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

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS18/19

Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws1819/grundlagen/

Blatt Nr. 13

Hausaufgabe 1

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.

Hausaufgabe 2

Gegeben sei die Anfrage:

```
select *
  from R, S, T
 where R.A = S.A and S.B = T.B and T.C = R.A
```

Desweiteren soll gelten:

- `S.A` und `T.C` seien Fremdschlüssel auf `R`
- `S.B` sei Fremdschlüssel auf `T`
- `R.A`, `T.B` seien Primärschlüssel von `R` respektive `T`
- Ihre Query-Engine "kann" nur nested loops-Join
- Kardinalitäten: |R|=100, |S|=1000, |T|=10
- Es gibt keine Indexe

Bestimmen Sie den günstigsten QEP (query evaluation plan) auch als Baum mit Kosten-/Kardinalitäts-Abschätzungen. Verwenden Sie den in der Vorlesung gezeigten kostenbasierten DP (dynamisches Programmieren)-Optimierer.

Lösung:

In diesem einfachen Beispiel-System setzen sich Ausführungspläne aus lediglich zwei (physischen) Operatoren zusammen:

- Tablescan: scan(R_i), wobei R_i eine Basisrelation ist
- Nested loops-Join: $P_1 \bowtie^{\text{NL}} P_2$, mit den Teilplänen P_1 und P_2

D.h. dass stets alle Tupel der Basisrelationen gelesen werden und dass im Falle von Joins alle Tupel-Paare der beiden Eingaben verglichen werden. Als Kosten für die Anfragebearbeitung können also die Tupel gezählt werden, die die Query-Engine "in die Hand nehmen muss".

Kostenfunktion für Ausführungspläne:

$$C(P) = \begin{cases} |P| & \text{falls } P = \text{ scan}(...) \\ |P_1| \cdot |P_2| + C(P_1) + C(P_2) & \text{falls } P = P_1 \bowtie^{\text{NL}} P_2 \end{cases}$$

Die Kostenfunktion ist symmetrisch aufgrund der Kommutativität des NL-Joins: $C(R \bowtie^{\text{NL}} S) = C(S \bowtie^{\text{NL}} R)$

Für die Kardinaliäten kann (analog zu Hausaufgabe 1) angenommen werden, dass

$$|P| = \begin{cases} |R_i| & \text{falls } P \text{ ein scan über die Basisrelation } R_i \text{ ist} \\ (\prod_{R_i \in P_1, R_j \in P_2} f_{i,j}) |P_1||P_2| & \text{falls } P = P_1 \bowtie^{\text{NL}} P_2 \end{cases}$$

Sei $p_{i,j}$ das Join Prädikat zwischen den Relationen 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 $|R_i \bowtie_{p_{i,j}} R_j| = f_{i,j}|R_i||R_j|$.

Für Equijoins über den Fremdschlüssel gilt die Abschätzung:

$$f_{i,j} = \frac{1}{|R_i|}, \text{mit Fremdschlüssel in } R_j$$

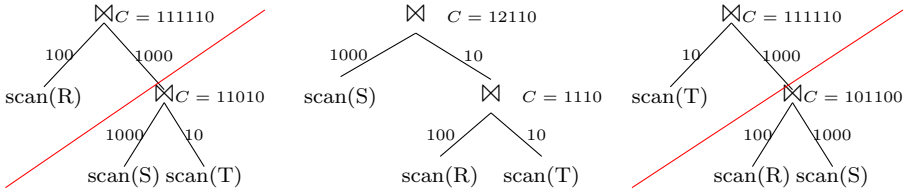
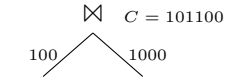
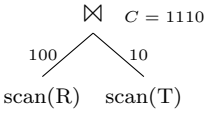
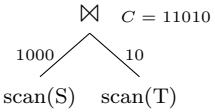
Da alle Relationen über Fremdschlüssel verbunden sind, gelten (vereinfachend) folgende Kardinalitäten:

$$|R \bowtie S| = |S \bowtie R| = \frac{1}{|R|} \cdot |R| \cdot |S| = |S| = 1000$$

$$|R \bowtie T| = |T \bowtie R| = \frac{1}{|R|} \cdot |R| \cdot |T| = |T| = 10$$

$$|S \bowtie T| = |T \bowtie S| = \frac{1}{|T|} \cdot |S| \cdot |T| = |S| = 1000$$

Für die Buttom-up-Berechnung der Kosten ergibt sich dann folgende DP-Tabelle:

BestePläneTabelle (DP table)		
Index	Pläne	Kosten
R,S,T		12110
R,S		101100
R,T		1110
S,T		11010
R	scan(R)	100
S	scan(S)	1000
T	scan(T)	10

Hausaufgabe 3

Wofür stehen die vier Buchstaben ACID? Erklären Sie für jeden der vier Konzepte, warum es für eine Datenbank wichtig ist. Geben Sie dazu jeweils ein Beispiel an, was passieren könnte, wenn dieses Konzept nicht gelten würde.

Lösung:

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Wenn eine Datenbank keine Atomarität garantieren kann, kann es zu inkonsistenten Daten kommen (unabhängig von der Konsistenzeigenschaft/Consistency). Hierzu dient eine Überweisung in einer Bank als Beispiel: Wenn eine Transaktion daraus besteht, einen Kontostand zu verringern und einen andern zu erhöhen, entsteht ein inkonsistenter Zustand, wenn nur eine der beiden Operationen tatsächlich gespeichert wird.

