$, um das Tupel t der Sorte $sort(R) \cup sort(S)$ mit $t|_{sort(R)} = t'$ und $t|_{sort(S)} = t''$ zu bezeichnen (bzw. den Wert *"undefiniert"*, falls es kein solches Tupel gibt, d.h. falls $t'|_{sort(R)\cap sort(S)} \neq t''|_{sort(R)\cap sort(S)}$).

Benannte Perspektive: Die SPJR-Algebra (5/7)

Operationen: Selektion σ, Projektion π, Join $\bowtie$, Umbenennung (Renaming) δ

Umbenennung (Renaming):

Operator δ_f, für eine Umbenennungsfunktion f, d.h. eine bijektive Funktion $f : U \to \mathbf{att}$, für eine beliebige endliche Menge U von Attribut-Namen.

Dieser Operator kann angewendet werden auf R-Relationen I, für die gilt: $U \subseteq sort(R)$ und $\big(sort(R) \setminus U\big) \cap f(U) = \emptyset$ und liefert die folgende Relation der Sorte $f(U) \cup \big(sort(R) \setminus U\big)$:

$$\delta_f(I) := \left\{ \text{Tupel } t \ : \begin{array}{l} \text{ex } t' \in I \text{ so dass f.a. } A \in U \text{ gilt } t'(A) = t(f(A)) \\ \text{und f.a. } A \in sort(R) \setminus U \text{ gilt } t'(A) = t(A) \end{array} \right\}$$

Oft schreiben wir $A_1 \cdots A_k \mapsto B_1 \cdots B_k$ um die Umbenennungsfunktion f mit Definitionsbereich $U = \{A_1, \ldots, A_k\}$ und Werten $f(A_i) = B_i$, für alle $i \in \{1, \ldots, k\}$, zu bezeichnen.

Benannte Perspektive: Die SPJR-Algebra (6/7)

Definition 2.26

Sei **R** ein Datenbankschema. Die Klasse der Anfragen der SPJR-Algebra über **R** (kurz: SPJR[**R**]) ist induktiv wie folgt definiert:

- Für alle Relations-Namen $R \in \mathbf{R}$ ist R eine SPJR[**R**]-Anfrage der Sorte $sort(R)$.
- Für alle Konstanten $c \in \mathbf{dom}$ und alle Attribut-Namen $A \in \mathbf{att}$ ist $\{\langle A : c \rangle\}$ eine SPJR[**R**]-Anfrage der Sorte $\{C\}$.
- Ist Q eine SPJR[**R**]-Anfrage der Sorte Σ, sind $A, B \in \Sigma$ und ist $a \in \mathbf{dom}$ eine Konstante, so sind auch $\sigma_{A=a}(Q)$ und $\sigma_{A=B}(Q)$ SPJR[**R**]-Anfragen der Sorte Σ.
- Ist Q eine SPJR[**R**]-Anfrage der Sorte Σ, ist $k \geqslant 0$ und sind $A_1, \ldots, A_k$ paarweise verschiedene Elemente aus Σ, so ist $\pi_{A_1,\ldots,A_k}(Q)$ eine SPJR[**R**]-Anfrage der Sorte $\{A_1, \ldots, A_k\}$.
- Sind Q und P zwei SPJR[**R**]-Anfragen der Sorten Σ und Π, so ist $(Q \bowtie P)$ eine SPJR[**R**]-Anfrage der Sorte $\Sigma \cup \Pi$.
- Ist Q eine SPJR[**R**]-Anfrage der Sorte Σ und ist $f : \Sigma \to \mathbf{att}$ eine Umbenennungsfunktion, so ist $\delta_f(Q)$ eine SPJR[**R**]-Anfrage der Sorte $f(\Sigma)$.

Benannte Perspektive: Die SPJR-Algebra (7/7)

Die Semantik $[\![Q]\!]$ von SPJR[**R**]-Anfragen Q ist induktiv auf die offensichtliche Art definiert.

Wie bei der SPC-Algebra lassen wir wieder eine Verallgemeinerung des Selektions-Operators zu:

- Eine positive konjunktive Selektionsbedingung ist eine Formel F der Form

$$\gamma_1 \wedge \cdots \wedge \gamma_n$$

  wobei $n \geqslant 1$ und jedes γ_i eine Selektionsbedingung der Form $A_i{=}a_i$ oder $A_i{=}B_i$ für Attribut-Namen A_i, B_i und Konstanten $a_i \in \mathbf{dom}$.

- Der Selektionsoperator σ_F hat dieselbe Wirkung wie die Hintereinanderausführung der Selektionsoperatoren σ_{γ_i} für alle $i \in \{1, \ldots, n\}$.

Eine Normalform für SPJR-Anfragen

Sei **R** ein Relationsschema.

Definition 2.27

Eine SPJR[**R**]-Anfrage ist in Normalform, falls sie von der Form

$$\pi_{B_1,\ldots,B_k}\Big(\{\langle A_1 : c_1\rangle\} \bowtie \cdots \bowtie \{\langle A_m : c_m\rangle\} \bowtie \sigma_F\big(\delta_{f_1}(R_1) \bowtie \cdots \bowtie \delta_{f_\ell}(R_\ell)\big)\Big)$$

ist, für $k, m, \ell \geqslant 0$, $B_1, \ldots, B_k, A_1, \ldots, A_m \in \mathbf{att}$ so dass $\{A_1, \ldots, A_m\} \subseteq \{B_1, \ldots, B_k\}$ und die $A_1, \ldots, A_m$ paarweise verschieden, $c_1, \ldots, c_m \in \mathbf{dom}$, $R_1, \ldots, R_\ell \in \mathbf{R}$, eine positive konjunktive Selektionsbedingung F und Umbenennungsfunktionen $f_1, \ldots, f_\ell$, so dass die Sorten von $\delta_{f_1}(R_1)$,..., $\delta_{f_\ell}(R_\ell)$ paarweise disjunkt sind und keins der $A_1, \ldots, A_m$ als Attribut von einem der $\delta_{f_j}(R_j)$ vorkommt.

Proposition 2.28

Für jede SPJR[**R**]*-Anfrage Q gibt es eine* SPJR[**R**]*-Anfrage Q' in Normalform, die dieselbe Anfragefunktion definiert.*

Beweis: Übung.

