Prof. Dr.-Ing. Dr. h. c. T. Härder
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Universität Kaiserslautern
www.haerder.de

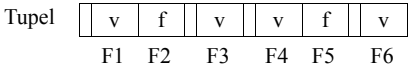
Übungsblatt 9

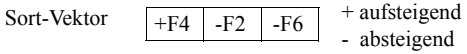
Unterlagen zur Vorlesung: „wwwdvs.informatik.uni-kl.de/courses/DBSREAL/“

Aufgabe 1: Sortieren variabel langer Schlüssel

Beim Sortieren von Tupeln kann ein Sortierschlüssel, der sich auf mehrere Felder in beliebiger Reihenfolge verteilt, spezifiziert werden. Zusätzlich ist die Angabe einer aufsteigenden oder absteigenden Sortierordnung pro Feld erlaubt. Es sind Felder konstanter Länge, für die die Längeninformation in den Beschreibungsdaten abgelegt ist, und Felder variabler Länge mit vorangestellter Längeninformation erlaubt. Die Feldinhalte seien vom Typ „binär“. Feldlänge Null ist im Fall variabler Felder möglich.

Beispiel:





Es gelten folgende Regeln für den Vergleich von Werten:

Für Felder fester Länge entscheidet über die Sortierreihenfolge der binäre Wert bei aufsteigender Sortierordnung. Bei absteigend definierter Sortierordnung dreht sich die Reihenfolge um. Bei Feldern variabler Länge gelten die Regeln des Falls fester Länge, wenn die zu vergleichenden Feldwerte gleiche Länge besitzen. Wenn Wert W1 kürzer als Wert W2 ist und die Länge von m1 Bytes hat, so wird er gegen den m1-Byte Präfix von W2 verglichen. Wenn keine Übereinstimmung auftritt, bestimmt dieser Vergleich die relative Ordnung. Bei Übereinstimmung wird W2 wegen seiner größeren Länge als größer als W1 bei aufsteigender Ordnung (kleiner als W1 bei absteigender Ordnung) eingestuft.

1.) Entwerfen Sie ein Algorithmus, der zwei Tupel mit verteilten Schlüsselfeldern entsprechend der oben gegebenen Vergleichsregeln sortiert.

2.) Geben Sie einen Algorithmus an, der die im Sort-Vektor spezifizierten Felder in einem variabel langen Feld so vercodet, dass jeweils zwei Sortierschlüssel direkt verglichen werden können. Durch die Kodierung muss natürlich die ursprüngliche Sortierfolge gewahrt bleiben. Die einfache Konkatenation der einzelnen Felder löst

diese Aufgabe wegen des Auftretens variabel langer Werte nicht. Eine solche Kodierung ist auch wichtig für den Fall eines Mehrfeld-Index.

3.) Entwerfen Sie einen Algorithmus zum Vergleich von zwei kodierten Schlüsseln.

4.) Modifizieren Sie den unter 2.) entworfenen Algorithmus derart, dass das letzte Sort-Feld nicht in die Kodierung einbezogen, sondern nur angehängt wird. Dies ist besonders wichtig im häufig auftretenden Fall eines einzigen Sortierfeldes.

Aufgabe 2: Übersetzung von SQL auf eine satzorientierte Schnittstelle

a) Schreiben Sie für folgende SQL-Anfrage ein Programm mit den Operationen auf der satzorientierten Schnittstelle:

```
SELECT  ENO, ENAME, SAL
FROM    EMP
WHERE   AGE > 45
        AND SAL > 50 000
        AND DNO >= K50 AND DNO <= K60
```

Folgende Grundannahme wird hier für die Verarbeitung über einen Index (Index-Scan) gemacht:

Für jeden Schlüsselwert eines Anfragebereichs werden aus der Indexseite die TIDs der zugehörigen Sätze gelesen, so daß dann unmittelbar auf die Sätze in einer Datenseite zugegriffen werden kann. Somit wird eine Seite für jeden Wert maximal einmal gelesen. Aufgrund der Pufferersetzung ist es jedoch möglich, daß eine Seite verdrängt wird, bevor sie für den Zugriff auf den nächsten Wert erneut gebraucht wird.

Eine solche Verarbeitung hat den Vorteil, daß die Sätze nach dem Indexattribut sortiert ausgegeben werden. Sie hat den Nachteil, daß eine Seite ggf. mehrfach gelesen werden muß. Wenn die Sätze nicht sortiert ausgegeben werden müssen, wäre es günstiger, zunächst alle TIDs zu allen Werten des Anfragebereichs zu lesen und nach ihrer Seitennummer zu sortieren (Merge-Sort ist möglich). Beim darauffolgenden Zugriff wird jede Datenseite maximal einmal gelesen und die Zugriffsbewegungen auf der Magnetplatte werden (bei einer ungestörten Verarbeitung) minimal.

