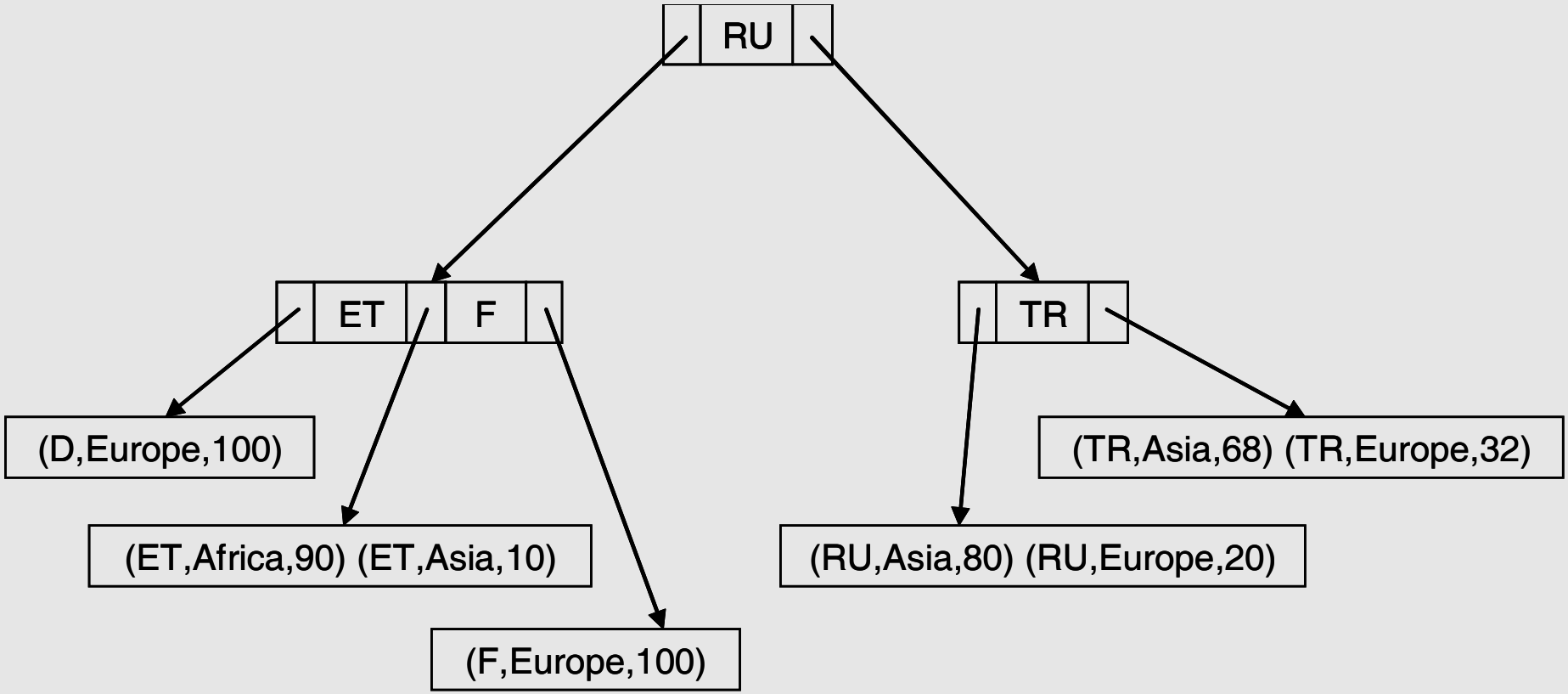
7.3 Baum-Indexstrukturen

B-Baum der Ordnung (m, l); $m > 2$, $l > 1$.

- Die Wurzel ist entweder ein Blatt oder hat mindestens zwei direkte Nachfolger.
- Jeder innere Knoten außer der Wurzel hat mindestens $\lceil \frac{m}{2} \rceil$ und höchstens m direkte Nachfolger.
- Die Länge des Pfades von der Wurzel zu einem Blatt ist für alle Blätter gleich.
- Die inneren Knoten haben die Form $(p_0, k_1, p_1, k_2, p_2, \ldots, k_n, p_n)$, wobei $\lfloor \frac{m}{2} \rfloor \leq n \leq m - 1$. Es gilt:
  - p_i ist ein Zeiger auf den $i + 1$-ten direkten Nachfolger und jedes k_i ist ein Suchschlüsselwert, $0 \leq i \leq n$.
  - Die Suchschlüsselwerte sind geordnet, d.h. $k_i < k_j$, für $1 \leq i < j \leq n$.
  - Alle Suchschlüsselwerte im linken (rechten) Teilbaum von k_i sind kleiner (größer oder gleich) als der Wert von k_i, $1 \leq i \leq n$.
- Die Blätter haben die Form $(k_1^*, k_2^*, \ldots, k_g^*)$, wobei $\frac{l}{2} \leq g \leq l$ und k_i^* das Tupel mit Suchschlüsselwert k_i.