Nicht-Erfüllbare Anfragen

Bemerkung:

Sowohl in der SPC-Algebra als auch in der SPJR-Algebra lassen sich unerfüllbare Anfragen ausdrücken.

Beispiel: Die Anfrage $Q :=$

$$\sigma_{3=\text{“George Clooney”}}\Big(\sigma_{3=\text{“Wolfgang Völz”}}\big(\mathit{Filme}\big)\Big)$$

wählt in einer Datenbank vom Schema **KINO** genau diejenigen Tupel $t = (a, b, c)$ aus der *Filme*-Relation aus, für deren dritte Komponente c gilt: $c =$ “George Clooney” und $c =$ “Wolfgang Völz”.

Solche Tupel kann es aber nicht geben!

Für jede Datenbank **I** vom Schema **KINO** gilt also: $[\![Q]\!](\mathbf{I}) = \emptyset$.

Somit ist die Anfrage Q nicht erfüllbar.

Äquivalenz der Ausdrucksstärke der SPC-Algebra und der SPJR-Algebra

Lemma 2.29

Die SPC-Algebra und die SPJR-Algebra können genau dieselben Anfragen ausdrücken.

Es gilt sogar: Jede Anfrage aus einer dieser Anfragesprachen kann in polynomieller Zeit in eine äquivalente Anfrage der anderen Anfragesprache übersetzt werden.

Beweis: siehe Tafel.

Äquivalenz des deskriptiven und des algebraischen Ansatzes

Theorem 2.30 (Äquivalenz konjunktiver Anfragesprachen)

Die folgenden Anfragesprachen können genau dieselben erfüllbaren Anfragefunktionen ausdrücken:

(a) *die Klasse der regelbasierten konjunktiven Anfragen*

(b) *die Klasse der Tableau-Anfragen*

(c) *die Klasse der Anfragen des konjunktiven Kalküls*

(d) *die Anfragen der SPC-Algebra*

(e) *die Anfragen der SPJR-Algebra.*

Es gilt sogar: Jede Anfrage aus einer dieser Anfragesprachen kann in polynomieller Zeit in äquivalente Anfragen der anderen Anfragesprachen übersetzt werden.

Beweis: siehe Tafel.

Relationale Algebra

3.1 Definition und Beispiele
3.2 Anfrageauswertung und Heuristische Optimierung

Relationale Algebra

Grenzen der Ausdrucksstärke konjunktiver Anfragen

Wir haben gesehen:

- konjunktive Anfragen können nur *monotone* Anfragefunktionen beschreiben (Satz 2.6) $\leadsto$ Anfragen mit Negationen der Art

  Welche Regisseure haben noch nie mit “Tom Cruise” gearbeitet?

  können nicht in SPC- bzw. SPJR-Algebra gestellt werden.

- konjunktive Anfragen können keine Ver-ODER-ungen der Art

  In welchen Kinos läuft “Capote” oder “Knallhart”?

  ausdrücken (Übung, Blatt 1, Aufgabe 2(b)).

Jetzt:

Erweitere SPC- bzw. SPJR-Algebra um die Möglichkeit, auch solche Anfragen zu beschreiben.

Vereinigung und Differenz

Operatoren $\cup$ und $-$:

Diese Operatoren können angewendet werden auf Relationen I und J, die dieselbe Sorte bzw. Stelligkeit haben und liefern als Ausgabe die Relationen

$$I \cup J := \{ t : t \in I \text{ oder } t \in J \}$$

bzw.

$$I - J := \{ t \in I : t \notin J \}$$

SPJRU, SPCU und relationale Algebra

Definition 3.1

Sei **R** ein Datenbankschema.

(a) Zur Definition der Klasse der Anfragen der SPJRU[**R**] (bzw. der SPCU[**R**]) werden die Definitionen von SPJR[**R**] (bzw. SPC[**R**]) um die folgende Regel erweitert:

- Sind Q und P zwei SPJRU[**R**]-Anfragen derselben Sorte Σ (bzw. SPCU[**R**]-Anfragen derselben Stelligkeit k), so ist $(Q \cup P)$ eine SPJRU[**R**]-Anfrage der Sorte Σ (bzw. eine SPCU[**R**]-Anfrage der Stelligkeit k).

(b) Zur Definition der Klasse der Anfragen der relationalen Algebra über **R** in der benannten (bzw. der unbenannten) Perspektive werden die Definitionen von SPJRU[**R**] (bzw. SPCU[**R**]) um die folgende Regel erweitert:

- Sind Q und P zwei Anfragen der relationalen Algebra derselben Sorte Σ (bzw. derselben Stelligkeit k), so ist $(Q - P)$ eine Anfrage relationalen Algebra der Sorte Σ (bzw. der Stelligkeit k).

Die Semantik $[\![Q]\!]$ solcher Anfragen Q ist induktiv auf die offensichtliche Art definiert. Mit SPJRU (bzw. SPCU) bezeichnen wir die Klasse aller SPJRU[**R**]-Anfragen (bzw. SPCU[**R**]-Anfragen) für alle Datenbankschemata **R**.

Beispiele

- In welchen Kinos läuft “Capote” oder “Knallhart”?

$$\pi_{\text{Kino}}\left(\sigma_{\text{Titel=”Capote”}}(\mathit{Programm}) \cup \sigma_{\text{Titel=”Knallhart”}}(\mathit{Programm})\right)$$

- Welche Regisseure haben noch nie mit “Tom Cruise” gearbeitet?

$$\pi_{\text{Regie}}(\mathit{Filme}) - \pi_{\text{Regie}}\left(\sigma_{\text{Schauspieler=“Tom Cruise”}}(\mathit{Filme})\right)$$

- Welche derzeit laufenden Filme haben nur Schauspieler, die schon mal in einem Film von “Stephen Spielberg” mitgespielt haben?

$$\pi_{\text{Titel}}(\mathit{Programm}) - \pi_{\text{Titel}}\Big(\mathit{Filme} \bowtie \underbrace{\left(\pi_{\substack{\text{Schau-}\\\text{spieler}}}(\mathit{Filme}) - \pi_{\substack{\text{Schau-}\\\text{spieler}}}\left(\sigma_{\substack{\text{Regie=“Stephen}\\\text{Spielberg”}}}(\mathit{Filme})\right)\right)}_{\text{Schauspieler, die noch nie mit Stephen Spielberg gearbeitet haben}}\Big)$$

