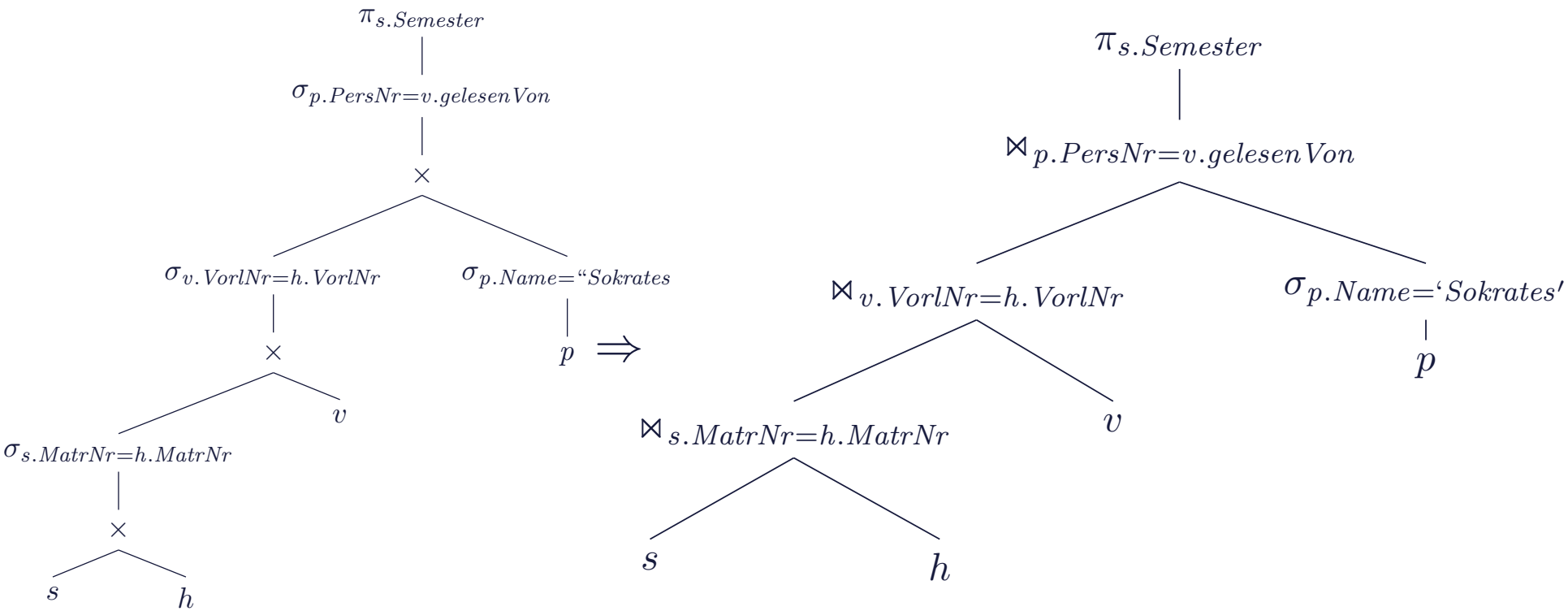
Datenbanksysteme

Wintersemester 2015/16

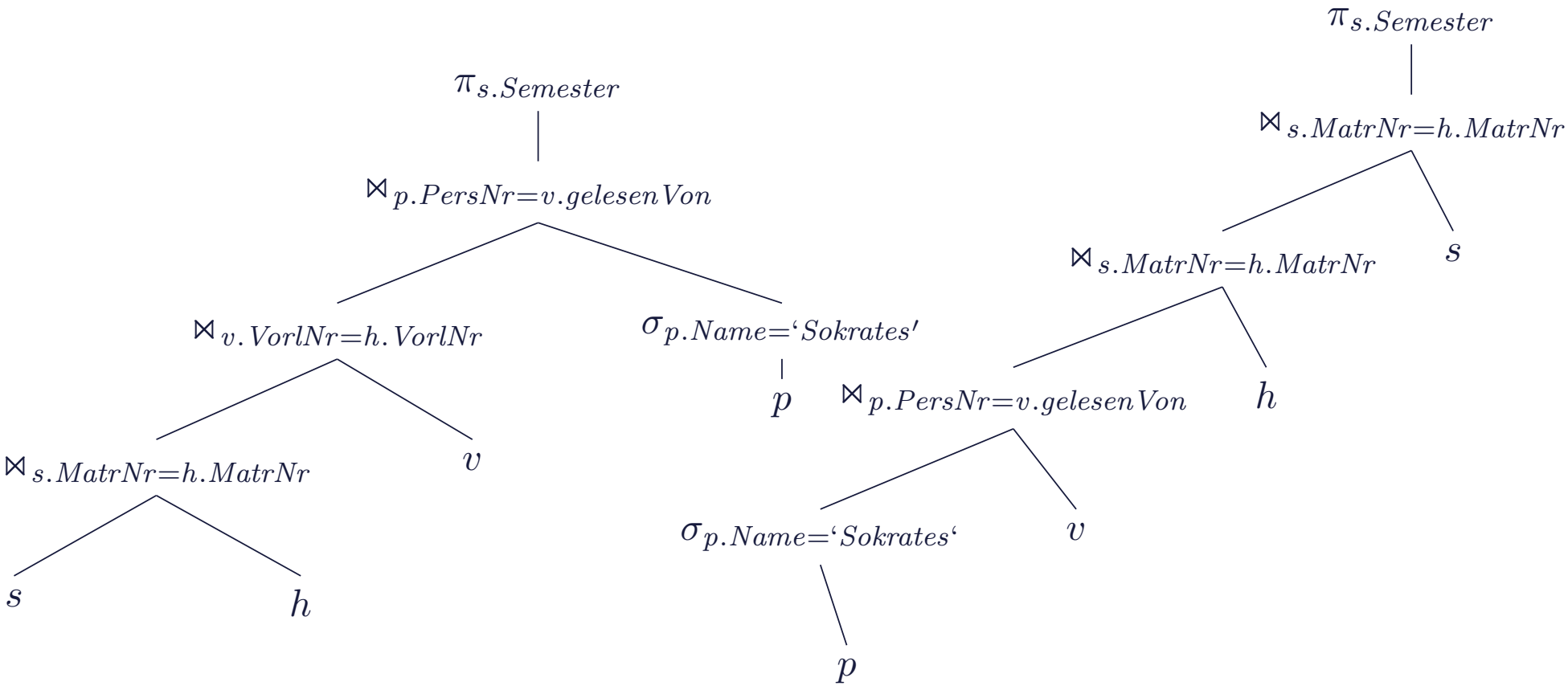
Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