Beispiel B-Baum $(3, 2)$



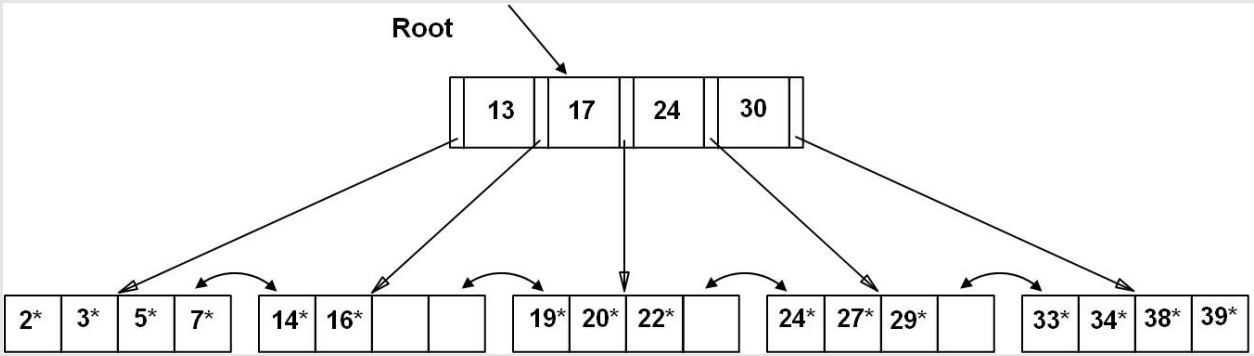
Höhe h:

- $\lceil log_m N \rceil \leq h \leq \lfloor 1 + log_{\frac{m}{2}} \frac{N}{2} \rfloor$; N Anzahl Blätter des Baumes.
- Anzahl Externzugriffe für Suchen, Einfügen und Löschen: $O(h)$.

B-Baum in der Praxis

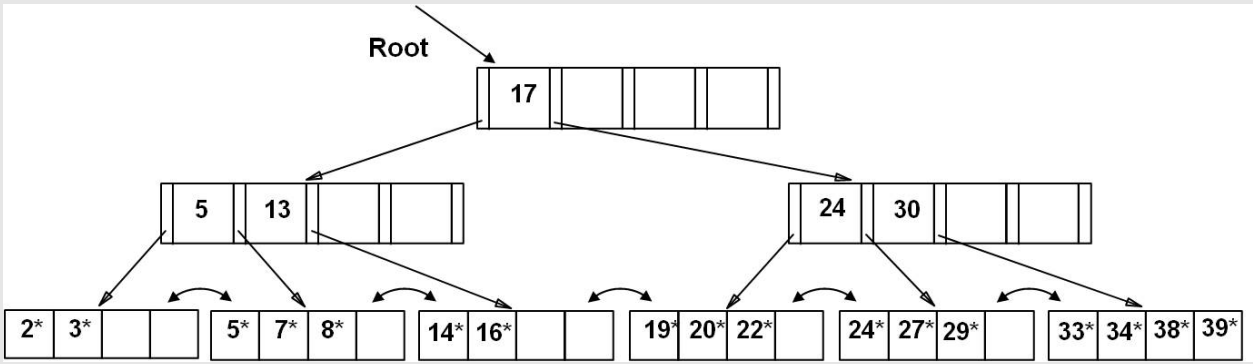
- Typisch: Ordnung $m = 200$, Füllungsgrad 67%, Verzweigungsgrad 133, Seitengröße 8Kbytes.
- $h = 1$: 133 Blätter, 1 Mbyte,
  $h = 2$: $133^2 = 17.689$ Blätter, 133 Mbyte,
  $h = 3$: $133^3 = 2.352.637$ Blätter, 17 Gbyte
  $h = 4$: $133^4 = 312.900.700$ Blätter, 2 Tbyte.
- Kann u.U. bis zur Höhe 2 im Internspeicher gehalten werden.
- Um die Effizienz von Zugriffen in Sortierfolge zu erhöhen werden die Blätter in der Sortierfolge benachbarter Tupel mit einander verkettet.

