3. Normalform

Ein Relationsschema $R = (V, \mathcal{F})$ ist in *3. Normalform* (3NF) genau dann, wenn jedes NSA $A \in V$ die folgende Bedingung erfüllt.

Wenn $X \to A \in \mathcal{F}$, $A \notin X$, dann ist X ein Superschlüssel.

Die Bedingung der 3NF verbietet nichttriviale funktionale Abhängigkeiten $X \to A$, in denen ein NSA A in der Weise von einem Schlüssel K transitiv funktional abhängt, dass $K \to X$, $K \not\subseteq X$ gilt und des Weiteren $X \to A$.

Boyce-Codd-Normalform

Ein Relationsschema $R = (V, \mathcal{F})$ ist in *Boyce-Codd-Normalform* (BCNF) genau dann, wenn die folgende Bedingung erfüllt ist.

Wenn $X \to A \in \mathcal{F}$, $A \notin X$, dann ist X ein Superschlüssel.

Die BCNF verschärft die 3NF.

- Sei $R = (V, \mathcal{F})$, wobei $V = \{$ `Stadt, Adresse, PLZ` $\}$, und $\mathcal{F} = \{$ `Stadt Adresse` $\to$ `PLZ, PLZ` $\to$ `Stadt`$\}$.
- R ist in 3NF, aber nicht in BCNF.
- Sei $\rho = \{$`Adresse PLZ, PLZ Stadt`$\}$ eine Zerlegung, dann erfüllt ρ die BCNF und ist verlustfrei, jedoch nicht abhängigkeitsbewahrend.

BCNF-Analyse: verlustfrei und nicht abhängigkeitsbewahrend

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

1. Sei $X \subset V$, $A \in V$ und $X \to A \in \mathcal{F}$ eine FA, die die BCNF-Bedingung verletzt.[1] Sei weiter $V' = V \setminus \{A\}$.

   Zerlege R in

   $$R_1 = (V', \pi[V']\mathcal{F}), \quad R_2 = (XA, \pi[XA]\mathcal{F}).$$

2. Teste die BCNF-Bedingung bzgl. R_1 und R_2 und wende den Algorithmus gegebenenfalls rekursiv an.

[1] Um eine in den gegebenen FAs ausgedrückte Zusammengehörigkeit von Attributen nicht zu verlieren, kann anstatt $X \to A$ die FA $X \to (X^+ \setminus X)$ betrachtet werden.

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, $i \geq 0$.
3. Bilde zu jeder Klasse $\mathcal{C}_i$, $i \geq 0$, ein Schema mit Format $V_{\mathcal{C}_i} = X_i \cup \{A_{i1}, A_{i2}, \ldots\}$.[2]
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.

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

3NF & BCNF

Seien gegeben

$$V = \{SNr, SName, LCode, LFlaeche, KName, KFlaeche, Prozent\},$$
$$\mathcal{F} = \{SNr \rightarrow SName\ LCode, LCode \rightarrow LFlaeche,$$
$$KName \rightarrow KFlaeche, LCode\ KName \rightarrow Prozent\}.$$

(a) Geben Sie eine verlustfreie und abhängigkeitsbewahrende 3NF-Zerlegung an, indem Sie den 3NF-Synthese-Algorithmus anwenden.

(b) Geben Sie eine verlustfreie und möglichst abhängigkeitsbewahrende BCNF-Zerlegung an, indem Sie den BCNF-Analyse-Algorithmus anwenden.

Kennzeichnen Sie jeweils die Schlüssel der Schemata der gefundenen Zerlegungen.

(a) Lösung mit 3NF-Synthese-Algorithmus

Es gilt

$$\mathcal{F} = \{SNr \rightarrow SName\ LCode, LCode \rightarrow LFlaeche,$$
$$KName \rightarrow KFlaeche, LCode\ KName \rightarrow Prozent\} = \mathcal{F}^{min}.$$

Einziger Schlüssel ist $\{SNr, KName\}$. Bei Anwendung des Algorithmus ergeben sich zunächst die Schemata mit den Formaten

- $V_{C_1} = \{\underline{SNr}, SName, LCode\}$,
- $V_{C_2} = \{\underline{LCode}, LFlaeche\}$,
- $V_{C_3} = \{\underline{KName}, KFlaeche\}$,
- $V_{C_4} = \{\underline{LCode}, \underline{KName}, Prozent\}$.