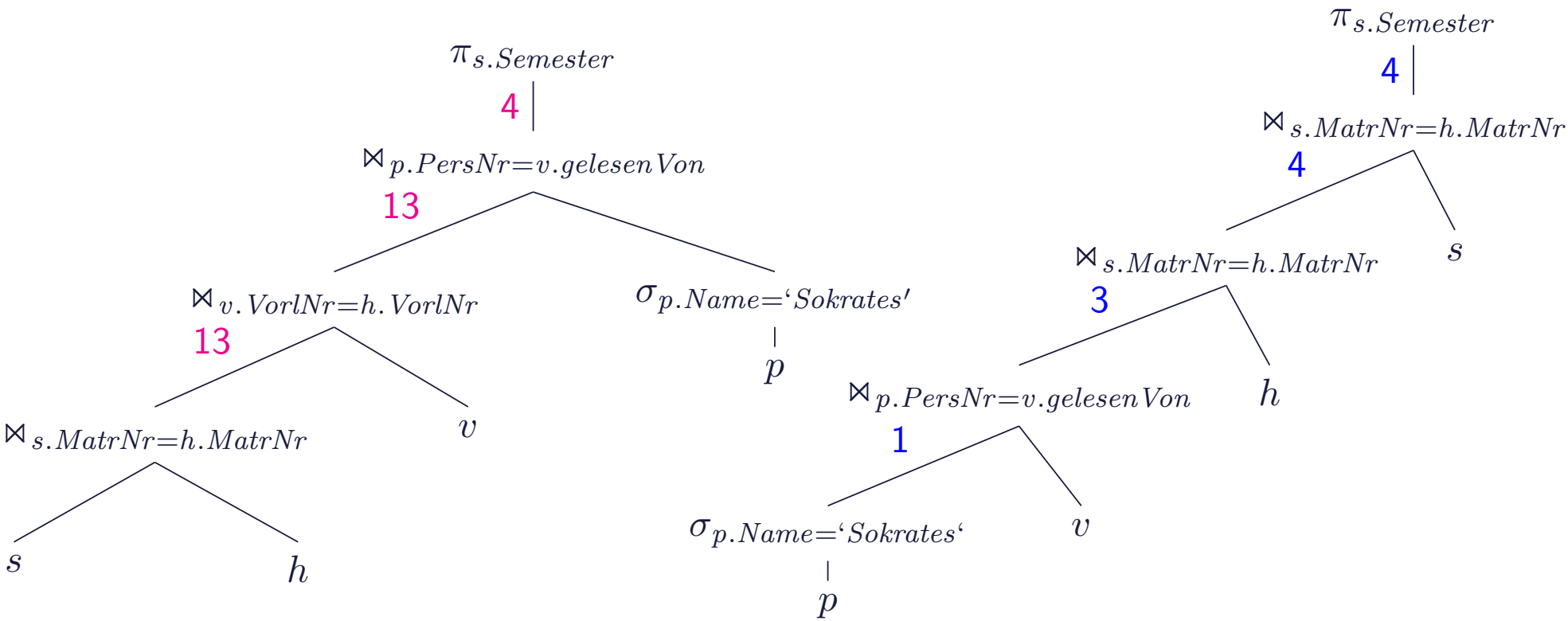
Wiederholung: Beispiel - Zusammenfassung von Selektionen und Kreuzprodukten zu Joins



