



Übungen zur Vorlesung "Datenbanksysteme II"

Wintersemester 2009/2010

Melanie Herschel (melanie.herschel@uni-tuebingen.de)

6. Übungsblatt

Ausgabe: 26. Januar 2010 · Besprechung: 9. Februar 2010

Auf diesem Übungsblatt gehen nur Aufgaben 1 und 2 in die "normale" Bewertung der Übungsleistung mit ein. Aufgaben 3 und 4 sind Bonusaufgaben, mit denen Sie Punkte aufholen können.

Bitte geben Sie, zusätzlich zu Ihrem **Namen**, auch Ihre **Matrikelnummer** auf dem Übungsblatt an!

Aufgabe 1: Von Anfragen zu Plänen

(8 Punkte)

Beachten Sie das folgende Teilschema einer Datenbank.

Studenten		
<u>MatrNr</u>	Vorname	Nachname
:	:	:

Vorlesungen	
<u>VorID</u>	Name
:	:

nimmt_teil	
<u>VorID</u>	<u>MatrNr</u>
:	:

Die Primärschlüssel sind unterstrichen. Es können die offensichtlichen Fremdschlüsselbeziehungen und Indizes auf Primär- und Fremdschlüsselspalten angenommen werden.

Für die Tabellen gelten die folgenden Kardinalitäten:

- |Studenten| = 2,000
- |Vorlesung| = 200
- |nimmt_teil| = 10,000

Basierend auf dem obigen Schema lösen Sie bitte folgende Aufgaben:

1. Drücken Sie die folgende, in natürlicher Sprache formulierte Anfrage in SQL aus:
Geben Sie die Nachnamen aller Studenten aus, die an der Vorlesung DBS2 teilgenommen haben.
2. Zeichnen Sie den zur Anfrage gehörenden Anfrage-Plan, der dem Ergebnis des Parsers entspricht (und auf dem daher noch keine Optimierungen vorgenommen wurden).

3. Wenn möglich, wenden Sie aus der Vorlesung bekannte Transformationsregeln auf den Plan der Teilaufgabe 1 an (Pushen von Selektion, Pushen von Projektion, Zusammenfassen von Kreuzprodukt und Selektion zu Join). Schätzen Sie die Kardinalitäten der Zwischenergebnisse Ihres Plans mittels Ihres Wissens über Kardinalitäten und Fremdschlüsselbeziehungen ab.
4. Wählen Sie die entsprechenden Zugriffspläne und physischen Implementationen der verschiedenen Operatoren (insbesondere Joins). Achten Sie hierbei darauf von den gegebenen Indizes und Sortierung der Zwischenergebnisse zu profitieren. Versuchen Sie, einen optimalen Plan zu finden. Ziehen Sie es in Betracht, die Reihenfolge der Joins zu ändern, falls dies nötig sein sollte.

Aufgabe 2: Greedy Join Enumeration

(8 Punkte)

Der Greedy Join Enumeration Algorithmus ist eine sehr einfache und intuitive Vorgehensweise, um einen guten Zugriffssplan zu finden. Natürlich garantiert dieser Algorithmus keinen optimalen Plan, findet aber in den meisten Fällen ausreichend gute Approximationen.

Beachten Sie folgende SQL-Anfrage:

```
SELECT *
  FROM R1, R2, R3, R4
 WHERE R1.a = R3.c
    AND R2.b = R4.d
    AND R4.y = R1.l
    AND R2.f = R3.j
    AND R1.g = R2.k
```

In das zugrunde liegende Kostenmodell fließen sowohl die Kosten der Eingabetabellen, als auch die Selektivität des Join-Prädikats mit ein:

$$\text{cost}(R_i \bowtie_p R_j) = (\text{cost}(R_i) + \text{cost}(R_j)) \cdot (1 + \text{sel}(p))$$

Die Selektivitäten für die einzelnen Joinprädikate sind die folgenden:

- $\text{sel}(R_1.a = R_3.c) = 0.9$
- $\text{sel}(R_2.b = R_4.d) = 0.5$
- $\text{sel}(R_4.y = R_1.l) = 0.1$
- $\text{sel}(R_2.f = R_3.j) = 0.8$
- $\text{sel}(R_1.g = R_2.k) = 0.2$

Auf den Tabellen gibt es keine Indizes, was bedeutet, dass der beste Zugriffssplan einer Tabelle R_i ein **Table-Scan** ist:

$$\forall i \in \{1, \dots, 4\} : \text{bestezugriffspfad}(R_i) = \text{TB-SCAN}(R_i)$$

Die Kosten für einen Table-Scan einer Tabelle R_i ermitteln sich aus den Seiten N_{R_i} , welche Sie beansprucht:

$$\forall i \in \{1, \dots, 4\} : \text{cost}(\text{TB-SCAN}(R_i)) = N_{R_i}$$

Die Seiten, welche die einzelnen Tabellen beanspruchen sind in folgender Tabelle aufgelistet.

Statistics	
Table	Pages
R_1	100
R_2	400
R_3	300
R_4	50

Finden Sie mittels des Greedy Join Algorithmus und der obigen Informationen einen guten Ausführungsplan für die SQL-Anfrage.

Aufgabe 3: Bonusaufgabe: Anfrageoptimierung

(12 Punkte)

Beachten Sie das folgende relationale Schema und die SQL Anfrage:

Suppliers		
sid	sname	city
⋮	⋮	⋮

Parts		
pid	pname	price
⋮	⋮	⋮

Supply	
sid	pid
⋮	⋮

```
SELECT S.sname, P.pname
FROM Suppliers S, Parts P, Supply Y
WHERE S.sid = Y.sid AND Y.pid = P.pid AND
      S.city = 'San Francisco' AND P.price <= 2,000
```

1. Welche Informationen über die Tabellen benötigt der Optimierer, um einen guten Ausführungsplan für die gegebene Anfrage zu finden?
2. Wieviele verschiedene Join-Reihenfolgen werden, beim Ausführen der gegebenen Anfrage, von einem System-R Optimierer betrachtet? Nehmen Sie an, dass Kreuzprodukte vom Optimierer nicht in Betracht gezogen werden.
3. Welche Indizes sind beim Ausführen der gegebenen Anfrage hilfreich? Erklären Sie kurz.
4. Wie wirkt sich das Hinzufügen von DISTINCT zur SELECT-Klausel der gegebenen Anfrage aus? Erklären Sie.
5. Wie wirkt sich das Hinzufügen von ORDER BY sname auf den Plan aus? Erklären Sie.
6. Wie wirkt sich das Hinzufügen von GROUP BY sname auf den Plan aus? Erklären Sie.

Aufgabe 4: Bonusaufgabe: Korrelierte Anfragen

(6 Punkte)

Als Weinliebhaber informieren Sie sich natürlich laufend in Fachzeitschriften, welches edle Tröpfchen Sie wohl als nächstes erwerben sollten. Die folgende Tabelle zeigt eine Auswahl von edlen Weinen, zusammen mit renommierten Kritikern, die sie empfohlen haben.

Weinempfehlungen	
Wein	Kritiker
La Rose Grand Cru	Parker
Pinot Noir	Parker
Riesling Reserve	Parker
La Rose Grand Cru	Clarke
Pinot Noir	Clarke
Riesling Reserve	Gault-Millau

Vertrauskritiker
Name
Parker
Clarke

Da Sie nicht jedem Kritiker vertrauen, haben Sie sich zusätzlich eine Tabelle mit den Kritikern Ihres Vertrauens angelegt (**Vertrauenskritiker**).

Schreiben Sie eine SQL Anfrage, welche genau jene Weine findet, die von **allen** Ihren Vertrauenskritikern empfohlen wurden!

Mit der obigen Ausprägung ist das Ergebnis also das folgende:

Ergebnis
Name
La Rose Grand Cru Pinot Noir