Filme mit mind. einem Schauspieler, der noch nie mit Stephen Spielberg gearbeitet hat

Ausdrucksstärke (1/2)

Proposition 3.2

(a) *Jede* SPCU*-Anfrage und jede* SPJRU*-Anfrage ist monoton.*

(b) *Für jede Datenbank* $\mathbf{I}$ *und jede Anfrage* Q *der relationalen Algebra gilt:*

$$adom\big([\![Q]\!](\mathbf{I})\big) \subseteq adom(Q, \mathbf{I}).$$

(c)
$$\begin{array}{ccccc} \text{SPC} & < & \text{SPCU} & < & \textit{relationale Algebra (unbenannte Perspektive)} \\ \equiv & & \equiv & & \equiv \\ \text{SPJR} & < & \text{SPJRU} & < & \textit{relationale Algebra (benannte Perspektive)} \end{array}$$

Beweis:

(a)+(b): Einfache Induktion nach dem Aufbau der Anfragen.

(c): Übung.

Ausdrucksstärke (2/2)

Proposition 3.3

(a) *Benannte Perspektive:*
Keiner der Operatoren σ, π, $\cup$, $-$, $\bowtie$, δ ist redundant.

D.h.: Weglassen jedes einzelnen dieser Operatoren führt zu einer Algebra, die manche in der relationalen Algebra ausdrückbaren Anfragefunktionen nicht beschreiben kann.

(b) *Unbenannte Perspektive:*

(i) *Der Operator σ kann durch Kombination der Operatoren π, $-$, $\times$ ausgedrückt werden.*

Beachte: um dies zu zeigen, muss man nutzen, dass bei der Proketion $\pi_{j_1,\ldots,j_k}$ die Indices j_i nicht paarweise verschieden sein müssen.

(ii) *Keiner der Operatoren π, $\cup$, $-$, $\times$ ist redundant.*

Beweis: Übung.

Theta-Join und Semijoin

Eine positive konjunktive Join-Bedingung ist ein Ausdruck θ der Form $\bigwedge_{\ell=1}^{m} x_{i_\ell}{=}y_{j_\ell}$, für natürliche Zahlen $m \geqslant 0$ und $i_1, \ldots, i_m, j_1, \ldots, j_m \geqslant 1$.

Zwei Tupel $\overline{a} = (a_1, \ldots, a_r) \in \mathbf{dom}^r$ und $\overline{b} = (b_1, \ldots, b_s) \in \mathbf{dom}^s$ mit $r \geqslant \max\{i_1, \ldots, i_m\}$ und $s \geqslant \max\{j_1, \ldots, j_m\}$ erfüllen θ

$$(\text{kurz: } (\overline{a}, \overline{b}) \models \theta, \text{ bzw. } \theta(\overline{a}, \overline{b})),$$

falls für alle $\ell \in \{1, \ldots, m\}$ gilt: $a_{i_\ell} = b_{j_\ell}$.

In der relationalen Algebra (unbenannte Perspektive) lassen sich u.a. die folgenden Operationen ausdrücken:

- Theta-Join $\bowtie_\theta$, wobei θ eine positive konjunktive Join-Bedingung ist.
  Semantik: $I \bowtie_\theta J := \{ (\overline{a}, \overline{b}) : \overline{a} \in I, \overline{b} \in J, \text{ so dass } (\overline{a}, \overline{b}) \models \theta \}$
- Semijoin $\ltimes_\theta$, wobei θ eine positive konjunktive Join-Bedingung ist.
  Semantik: $I \ltimes_\theta J := \{ \overline{a} \in I : \text{ex. } \overline{b} \in J, \text{ so dass } (\overline{a}, \overline{b}) \models \theta \}$

Relationale Algebra

Anfrageauswertung und Heuristische Optimierung

. . . hat viele Aspekte:

- Speicher- und Indexstrukturen
- Betriebssystem
- Seitenersetzungsstrategien
- Statistische Eigenschaften der Daten
- Statistische Informationen über Anfragen
- Implementierung der einzelnen Operatoren
- Ausdrucksstärke der Anfragesprache

Hier: Überblick und einige Teilaspekte.

Details: DBS I+II Vorlesung von Prof. Zicari

Anfrageauswertung allgemein

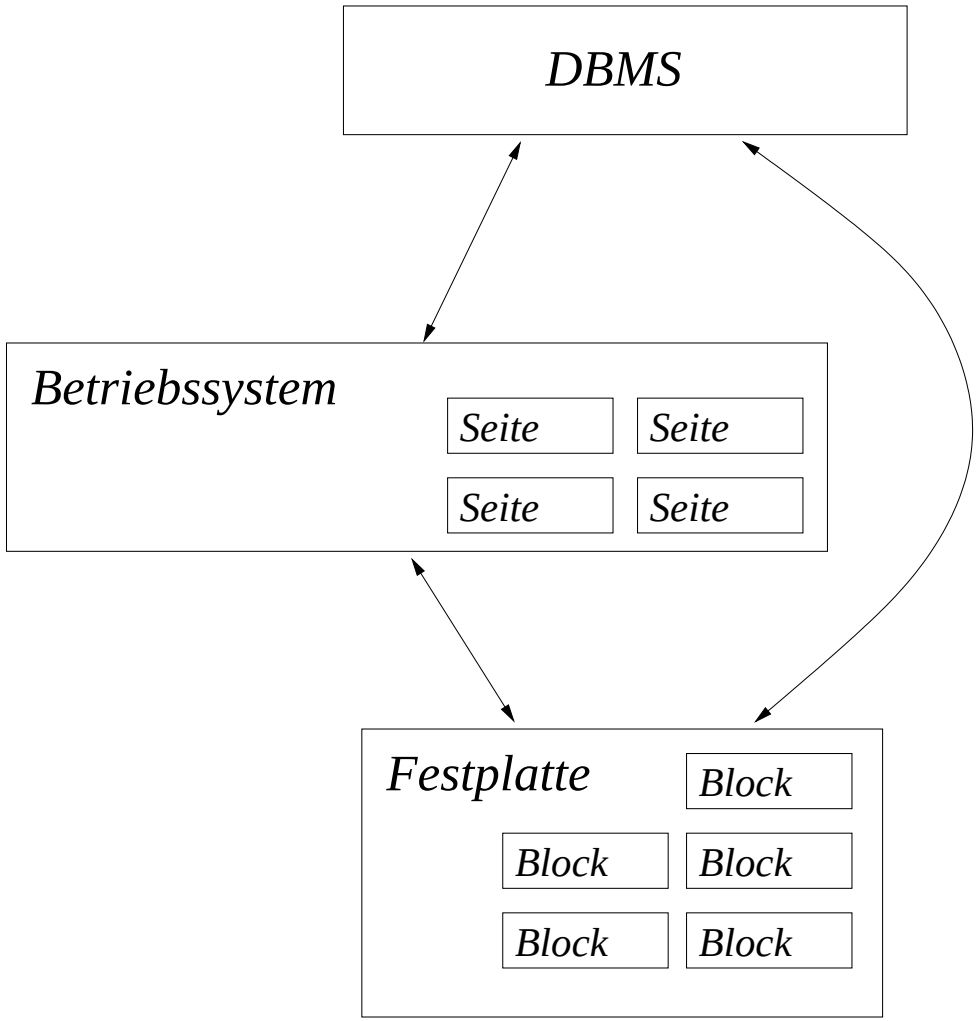
Vorbemerkung:

