3 Einbringstrategien

Terminus: Page Materialization (in German: “Einbringen”)

Types of page materialization

- direct page assigment („Seitenzuordnung“)
- indirect page assignment

shadow page concept (Schattenspeicherkonzept)

twin-file concept (Konzept der Zusatzdatei)

*Einbringen = **Ablegen auf einem nicht-flüchtigen Speicher ("materialisieren")***

Grundlegende Idee

- Speichern muss nicht notwendigerweise identisch mit dem Einbringen einer Seite in den Datenbestand sein !
- Einbringen kann verzögert stattfinden

Zeitpunkt des Einbringens im Kontext einer Transaktion

- vor einem Commit ("steal" versus "no steal")
- nach einem Commit ("force" versus "no force")

*Eine Einbringstrategie legt fest, in welchen Block eine modifizierte Seite abgespeichert wird -> **Seitenzuordnung***

Unterscheidung in

- direkte Seitenzuordnung
- indirekte Seitenzuordnung

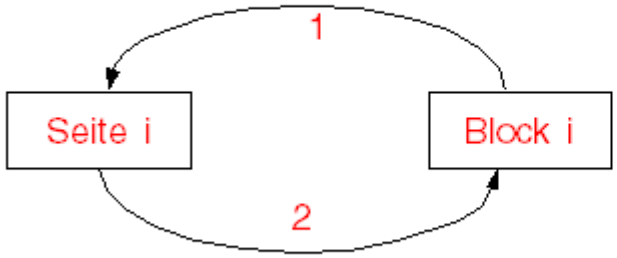
… vor COMMIT Zeitpunkt

- STEAL
  - geänderte Seiten können jederzeit, insbesondere vor EOT der ändernden Transaktion, ersetzt und in die Datenbank eingebracht werden
  - Flexibilität mit Blick auf Seitenersetzung
  - Transaktionsfehler können in schmutzigen Seiten münden → UNDO-Recovery
- NOSTEAL (¬ STEAL)
  - Seiten mit schmutzigen Änderungen dürfen nicht ersetzt werden
  - keine UNDO-Recovery auf der materialisierten DB vorzusehen
  - Probleme bei langen Änderungstransaktionen

… nach COMMIT Zeitpunkt

- FORCE
  - alle geänderten Seiten werden spätestens im Zuge der Commit-Behandlung in die materialisierte Datenbank eingebracht
  - Hoher punktueller Schreibaufwand aber keine REDO-Recovery notwendig
- NOFORCE (¬ FORCE)
  - Geänderte Seiten dürfen über den Commit-Zeitpunkt im Hauptspeicher verbleiben
  - Nachziehen festgeschriebener Änderungen notwendig → REDO-Recovery