Bei einer Datenbank, die nicht konsistent ist, können weitreichende Probleme entstehen. Wenn z.B. garantiert werden muss, dass Kontonummern eindeutig sind, die Datenbank aber inkonsistente Daten zulässt, kann nicht mehr garantiert werden, dass Überweisungen tatsächlich beim richtigen Kontobesitzer ankommen.

Bei dem Beispiel einer Überweisung kann es auch zu Problemen kommen, wenn die Datenbank Transaktionen nicht korrekt voneinander isoliert. Zwei parallele Überweisungen können gleichzeitig Kontostände ändern, was zu inkorrekten Kontoständen nach der Ausführung beider Transaktionen führen kann.

Wenn eine Datenbank eingesetzt wird, die keine Dauerhaftigkeit garantieren kann, kann nie darauf vertraut werden, dass ein commit eine Transaktion tatsächlich festschreibt. Das ist aber z.B. bei einem Geldautomaten notwendig, der erst Geld ausgeben sollte, sobald die Datenbank garantieren kann, dass die Abhebetransaktion verbucht ist.

Hausaufgabe 4

Formulieren Sie die folgende Anfrage auf dem bekannten Unischema in SQL: Ermitteln Sie für jede Vorlesung, wie viele Studenten diese vorgezogen haben. Ein Student hat eine Vorlesung vorgezogen, wenn er in einem früheren Semester ist als der „Modus“ der Semester der Hörer dieser Vorlesung. Der Modus ist definiert als der Wert, der am häufigsten vorkommt – für diese Anfrage also das Semester, in dem die meisten Hörer dieser Vorlesung sind. Falls es mehrere Semester dieser Art gibt, soll nur das niedrigste zählen.

Beachten Sie, dass auch Vorlesungen ohne Hörer, sowie Vorlesungen deren Hörer alle im gleichen Semester sind, ausgegeben werden sollen.

Geben Sie für jede Vorlesung die Vorlesungsnummer, den Titel und die Anzahl der „Vorzieher“ aus.

Lösung:

Die Aufgabe lässt sich am leichtesten lösen, wenn man sie in mehrere Teile aufbricht:

Zunächst erstellen wir eine Anfrage, welche für jede Vorlesung aufschlüsselt, von wie vielen Stundenten sie pro Semester gehört wird:

```
with vorl_semester_anz as (
    select h.vorlnr, s.semester, count(*) as anzahl
    from hoeren h, Studenten s
    where h.matrnr = s.matrnr
    group by h.vorlnr, s.semester
)
```

Mithilfe dieser Sicht können wir nun für jede Vorlesung den Modus der Hörersemester bestimmen. Der Modus ist dasjenige Semester, dem die meisten Anhörer angehören. Unter diesen Einträgen könnten auch Duplikate sein. Wenn eine Vorlesung z.B. gleich oft von Erst- und Drittsemestern gehört wird, wollen wir sie dem ersten Semester zuordnen. Mit einem einfachen *group by* finden wir das Minimum.

```
with vorl_modus as (
    select v1.vorlnr, min(v1.semester) as modus
    from vorl_semester_anz v1
    where v1.anzahl = (
        select max(v2.anzahl)
        from vorl_semester_anz v2
        where v1.vorlnr = v2.vorlnr
    )
    group by v1.vorlnr
)
```

Nun müssen wir nur noch für jedes dieser Vorlesung-Semester-Paare alle diejenigen Studenten finden und aufsummieren, welche die Vorlesung hören und in einem niedrigeren Semester als der Modus sind. Da wir auch Vorlesungen ohne Hörer ausgeben wollen, müssen wir den *left outer join* verwenden. Wichtig: Die Bedingung `s.semester < vm.semester` muss als Bedingung des outer joins angegeben werden, und *nicht* in der where-Klausel, da ansonsten alle Vorlesungen ohne Studenten aus niedrigeren Semestern wieder herausgefiltert werden würden.

```
    select v.vorlnr, v.titel, count(s.matrnr) as anzahl
    from
        vorlesungen v left outer join
        vorl_modus vm on v.vorlnr = vm.vorlnr left outer join
        hoeren h on h.vorlnr = v.vorlnr left outer join
        studenten s on s.matrnr = h.matrnr and s.semester < vm.modus
    group by v.vorlnr, v.titel
```

Insgesamt ergibt sich also beispielsweise folgende Anfrage:

```
with vorl_semester_anz as (
    select h.vorlnr, s.semester, count(*) as anzahl
    from hoeren h, Studenten s
    where h.matrnr = s.matrnr
    group by h.vorlnr, s.semester
), vorl_modus as (
    select v1.vorlnr, min(v1.semester) as modus
    from vorl_semester_anz v1
    where v1.anzahl = (
        select max(v2.anzahl)
        from vorl_semester_anz v2
        where v1.vorlnr = v2.vorlnr
    )
    group by v1.vorlnr
)

select v.vorlnr, v.titel, count(s.matrnr) as anzahl
from
    vorlesungen v left outer join
    vorl_modus vm on v.vorlnr = vm.vorlnr left outer join
    hoeren h on h.vorlnr = v.vorlnr left outer join
    studenten s on s.matrnr = h.matrnr and s.semester < vm.modus
group by v.vorlnr, v.titel
```