Wiederholung: Beispiel - Optimierung der Joinreihenfolge



Wiederholung: Beispiel - Effekt: Reduzierung der Zwischenergebnisse



Diese Zwischenkosten müssen natürlich geschätzt werden. Der Optimierer kann dann den günstigsten Plan bzgl. dieser geschätzten Kosten auswählen .

Anfrageoptimierung - Join Ordering

Literatur hierzu, Übersicht im Buch (Under Construction): "Building Query Compilers" von Guido Moerkotte (Uni Mannheim) (enthält Verweise auf Originalarbeiten). Großteil der Folien im Folgenden basierend auf Folien von Thomas Neumann (TUM) – basierend auf diesem Buch.

Problemstellung und Setup

- Wir haben bereits gesehen, dass der **Join-Operator kommutativ und assoziativ** ist, d.h. $R \bowtie S = S \bowtie R$ und $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- Wir betrachten nun in welcher Reihenfolge die Joins eines Anfrageplans ausgeführt (geordnet) werden sollen bzw. können.

Die beteiligten Relationen sind $R_1, \ldots, R_n$ und wir betrachten Anfragen der Form:

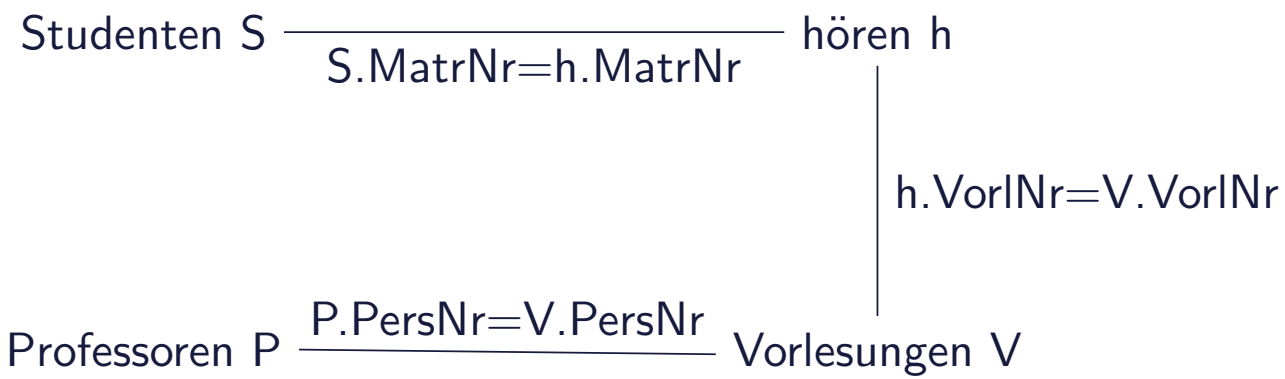
- Selektionen sind Konjunktionen über einfachen Prädikate der Form $x = y$

select ...
from $R_1, \ldots, R_n$
where $R_1.a = R_2.b$ **and** $R_1.a = R_3.c$ ...

