6.4 Minimale Überdeckung: Basis für Normalisierungsalgorithmen

- Seien $\mathcal{F}$ Mengen von funktionalen Abhängigkeiten.
- Wir suchen eine *"minimale" Überdeckung* von $\mathcal{F}$.

  $\mathcal{G}$ überdeckt $\mathcal{F}$, wenn $\mathcal{F} \equiv \mathcal{G}$, d.h. $\mathcal{F}^+ = \mathcal{G}^+$.
- Strategie: Bilde $\mathcal{G}$ durch Streichen von Attributen in den FAs von $\mathcal{F}$ oder Entfernen von FAs in $\mathcal{F}$ in einer Weise, die die Äquivalenz nicht zerstört.

Beispiel Rechtsreduktion

- $\mathcal{F}_1 = \{A \to B, B \to A, B \to C, A \to C, C \to A\}$.

  Kann die FA $B \to C$ zu $B \to \emptyset$ reduziert werden, d.h. gestrichen werden?

  Sei $\mathcal{F}'_1 = \{A \to B, B \to A, A \to C, C \to A\}$.

  Gilt $\mathcal{F}_1^+ = \mathcal{F}_1'^+$? Ja, denn

  (a) $\mathcal{F}_1^+ \subseteq \mathcal{F}_1'^+$ wegen $XPlus(B, C, \mathcal{F}'_1)$.
  (b) $\mathcal{F}_1^+ \supseteq \mathcal{F}_1'^+$ wegen $\mathcal{F}_1 \supseteq \mathcal{F}'_1$.

Beispiel Linksreduktion

- $\mathcal{F}_2 = \{AB \to C, A \to B, B \to A\}$.

  Kann die FA $AB \to C$ zu $B \to C$ reduziert werden, d.h. $AB \to C$ durch $B \to C$ ersetzt werden?

  Sei $\mathcal{F}_2' = \{B \to C, A \to B, B \to A\}$.

  Gilt $\mathcal{F}_2^+ = \mathcal{F}_2'^+$? Ja, denn

  (a) $\mathcal{F}_2^+ \subseteq \mathcal{F}_2'^+$ wegen (A2) und (A6) angewendet auf $B \to C$.
  (b) $\mathcal{F}_2^+ \supseteq \mathcal{F}_2'^+$ wegen $XPlus(B, C, \mathcal{F}_2)$.

Links- und Rechtsreduktion

- Eine Menge $\mathcal{F}$ funktionaler Abhängigkeiten heißt *linksreduziert*, wenn sie die folgende Eigenschaft erfüllt.

  Wenn $X \to Y \in \mathcal{F}, Z \subseteq X$, dann $\mathcal{F}' = (\mathcal{F} \setminus \{X \to Y\}) \cup \{Z \to Y\}$ nicht äquivalent zu $\mathcal{F}$.
- *Linksreduktion:* ersetze $X \to Y$ in $\mathcal{F}$ durch $Z \to Y$.
- Sie heißt *rechtsreduziert*, wenn $X \to Y \in \mathcal{F}, Z \subseteq Y$, dann $\mathcal{F}' = (\mathcal{F} \setminus \{X \to Y\}) \cup \{X \to Z\}$ nicht äquivalent zu $\mathcal{F}$.
- *Rechtsreduktion:* ersetze $X \to Y$ in $\mathcal{F}$ durch $X \to Z$.

Entscheidung mittels XPlus-Algorithmus

- Sei $X \to Y$ eine Abhängigkeit in $\mathcal{F}$ und sei $Z \to Y$, wobei $Z \subseteq X$.

  Wir führen die entsprechende Linksreduktion durch, wenn $XPlus(Z, Y, \mathcal{F})$ das Ergebnis `true` liefert.
- Sei $X \to Y$ eine Abhängigkeit in $\mathcal{F}$ und sei $X \to Z$, wobei $Z \subseteq Y$.

  Wir führen die entsprechende Rechtsreduktion durch, wenn $XPlus(X, Y, \mathcal{F}')$ das Ergebnis `true` liefert.

Satz

Sei eine Menge funktionaler Abhängigkeiten $\mathcal{F}$ gegeben und sei $\mathcal{F}'$ aus $\mathcal{F}$ durch eine Links- oder Rechtsreduktion hervorgegangen.