B-Baum (5, 4). Einfügen eines Tupels mit Schlüsselwert 8

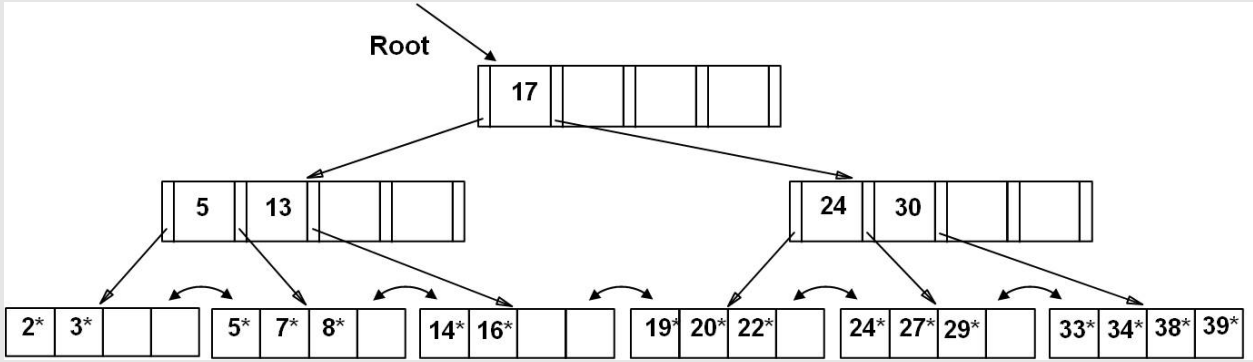


- Einfügen in das zugehörige Blatt mit möglichem Überlauf.
- In diesem Fall durch Aufteilen ein neues Blatt, bzw. einen neuen Knoten des Baumes erzeugen und die zugehörige Verzweigungsinformation in den Elterknoten rekursiv einfügen.
- Die Höhe des Baumes kann bei diesem Verfahren um maximal 1 wachsen.