Anfragegraph

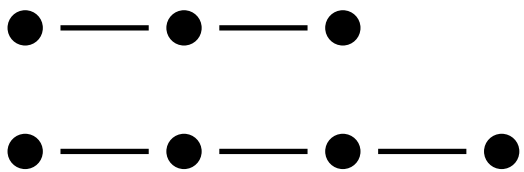
Anfragen dieses Typs können als Graph dargestellt werden:

- Der **Anfragegraph ist ein ungerichteter Graph** mit $R_1, \ldots, R_n$ als Knoten
- Ein **Pädiktat** der Form $a_1 = a_2$, wobei $a_1 \in R_i$ und $a_2 \in R_j$ **erzeugt** eine Kante zwischen R_i und R_j, beschriftet mit dem Prädikat

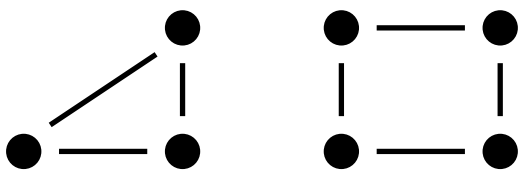


Für zwei Relationen, die nicht via einer Kante verbunden sind, kann nur ein Kreuzprodukt berechnet werden. Wir unterscheiden später ob Kreuzpodukte überhaupt zugelassen werden oder nicht.

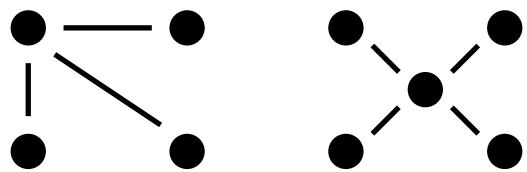
Formen von Anfragegraphen



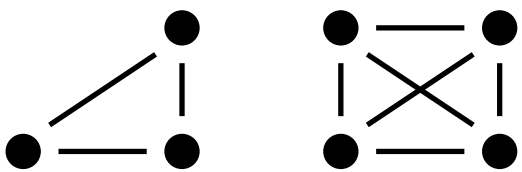
Ketten (chains)



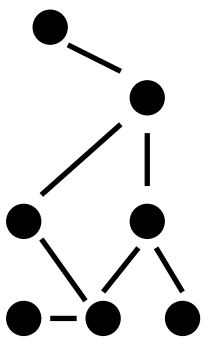
Ringe (cycles)



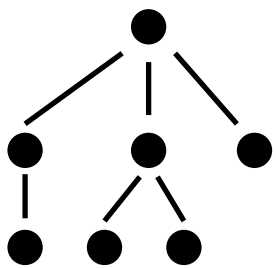
Sterne (stars)



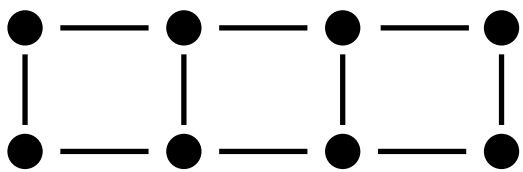
(cliques)



(cyclic)



Baum (tree)



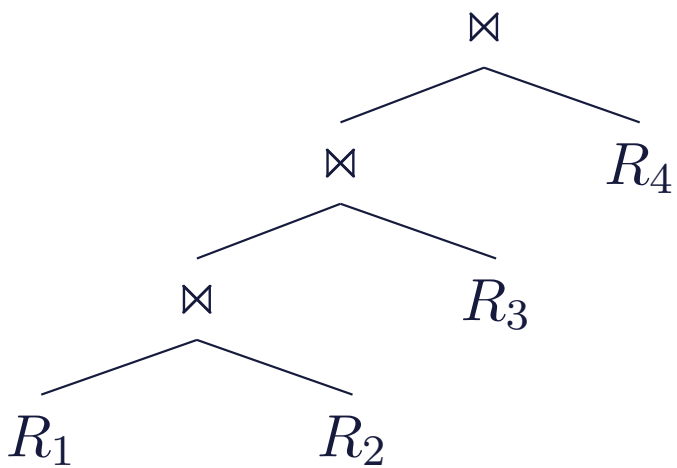
(grid)