Dann $\mathcal{F} \equiv \mathcal{F}'$.

minimale Überdeckung

Eine Menge funktionaler Abhängigkeiten $\mathcal{F}^{min}$ ist eine *minimale Überdeckung* zu $\mathcal{F}$, wenn wir sie durch Anwendung der folgenden Schritte erzeugen können:

- Führe alle möglichen Linksreduktionen durch.
- Führe alle möglichen Rechtsreduktionen durch.
- Streiche alle trivialen funktionalen Abhängigkeiten der Form $X \rightarrow \emptyset$.
- Vereinige alle funktionalen Abhängigkeiten mit gleicher linker Seite $X \rightarrow Y_1$, ..., $X \rightarrow Y_n$ zu einer einzigen FA der Form $X \rightarrow Y_1 \ldots Y_n$.

6.5 Algorithmus zur Normalisierung

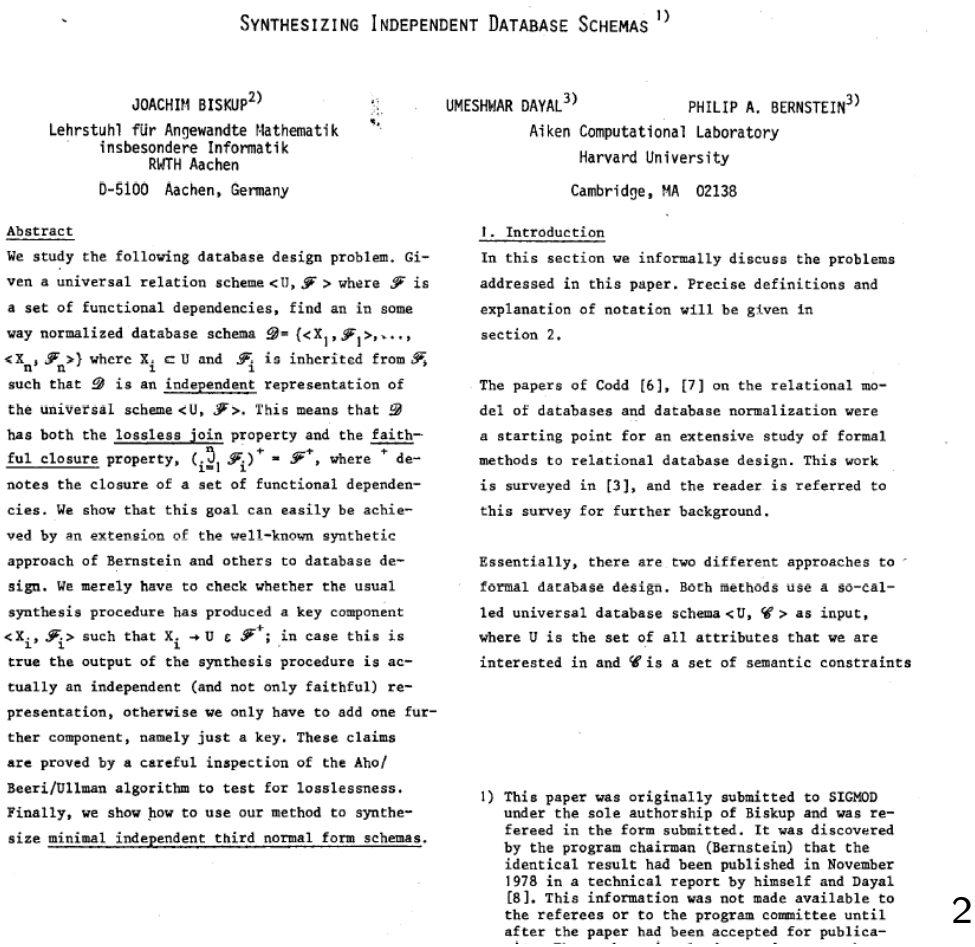
3NF-Synthese: verlustfrei und abhängigkeitsbewahrend

Sei $R = (V, \mathcal{F})$ ein Relationsschema.

1. Sei $\mathcal{F}^{min}$ eine minimale Überdeckung zu $\mathcal{F}$.
2. Betrachte jeweils maximale Klassen von funktionalen Abhängigkeiten aus $\mathcal{F}^{min}$ mit derselben linken Seite. Seien $\mathcal{C}_i = \{X_i \to A_{i1}, X_i \to A_{i2}, \ldots\}$ die so gebildeten Klassen, $i \geq 0$.[1]
3. Bilde zu jeder Klasse $\mathcal{C}_i$, $i \geq 0$, ein Schema mit Format $V_{\mathcal{C}_i} = X_i \cup \{A_{i1}, A_{i2}, \ldots\}$.
4. Sofern keines der gebildeten Formate $V_{\mathcal{C}_i}$ einen Schlüssel für R enthält, berechne einen Schlüssel für R. Sei Y ein solcher Schlüssel. Bilde zu Y ein Schema mit Format $V_K = Y$.
5. $\rho = \{V_K, V_{\mathcal{C}_1}, V_{\mathcal{C}_2}, \ldots\}$ ist eine verlustfreie und abhängigkeitsbewahrende Zerlegung von R in 3NF.