Direkte Seitenzuordnung



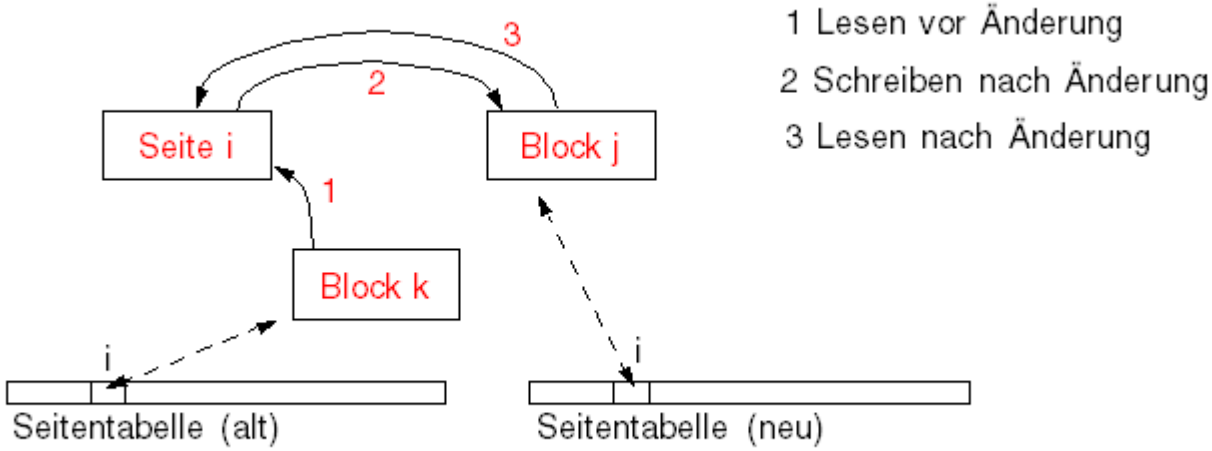
1 Lesen vor Änderung

2 Schreiben nach Änderung

- Beim Ausschreiben aus dem Hauptspeicher **ersetzt** eine Seite genau den Block, von dem sie beim Einlagern in den Puffer gelesen wurde (“Update-in-place”).
- **Vorteil:** Einfachheit; immer nur ein Block pro Seite.
- **Nachteil:** Keine Unterstützung der Recovery.
  Der alte Zustand der Seite muss als Log-Information **vor** dem Einbringen der geänderten Seite auf einen sicheren Speicher geschrieben werden (“**W**rite-**A**head-**L**og,” WAL-Prinzip).

Indirekte Seitenzuordnung

- Beim Ausschreiben aus dem Hauptspeicher wird eine Seite in einen freien Block geschrieben. Der ursprüngliche Block bleibt unverändert.
- Auch nach einem Hauptspeicherfehler ist eine konsistente Datenbank in den alten Blöcken erhalten!



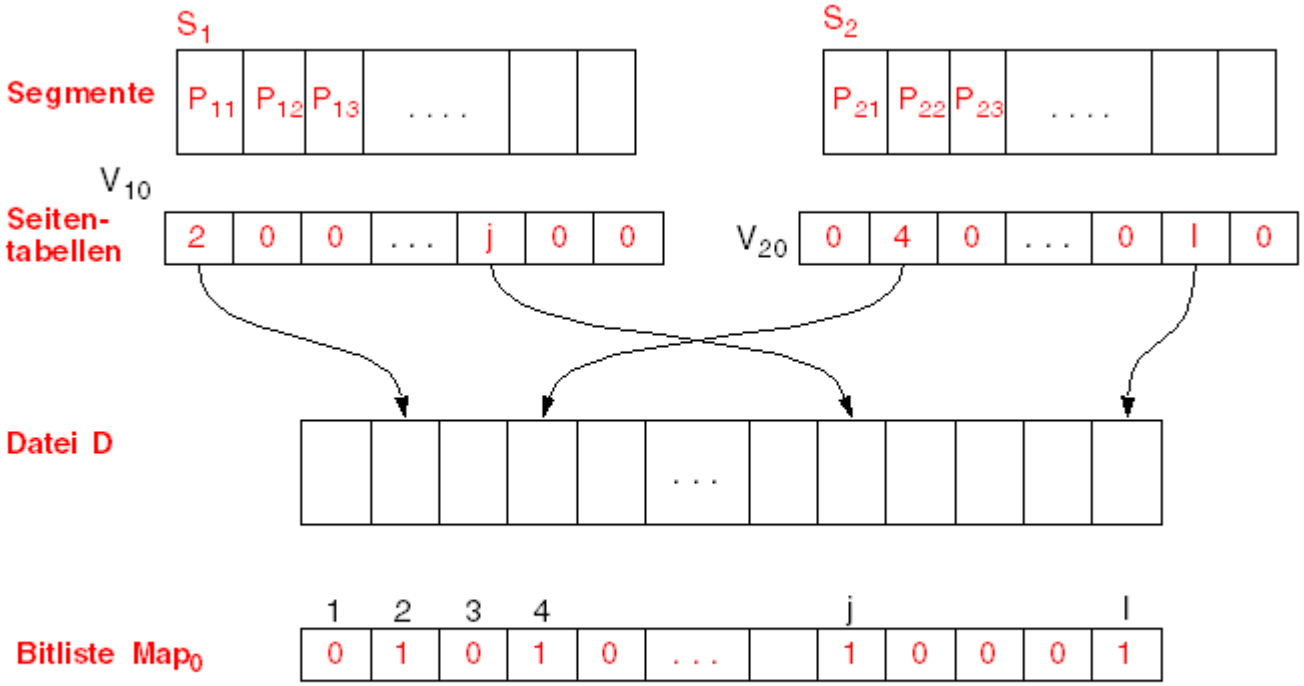
Beispiel einer indirekten Seitenzuordnung