$\Rightarrow$ Keines dieser Schemata enthält den Schlüssel $\{SNr, KName\}$.
Um Verlustfreiheit zu gewährleisten muss das Schema mit dem Format $V_K = \{\underline{SNr}, \underline{KName}\}$ hinzugenommen werden.

(b) Lösung mit BCNF-Analyse-Algorithmus

Zerlegung 1:

(1) Wähle FD $SNr \rightarrow (SNr^+ \setminus SNr)$, d.h. $SNr \rightarrow SName\ LCode\ LFlaeche$
- (1.1) $R_1 = (\{SNr, KName, KFlaeche, Prozent\}, \pi[V_{R_1}]\mathcal{F})$ und
- (1.2) $R_2 = (\{SNr, SName, LCode, LFlaeche\}, \pi[V_{R_2}]\mathcal{F})$

(2) Weitere Zerlegung von R_2 mit $LCode \rightarrow LFlaeche$ liefert:
- (2.1) $R_3 = (\{SNr, SName, LCode\}, \pi[V_{R_3}]\mathcal{F})$ und
- (2.2) $R_4 = (\{LCode, LFlaeche\}, \pi[V_{R_4}]\mathcal{F})$

(3) Weitere Zerlegung von R_1 mit $KName \rightarrow KFlaeche$ liefert:
- (3.1) $R_5 = (\{SNr, KName, Prozent\}, \pi[V_{R_5}]\mathcal{F})$ und
- (3.2) $R_6 = (\{KName, KFlaeche\}, \pi[V_{R_6}]\mathcal{F})$

Die Zerlegung $\rho_1 = \{R_3, R_4, R_5, R_6\}$ ist *nicht* abhängigkeitsbewahrend.
$\Rightarrow$ Verliere FD $LCode\ KName \rightarrow Prozent$[3].

[3] Diese kann nicht aus $SNr \rightarrow LCode, SNr\ KName \rightarrow Prozent$ gefolgert werden.

Lösung mit BCNF-Analyse-Algorithmus

Zerlegung 2:

(1) Wähle FD $LCode\ KName \rightarrow Prozent$:
- (1.1) $R_1 = (\{SNr, SName, LCode, LFlaeche, KName, KFlaeche\}, \pi[V_{R_1}]\mathcal{F})$ und
- (1.2) $R_2 = (\{LCode, KName, Prozent\}, \pi[V_{R_2}]\mathcal{F})$

(2) Weitere Zerlegung von R_1 mit $LCode \rightarrow LFlaeche$ liefert:
- (2.1) $R_3 = (\{SNr, SName, LCode, KName, KFlaeche\}, \pi[V_{R_3}]\mathcal{F})$ und
- (2.2) $R_4 = (\{LCode, LFlaeche\}, \pi[V_{R_4}]\mathcal{F})$

(3) Weitere Zerlegung von R_3 mit $KName \rightarrow KFlaeche$ liefert:
- (3.1) $R_5 = (\{SNr, SName, LCode, KName\}, \pi[V_{R_5}]\mathcal{F})$ und
- (3.2) $R_6 = (\{KName, KFlaeche\}, \pi[V_{R_6}]\mathcal{F})$

(4) Weitere Zerlegung von R_5 mit $SNr \rightarrow SName\ LCode$ liefert:
- (4.1) $R_7 = (\{SNr, KName\}, \pi[V_{R_7}]\mathcal{F})$ und
- (4.2) $R_8 = (\{SNr, SName, LCode\}, \pi[V_{R_8}]\mathcal{F})$

Somit ergibt sich die verlustfreie *und* abhängigkeitsbewahrende Zerlegung $\rho_2 = \{R_2, R_4, R_6, R_7, R_8\}$.

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 verfügbaren 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.