[1] Der von uns betrachtete Algorithmus zur Berechnung von $\mathcal{F}^{min}$ hat diese Klassenbildung bereits vorgenommen.

6.6 empfohlene Lektüre



SYNTHESIZING INDEPENDENT DATABASE SCHEMAS [1)]

JOACHIM BISKUP[2)]
Lehrstuhl für Angewandte Mathematik
insbesondere Informatik
RWTH Aachen
D-5100 Aachen, Germany

UMESHWAR DAYAL[3)] PHILIP A. BERNSTEIN[3)]
Aiken Computational Laboratory
Harvard University
Cambridge, MA 02138

Abstract

We study the following database design problem. Given a universal relation scheme $\langle U, \mathcal{F} \rangle$ where $\mathcal{F}$ is a set of functional dependencies, find an in some way normalized database schema $\mathcal{D} = \{\langle X_1, \mathcal{F}_1 \rangle, \ldots, \langle X_n, \mathcal{F}_n \rangle\}$ where $X_i \subset U$ and $\mathcal{F}_i$ is inherited from $\mathcal{F}$, such that $\mathcal{D}$ is an independent representation of the universal scheme $\langle U, \mathcal{F} \rangle$. This means that $\mathcal{D}$ has both the lossless join property and the faithful closure property, $(\bigcup_{i=1}^{n} \mathcal{F}_i)^+ = \mathcal{F}^+$, where $^+$ denotes the closure of a set of functional dependencies. We show that this goal can easily be achieved by an extension of the well-known synthetic approach of Bernstein and others to database design. We merely have to check whether the usual synthesis procedure has produced a key component $\langle X_i, \mathcal{F}_i \rangle$ such that $X_i \rightarrow U \in \mathcal{F}^+$; in case this is true the output of the synthesis procedure is actually an independent (and not only faithful) representation, otherwise we only have to add one further component, namely just a key. These claims are proved by a careful inspection of the Aho/Beeri/Ullman algorithm to test for losslessness. Finally, we show how to use our method to synthesize minimal independent third normal form schemas.

1. Introduction

In this section we informally discuss the problems addressed in this paper. Precise definitions and explanation of notation will be given in section 2.

The papers of Codd [6], [7] on the relational model of databases and database normalization were a starting point for an extensive study of formal methods to relational database design. This work is surveyed in [3], and the reader is referred to this survey for further background.

Essentially, there are two different approaches to formal database design. Both methods use a so-called universal database schema $\langle U, \mathcal{G} \rangle$ as input, where U is the set of all attributes that we are interested in and $\mathcal{G}$ is a set of semantic constraints

1) This paper was originally submitted to SIGMOD under the sole authorship of Biskup and was refereed in the form submitted. It was discovered by the program chairman (Bernstein) that the identical result had been published in November 1978 in a technical report by himself and Dayal [8]. This information was not made available to the referees or to the program committee until after the paper had been accepted for publica-

2

[2] In: Proceeding SIGMOD '79 Proceedings of the 1979 ACM SIGMOD international conference on Management of data. Kann gegoogelt werden.

Kapitel 7: Physischer Datenbankentwurf

- Speicherung und Verwaltung der Relationen einer relationalen Datenbank so, dass eine möglichst große Effizienz der einzelnen Anwendungen auf der Datenbank ermöglicht wird.
- Vor dem Hintergrund eines logischen Schemas und den Anforderungen der geplanten Anwendungen wird ein geeignetes *physisches Schema* definiert.

7.1 Grundlagen

Begriffe

- physische Datenbank: Menge von *Dateien*,
- Datei: Menge von gleichartigen *Sätzen*,
- Satz: In ein oder mehrere *Felder* unterteilter Speicherbereich.
- relationale Datenbanken: Relationen, Tupel und Attribute.

Indexstrukturen

Hilfsstrukturen zur Gewährleistung eines effizienten Zugriffs zu den Daten einer physischen Datenbank.

Zugriffseinheiten

- *Seiten*, oder auch *Blöcke*,
- ein Block besteht aus einer Reihe aufeinander folgender Seiten,
- eine Seite enthält in der Regel mehrere Tupel.
- Typischerweise hat eine Seite eine Größe von 4KB bis 8KB und ein Block eine Größe von bis zu 32KB.