Ergebnis

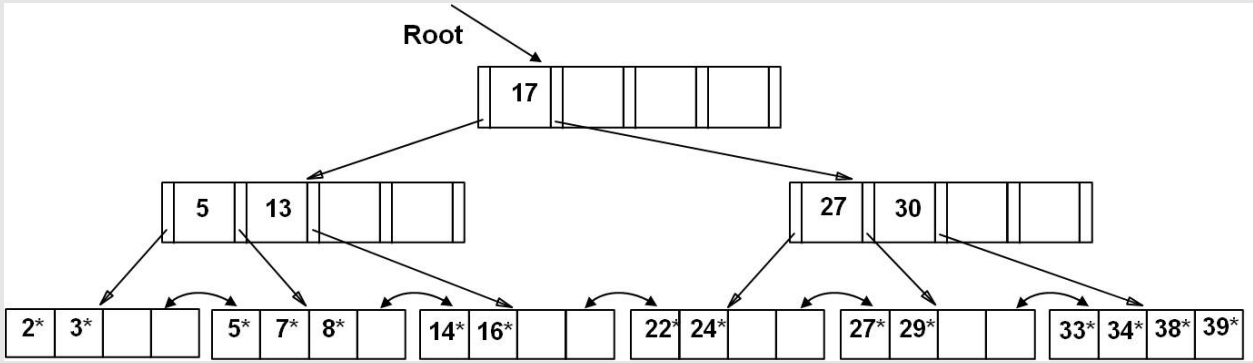


Löschen der Tupel mit Schlüsselwerten 19 und 20

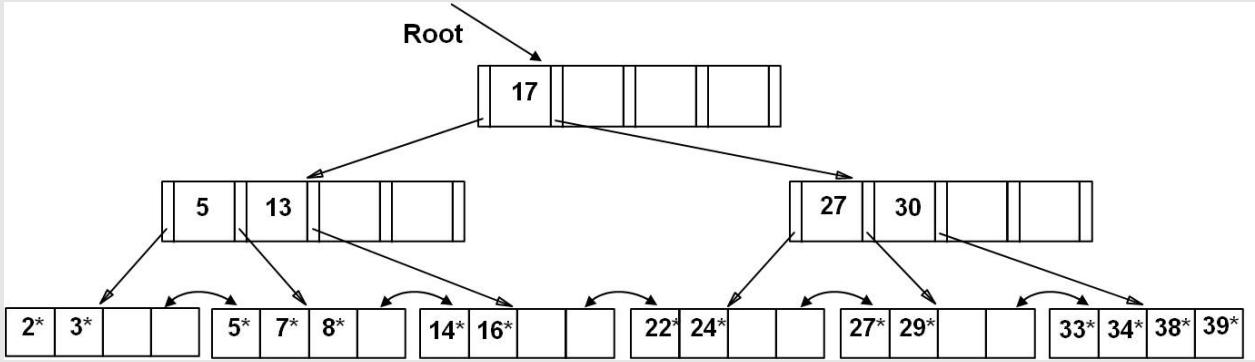


- Löschen des Tupels 19 ist unproblematisch.
- Nachfolgendes Löschen des Tupels 20 erzeugt einen Unterlauf im Blatt.
- Neuverteilen der Tupel mit einem Nachbarblatt so, dass beide Blätter die Mindestfüllung erreichen.
- Falls dies gelingt, Anpassen der Verzweigungsinformation im Elterknoten.

Ergebnis

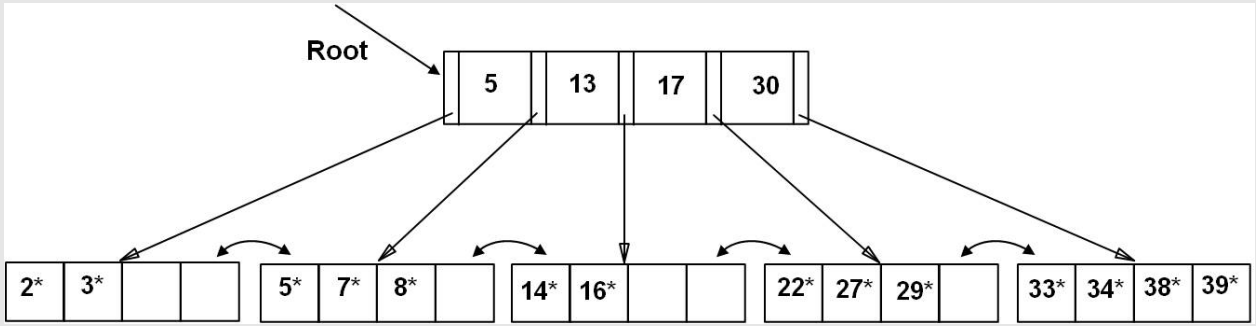


Löschen des Tupels mit Schlüsselwert 24



- Löschen des Tupels 24 erzeugt einen Unterlauf im Blatt.
- Neuverteilen der Tupel mit einem Nachbarblatt so, dass beide Blätter die Mindestfüllung erreichen.
- Dies gelingt nicht, somit Verschmelzen beider Blätter zu einem und rekursives Löschen, jetzt der Verzweigungsinformation im Elterknoten.
- Die Höhe des Baumes kann sich um maximal 1 verringern.

Ergebnis



- B-Bäume sind auch geeignet für Bereichsanfragen der Form *Searchkey op value* mit arithmetischem Vergleichsoperator *op*.
- B-Bäume sind ebenfalls geeignet für Sekundärindices: die Einträge k^* in den Blättern sind dann der Form (k, Tids), wobei *Tids* die Adressen der Tupel mit Schlüsselwert k.

mehrattributiger Suchschlüssel

- Konkatenation der einzelnen Attributwerte. In welcher Reihenfolge?
- Unabhängige Indexstrukturen über den einzelnen Attributen. Nachbarschaftsbeziehungen?
- $\Longrightarrow$ mehrdimensionale Zugriffsstrukturen.

7.4 Hash-Indexstrukturen

- Eine gegebene Relation wird mittels einer Streuungsfunktion (*Hashfunktion*) h in Seiten aufgeteilt. Die Seiten seien numeriert von 0, 1, ..., $K-1$. Auf der Menge der Suchschlüsselwerte k sollte $h(k)$ möglichst gleichverteilt über 0, 1, ..., $K-1$ sein.
- Mittels h wird jedem Tupel in Abhängigkeit seines Suchschlüsselwertes k eine Seite $h(k) = k \ mod \ K$.

