# Satzverwaltung

## Aufgabe

*Abbildung von Sätzen auf Blöcken*

- Techniken der Satzadressierung
  - TID Konzept (direkt)
  - Zuordnungstabelle (indirekt)
- Abbildung von Sätzen
- Realisierung "langer Felder"
- Freispeicherverwaltung

# Slotted Pages

*Wie ordnet man eine Menge von Sätzen auf eine Seite an?*

- Seite besteht aus drei Teilen
  - fixer Seitenkopf (z.B. mit Seitennr, LSN)
  - Slots (Verweise auf einzelne Tupel)
  - Repräsentation der Sätze
- Slot = $\langle$ Zeiger, Länge $\rangle$
- Platz für Slots wird von vorne nach hinten allokiert
- Platz für die Sätze wird von hinten nach vorne allokiert

**Fazit**

- Adresse eines Satzes (TID):

  $\langle$ Segmentnr., Seitennr., Slotnr. $\rangle$
- Sätze können problemlos innerhalb einer Seite migrieren

# Direkte Satzadressierung

## Das TID Konzept

- Zeiger werden als TID realisiert
- (Indexe enthalten z.B. Zeiger auf Sätze)
- Zugriff trivial, wenn Satz nicht auf eine andere Seite migriert
- (Migration z.B. notwendig weil Satz wächst und nicht mehr auf die Seite paßt)
- bei Migration wird ein *forward* in der Heimatseite angelegt
- Zugriff erfolgt durch Indirektion mit dem *forward*
- bei jeder weiteren Migration wird der *forward* angepaßt

## Bewertung

- minimal 1 Seitenzugriff erforderlich
- maximal 2 Seitenzugriffe erforderlich

# Indirekte Satzadressierung

- separate Zuordnungstabelle speichert Adresse eines Satzes
- bei Migration eines Satzes wird der Eintrag des Satzes in der Zuordnungstabelle angepaßt
- Zeiger werden als Indizes in der Zuordnungstabelle realisiert

## Bewertung

- Nachteil (?): immer genau 2 Seitenzugriffe erforderlich
- Vorteil: Indizes in der Zuordnungstabelle sind kleiner als TIDs
- Vorteil: kein Verschnitt in Datenseiten durch *forwards*; i.e., Zuordnungstabelle kann sehr kompakt gespeichert werden
- mehr zur Bewertung: siehe Übung

# Indirekte Satzadressierung, Alternative

## Prinzip

- Implementiere Zeiger auf Satz als Wert des Primärschlüssel des Satzes
- Speicherung von Sätzen in $B^+$-Bäumen

## Bewertung

- Vorteil: Sortierung der Tabelle wird garantiert
- Nachteil: teurer Zugriff durch mehrstufigen $B^+$ Baum
- Vorteil: keine zusätzliche Freispeicherverwaltung notwendig
- Nachteil: Speicherauslastung schlecht (max. 70% in der Regel)

# Probable Page Pointers

## Prinzip

- Erweiterung bisheriger Verfahren mit zusätzlichem Hint
- i.e., Zeiger wird mit zusätzlichem PPP versehen
- suche nach Datensatz zuerst unter Verwendung des PPP
- nur bei Mißerfolg Dereferenzierung nach TID Konzept oder mit Zuordnungstabelle
- update des PPP bei Mißerfolg

## Bewertung

- Vorteil: in der Regel kommt man mit einem Seitenzugriff aus (richtiger PPP)
- Nachteil: im schlimmsten Fall sind drei Seitenzugriffe notwendig
- Nachteil: große Zeiger
- Nachteil: zusätzliche Kosten durch Update des PPP

# Abbildung von Sätzen

## Trennung von Metadaten und Nutzdaten

- Speichere im Katalog
  - Attributname
  - Typ
  - Offset oder Position
- Speichere im Datensatz
  - Wert
  - (Repräsentation hängt vom Typ ab)

## N.B.:

- Die XML Welt zeichnet sich genau dadurch aus, daß Attributnamen (und Elementnamen) in jedem Datensatz zusätzlich gespeichert werden.
- Erstaunlicherweise sind XML Dokumente trotzdem kleiner als relationale Datenbasen, die dieselbe Information speichern.
- Mehr dazu in Vorlesung: Verteilte Datenbanken.

# Satzlayout

## Teil fixer Länge

- speichert alle Werte, die einen Typ fixer Länge haben
- z.B. *number(10,2), date, char[100]*
- Vorteil: auf solche Werte kann direkt zugegriffen werden

## Teil variabler Länge

- notwendig für *varchars*
- speichere *Länge* und *Zeiger* im fixen Teil
- speichere eigentlichen Wert im variablen Teil
- Wichtig: die Anwesenheit von *varchars* stört den direkten Zugriff auf *dates* etc. nicht!

## NULL Werte

- kleine Bitmap (fixer Länge) am Anfang des Satzes
- "1" falls Attribut den Wert NULL hat; "0" ansonsten
- Vorteil dieses Verfahrens: einfach und schnell
- Nachteil: Platzverschwendung, wenn viele NULL Werte

## Komprimierung

- ganze Sätze vs. einzelne Felder
- Sieger: einzelne Felder
- spezielle Dekodierungsfunktionen für Effizienz
- sehr effiziente Repräsentation von NULL Werte Teil des Komprimierungsschemas
- Literatur: Westmann, Kossmann, Helmer, Moerkotte: The Implementation and Performance of Compressed Databases; Technischer Bericht, Universität Mannheim, 1998.