Join-Baum

Ein **Join-Baum ist ein Binärbaum** mit

- Join-Operatoren als innere Knoten
- Relationen als Blätter

Beispiel:

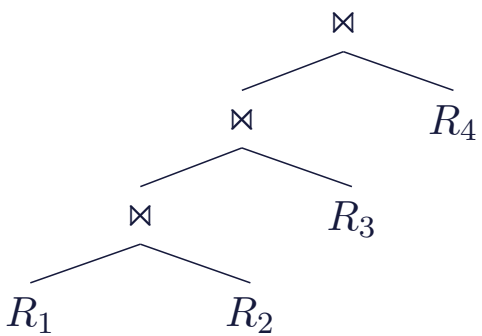


Beschreibt eine tatsächliche Realisierung (Ordnung!) des Joins über den beteiligten Relationen eines Anfragegraphen.

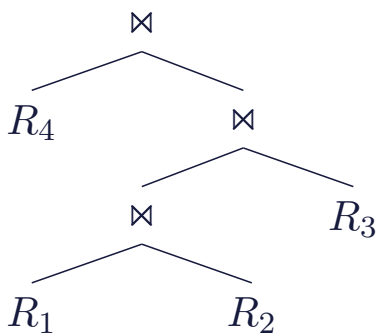
Gestalt von Join-Bäumen

- **links-tiefer Baum**
- **rechts-tiefer Baum**
- **zigzag Baum** (mindestens eine Eingabe ist eine Relation)
- **buschiger (bushy) Baum**

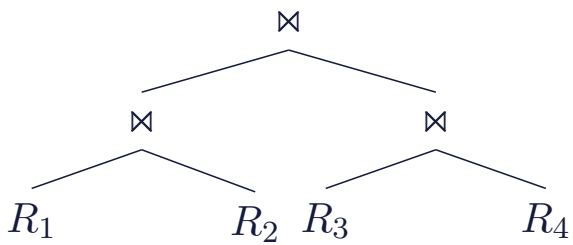
Die ersten drei Klassen werden auch zusammengefasst als **lineare Bäume**.



(links-tief)



(zigzag)



(buschig)

Selektivität von Joins

Eingabe:

- Kardinalitäten $|R_i|$
- Selektivitäten $f_{i,j}$: falls $p_{i,j}$ das Join-Präditkat zwischen R_i und R_j ist dann definieren wir

$$f_{i,j} = \frac{|R_i \bowtie_{p_{i,j}} R_j|}{|R_i \times R_j|}$$

Berechne:

- Kardinalität des Ergebnisses:

$$|R_i \bowtie_{p_{i,j}} R_j| = f_{i,j} * |R_i| * |R_j|$$

Idee dahinter: Die Selektivität kann (idealerweise) recht einfach berechnet/geschätzt werden.

Kardinalität für Join-Bäume

Gegeben ein Join-Baum T, die **Kardinalität des Anfrageergebnisses** $|T|$ kann rekursiv berechnet werden durch

$$|T| = \begin{cases} |R_i| & \text{falls } T \text{ ein Blatt } R_i \text{ ist} \\ \left(\prod_{R_i \in T_1, R_j \in T_2} f_{i,j}\right) * |T_1| * |T_2| & \text{falls } T = T_1 \bowtie T_2 \end{cases}$$

- Erlaubt einen einfache Berechnung der Kardinalität eines Joins
- Benötigt nur die Kardinalitäten der zugrunde liegenden Relationen und Selektivitäten
- **Setzt Unabhängigkeit der Prädikate voraus**

Kostenfunktion

Gegeben ein Join-Baum T, dann ist die **Kostenfunktion** C_{out} definiert als

$$C_{out}(T) = \begin{cases} 0 & \text{falls } T \text{ ist ein Blatt } R_i \\ |T| + C_{out}(T_1) + C_{out}(T_2) & \text{falls } T = T_1 \bowtie T_2 \end{cases}$$

- **Addiert die Größen der (Zwischen)ergebnisse auf**
- Idee dahinter: Größere Zwischenergebnisse erfordern mehr Arbeit
- Die Kosten der einzelnen Relationen werden hier weggelassen (da sie sowieso gelesen werden müssen)