- Direkter Zugriff zu den Tupeln einer Relation mit einem einzigen Externzugriff, sofern hinreichend viele Seiten zur Verfügung gestellt werden und die Hashfunktion annähernd eine Gleichverteilung der Tupel in den Seiten bewirkt.
- Werden mehr Tupel einer Seite zugeordnet, als diese aufnehmen kann, so müssen der Seite *Überlaufseiten* zugeordnet werden.
- Typischerweise werden die Seiten im Mittel beim Aufbau eines Hashindex nur zu 80% gefüllt.
- Zugriff zu den Tupeln gemäß einer Sortierfolge wird nicht unterstützt.

- Bereichsanfragen können nicht unterstützt werden.
- Sekundärindex analog zu B-Baum.
- Behandlung mehrattributiger Schlüssel analog zu B-Baum.

7.5 empfohlene Lektüre

ORGANIZATION AND MAINTENANCE OF LARGE

ORDERED INDICES

by

R. Bayer

and

E. McCreight

Mathematical and Information Sciences Report No. 20

Mathematical and Information Sciences Laboratory

BOEING SCIENTIFIC RESEARCH LABORATORIES

July 1970

ABSTRACT

Organization and maintenance of an index for a dynamic random access file is considered. It is assumed that the index must be kept on some pseudo random access backup store like a disc or a drum. The index organization described allows retrieval, insertion, and deletion of keys in time proportional to $\log_k I$ where I is the size of the index and k is a device dependent natural number such that the performance of the scheme becomes near optimal. Storage utilization is at least 50% but generally much higher. The pages of the index are organized in a special data-structure, so-called B-trees. The scheme is analyzed, performance bounds are obtained, and a near optimal k is computed. Experiments have been performed with indices up to 100,000 keys. An index of size 15,000 (100,000) can be maintained with an average of 9 (at least 4) transactions per second on an IBM 360/44 with a 2311 disc.

Key Words and Phrases: Data structures, random access files, dynamic index maintenance, key insertion, key deletion, key retrieval, paging, information retrieval.

1

[1] In: Acta Informatica 1: 173-189 (1972). Kann als Report gegoogelt werden.

8 Auswertung von Anfrageoperatoren

8.1 Selektion

Auswertung von $\sigma[A\ op\ val]R$.

Sofern vorhanden, kann

- Index zu A,
- Sortierung zu A

ausgenutzt werden. Ansonsten wird ein Scan über R erforderlich.

8.2 Projektion

Auswertung von $\pi[A_1, \ldots, A_m]R$.

Scan über R – anschließend müssen gegebenenfalls Duplikate entfernt werden mittels

- Sortierung, oder
- eines Hashverfahrens.

8.3 Verbund

Auswertung von $R \bowtie S$.

Sei X das Format von R und Y das Format von S. Sei n die Anzahl Tupel der betrachteten Relation r zu R und m entsprechend die Anzahl Tupel von s zu S.

- *Nested-Loop-Verbund*,
- *Sort-Merge-Verbund*,
- *Hash-Verbund*.

Nested-Loop-Verbund

FOR EACH tuple $\mu \in r$ DO
    FOR EACH tuple $\nu \in s$ DO
        IF $\mu[X \cap Y] = \nu[X \cap Y]$ THEN add $\mu \bowtie \nu$ to result

Variante Block-Nested-Loop-Verbund

Angenommen der gesamte Datenbankpuffer bestehend aus B Seiten steht zur Verfügung.

FOR EACH block of $B - 2$ pages of r DO
    FOR EACH page of s DO
        for all matching current in-memory tuples μ of the r-block and ν of the current s-page add $\mu \bowtie \nu$ to result.

Sei N die Anzahl Seiten von r und M die Anzahl Seiten von s.

- Sei o.B.d.A. $N \leq M$. Wähle r als die äußere Relation.
- Sei r die äußere Relation. Die Anzahl der gelesenen Seiten ergibt sich zu $N + N/(B-2) \times M$.

Nested-Loop-Verbund

FOR EACH tuple $\mu \in r$ DO
    FOR EACH tuple $\nu \in s$ DO
        IF $\mu[X \cap Y] = \nu[X \cap Y]$ THEN add $\mu \bowtie \nu$ to result

Variante Index-Nested-Loop-Verbund

- Existiert ein Index über die Verbundattribute einer Relation, dann wähle diese als die innere Relation.