Speicherhierarchie

- *Primärspeicher* (*interner Speicher*) bestehend aus *Cache* und einem in Seitenrahmen unterteilten *Hauptspeicher*. Direkter Zugriff im Nanosekundenbereich.
- *Sekundärspeicher* (*externer Speicher*), typischerweise in Seiten unterteilter externer Plattenpeicher. Direkter Zugriff im Millisekundenbereich.
- *Tertiärspeicher* (*Archiv*): kein direkter Zugriff möglich.

Pufferverwaltung

- Anzahl benötigter Seiten einer Datenbank typischerweise erheblich größer, als die Anzahl zur Verfügung stehenden Seitenrahmen.
- *Datenbankpuffer*: im Hauptspeicher für die Datenbank verfügbare Seitenrahmen.
- $\Longrightarrow$ *Pufferverwaltung*: für angeforderte Seiten der Datenbank muss ein Seitenrahmen im Puffer gefunden werden; sonst *Seitenfehler*.
- Für jeden Seitenrahmen unterhält die Pufferverwaltung die Variablen *pin-count* und *dirty*; *pin-count* gibt die Anzahl Prozesse an, die die aktuelle Seite im Rahmen angefordert haben und noch nicht freigegeben haben, *dirty* gibt an, ob der Inhalt der Seite verändert wurde.

- Es ist aus Effizienzgründen häufig sinnvoll,
  - geänderte Seiten nicht direkt in die Datenbank zurück zuschreiben,
  - ein *Prefetching* der Seiten vorzunehmen.
- Eine Änderung heißt *materialisiert*, wenn die entsprechende Seite in den externen Speicher der Datenbank zurückgeschrieben ist.

Adressierung

- *Tupelidentifikatoren* der Form $Tid = (b, s)$, wobei b eine Seitenummer und s eine Tupelnummer innerhalb der Seite b.
- Mittels eines in der betreffenden Seite abgelegten *Adressbuchs* (*Directory*) kann in Abhängigkeit des Wertes s die physische Adresse des Tupels in der Seite gefunden werden.
- Tupel sind lediglich in ihrer Seite frei verschiebbar.
- Verschiebbarkeit über Seitengrenzen mittels *Verweisketten*.
- Tupel mit variabel langen Attributwerten verlangen *Längenzähler*, bzw. eine *Zeigerliste*.

7.2 Dateiorganisationsformen und Indexstrukturen

Zugriffsarten

- *Durchsuchen* (*scan*) einer gesamten Relation.
- *Suchen* (*search*). Lesen derjenigen Tupel, die eine über ihren Attributen formulierte Bedingung erfüllen.

  Bereichssuchen (*range search*): in den Bedingungen über den Attributen stehen nicht ausschließlich Gleichheitsoperatoren, sondern beliebige andere arithmetische Vergleichsoperatoren.
- *Einfügen* (*insert*) eines Tupels in eine Seite.
- *Löschen* (*delete*) eines Tupels in einer Seite.
- *Ändern* (*update*) des Inhalts eines Tupels einer Seite.

Haufenorganisation

- Zufällige Anordnung der Tupel in den Seiten.
- Suchen eines Tupels benötigt im Mittel $\frac{n}{2}$ Seitenzugriffe.
- Einfügen eines neuen Tupels benötigt 2 Seitenzugriffe.
- Löschen und Ändern eines Tupels benötigen Anzahl-Suchen plus 1 Seitenzugriffe.

Suchbaumindex

- Die Tupel sind nach dem Suchschlüssel sortiert.
- Suchen eines Tupels benötigt eine logarithmische Anzahl Seitenzugriffe.
- Einfügen, Löschen und Ändern eines Tupels: s. später.

Hashindex

- Mittels einer Hashfunktion wird einem Suchschlüsselwert eine Seitennummer zugeordnet.
- Suchen eines Tupels benötigt in der Regel ein bis zwei Seitenzugriffe.
- Einfügen, Löschen und Ändern eines Tupels benötigen in der Regel zwei bis drei Seitenzugriffe.

