TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

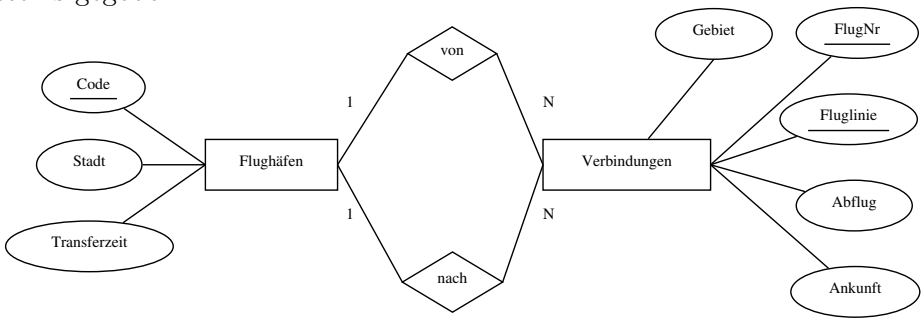
Übung zur Vorlesung *Grundlagen: Datenbanken* im WS20/21

Christoph Anneser, Josef Schmeißer, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws2021/grundlagen/

Blatt Nr. 10

Hausaufgabe 1

Betrachten Sie die Anfrage *„Finde alle Flüge von New-York nach Sydney mit einmaligem Umsteigen“*. Dazu sei das nachfolgende (vereinfachte) ER-Diagramm eines Fluginformationssystems gegeben:



a) Geben Sie eine SQL-Query für die oben genannte Anfrage an.

b) Führen Sie die kanonische Übersetzung des SQL-Statements in die relationale Algebra durch.

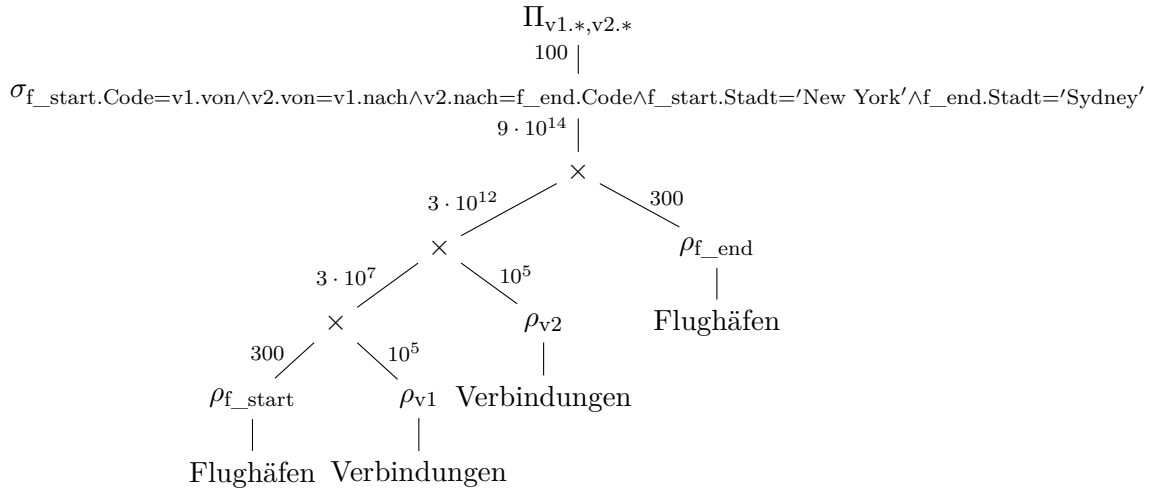
c) Schätzen Sie die Relationsgrößen sinnvoll ab (z.B. so wie in den Beispielen der Vorlesung) und transformieren Sie den kanonischen Operatorbaum aus Teilaufgabe b) zur optimalen Form. Wie haben sich die Kosten dabei geändert? (Kosten = Anzahl der Zwischenergebnistupel)

Lösung:

a) SQL-Anfrage:

```
SELECT DISTINCT v1.*, v2.*
FROM Flughäfen f_start, Verbindungen v1, Verbindungen v2, Flughäfen f_end
WHERE f_start.Stadt = "New York"
  AND f_end.Stadt = "Sydney"
  AND v2.von = v1.nach
  AND v2.nach = f_end.Code
  AND f_start.Code = v1.von
```