  Für jedes μ der äußeren Relation können die potentiellen Verbundpartner ν der inneren Relation mittels Index-Look-Up bestimmt werden.
- Ein Index für die innere Relation kann auch während des ersten Durchlaufs dynamisch berechnet werden: *index-on-the-fly*.

Sort-Merge-Verbund

Beide Relationen sind nach den Verbundattributen aufsteigend sortiert.

- Für jede Relation wird ein Zeiger definiert, der zu Beginn jeweils das erste Tupel referenziert.
- Erfüllen die referenzierten Tupel die Verbundbedingung, dann führe für sie den Verbund aus.
- Anderenfalls verschiebe den Zeiger, der das Tupel mit kleinerem Verbundwert referenziert, bis beide Zeiger ein weiteres Paar von Verbundpartnern referenzieren, oder der gerade bewegte Zeiger ein Tupel mit einem größeren Wert referenziert.
- In diesem Fall wird das Verfahren fortgesetzt indem der zuletzt nicht bewegte Zeiger in entsprechender Weise weiter geschoben wird.

Sei N die Anzahl Seiten von r und M die Anzahl Seiten von s.

- Es werden o.B.d.A. mindestens $M + 1$ Seiten gelesen: Suchschlüsselwert des ersten Tupels von r größer als alle Suchschlüsselwerte in s.
- Es werden höchstens $N \times M$ Seiten gelesen: in r und s existiert nur ein und derselbe Suchschlüsselwert.

Hash-Verbund

Idee

Nur solche Tupel $\mu \in r$ und $\nu \in s$ können Partner bei Berechnung eines natürlichen Verbundes sein, für die für eine gegebene Hashfunktion h gerade $h(\mu) = h(\nu)$.

Verfahren

Sei N die Anzahl Seiten von r und sei M die Anzahl Seiten von s, wobei $N \leq M$. Die Relationen werden in Form von Blöcken gespeichert. Ein Block besteht aus einer Menge von Seiten. Die Anzahl der zur Verbundberechnung im Hauptspeicher zur Verfügung stehenden Seiten sei k.

Sei h_1 eine Hashfunktion, die jedem Tupel einer Relation einen Block zuordnet. Sei h_2 eine von h_1 verschiedene Hashfunktion, die jedem Tupel einer Relation einen Hauptspeicherbereich zuordnet.

(1) Wende h_1 jeweils auf r und s an und bilde so je eine Partition von r und s.

(2) Bilde dann den Verbund,

(2i) indem ein Block der Partition von r gelesen wird um die Tupel in den Seiten des eingelesenen Blockes mittels h_2 im Hauptspeicher zu verteilen,

(2ii) und dann die einzelnen Seiten des gemäß h_1 korrespondierenden Blockes der Partition von s nacheinander gelesen werden und für jedes Tupel einer solchen Seite die korrespondierenden Tupel von r mittels h_2 bestimmt werden.

(3) Iteriere das Verfahren bis alle Blöcke der Partition von r bearbeitet.

8.4 Mengenoperatoren und Aggregierung

$\cap$, $\cup$ und $-$

Gleichheit der beteiligten Formate!

- $r \cap s = r \bowtie s$.
- $\cup$ benötigt Duplikateliminierung.
  - ist eine Variante einer Duplikateliminierung.

GROUP BY, SUM, AVG, MAX, MIN und COUNT

- Sortierung nach den Gruppierungsattributen,

  Scan der einzelnen Gruppen.
- Existiert eine Indexstruktur zu der gegebenen Relation, dann können u.U. anstatt der eigentlichen Tupel der Relationen die entsprechenden Suchschlüsselwerte verarbeitet werden.
  - Enthält der Suchschlüssel des Index alle für die Aggregierung benötigten Attribute, dann erübrigen sich Zugriffe zu den eigentlichen Tupeln.
  - Bilden die Gruppierungsattribute einen Prefix des Suchschlüssels, dann wird der initiale Sortierschritt überflüssig.