Kostenfunktion für Joins

Für einzelne Joins

$$\begin{aligned}
\textbf{Nested-Loops Join:} C_{nlj}(e_1 \bowtie e_2) &= |e_1||e_2| \\
\textbf{Hash-Join:} C_{hj}(e_1 \bowtie e_2) &= 1.2|e_1| \\
\textbf{Sort-Merge-Join:} C_{smj}(e_1 \bowtie e_2) &= |e_1|\log(|e_1|) + |e_2|\log(|e_2|)
\end{aligned}$$

Für Sequenzen von Join-Operatoren $s = s_1 \bowtie \ldots \bowtie s_n$**:**

$$\begin{aligned}
C_{nlj}(s) &= \sum_{i=2}^{n} |s_1 \bowtie \ldots \bowtie s_{i-1}||s_i| \\
C_{hj}(s) &= \sum_{i=2}^{n} 1.2|s_1 \bowtie \ldots \bowtie s_{i-1}| \\
C_{smj}(s) &= \sum_{i=2}^{n} |s_1 \bowtie \ldots \bowtie s_{i-1}|\log(|s_1 \bowtie \ldots \bowtie s_{i-1}|) + \sum_{i=2}^{n} |s_i|\log(|s_i|)
\end{aligned}$$

Anmerkungen zu diesen Kostenfunktionen

- Kostenfunktionen sehr einfach
- Join-Implementierungen sind sehr einfach modelliert (z.B. Faktor 1.2, kein n-way sort/merge)
- Designed für links-tiefe Bäume
- C_{hj} und C_{smj} funktionieren nicht für Kreuzprodukte (Korrektur dafür: Betrachte Kardinalität der Ausgabe, d.h C_{nl})
- Kostenfunktionen nehmen an, dass die gleiche Join-Implementierung (Algorithmus) für den gesamten Join-Baum benutzt wird

Beispiel Statistiken und Hinweis zu Kreuzprodukten

Als im Folgenden verwendetes Beispiel nehmen wir an:

$$\begin{aligned} |R_1| &= 10 \\ |R_2| &= 100 \\ |R_3| &= 1000 \\ f_{1,2} &= 0.1 \\ f_{2,3} &= 0.2 \end{aligned}$$

- Daraus folgt der Anfragegraph $R_1 - R_2 - R_3$

Hinweis zu Kreuzprodukten

- Relationen, die nicht durch Kanten verbunden sind können nur mit Hilfe eines Kreuzproduktes (d.h. $f_{i,j} = 1$) berechnet werden.
- Das Zulassen von Kreuzprodukten vergrößert den Suchraum. Ob diese zugelassen werden oder nicht ist später essenziell bei Algorithmen und Betrachtung der Größe des Suchraums.

Beispiel für Kostenberechnung

	C_{out}	C_{nl}	C_{hj}	C_{smj}
$R_1 \bowtie R_2$	100	1000	12	697.61
$R_2 \bowtie R_3$	20000	100000	120	10630.26
$R_1 \times R_3$	10000	10000	10000	10000.00
$(R_1 \bowtie R_2) \bowtie R_3$	20100	101000	132	11327.86
$(R_2 \bowtie R_3) \bowtie R_1$	40000	300000	24120	32595.00
$(R_1 \times R_3) \bowtie R_2$	30000	1010000	22000	143542.00

Beobachtungen:

- Kosten variieren sehr stark
- Join-Bäume mit Kreuzprodukten sind sehr teuer
- Join-Ordnung essenziell

Weitere Beispiele

Für $|R_1| = 1000, |R_2| = 2, |R_3| = 2, f_{1,2} = 0.1, f_{1,3} = 0.1$
haben wir die Kosten

	C_{out}
$R_1 \bowtie R_2$	200
$R_2 \times R_3$	4
$R_1 \bowtie R_3$	200
$(R_1 \bowtie R_2) \bowtie R_3$	240
$(R_2 \times R_3) \bowtie R_1$	44
$(R_1 \bowtie R_3) \bowtie R_2$	240

- Hier ist der Baum mit Kreuzprodukt am besten
- Aber nur weil die Kardinalitäten von $|R_2|$ und $|R_3|$ sehr klein
- Kann daher durchaus (im Allgemeinen) eine attraktive Lösung sein, eben wenn Kardinalitäten klein

Weitere Beispiele

Für $|R_1| = 10, |R_2| = 20, |R_3| = 20, |R_4| = 10, f_{1,2} = 0.01,$ $f_{2,3} = 0.5, f_{3,4} = 0.01$ haben wir die Kosten