- Datenbanken sind sehr groß
- werden auf Sekundärspeicher (Festplatte) gespeichert
- Aufwand wird dominiert durch die Anzahl der Plattenzugriffe (“Seitenzugriffe”)
  Denn: In derselben Zeit, die für einen “Random Access” auf der Festplatte benötigt wird, können zigtausende Operationen im Hauptspeicher durchgeführt werden.

Allgemeines Vorgehen:

- durch Erzeugen, Filtern, Manipulieren und Kombinieren von Tupelströmen
- dabei evtl. Verwendung von Indexstrukturen (“Wegweiser”), Hashing und Sortier-Schritten
- wünschenswert: möglichst wenig auf Festplatte zwischenspeichern
- Operationen: Operationen der relationalen Algebra, Sortieren, Duplikatelimination, . . .

Verwaltung des Sekundärspeichers



1 Plattenzugriff $\triangleq$ Lesen eines Blocks bzw. einer Seite

Wichtige Parameter einer Datenbankrelation I

- n_I: Anzahl der Tupel in Relation I
- s_I: (mittlere) Größe eines Tupels aus I
- f_I: Blockungsfaktor ("Wie viele Tupel aus I passen in einen Block?")

$$f_I \approx \frac{\text{Blockgröße}}{s_I}$$

- b_I: Anzahl der Blöcke (Seiten) der Festplatte, die Tupel aus I beinhalten

$$b_I \approx \frac{n_I}{f_I}$$

Lesen eines Blocks (bzw. einer Seite) $\stackrel{\triangle}{=}$ 1 Zugriff auf Platte

Operationen der relationalen Algebra (1/3)

Selektion $\sigma_F(I)$:

- Selektion meist als Filter auf einem Tupelstrom:
  $\mathcal{O}(b_I)$ Zugriffe auf Festplatte; $\mathcal{O}(n_I)$ Schritte insgesamt
- evtl. Verwendung eines Index;
  dann schneller, falls nur sehr wenige Tupel im Ergebnis

Operationen der relationalen Algebra (2/3)

Projektion $\pi_{j_1,\ldots,j_k}(I)$:

- 2 Komponenten:
  - Ändern der einzelnen Tupel (Auswahl und Reihenfolge von Spalten)
  - Duplikatelimination

- Tupeländerung: als Filter auf einem Tupelstrom

  $\mathcal{O}(b_I)$ Zugriffe auf Festplatte; $\mathcal{O}(n_I)$ Schritte insgesamt

  Dabei können Duplikate “entstehen”

- Duplikatelimination:
  - in SQL i.d.R. nicht verlangt (außer bei `SELECT DISTINCT`)
  - sind die Tupel sortiert, so können Duplikate durch einen Scan leicht erkannt werden
  - Sortieren: durch Merge-Sort möglich mit $\mathcal{O}(b_I \cdot \log b_I)$ Plattenzugriffen und $\mathcal{O}(n_I \cdot \log n_I)$ Schritten insgesamt
  - Alternative: Hashing
    - Abbilden der Tupel durch eine Hash-Funktion
    - $\rightsquigarrow$ Duplikate werden auf denselben Wert abgebildet und dadurch erkannt
    - bei idealer Hash-Funktion: lineare Zeit

Operationen der relationalen Algebra (3/3)

Binäre Operationen auf zwei Relationen I und J: $\cup,\ -,\ \times,\ \bowtie_\theta,\ \ltimes_\theta$

- Nested-Loops-Methode: (Schleifeniteration)

  für jedes Tupel $t \in I$ (bzw. jede Seite) wird die gesamte Relation J durchlaufen

  $\mathcal{O}(b_I \cdot b_J)$ Plattenzugriffe; $\mathcal{O}(n_I \cdot n_J)$ Schritte insgesamt

- Merge-Methode: (weniger sinnvoll für $\times$)

  I und J sortiert $\rightsquigarrow$ schrittweise in der vorgegebenen Tupelreihenfolge durchlaufen;
  Für $\ltimes_\theta$: $\mathcal{O}(b_I + b_J)$ Plattenzugriffe; $\mathcal{O}(n_I + n_J)$ Gesamtschritte

  Evtl. vorher nötig: Sortieren von I und/oder J (durch Merge-Sort)
  $\mathcal{O}(b_I \cdot \log b_I)$ und/oder $\mathcal{O}(b_J \cdot \log b_J)$ Plattenzugriffe;
  $\mathcal{O}(n_I \cdot \log n_I)$ und/oder $\mathcal{O}(n_J \cdot \log n_J)$ Gesamtschritte

- Hash-Methode: (weniger sinnvoll für $\times$)

  die kleinere der beiden Relationen in Hash-Tabelle;
  Tupel der zweiten Relation finden ihren Vergleichspartner mittels Hash-Funktion;
  bei idealer Hash-Funktion: Aufwand $\mathcal{O}(n_I + n_J)$

Beispiel für Merge-Technik

Berechne $I \bowtie_\theta J$ für Join-Bedingung $\theta := x_1{=}y_4 \wedge x_2{=}y_1$

1. Sortiere I lexikographisch nach "1-te Spalte; 2-te Spalte"
2. Sortiere J lexikographisch nach "4-te Spalte; 1-te Spalte"
3. Seien t und s die ersten Tupel von I und J
4. Falls $(t_1, t_2) < (s_4, s_1)$, so lies nächstes Tupel t aus I.
5. Falls $(t_1, t_2) > (s_4, s_1)$, so lies nächstes Tupel s aus J.
6. Falls $(t_1, t_2) = (s_4, s_1)$, so gib die Tupel (t, s) und (t, s') für alle Nachfolger s' von s in J mit $(s'_4, s'_1) = (s_4, s_1)$ aus
7. Lies nächstes Tupel t aus I und gehe zu Zeile 4.