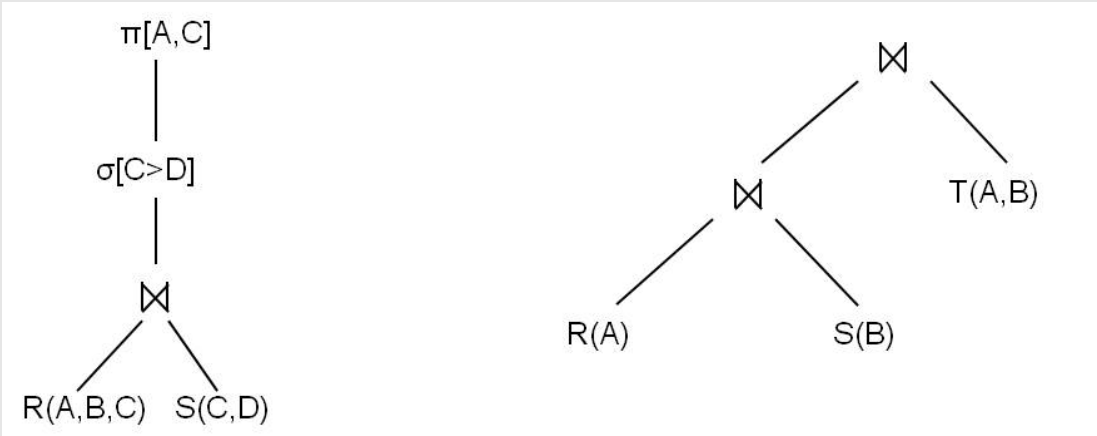
8.5 Optimierung

Optimierungstechniken

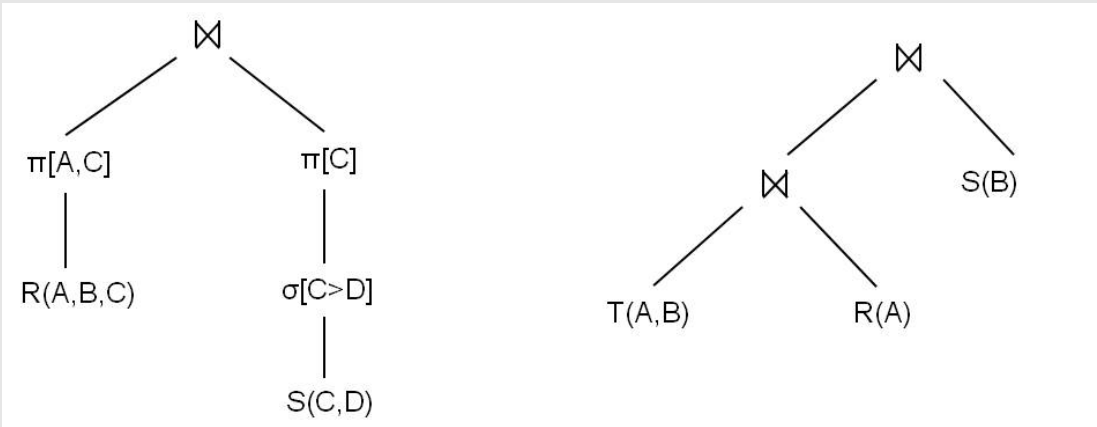
- *semantische*: Ausnutzen von Informationen über Integritätsbedingungen und (funktionalen) Abhängigkeiten,
- *algebraische*: äquivalenzbewahrende Umformungen,
- *physische*: Auswählen von Zugriffspfaden (Indexstrukturen).