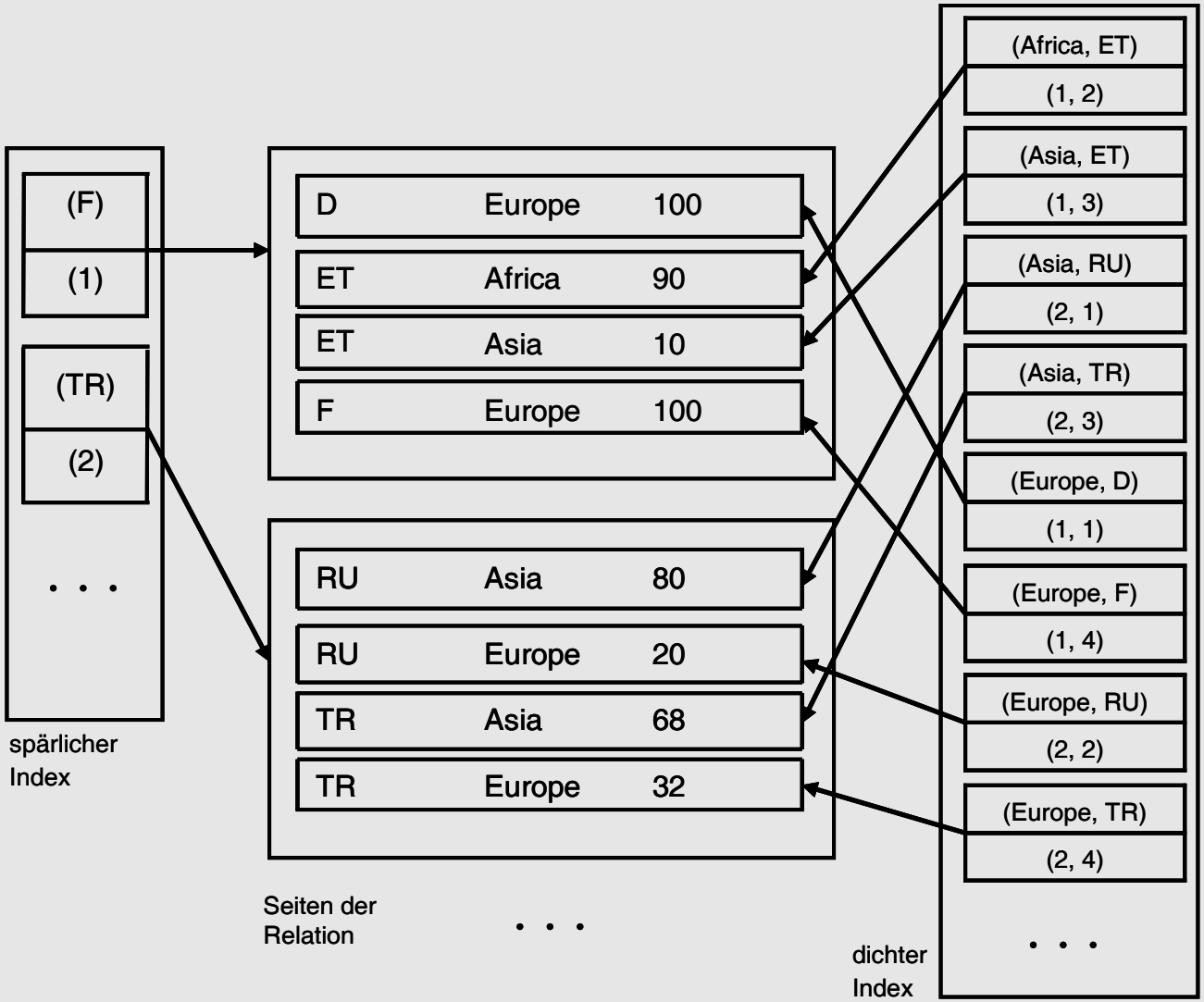
Primär- und Sekundärindex

- Ein *Suchschlüssel* ist die einem Index zugrunde liegende Attributkombination.
- Bei einem *Primärindex* ist der Primärschlüssel der Relation im Suchschlüssel enthalten.

  Sekundärindex anderenfalls. Ein Suchschlüsselwert identifiziert i.A. eine Menge von Tupeln.

Zu einer Relation existieren typischerweise mehrere Indexstrukturen über jeweils unterschiedlichen Suchschlüsseln.

Beispiel


(F)
(1)
(TR)
(2)
spärlicher Index
D Europe 100
ET Africa 90
ET Asia 10
F Europe 100
RU Asia 80
RU Europe 20
TR Asia 68
TR Europe 32
Seiten der Relation
(Africa, ET)
(1, 2)
(Asia, ET)
(1, 3)
(Asia, RU)
(2, 1)
(Asia, TR)
(2, 3)
(Europe, D)
(1, 1)
(Europe, F)
(1, 4)
(Europe, RU)
(2, 2)
(Europe, TR)
(2, 4)
dichter Index

weitere Indexformen

- *dicht*: pro Suchschlüsselwert der Tupel der betreffenden Relation einen Verweis,
- *spärlich*: pro Intervall von Suchschlüsselwerten einen Verweis.
- *geballt (engl. clustered)*: logisch zusammengehörende Tupel sind auch physisch benachbart gespeichert.