# Realisierung langer Felder

## Problem

- Was ist mit Datensätzen, die größer als eine Seite sind?

## Lösung

- Satz / Feld wird in seitengroße Stücke zerteilt
- ein Index (Hashtabelle oder B-Baum) wird für diese Stücke angelegt
- etwas komplizierterer Index und Puffermechanismus notwendig, wenn der wahlfreie Zugriff auf einzelne Bereiche unterstützt werden soll (i.e., Listenstrukturen)
- **Mehr dazu in OO-OR Vorlesung**

# Freispeicherverwaltung

## Problem

- finde Heimat für einen neuen Satz
- oder finde neue Heimat, falls ein Satz für seine alte Heimat zu groß wird

## Ansätze

- Speicherung von Sätzen in $B^+$ Bäumen
- Append Only
- Best Fit, Next Fit
- Hybrid Verfahren

## Kriterien

- Geschwindigkeit der Suche
- Speicherauslastung
- geringer Hauptspeicherbedarf

**Literatur:** McAuliffe et al., SIGMOD 1996

# Append Only

## Prinzip

- betrachte immer nur die letzte erzeugte Seite
- wenn der Satz in dieser paßt, okay
- ansonsten, erzeuge eine neue Seite

## Bewertung

- sehr schnelles Einfügen
- schlechte Speicherauslastung
- quasi kein zusätzlicher Hauptspeicherbedarf

## Verallgemeinerung AO(n)

- betrachte die letzten $n$ erzeugten Seiten
- erhöht Speicherauslastung
- erhöht Hauptspeicherbedarf und Antwortzeit

# Best Fit, First Fit, Next Fit

**Auf alle Fälle** halte eine Space Map

- **Best Fit.**
  - suche vom Anfang der Liste die geeigneteste Seite
  - Aufwand: durchsuche komplette Liste jedes Mal
- **First Fit**
  - suche vom Anfang der Liste bis passende Seite
  - Nachteil: bald sind alle Seiten am Anfang voll; nutzlose Suche am Anfang
- **Next Fit**
  - halte *Cursor*; starte Suche immer beim Cursor
  - für BS super; hier schlecht, falls kein Platz vorhanden
  - beachte: suche verursacht Festplattenzugriffe

# Die Space Map

- klassifiziere jede Seite gemäß des freien Platzes
- einfachste Variante:
  - *open:* Seite hat freien Platz
  - *closed:* Seite hat keinen freien Platz
- fein granulare Varianten:
  - *bucket 0:* Seite ist zu 75% bis 100% belegt
  - *bucket 1:* Seite ist zu 50% bis 75% belegt
  - *bucket 2:* Seite ist zu 25% bis 50% belegt
  - *bucket 3:* Seite ist zu 0% bis 25% belegt
- je feiner die Granularität, desto größer die Space Map, desto genauer aber auch die Freispeicherverwaltung

# Histogramme und *Witnesses*

## Histogramme

- materialisiere Anzahl der Seiten für jeden Bucket
- Vorteil: erfolglose Suche ohne viel Aufwand

## Witnesses

- merke Dir, falls möglich, zu jedem Bucket eine Seitennr. als Repräsentanten (i.e., Witness)
- erlaube *unknown* Werte, wenn es zu einem Bucket keinen bekannten Witness gibt
- wenn es Witness gibt, dann verwende Witness; ansonsten normale Next-Fit Suche mit Histogrammen
- setze Witness z.B. beim Next-Fit Scan durch die Space Map
- Vorteil: Abkürzung zum Glück
- (ähnlich wie ein PID Cache)

# Zwischenfazit

## AO(n)

- sehr schnell
- speicherineffizient

## Next-Fit

- Next-Fit mit Histogrammen und Witnesses sehr gut (schnell und speichereffizient) in vielen Fällen
- sehr langsam allerdings, wenn es viele kleine Löcher gibt, weil es versucht alle Löcher zu stopfen

# Hybride Verfahren: HY(n,u)

## Prinzip

- verwende AO(n) solange Speicherauslastung besser als $u$
- (N.B. Histogramme erlauben jederzeit leichte Berechnung der Speicherauslastung)
- verwende Next-Fit sobald Speicherauslastung zu schlecht wird.

## Bewertung

- das beste aus allen Welten

# Alternative: Freispeicherliste

## Prinzip

- Liste mit Seitennr. von Seiten, die noch Platz haben
- Platz haben definiert als: freier Platz > Schwellwert
- hänge neue Seiten an Liste an
- hänge Seiten nach Löschungen an Liste an
- entferne Seite nach dem Erzeugen von neuen Objekten
- aus der Freispeicherliste wählt man mit First-Fit aus

## Bewertung

- Vorteil: man hält nicht Informationen für alle Seiten sondern nur für Seiten mit Platz
- Nachteil: Wahl des Schwellwertes ist kritisch; zu hoch, dann schlechte Speicherauslastung; zu niedrig, dann degeneriert Freispeicherliste zu einer Space Map
- Nachteil: Aufwendige Transaktionsverwaltung
- nur Oracle verwendet Freispeicherlisten