Algebraische Optimierung

Anfragebäume zweier Algebraausdrücke

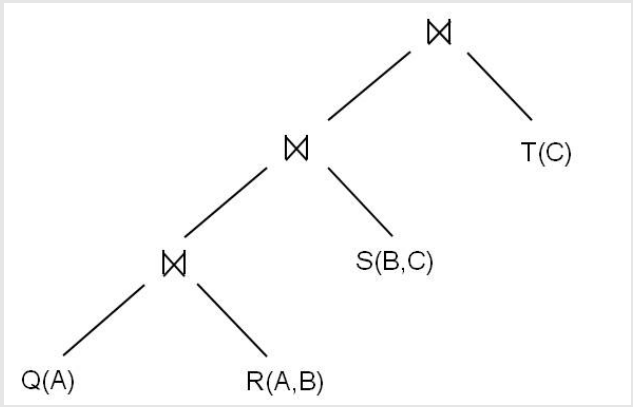


äquivalenzerhaltende Umformungen obiger Ausdrücke

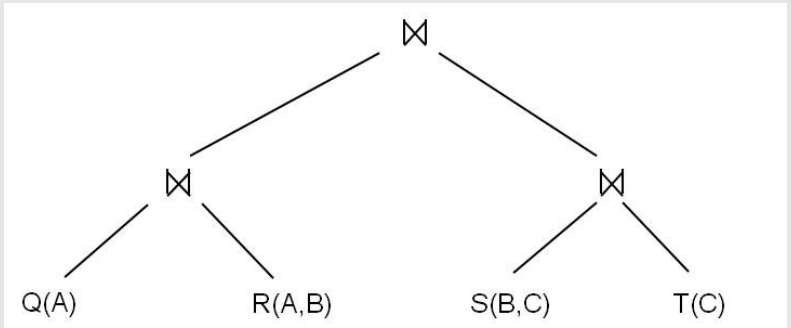


Aufgrund der Gültigkeit des Assoziativgesetzes für den Verbund ergeben sich vielfältige Möglichkeiten der Auswertung eines Verbundausdrucks mit $n > 2$ Relationen.

left-deep-tree Join-Auswertung



bushy-tree Join-Auswertung



Bei einer *left-deep-tree* Join-Auswertung ist im Unterschied zur *bushy-tree* Join-Auswertung ein Verbund-Partner jeweils immer bereits berechnet.

Physische Optimierung

- Grundlage für Auswahlentscheidungen der physischen Optimierung sind Informationen über die Relationen und ihre Indexstrukturen.
- Diese Informationen werden im *Katalog* gehalten.
- Für die Entscheidung, ob ein Index für die Auswertung einer Anfrage von Nutzen ist, ist die *Selektivität* des Index eine wichtige Information.

  Die Selektivität ist umso stärker, je kleiner ihr Wert ist.
  - $\sigma[\alpha](R)$: $sel_\alpha = \frac{|\sigma[\alpha](R)|}{|R|}$.
    Spezialfall $\alpha \equiv A = c$ und A Schlüssel von R?
    $sel_\alpha = \frac{1}{|R|}$.
  - $R \bowtie S$: $sel_{RS} := \frac{|R \bowtie S|}{|R \times S|} = \frac{|R \bowtie S|}{|R| \cdot |S|}$.
    Spezialfall $R \bowtie_{R.A=S.B} S$ und A Schlüssel?
    $sel_{RS} \leq \frac{1}{|R|}$ wegen $| R \bowtie_{R.A=S.B} S | \leq |S|$.

left-deep-tree vs. *bushy-tree* Join-Auswertung

- Bei einer *left-deep-tree* Join-Auswertung, z.B. $(R \bowtie S) \bowtie T$, kann auf eine vollständige Materialisierung der Zwischenergebnisse von $R \bowtie S$ verzichtet werden; Zwischenresultate werden direkt an den Folgeoperator weitergereicht, hier $\bowtie T$.

  Diese Technik ist allgemeiner anwendbar, z.B. auch bei $\sigma[A = 5 \wedge B > 6]R$, wobei je ein Index über A und B angenommen wird; man redet von *Pipelining*.
- Bei einer *bushy-tree* Join-Auswertung, z.B. $(Q \bowtie R) \bowtie (S \bowtie T)$ müssen in der Regel beide Zwischenergebnisse zuerst berechnet werden, bevor der Verbund berechnet werden kann; das Berechnen der Zwischenergebnisse zu $(Q \bowtie R)$ und $(S \bowtie T)$ kann jedoch *parallel* erfolgen.

8.6 empfohlene Lektüre

Access Path Selection
in a Relational Database Management System

P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and complex queries (such as joins), given a user specification of desired data as a boolean expression of predicates. System R is an experimental database management system developed to carry out research on the relational model of data. System R was designed and built by members of the IBM San Jose Research Laboratory.

retrieval. Nor does a user specify in what order joins are to be performed. The System R optimizer chooses both join order and an access path for each table in the SQL statement. Of the many possible choices, the optimizer chooses the one which minimizes "total access cost" for performing the entire statement.

This paper will address the issues of access path selection for queries. Retrieval for data manipulation (UPDATE, DELETE) is treated similarly. Section 2 will describe the place of the optimizer in the processing of a SQL statement, and

2

[2] In: ACM SIGMOD Conference, 1979. Kann gegoogelt werden.