Es soll versucht werden, ein optimales Zugriffsprogramm mit Hilfe der Scan-Technik zu entwerfen, wenn

1. je ein Index für AGE, SAL und DNO vorhanden ist. Welchen Einfluß auf die Optimierung hat die Existenz eines Index mit Cluster-Bildung für eines der drei Attribute?
2. je ein Index für AGE und SAL und zusätzlich ein Index mit Cluster-Bildung für ENO vorhanden ist.
3. für alle Attribute der Qualifikationsbedingung Indexstrukturen ohne Cluster-Bildung vorhanden sind, die Tabelle EMP jedoch alleine in einem Segment mit hoher Lokalität (Belegungsfaktor b > 0.9) gespeichert ist.

b) In welcher Weise kann folgende SQL-Anfrage auf der satzorientierten Schnittstelle ausgewertet werden:

```
SELECT  X.ENAME, Y.ENAME
FROM    EMP X, EMP Y
WHERE   X.MNO = Y.PNO
        AND X.SAL > Y.SAL
```

Diskutieren Sie mögliche Lösungswege, wenn

1. Indexstrukturen für MNO, PNO und SAL vorhanden sind.

2. keine Zugriffspfade zur Unterstützung der Auswertung herangezogen werden können.

c) In welcher Weise würden Sie folgende SQL-Anfrage, die eine Partitionierung der (Zwischen-)Ergebnismenge nach den Attributwerten eines Attributs erfordert, auswerten:

```
SELECT   DNO
FROM     EMP
WHERE    JOB = 'PROGRAMMER'
GROUP BY DNO
HAVING COUNT (x) > 10
```

Geben Sie ein Programm für die satzorientierte Schnittstelle an, wenn

1. eine Indexstruktur für das Attribut JOB vorhanden ist

2. eine Indexstruktur für das Attribut DNO vorhanden ist

Welches ist die beste Auswertungsstrategie?

Aufgabe 3: Join-Programmierung auf der satzorientierten DBS-Schnittstelle 127

Skizzieren Sie in Pseudo-Code den Ablauf eines Sort Merge Join sowie den Ablauf eines Hash Join, wenn zwischen den Verbundattributen eine n:m-Beziehung besteht und keine Indexstrukturen zur Verfügung stehen. Ein SORT-Operator ist vorhanden. Die zur Partitionierung herangezogene Hash-Funktion bilde die Werte von ORT und WOHNORT direkt auf die Nummer der Partition ab.

```
SELECT  P.PNR, P.WOHNORT, P.ANR
FROM    ABT A, PERS P
WHERE   P.WOHNORT = A.ORT AND
        P.ALTER > 30 AND
        A.ANZ_MITARBEITER < 10
```

Aufgabe 4: Kostenmodelle für die Selektionsoperation

Gegeben sei Tabelle R(A1, A2, A3, ..., An) mit zusammenhängender Speicherung der Sätze in Seiten des Segmentes S.

Annahmen:

- Segment S habe $M_S = 10^4$ Seiten
- Tabelle R habe
  - $N_R = 10^5$ Sätze und ggf.
  - Cluster-Faktor $c_R = 50$
- Indexe
  - $I_R(A1)$ mit $j_{A1} = 100$

  - $I_R(A2)$ mit Cluster-Bildung und $j_{A2} = 10$
  - als B*-Bäume mit Höhe $h_B = 2$ und $N_B = 100$ Blattseiten realisiert

a) Wie teuer ist die Auswertung der SQL-Anfrage

```
Select  *
From    R
Where   A3 = 'x'
```

1. bei einem Tabellen-Scan
2. bei Nutzung von $I_R(A1)$
3. bei Nutzung von $I_R(A2)$
4. Wie teuer ist die Auswertung der obigen Anfrage, wenn die Tabelle als Hash-Struktur mit A3 als Primärschlüssel angelegt ist?

b) A1 habe 100 Werte, die von 1 bis 100 gleichverteilt vorkommen ($j_{A1} = 100$)

Wie teuer ist die Auswertung der SQL-Anfrage

```
Select  *
From    R
Where   A1 > 50
```

5. bei Index-Nutzung
6. bei Annahme einer Cluster-Bildung bei $I_R(A1)$
7. ohne Indexnutzung

c) Welche Kosten verursacht die SQL-Anfrage?

```
Select  *
From    R
Where   A1 = 50   AND   A2 = 10
```

8. bei Nutzung von $I_R(A1)$ und $I_R(A2)$ (ohne Cluster-Bildung)
9. bei zusätzlicher Annahme einer Cluster-Bildung bei $I_R(A2)$ und Zugriff über beide Indexe
10. bei Zugriff nur über $I_R(A2)$ mit Cluster-Bildung
11. bei Zugriff nur über $I_R(A1)$ mit Cluster-Bildung
12. bei k-d-Scan auf Grid-File für A1, A2