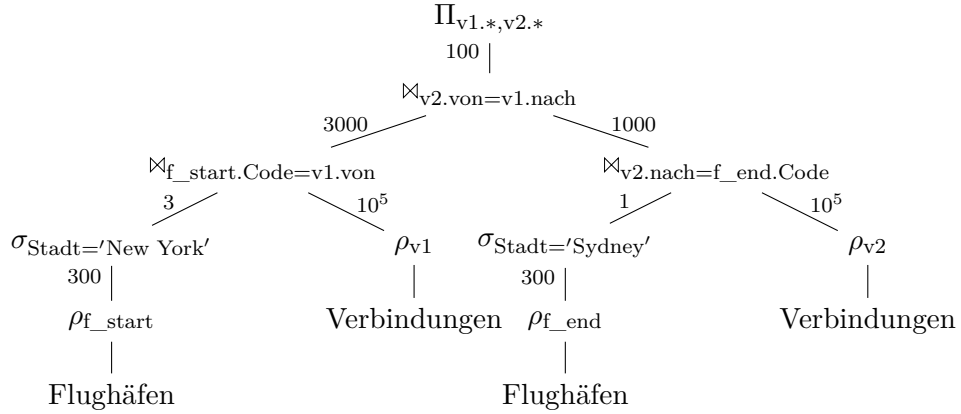
b) **Kanonische Übersetzung:**
Wir gehen davon aus, dass es weltweit 300 (relevante) Flughafen gibt und 100.000 Verbindungen zwischen Flughäfen. Weiterhin nehmen wir an, dass es 100 Möglichkeiten gibt, über einen Zwischenstop von New York nach Sydney zu fliegen. Andere Werte sind natürlich auch in Ordnung. Hier reicht es aus, grob zu schätzen.



Diese Anfrage kostet über 900 Billionen Tupel. Wenn eine hypotetische CPU mit 3 GHz ein Tupel pro Takt verarbeiten könnte, dann liefe die Anfrage über 3 Tage und 11 Stunden.

c) **Optimaler Ausführungsplan:**

Wir gehen hier zusätzlich davon aus, dass es drei Flughafen in New York gibt und einen in Sydney. Weiterhin gehen wir davon aus, dass es von jedem New Yorker Flughafen sowie zu dem Flughafen in Sydney jeweils 1000 Verbindungen gibt.



Die optimierte Anfrage kostet nur ca. 205000 Tupel. Unsere hypotetische 3 GHz-CPU würde hier nur noch 68 Mikrosekunden benötigen.

Hausaufgabe 2

a) Was ist ein Equi-Join?

b) Bei welchen Join-Prädikaten ($<, =, >$) kann man sinnvoll einen Hashjoin einsetzen?

c) Gegeben die Relation Profs = $\{\underline{\text{PersNr}}, \text{Name}\}$ und Raeume = $\{\underline{\text{PersNr}}, \text{RaumNr}\}$.

1) Skizzieren Sie eine geschickte Möglichkeit, den Equi-Join Profs $\bowtie$ Raeume durchzuführen.

2) In welchem Fall wäre selbst ein Ausdruck wie

$$\text{Profs} \bowtie_{\text{Profs.Persnr} < \text{Raeume.PersNr}} \text{Raeume}$$

effizient auswertbar?

d) Der Student Maier hat einen Algorithmus gefunden, der den Ausdruck $A \times B$ in einer Laufzeit von $O(|A|)$ materialisiert. Was sagen Sie Herrn Maier?

Lösung:

a) Ein Equi-Join hat eine Äquivalenz als Joinbedingung, etwa die Gleichheit zweier Attribute.

b) Ein Hash Join bietet sich nur für Equi-Joins an, da lediglich ein Join-Partner mit gleichem Attributwert effizient auffindbar ist. Das Finden eines Partners, dessen Attributwert beispielsweise kleiner sein soll kann mittels Hashing i.A. nicht effizient bearbeitet werden.

c) 1) Offenbar ist das Joinattribut gerade der Primärschlüssel, womit von der Existenz eines Indexes ausgegangen werden kann. Somit bietet sich ein Index-basierter Join an, etwa dadurch, dass die eine Relation Element für Element abgearbeitet wird, während Joinpartner aus der anderen Relation mittels des Indexes gefunden werden.

2) Falls der Index sortiert ist, dies wäre etwa bei einem B-Baum der Fall. Dadurch liegen Joinpartner zumindest nacheinander im Index, anders als bei einer Implementierung des Indexes mittels Hash.

d) Dies ist mit Sicherheit nicht der Fall, da ein Algorithmus keine bessere Komplexitätsklasse haben kann als sein Ergebnis wächst. Mit anderen Worten, $A \times B$ hat eine Ergebnisgröße von $|A| \cdot |B|$ und dieses Ergebnis kann sicher nicht schneller als in $O(|A| \cdot |B|)$ materialisiert werden.