- Verwendung: fehlertolerante Systeme
- Realisiert im System R, weiterentwickelt im TOSP (Transaktions-orientiertes Schattenspeicherkonzept)

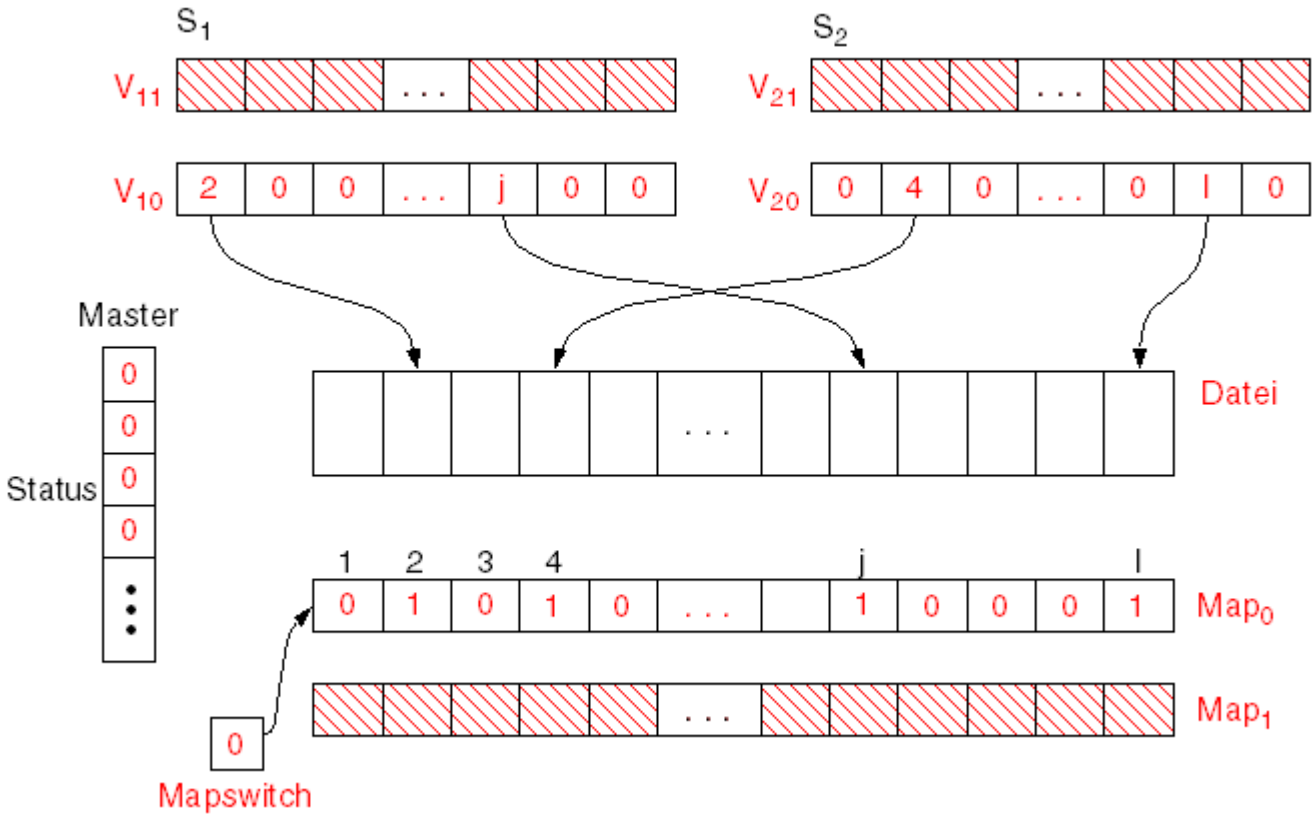
Grundidee

- Inhalte aller Seiten eines Segments werden in einem Sicherungsintervall Dt (z.B. 5min) in einem speicherkonsistenten Zustand unverändert gehalten
- Sicherungspunkte sind segmentorientiert (nicht transaktionsorientiert)
- Ist ein Segment geschlossen, erfüllt sein Inhalt bestimmte, von höheren Schichten kontrollierte Konsistenzbedingungen