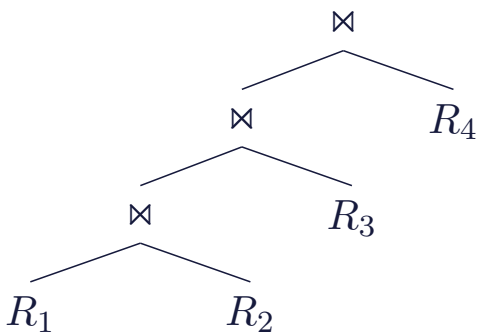
	C_{out}
$R_1 \bowtie R_2$	2
$R_2 \bowtie R_3$	200
$R_3 \bowtie R_4$	2
$((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$	24
$((R_2 \times R_3) \bowtie R_1) \bowtie R_4$	222
$(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$	6

- Der buschige Baum ist hier besser als alle anderen Möglichkeiten

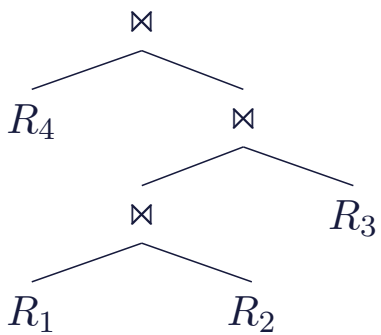
Wiederholung: Gestalt von Join-Bäumen

- **links-tiefer Baum**
- **rechts-tiefer Baum**
- **zigzag Baum** (mindestens eine Eingabe ist eine Relation)
- **buschiger (bushy) Baum**

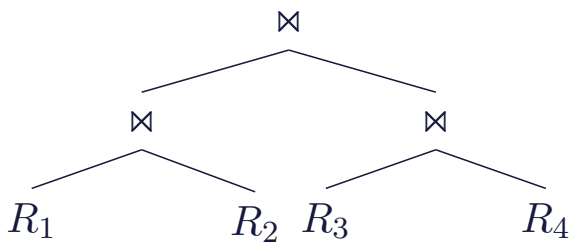
Die ersten drei Klassen werden auch zusammengefasst als **lineare Bäume**.



(links-tief)



(zigzag)



(buschig)

Klassifikation der Join-Ordering-Probleme

Die *hier* betrachteten Probleme können anhand der folgenden Kriterien klassifiziert werden:

1. **Anfragegraph:** *Kette*, *Cycle*, *Stern* und *Clique*
2. **Struktur des Join-Baums**: *links-tief*, *zigzag* oder *buschig* Bäume
3. **Kreuzprodukte:** *mit* oder *ohne* Kreuzprodukte

Catalan-Zahl

Die Anzahl von Binärbäumen mit n Blättern ist gegeben durch $\mathcal{C}(n-1)$, wobei $\mathcal{C}(n)$ definiert ist durch

$$\mathcal{C}(n) = \begin{cases} 1 & \text{if } n = 0 \\ \sum_{k=0}^{n-1} \mathcal{C}(k)\mathcal{C}(n-k-1) & \text{if } n > 0 \end{cases}$$

Dies kann in geschlossener Form geschrieben werden als

$$\mathcal{C}(n) = \frac{1}{n+1}\binom{2n}{n}$$

Die Catalan-Zahlen wachsen in der Ordnung von $\Theta(4^n/n^{\frac{3}{2}})$

Anzahl Join-Bäume mit Kreuzprodukte

links-tief		$n!$
rechts-tief		$n!$
zigzag		$n!2^{n-2}$
buschig		$n!\mathcal{C}(n-1)$
	$=$	$\frac{(2n-2)!}{(n-1)!}$

- Idee: Anzahl der Kombination der Blätter $(n!) * $ Möglichkeiten einen Baum zu bilden
- (bei zigzag im Vergleich zu links bzw. rechtstief: Vertauschen von Eingaben möglich, d.h. Join oder Relation links oder rechts)
- Wächst exponentiell
- und je flexibler die Baumstruktur ist, desto schneller

Ketten Anfragen, keine Kreuzprodukte

Wir bezeichnen mit $f(n)$ die Anzahl links-tiefer Bäume für eine Ketten Anfrage $R_1 - \ldots - R_n$

- Offensichtlich gilt $f(0) = 1, f(1) = 1$
- Für $n > 1$, R_n kann zu allen Bäumen für $R_1 - \ldots - R_{n-1}$ hinzugefügt werden
- R_n kann also an jede Position nach(!) R_{n-1} eingefügt werden
- Sei k (in $[1,n-1]$) die Position von R_{n-1} von unten betrachtet
- Es gibt $n - k$ Join-Bäume für Einfügen von R_n nach R_{n-1}
- Plus einen Weiteren für $k = 1$, da R_n auch vor R_{n-1} eingefügt werden kann
- Wenn R_{n-1} auf Platz k, dann $f(k-1)$ Bäume darunter.