Hausaufgabe 3

Sie verwenden ein Datenbanksystem, welches die folgenden Joinalgorithmen unterstützt:

- (Blockwise-)Nested-Loop-Join
- Index-Join
- Sort-Merge-Join
- Hash-Join

Geben Sie für jeden der vier Algorithmen ein Beispiel an, bei dem dieses Datenbanksystem diesen Algorithmus verwenden würde. Geben Sie dazu für jedes Beispiel ein Schema und eine SQL-Anfrage an. Begründen Sie, warum der gewählte Joinalgorithmus den anderen vorgezogen wird.

Lösung:

- (Blockwise-)Nested-Loop-Join: Schema: Relationen R und S. Die Attribute der Relationen sind beliebig.

  Anfrage: `select * from R, S`

Begründung: Die Anfrage enthält ein Kreuzprodukt. Also ist die Kardinalität der Ergebnismenge zwangsläufig $|R \times S|$. Damit haben alle Joinalgorithmen die selbe (theoretische) Laufzeit. Um sich den unnötigen Overhead des Sortierens oder des Hashings zu sparen, verwendet die Datenbank daher einen Nested-Loop-Join.

- Index-Join: Schema: Relationen: $R : \{[\underline{a}]\}$ und $S : \{[b]\}$.

  Anfrage: `select * from R, S where a = b`

  Begründung: Da a in R ein Schlüssel ist, existiert bereits ein Index. Also kann dieser für einen effizienten Index-Join verwendet werden. Damit wird ebenfalls der Overhead des Sortierens oder Hashings gespart.

- Sort-Merge-Join: Schema: Relationen: $R : \{[a]\}$ und $S : \{[b]\}$.

  Anfrage: `select * from R, S where a = b order by a`

  Begründung: Da das Ergebnis ohnehin nach dem Attribut a sortiert werden muss, kann die Sortierung bereits vor dem Join durchgeführt werden, damit die Sortierung für einen Sort-Merge-Join verwendet werden kann.

- Hash-Join: Schema: Relationen: $R : \{[a]\}$ und $S : \{[b]\}$.

  Anfrage: `select * from R, S where a = b`

  Begründung: Hier wählt das Datenbanksystem den Hash-Join, da seine Komplexität die niedrigste ist.

Hausaufgabe 4

Gegeben sind die beiden Relationenausprägungen:

R	
	A
...	0
...	5
...	7
...	8
...	8
...	10
⋮	⋮

S	
B	
5	...
6	...
7	...
8	...
8	...
11	...
⋮	⋮

Werten Sie den Join $R \bowtie_{R.A=S.B} S$ mithilfe des Nested-Loop- sowie des Sort/Merge-Algorithmus aus. Machen Sie deutlich, in welcher Reihenfolge die Tupel der beiden Relationen verglichen werden und kennzeichnen Sie die Tupel, die in die Ergebnismenge übernommen werden. Vervollständigen Sie hierzu die beiden folgenden Tabellen:

		S.B					
		5	6	7	8	8	11
R.A	0	1	2	3			
	5						
	7						
	8						
	8						
	10						

Nested-Loop-Join

		S.B					
		5	6	7	8	8	11
R.A	0	1					
	5	2✔					
	7						
	8						
	8						
	10						

Sort/Merge-Join

Lösung:

		S.B					
		5	6	7	8	8	11
R.A	0	1	2	3	4	5	6
	5	7✔	8	9	10	11	12
	7	13	14	15✔	16	17	18
	8	19	20	21	22✔	23✔	24
	8	25	26	27	28✔	29✔	30
	10	31	32	33	34	35	36

Nested-Loop-Join

		S.B					
		5	6	7	8	8	11
R.A	0	1					
	5	2✔	3				
	7		4	5✔			
	8			6	7✔	10✔	
	8				8✔	11✔	
	10				9	12	13

Sort/Merge-Join

Ausführliche Lösung:
`http://www-db.in.tum.de/teaching/ws1415/grundlagen/Loesung11_sort_merge_join.pdf`

Hausaufgabe 5

Für einen Join-Baum T sei folgende Kostenfunktion gegeben

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

Die Kardinaliät sei dabei

$$|T| = \begin{cases} |R_i| & \text{falls } T \text{ eine Basisrelation } R_i \text{ ist} \\ (\prod_{R_i \in T_1, R_j \in T_2} f_{i,j}) |T_1||T_2| & \text{falls } T = T_1 \bowtie T_2 \end{cases}$$

Sei $p_{i,j}$ das Join Prädikat zwischen R_i und R_j, dann sei

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

und die Kardinaliät eines Join-Resultats ist $|R_i \bowtie_{p_{i,j}} R_j| = f_{i,j}|R_i||R_j|$.