- Auf einer Blockmenge D können gleichzeitig mehrere Segmente für den Änderungsbetrieb geöffnet werden
- Wenn in einem Segment S_i mit den Seiten P_{ij} $(1 \leq j \leq s_i)$ insgesamt $h_i \leq s_i$ belegt sind, so gilt für den Speicherbedarf z: $h_i \leq z \leq 2h_i$

> Indirekte Seitenadressierung für Schattenspeicherkonzept



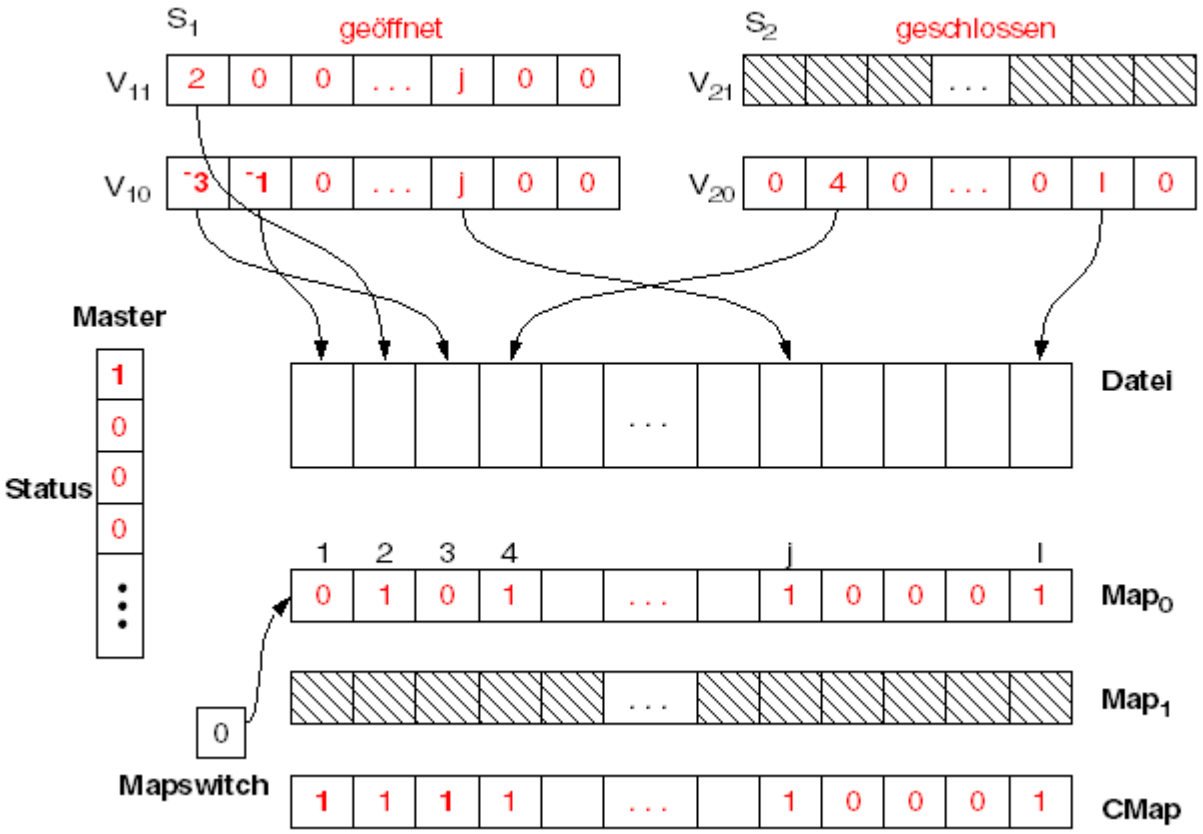
> Datenstrukturen im Schattenspeicherkonzept