Also insgesamt $(n > 1)$:

$$f(n) = 1 + \sum_{k=1}^{n-1} f(k-1) * (n-k)$$

Ketten-Anfragen, keine Kreuzprodukte (2)

Die **Anzahl für links-tiefe Bäume für Ketten-Anfragen der Größe n ist**

$$f(n) = \begin{cases} 1 & \text{if } n < 2 \\ 1 + \sum_{k=1}^{n-1} f(k-1) * (n-k) & \text{if } n \geq 2 \end{cases}$$

Kann in geschlossener Form geschrieben werden als

$$f(n) = 2^{n-1}$$

- **Generalisierung für zigzag wie zuvor**

Ketten-Anfragen, keine Kreuzprodukte (3)

Die Generalisierung zu **buschigen Bäumen** ist nicht so trivial:

- jeder Teilbaum muss eine Teil-Kette enthalten um Kreuzprodukte zu vermeiden
- Kette $R_1 - \ldots - R_n$ muss überall geteilt werden
- Kommutativität ist zu berücksichtigen (=2 Möglichkeiten)

Das führt zu folgender Formel:

$$f(n) = \begin{cases} 1 & \text{if } n < 2 \\ \sum_{k=1}^{n-1} 2f(k)f(n-k) & \text{if } n \geq 2 \end{cases}$$

In geschlossener Form:

$$f(n) = 2^{n-1}\mathcal{C}(n-1)$$

Stern-Anfragen, keine Kreuzprodukte

Betrachten wir eine Stern-Anfrage R_1 in der Mitte und $R_2, \ldots, R_n$ als "Satelliten" außen herum.

- **Klar, der erste Join muss R_1 enthalten.**
- **Danach können die restlichen Relationen beliebig hinzugefügt werden**

Das führt zu den folgenden Formeln:

- links-tief: $2 * (n-1)!$
- zigzag: $2 * (n-1)! * 2^{n-2} = (n-1)! * 2^{n-1}$
- buschig: Es sind keine buschigen Bäume möglich (R_1 muss immer vorhanden sein damit Join möglich), also gleiche Anzahl wie bei zigzag. Achtung, dies meint hier buschige Bäume, die nicht linear sind.

Clique-Anfragen, keine Kreuzprodukte

- **In einer Clique-Anfrage kann jede Relation mit jeder anderen verbunden werden**
- Es gibt hier gar keine Bäume die Kreuzprodukte enthalten
- Die Anzahl ist die gleiche wie mit Kreuzprodukten

Beispiel Zahlen, ohne Kreuzprodukte

	Ketten-Anfragen			Stern-Anfragen	
n	Links-Tief 2^{n-1}	ZigZag 2^{2n-3}	Buschig $2^{n-1}\mathcal{C}(n-1)$	Links-Tief $2(n-1)!$	ZigZag/Buschig $2^{n-1}(n-1)!$
1	1	1	1	1	1
2	2	2	2	2	2
3	4	8	8	4	8
4	8	32	40	12	48
5	16	128	224	48	384
6	32	512	1344	240	3840
7	64	2048	8448	1440	46080
8	128	8192	54912	10080	645120
9	256	32768	366080	80640	10321920
10	512	131072	2489344	725760	18579450

Beispiel Zahlen, mit Kreuzprodukte

n	**Links-Tief** $n!$	**ZigZag** $n!2^{n-2}$	**Buschig** $n!\mathcal{C}(n-1)$
1	1	1	1
2	2	2	2
3	6	12	12
4	24	96	120
5	120	960	1680
6	720	11520	30240
7	5040	161280	665280
8	40320	2580480	17297280
9	362880	46448640	518918400
10	3628800	968972800	17643225600