Aufwand für Zeilen 3–7:

- falls alle Tupel den gleichen Wert in den Spalten 1,2 bzw. 4,1 haben: ca. $n_I \cdot n_J$ Gesamtschritte
- falls alle Tupel aus J in (s_4, s_1) unterschiedliche Werte haben: ca. $n_I + n_J$ Gesamtschritte :-)
- bei Semijoin $\ltimes_\theta$ statt Theta-Join $\bowtie_\theta$ reichen immer $n_I + n_J$ Gesamtschritte

Anfrageauswertung

Proposition 3.4

Das Auswertungsproblem für die relationale Algebra läßt sich in Zeit $(k+n)^{\mathcal{O}(k)}$ *lösen.*

Beweis:

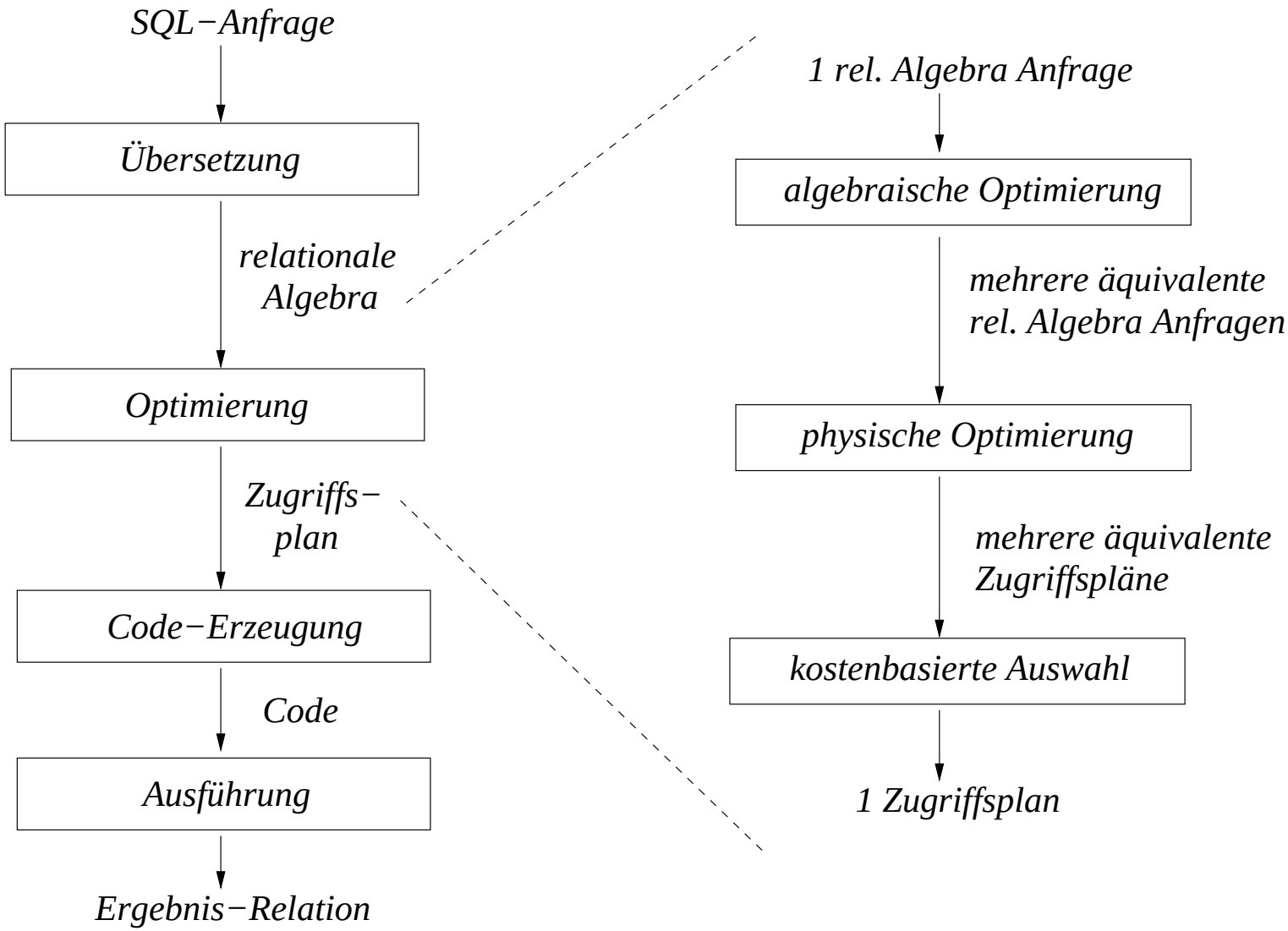
Zeige per Induktion nach dem Aufbau von Anfragen der relationalen Algebra, dass für jede Anfrage Q der Länge k und jede Datenbank $\mathbf{I}$ der Größe n gilt:

(1) $|| [\![Q]\!](\mathbf{I}) || \leqslant (k+n)^k$

(2) Q kann auf $\mathbf{I}$ in $\mathcal{O}\big((k+n)^{2k}\big)$ Elementarschritten ausgewertet werden.

Details: *Übung*.

Anfragebearbeitung durch ein DBMS



Ziel der Optimierung

- möglichst schnelle Auswertung der Anfrage
- möglichst wenige Zugriffe auf Festplatte
- möglichst in allen Operationen so wenig Seiten wie möglich berücksichtigen

Grundregeln:

(1) Selektionen so früh wie möglich

(2) auch Projektionen früh, aber evtl. Duplikatelimination vermeiden

(3) Basisoperationen zusammenfassen und wenn möglich ohne Zwischenspeicherung realisieren
(Bsp: $\bowtie_\theta$ besser als $\times$; $\ltimes_\theta$ besser als $\bowtie_\theta$)

(4) Redundante Operationen oder leere Zwischenrelationen entfernen

(5) Zusammenfassung gleicher Teilausdrücke:
Wiederverwendung von Zwischenergebnissen

Anfrageauswertung an einem Beispiel (1/4)

Anfrage:

Welche Kinos (Name + Adresse) spielen einen Film von “Stephen Spielberg”?