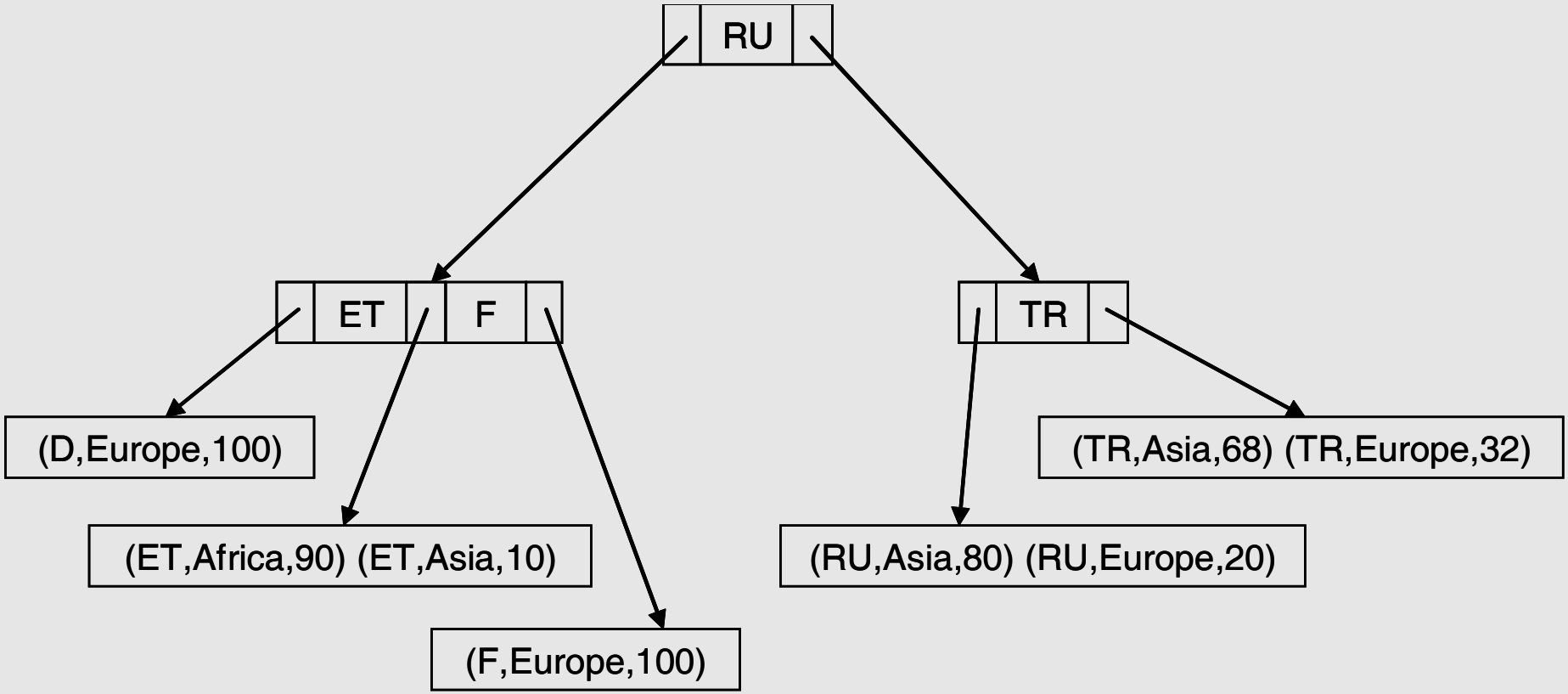
- Eine Relation heißt *invertiert* bzgl. einem Attribut, wenn ein dichter Sekundärindex zu diesem Attribut existiert,
- sie heißt *voll invertiert*, wenn zu jedem Attribut ein dichter Sekundärindex existiert.

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 $(3, 2)$



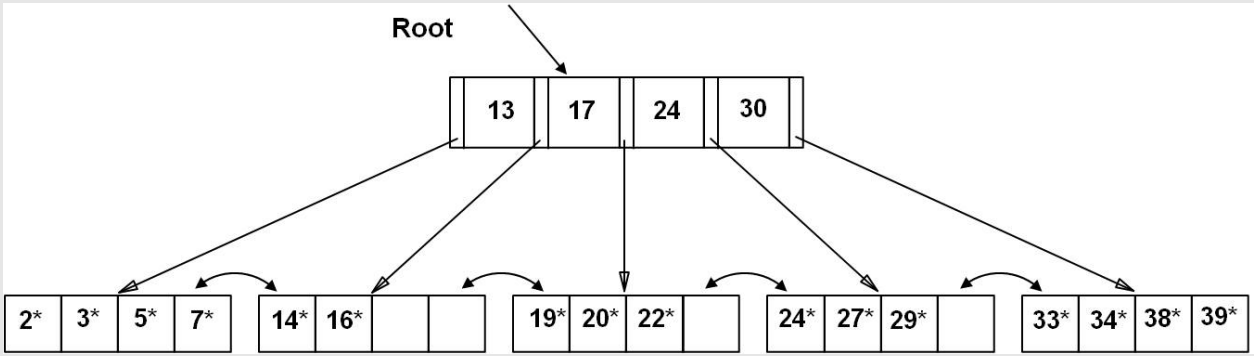
Höhe h:

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

B-Baum in der Praxis

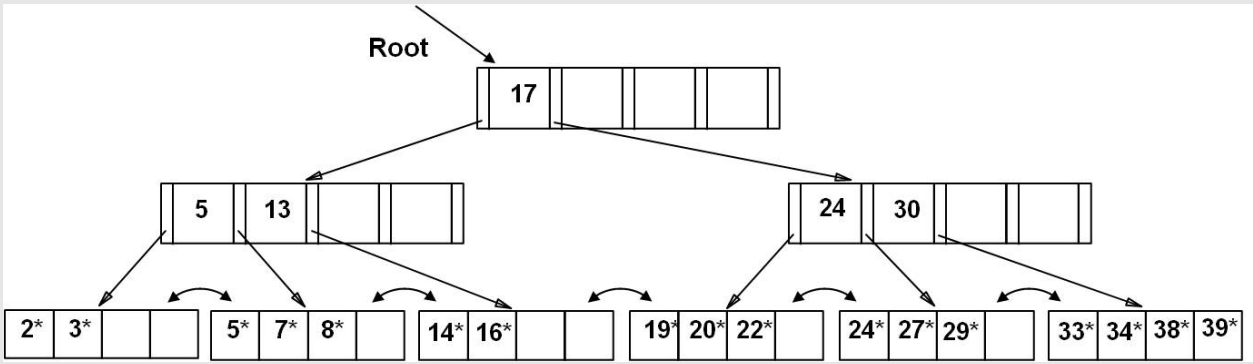
- Typisch: Ordnung $m = 200$, Füllungsgrad 67%, Verzweigungsgrad 133, Seitengröße 8Kbytes.
- $h = 1$: 133 Blätter, 1 Mbyte,
  $h = 2$: $133^2 = 17.689$ Blätter, 133 Mbyte,
  $h = 3$: $133^3 = 2.352.637$ Blätter, 17 Gbyte
  $h = 4$: $133^4 = 312.900.700$ Blätter, 2 Tbyte.
- Kann u.U. bis zur Höhe 2 im 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

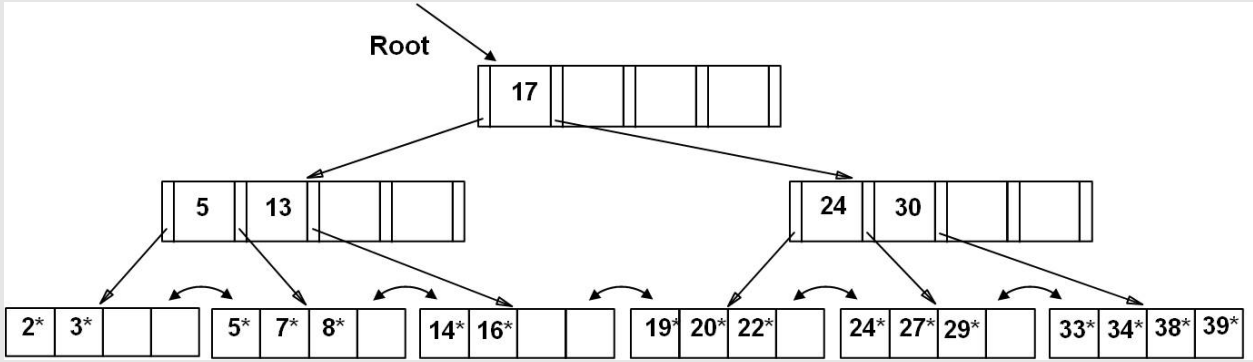


- Einfügen in das zugehörige Blatt mit möglichem Überlauf.
- In diesem Fall durch Aufteilen ein neues Blatt, bzw. einen neuen Knoten des Baumes erzeugen und die zugehörige Verzweigungsinformation in den Elterknoten rekursiv einfügen.
- Die Höhe des Baumes kann bei diesem Verfahren um maximal 1 wachsen.

