

Prof. Dr. Guido Moerkotte

Email: moer@pi3.informatik.uni-mannheim.de

Pit Fender

B6, 29, Raum C0.05

68131 Mannheim

Telefon: (0621) 181-2517

Email: pfender@pi3.informatik.uni-mannheim.de

Anfrageoptimierung

Herbst-/Wintersemester 07

9. Übungsblatt

21. November 2007

Aufgabe 1

Bestimmen Sie den break-even point, bis wann sich der Einsatz eines Index gegenüber einem Scan lohnt.

Lösung

Für den break-even point sind mehrere Faktoren von Relevanz

- Clusterung des Index
- Verhältnis zwischen Anzahl der Blattseiten im Index und in Datenseiten der Tabelle
- Cashingverhalten der Seiten des Index, z.B. Wurzelseite verbleibt häufig im Datenbankpuffer
- Art des Prädikats: Punktanfrage, Range Predicates

Bei Anfrage an den Index ist der Abstieg zur und inklusive der ersten Blattseite teuer (random access). Bei Bereichsprädikaten dann sequentieller Zugriff. Bei einer nicht nach dem Index geclusterten Tabelle wird eine Sortierung der TIDs notwendig. Zugriff auf die Tabelle kann bei großer Dichte von TIDs ebenfalls im sequentiellen Zugriff erfolgen.

Im günstigsten Fall $\log n$ Seek-Kosten im Index inklusive einmaliger Seek-Kosten für den Zugriff auf die erste Seite der Tabelle.

Break-even point entspricht dann einer bestimmten Selektivität.

Aufgabe 2

Angenommen der Optimierer hat die Kosten für einen scan um einen Faktor $a, a > 1$ unterschätzt und die Kosten für den Index um einen Faktor $b, b < 1$ überschätzt. Wie groß ist der maximale Fehler durch diese Fehleinschätzung?

Lösung

- Problem nur wenn Scan teurer als Index

- im schlimmsten Fall ist der Unterschied so groß, dass der Scan scheinbar gerade billiger wird
- d.h. der maximale Fehler ist b/a

Aufgabe 3

Geben Sie für die folgenden Anfragen je einen Zugriffplan an, der die Attributzugriffe explizit bzw. implizit ausführt.

Aufgabe 3 a)

```
select s.name, s.matrnr
from studenten s
where s.age>27 and
      count(s.hoert)=0
```

Lösung

implizit : $\Pi_{matrnr}(\sigma_{count(hoert)=0}(studenten[s; age > 27]))$

explizit : $\chi_{matrnr:s.matrnr}(\sigma_{count=0}(\chi_{count:count(hoert)}(\chi_{hoert:s.hoert}(\sigma_{age>27}(\chi_{age:s.age}(studenten[s]))))))))$

Aufgabe 3 b)

```
select s.name, s.matrnr
from studenten s
where s.age>27 and
      count(s.hoert)<5 and
      exists v in s.hoert:
        v.name="AO"
```

Lösung

implizit : $\Pi_{matrnr}(\sigma_{\exists v \in hoert:v.name="AO"}(\sigma_{count(hoert)<5}(studenten[s; age > 27])))$

explizit : $\chi_{matrnr:s.matrnr}(\sigma_{\exists v \in hoert:v.name="AO"}(\sigma_{count<5}(\chi_{count:count(hoert)}(\chi_{hoert:s.hoert}(\sigma_{age>27}(\chi_{age:s.age}(studenten[s]))))))))$

Aufgabe 4

Finden Sie für die folgenden Anfragen sinnvolle Pläne unter Verwendung von Indizes. Geben Sie jeweils an, welche Annahmen bzgl. des Index Sie treffen.

Aufgabe 4 a)

```
select a.v
from   a a
where  a.w > 10
       and a.w < 30
```

Lösung

- Index-Scan unter Verwendung von oberer und unterer Schranke (10 bzw. 30).
- (B-Baum) Index auf Attribut w muss vorhanden sein.

Aufgabe 4 b)

```
select *
from   a
where  exists (select *
               from   b
               where  a.v = b.key)
```

Lösung

- Index-Scan auf Relation b . Der Index muß für das Schlüsselattribut $b.key$ vorhanden sein.
- Da Vergleich auf Schlüssel von Relation b ausgeführt wird, kann man einen effizienten Index-NL-Join oder D-Join ausführen. Am besten wird Relation a vorher aufsteigend nach $a.v$ sortiert.

$$\sigma_{0 < \text{count}(\sigma_{a.v=b.key}(B[b]))}(A[a])$$
$$\sigma_{0 < \text{count}(\chi_{\text{count:count}(\sigma_{a.v=b.key}(B[b]))})(A[a])$$
$$\sigma_{\text{count} > 0}(A[a] < \chi_{\text{count:count}(B_{key}[x:key=a.v])}(\square) >)$$

Aufgabe 4 c)

```
select a.b, min(a.c)
from a
group by a.b
```

Lösung

- Index-Scan auf Attribute $a.b, a.c$ von Relation a .
- Beginne mit minimalen Wert für $a.b$. Benutze dann Gap-Skipping, um zum nächsten Wert von $a.b$ zu springen. Der erste gefundene Wert ist jeweils das Minimum.

Aufgabe 5

Aufgabe 5 a)

Untersuchen Sie den Operator `Indexscan` in `tinydb`. Wie lassen sich Prädikate auswerten? Welches Zugriffsverhalten hat der Operator?

Lösung

- Start/Stop-Bedingung auf genau einem Attribut je Index
- jedes Tupel (im Bereich) ein random read

Aufgabe 5 b)

Schreiben Sie ein Java-Programm, das z.B. analog zu `JoinSample.java` unter Verwendung des `Tablescan` bzw. `Indexscan` die folgende Anfrage auswertet.

```
SELECT *
FROM CUSTOMER, NATION
WHERE c_nationkey = n_nationkey
      AND n_name = "GERMANY"
      AND c_custkey = 300
```

Vergleichen Sie die Laufzeit der beiden Programme.

Lösung

siehe `IndexSample.java`
Index: 2 ms
Scan: 33 ms

Aufgabe 5 c)

Erläutern Sie wie der **Indexscan** zusätzlich zu **Tablescans** in einem der join ordering Algorithmen, z.B. in GOO, als Planalternativen berücksichtigt werden können.

Lösung

- Anwendbarkeit von Prädikaten wie bei normalen Tablescans auch testen
- Finde alle Prädikate, die auf einem Attribut arbeiten (keine Equi-Join Prädikate)
- Teste, ob ein Index für das Attribut verfügbar ist
- Wenn ja, dann benutze das selektivste anwendbare Prädikat für den Index als upper und lower bound (mehrwertige Schlüssel nicht erlaubt)
- Range queries werden vom Parser und auch von der Selektion noch nicht unterstützt. Daher ist der Index noch nicht sinnvoll für solche Anfragen einsetzbar.
- Nun müssen die Kosten der Scans auch bei C_{out} berücksichtigt werden! Z.B. Kosten des Tablescans ist die Anzahl der Tupel in der Tabelle und Kosten des Indexscans ist die $n \lg n$ bei n Ergebnistupeln.