In SQL:

```
SELECT Orte.Kino, Orte.Adresse
FROM Orte, Filme, Programm
WHERE Orte.Kino = Programm.Kino and
      Programm.Titel = Filme.Titel and
      Filme.Regie = “Stephen Spielberg”
```

Direkte Übersetzung in relationale Algebra:

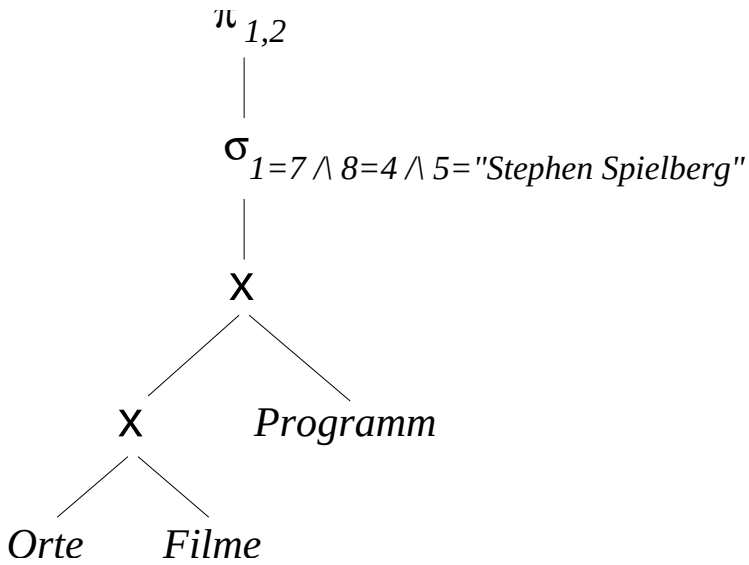
$$\pi_{1,2}\left(\sigma_{\substack{1=7\wedge 8=4\wedge \\ 5=\text{“Stephen Spielberg”}}}\left(\mathit{Orte}\times\mathit{Filme}\times\mathit{Programm}\right)\right)$$

Anfrageauswertung an einem Beispiel (2/4)

Original-Anfrage

$$\pi_{1,2}\left(\sigma_{\substack{1=7\wedge 8=4\wedge \\ 5=\text{“Stephen Spielberg”}}}\left(Orte \times Filme \times Programm\right)\right)$$

dargestellt als Anfrage-Baum:

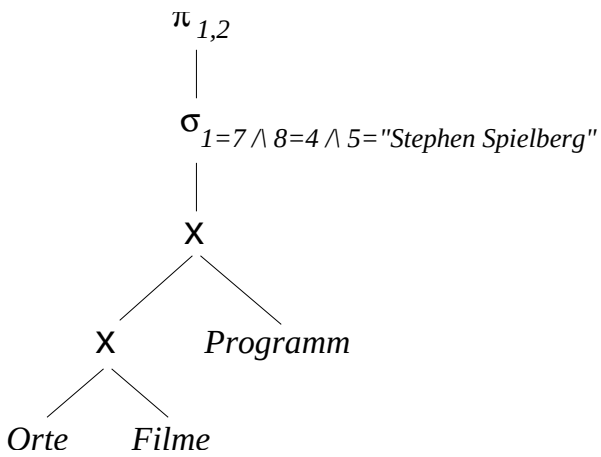


Anfrageauswertung an einem Beispiel (3/4)

Anfrage auswerten auf folgender Beispiel-Datenbank:

- *Filme*: 10.000 Tupel auf 200 Seiten (je 50 pro Seite); je 5 Tupel pro Film, 10 Filme von Stephen Spielberg
- *Programm*: 200 Tupel auf 4 Seiten (je 50 pro Seite); davon 3 Spielberg-Filme in 4 Kinos
- *Orte*: 100 Tupel auf 2 Seiten (je 50 pro Seite)

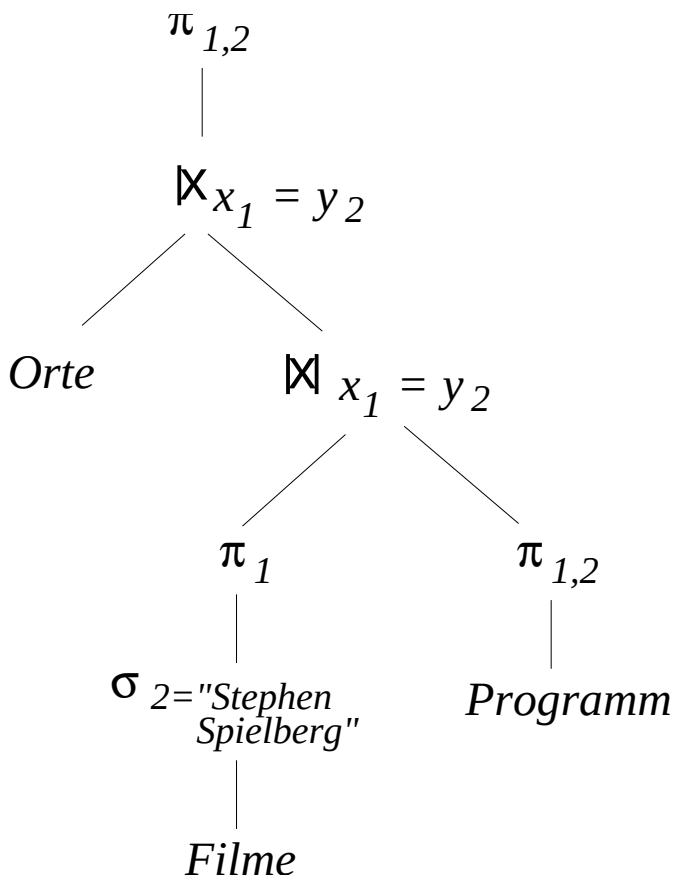
Anfrage-Baum:



Direkte Auswertung dieser Anfrage führt zu über 10.000.000 Plattenzugriffen.