Gegeben sei eine Anfrage über die Relationen R_1, R_2, R_3 und R_4 mit $|R_1| = 10, |R_2| = 20, |R_3| = 20, |R_4| = 10$. Die Selektivitäten der Joins seien $f_{1,2} = 0.01, f_{2,3} = 0.5, f_{3,4} = 0.01$, alle nicht gegebenen Selektivitäten sind offensichtlich 1 (Warum?). Berechnen Sie den

optimalen (niedrigste Kosten) Join-Tree. Als Vereinfachung reicht es, wenn Sie nur Joins mit Prädikat und keine Kreuzprodukte betrachten.

Lösung:

Es ist kein Algorithmus angegeben. Aufgrund der geringen Anzahl von Relationen ist es möglich, die Kosten aller möglichen Join-Bäume zu berechnen und den kostengünstigsten auszuwählen (Bruteforce).

Zunächst gilt es zu überlegen, für welche Join-Bäume die Kosten tatsächlich zu berechnen sind.

Left-Deep:

$$((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4 \tag{1}$$
$$((R_4 \bowtie R_3) \bowtie R_2) \bowtie R_1 \tag{2}$$
$$((R_3 \bowtie R_2) \bowtie R_1) \bowtie R_4 \tag{3}$$
$$((R_3 \bowtie R_2) \bowtie R_4) \bowtie R_1 \tag{4}$$

Bushy:

$$(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4) \tag{5}$$

Alle anderen Left Deep oder Bushy Trees enthalten Kreuzprodukte oder sind im Bezug auf die Kosten äquivalent. Ersteres entsteht, wenn Relationen in einer Reihenfolge gejoint werden, in der bei einem der Joins kein Prädikt möglich ist, beispielsweise ist dies für den Left-Deep Tree

$$((R_1 \bowtie R_2) \times R_4) \bowtie R_3$$

der Fall. Im Bezug auf die Kosten bei der gegebenen Kostenfunktion äquivalent sind Join-Trees, bei denen die Kinder eines Join Operators vertauscht wurden, etwa

$$(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$$

und

$$(R_3 \bowtie R_4) \bowtie (R_1 \bowtie R_2).$$

Im Beispiel müssen lediglich die Kosten für die Join-Reihenfolgen 1, 3 und 5 berechnet werden. Dies liegt am Aufbau der Kostenfunktion sowie den symmetrischen Größen der Relationen sowie ihrer Join Selektivitäten.

Die Berechnung von $C_{out}((R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4))$ sei hier exemplarisch in epischer Breite ausgeführt (Machen Sie es selber; Sie erkennen äußerst schnell ein Muster und müssen keine derartigen Formel-Konvolute schreiben):

$$
\begin{aligned}
& C_{out}((R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)) \\
= \; & |(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)| + C_{out}(R_1 \bowtie R_2) + C_{out}(R_3 \bowtie R_4) \\
= \; & |(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)| + |R_1 \bowtie R_2| + C_{out}(R_1) + C_{out}(R_2) + |R_3 \bowtie R_4| + C_{out}(R_3) + C_{out}(R_4) \\
= \; & |(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)| + |R_1 \bowtie R_2| + |R_3 \bowtie R_4| \\
= \; & f_{1,3} \cdot f_{1,4} \cdot f_{2,3} \cdot f_{2,4} \cdot |R_1 \bowtie R_2| \cdot |R_3 \bowtie R_4| + |R_1 \bowtie R_2| + |R_3 \bowtie R_4| \\
= \; & 0.5 \cdot (0.01 \cdot 10 \cdot 20) \cdot (0.01 \cdot 20 \cdot 10) + (0.01 \cdot 10 \cdot 20) + (0.01 \cdot 20 \cdot 10) \\
= \; & 2 + 2 + 2 \\
= \; & 6
\end{aligned}
$$

Die Ergebnisse der anderen Relevanten Join-Reihenfolgen sind:

$((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$	24
$((R_3 \bowtie R_2) \bowtie R_1) \bowtie R_4$	222
$(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$	6

Der Bushy-Tree 5 ist also der optimale Join-Tree.