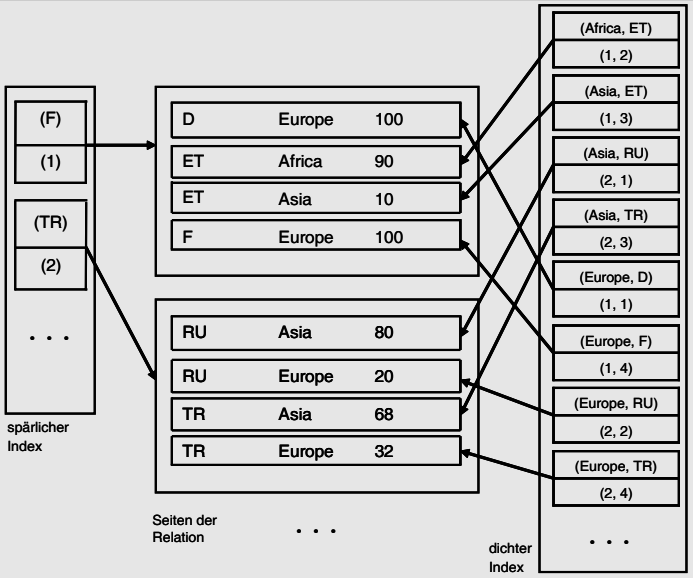
Primär- und Sekundärindex

- 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



weitere Indexformen

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

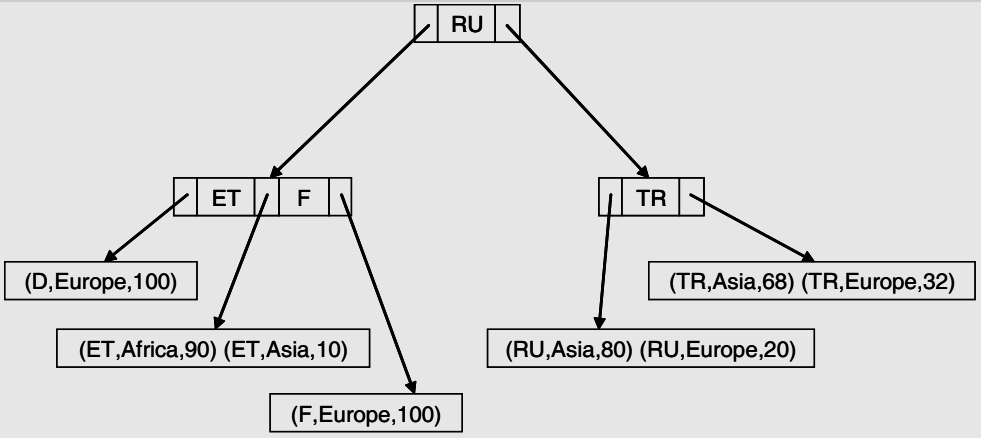
- Eine Relation heißt *invertiert* bzgl. einem Attribut, wenn ein dichter Sekundärindex zu diesem Attribut existiert,
- sie heißt *voll invertiert*, wenn zu jedem Attribut ein dichter Sekundärindex existiert.

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 $(3, 2)$



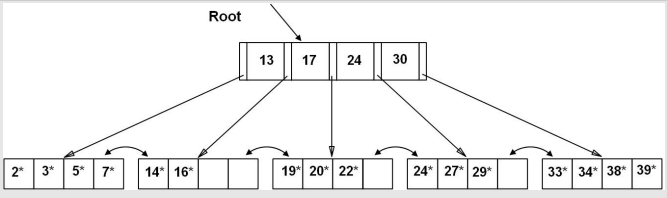
Höhe h:

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

B-Baum in der Praxis

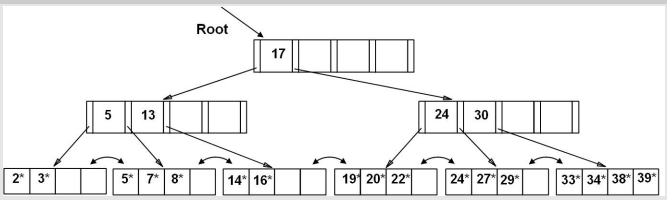
- Typisch: Ordnung $m = 200$, Füllungsgrad 67%, Verzweigungsgrad 133, Seitengröße 8Kbytes.
- $h = 1$: 133 Blätter, 1 Mbyte,
  $h = 2$: $133^2 = 17.689$ Blätter, 133 Mbyte,
  $h = 3$: $133^3 = 2.352.637$ Blätter, 17 Gbyte
  $h = 4$: $133^4 = 312.900.700$ Blätter, 2 Tbyte.
- Kann u.U. bis zur Höhe 2 im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