Anfrageauswertung an einem Beispiel (4/4)

Viel besserer Plan:



Auswertung dieses Plans in unserer Beispiel-Datenbank führt zu weniger als 250 Plattenzugriffen.

Heuristische Optimierung (1/3)

- Die heuristische Optimierung wendet allgemeine Regeln zur Umformung einer Anfrage der relationalen Algebra in eine äquivalente Anfrage an, die zu einem vermutlich effizienteren Auswertungsplan führt
- Grundregel: Selektionen so früh wie möglich
- Projektionen auch früh, aber evtl. Duplikatelimination vermeiden
- Anwendung von algebraischen Umformungsregeln
- Ziel: Operationen verschieben, um kleinere Zwischenergebnisse zu erhalten; wenn möglich Redundanzen erkennen

Heuristische Optimierung (2/3)

Einige algebraische Umformungsregeln:

(1) Cartesische Produkte und Joins sind kommutativ und assoziativ:

$Q_1 \bowtie_\theta Q_2 \quad \longleftrightarrow \quad Q_2 \bowtie_{\tilde{\theta}} Q_1$

$(Q_1 \bowtie_{\theta_1} Q_2) \bowtie_{\theta_2} Q_3 \quad \longleftrightarrow \quad Q_1 \bowtie_{\tilde{\theta}_1} (Q_2 \bowtie_{\tilde{\theta}_2} Q_3)$

($\tilde{\theta}$ entsteht aus θ durch "Zurückrechnen" der Spaltennummern)

(2) Ketten von Selektionen (bzw. Projektionen) zusammenfassen:

$\sigma_{F_1}(\sigma_{F_2}(Q)) \quad \longleftrightarrow \quad \sigma_{F_1 \wedge F_2}(Q) \quad \longleftrightarrow \quad \sigma_{F_2}(\sigma_{F_1}(Q))$

$\pi_X(\pi_Y(Q)) \quad \longleftrightarrow \quad \pi_{\tilde{X}}(Q)$

($\tilde{X}$ entsteht aus X durch "Zurückrechnen" der Spaltennummern)

(3) Vertauschen von Selektion und Join:

$\sigma_{F_1 \wedge F_2 \wedge F_3}(Q_1 \times Q_2) \quad \longleftrightarrow \quad \sigma_{F_1}(Q_1) \bowtie_{\theta_3} \sigma_{\tilde{F}_2}(Q_2)$,

wobei die Selektionsbed. F_1 (F_2) sich nur auf Spalten von Q_1 (Q_2) bezieht und F_3 Spalten von Q_1 mit Spalten von Q_2 vergleicht. $\tilde{F}_2$ und θ_3 entstehen aus F_2 und F_3 durch "Zurückrechnen" der Spaltennummern.

(4) Einführung von Semijoins, falls X nur Spalten von Q_1 beinhaltet:

$\pi_X(Q_1 \bowtie_\theta Q_2) \quad \longleftrightarrow \quad \pi_X(Q_1 \ltimes_\theta Q_2)$

Heuristische Optimierung (3/3)

Einige algebraische Umformungsregeln:

(5) Vertauschen von Selektion und Vereinigung bzw. Differenz:

$$\sigma_F(Q_1 \cup Q_2) \quad \longleftrightarrow \quad \sigma_F(Q_1) \cup \sigma_F(Q_2)$$
$$\sigma_F(Q_1 - Q_2) \quad \longleftrightarrow \quad \sigma_F(Q_1) - \sigma_F(Q_2)$$

(6) Analog: Vertauschen von Projektion und Vereinigung bzw. Differenz.

(7) Vertauschen von Projektion und Selektion unter bestimmten Bedingungen

(8) Vertauschen von Projektion und Join unter bestimmten Bedingungen

(9) Löschen von Redundanzen:

$$Q \cup Q \longrightarrow Q, \qquad Q \cap Q \longrightarrow Q, \qquad Q \bowtie Q \longrightarrow Q$$

(10) Löschen leerer Zwischenergebnisse:

$$Q - Q \longrightarrow \emptyset, \qquad Q \cap \emptyset \longrightarrow \emptyset, \qquad Q \cup \emptyset \longrightarrow Q, \qquad Q \bowtie \emptyset \longrightarrow \emptyset,$$
$$Q - \emptyset \longrightarrow Q, \qquad \emptyset - Q \longrightarrow \emptyset$$

(11) . . . usw. . . .

Wunschliste für bessere Optimierung:

- zum “Löschen leerer Zwischenergebnisse”:
  Test, ob eine gegebene (Teil-)Anfrage Q nicht erfüllbar ist (“$Q \equiv \emptyset$”)
- zum “Löschen von Redundanzen”:
  Test, ob zwei (Teil-)Anfragen Q und P äquivalent sind (“$Q \equiv P$”)

In späteren Kapiteln:

- Algorithmen zum Lösen dieser Probleme für konjunktive Anfragen
- Nicht-Entscheidbarkeit dieser Probleme für allgemeinere Anfragen (relationale Algebra)

Vorgehensweise eines Optimierers in einem DBMS

- Erzeugung verschiedener rel. Algebra Ausdrücke:
  unter Verwendung heuristischer Optimierungsregeln
- Erzeugung von verschiedenen Auswertungsplänen:
  Anfrage-Bäume, erweitert um Informationen über zu verwendende Zugriffsstrukturen und Algorithmen für die einzelnen Operationen
- Abschätzung der Kosten für jeden erzeugten Auswertungsplan:
  unter Verwendung von statistischen Informationen über die Daten
  ($\rightsquigarrow$ kostenbasierte Optimierung)
- Auswahl des am günstigsten erscheinenden Plans

Generell:

Je häufiger die Anfrage ausgewertet werden soll, desto mehr Aufwand sollte für die Optimierung verwendet werden.