Ergebnis

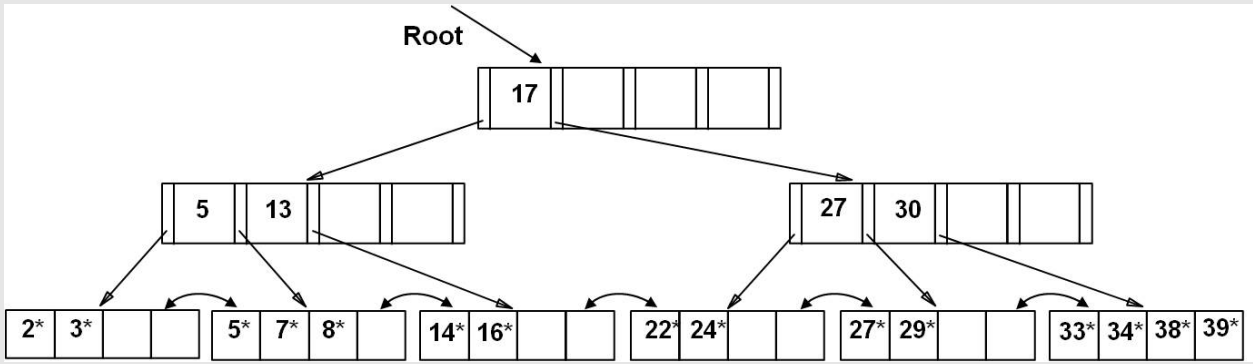


Löschen der Tupel mit Schlüsselwerten 19 und 20

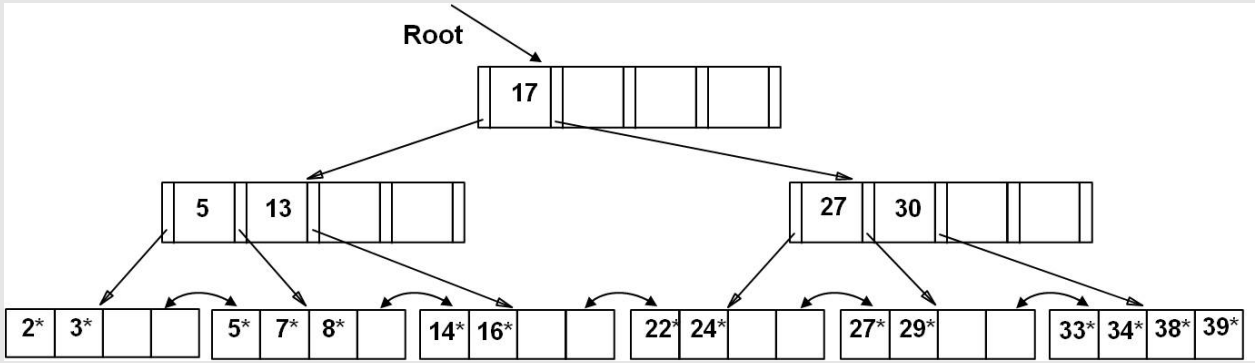


- Löschen des Tupels 19 ist unproblematisch.
- Nachfolgendes Löschen des Tupels 20 erzeugt einen Unterlauf im Blatt.
- Neuverteilen der Tupel mit einem Nachbarblatt so, dass beide Blätter die Mindestfüllung erreichen.
- Falls dies gelingt, Anpassen der Verzweigungsinformation im Elterknoten.

Ergebnis

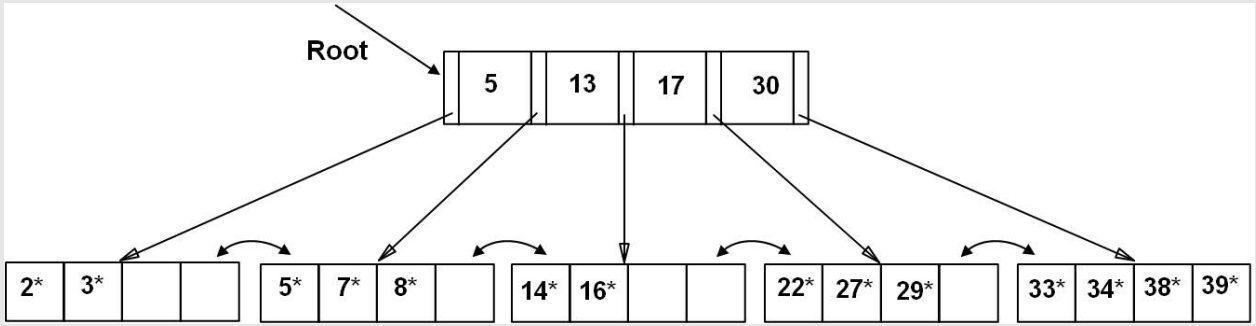


Löschen des Tupels mit Schlüsselwert 24



- 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.

3

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