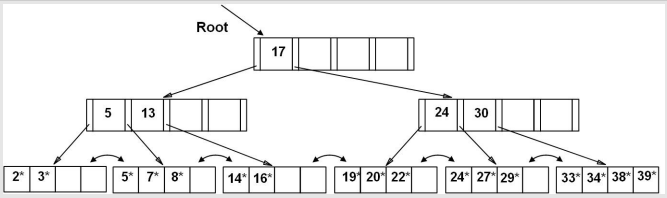


- Einfügen in das zugehörige Blatt mit möglichem Überlauf.
- In diesem Fall durch Aufteilen ein neues Blatt, bzw. einen neuen Knoten des Baumes erzeugen und die zugehörige Verzweigungsinformation in den Elterknoten rekursiv einfügen.
- Die Höhe des Baumes kann bei diesem Verfahren um maximal 1 wachsen.

Ergebnis

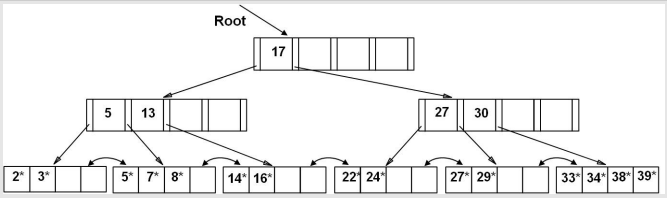


Löschen der Tupel mit Schlüsselwerten 19 und 20

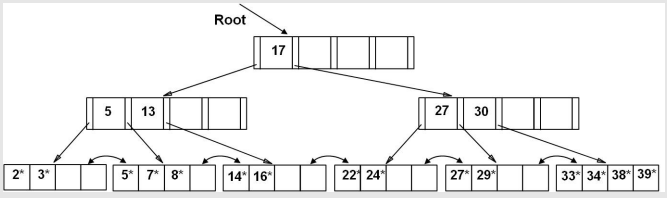


- Löschen des Tupels 19 ist unproblematisch.
- Nachfolgendes Löschen des Tupels 20 erzeugt einen Unterlauf im Blatt.
- Neuverteilen der Tupel mit einem Nachbarblatt so, dass beide Blätter die Mindestfüllung erreichen.
- Falls dies gelingt, Anpassen der Verzweigungsinformation im Elterknoten.

Ergebnis

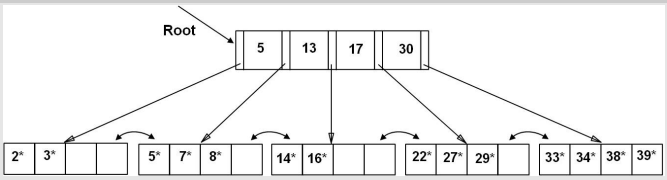


Löschen des Tupels mit Schlüsselwert 24



- 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, Tid), wobei *Tid* die Adresse des Tupels 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.

4

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

Ausblick

Berechnung des θ-Verbund

Seien ein Relationsschemata $R(A, B)$ und $S(C, D)$ mit Instanzen r und s gegeben, wobei gilt:

- r benötigt N und s benötigt M Seiten der Größe K,
- die Tupel in r und s haben gleiche Länge und eine Seite kann maximal k solcher Tupel aufnehmen.

$\Rightarrow$ Es soll der θ-Verbund $R \bowtie_{B=C} S$ berechnet werden.

Frage: Wieviele Seiten benötigt das Ergebnis der Verbundberechnung mindestens und höchstens[5]?

[5] Jede Seite soll jeweils maximal viele Tupel des Ergebnisses aufnehmen.