Join-Reihenfolge

- Joins sind kommutativ und assoziativ
  $\rightsquigarrow$ Änderung der Klammerung bzw. Reihenfolge von Join-Operationen ändert nicht das Ergebnis der Anfrage (modulo Ändern der Spalten-Reihenfolge)
- Aber: Die Größe der Zwischenergebnisse und somit der Auswertungs-Aufwand kann sich drastisch ändern.
- Unter Umständen lässt sich sogar die Anzahl der Joins verkleinern (mehr dazu in Kapitel 5)

- Klassische Vorgehensweise in DBMS: Nur Auswertungspläne betrachten, die Joins von links nach rechts klammern ("left-deep-trees")
  $\rightsquigarrow$ durch Umordnen sind immerhin alle Reihenfolgen möglich (immer noch exponentiell viele Möglichkeiten)
- (Es ist bekannt, dass diese Einschränkung nicht immer sinnvoll und nötig ist)

Jetzt: Heuristische Join-Optimierung bzgl. left-deep-trees

Beispiele (1/3)

Beispiel 3.5

R : 2-stellige Relation mit Attributen A, B und 1.000 Tupeln

S : 2-stellige Relation mit Attributen C, D und 10.000 Tupeln

T : 2-stellige Relation mit Attributen B, C und 100 Tupeln

Pro Tupel $(b, c) \in T$ gibt es ein Tupel $(\cdot, b) \in R$ und ein Tupel $(c, \cdot) \in S$.

Anfrage: $Ans(x) \leftarrow R(x, x_1),\ S(x_2, x_3),\ T(x_1, x_2)$

Aufwand bei Links-nach-rechts-Auswertung:

10.000.000 Tupel nach erstem Join, 100 Tupel nach zweitem Join

---

Andere Join-Reihenfolge: $Ans(x) \leftarrow R(x, x_1),\ T(x_1, x_2),\ S(x_2, x_3)$

Aufwand bei Links-nach-rechts-Auswertung:

100 Tupel nach erstem Join, 100 Tupel nach zweitem Join

Beispiele (2/3)

Beispiel 3.6

R : 2-stellige Relation mit Attributen A, B und 1.000 Tupeln

S : 2-stellige Relation mit Attributen C, D und 10.000 Tupeln

T : 2-stellige Relation mit Attributen B, C und 100 Tupeln

Pro Tupel $(b, c) \in T$ gibt es ein Tupel $(\cdot, b) \in R$ und ein Tupel $(c, \cdot) \in S$.

Für die Konstante d gibt es nur 1 Tupel $(\cdot, d)$ in S.

Anfrage: $Ans(x) \leftarrow R(x, x_1),\ T(x_1, x_2),\ S(x_2, d)$

Aufwand bei Links-nach-rechts-Auswertung:

100 Tupel nach erstem Join, 1 Tupel nach zweitem Join

Andere Join-Reihenfolge: $Ans(x) \leftarrow S(x_2, d),\ T(x_1, x_2),\ R(x, x_1)$

Aufwand bei Links-nach-rechts-Auswertung:

1 Tupel nach erstem Join, 1 Tupel nach zweitem Join

Beispiele (3/3)

Noch ein Beispiel:

- Auswertung von links nach rechts
- Anfrage: $Ans(z) \leftarrow R(v,w,y),\ v \leqslant x,\ S(x,y,z),\ P(a,v),\ Q(b,w,x)$
- Besserer Auswertungsplan:
  $Ans(z) \leftarrow P(a,v),\ Q(b,w,x),\ R(v,w,y),\ S(x,y,z),\ v \leqslant x,$
- Denn:
  - wahrscheinlich wenige Tupel der Form $(a,\cdot)$ in P
  - wahrscheinlich wenige Tupel der Form $(b,\cdot,\cdot)$ in Q
  - wahrscheinlich wenige Tupel, die $P(a,v),\ Q(b,w,x),\ R(v,w,y)$ erfüllen

Heuristik ("Sideways-Information-Passing"):

- Relations-Atome mit Konstanten zuerst auswerten
- Wenn möglich, Relations-Atome erst dann, wenn weiter links schon eine ihrer Variablen steht.
- Vergleichsoperatoren ($\leqslant$, $<$) möglichst erst dann verwenden, wenn beide Variablen schon verwendet wurden.

SIP-Graph und SIP-Strategie (1/2)

Definitionen:

- Regel $Ans(u) \leftarrow R_1(u_1), \ldots, R_k(u_k), E_1, \ldots, E_\ell, C_1, \ldots, C_m$
  wobei E_i Vergleich mit $=$; C_i Vergleich mit $<$ oder $\leqslant$ (dafür sei $<$ eine lineare Ordnung auf **dom**).
- SIP-Graph
  - Knotenmenge: Rel.-Atome $R_1(u_1), \ldots, R_k(u_k)$ und "="-Atome $E_1, \ldots, E_\ell$
  - Kante zwischen zwei Knoten, falls diese (mind.) eine Variable gemeinsam haben
  - Knoten ist markiert, falls er (mind.) eine Konstante enthält
- Falls der SIP-Graph zusammenhängend ist, so ist eine SIP-Strategie eine Anordnung $A_1, \ldots, A_{k+\ell+m}$ der Atome, so dass für jedes $j > 1$ gilt:
  - A_j ist ein markierter Knoten des SIP-Graphen, oder
  - A_j ist ein Knoten des SIP-Graphen und es gibt ein $j' < j$, so dass es im SIP-Graph eine Kante zwischen A_j und $A_{j'}$ gibt, oder
  - A_j ist ein C_i und für jede Variable x in C_i gibt es $j' < j$, so dass $A_{j'}$ ein Knoten des SIP-Graphen ist, in dem x vorkommt
- Außerdem: Falls ex. $R_i(u_i)$ od. E_i mit einer Konstanten, so ist A_1 ein solches Atom
- Falls der SIP-Graph nicht zusammenhängend ist: SIP-Strategie für jede Zusammenhangskomponente.

SIP-Graph und SIP-Strategie (2/2)

Beispiele:

- $Ans(z) \leftarrow P(a,v),\ Q(b,w,x),\ R(v,w,y),\ S(x,y,z),\ v \leqslant x$
  ist in der Reihenfolge einer SIP-Stragegie
- $Ans(z) \leftarrow R(v,w,y),\ v \leqslant x,\ S(x,y,z),\ P(a,v),\ Q(b,w,x)$
  ist nicht in der Reihenfolge einer SIP-Stragegie

Bemerkungen:

- Zu jeder regelbasierten konjunktiven Anfrage (evtl. mit $=$ und $\leqslant$; dann aber bereichsbeschränkt) gibt es eine SIP-Strategie.
- Eine SIP-Strategie für eine gegebene Anfrage lässt sich in polynomieller Zeit berechnen.
- Wird eine Variable weiter rechts nicht mehr benötigt, so kann sie aus dem Zwischenergebnis “heraus projiziert” werden.