Erläuterung

- schraffierte Strukturen
  - in der Ausgangssituation nicht benutzte Speicherbereiche
  - werden erst nach der Eröffnung für den Änderungsbetrieb erforderlich
- Status (i)
  - enthält den Eröffnungszustand für Segment i (zu Beginn: alle Segmente geschlossen)
- MAPSWITCH
  - zeigt an, welcher der beiden (gleichberechtigten) Freispeichertabellen Map_0 und Map_1 das aktuelle Verzeichnis belegter Blöcke enthält

> Änderungsbetrieb im Schattenspeicherkonzept



Eröffnung eines Segmentes k für Änderungen

- kopiere V_{k0} nach V_{k1} // Anlegen der Schattenseitenzuordnungstabelle
- STATUS(k) := 1
- Schreibe MASTER in einer ununterbrechbaren Operation aus
- Lege im Hauptspeicher eine Arbeitskopie CMAP von MAP_0 an

Erstmalige Änderung einer Seite P_i
(erstmalig seit Eröffnung des Segments)

- Lies Seite P_i aus Block j = $V_{k0}(i)$
- Finde einen freien Block j' in CMAP
- $V_{k0}(i)$ = j'
- Markiere Seite P_i in $V_{k0}(i)$ als geändert
- Bei weiteren Änderungen von P_i wird Block j' verwendet, d.h. einer Seite wird nur bei der erstmaligen Änderung nach Eröffnen des Segmentes ein neuer Block zugewiesen.

> Funktionsprinzip des Schattenspeicherkonzepts (2)

Beenden eines Änderungsintervalls

- Erzeuge die Bitliste mit der aktuellen Speicherbelegung in MAP_1 (neue Blöcke belegt, alte Blöcke freigegeben)
  - Merke:
    - Blöcke mit neuen/modifizierten Seiten können durch den Änderungsvermerk aufgefunden werden
    - Die Freigabe der alten Blöcke kann erst zu diesem Zeitpunkt erfolgen, da sonst die alten Blöcke vor dem Einbringen der neuen Version mit anderen Inhalten überschrieben werden könnten.
- Schreibe MAP_1 (kein Überschreiben von MAP_0)
- Schreibe V_{k0}
- Schreibe alle geänderten Blöcke
- STATUS(k) := 0, MAPSWITCH = 1 (MAP_1 ist aktuell)
- Schreibe MASTER in einer ununterbrechbaren Operation aus.

Merke

- Zum Zurücksetzen geöffneter Segmente muss lediglich V_{k1} in V_{k0} kopiert und STATUS(k) auf 0 gesetzt werden

Vorteile für Recovery

- Rücksetzen auf den letzten konsistenten Zustand ist einfach und billig
- Protokollieren des Zustandes vor einer Änderung (WAL-Protokoll) kann vermieden werden (flexibleres Schreiben der Log-Files)
- logisches Protokollieren ist möglich, da stets operationskonsistenter Zustand verfügbar ist

Nachteile

- Hilfsstrukturen (Seitentabellen V und Bittabellen M) werden so groß, dass sie in Blöcke zerlegt und durch einen speziellen Ersetzungsalgorithmus in einem eigenen Puffer verwaltet werden müssen
  (Seitentabellen belegen etwa 0.1-0.2% der gesamten Datenbank)
- zusätzlicher Speicherplatz für die Doppelbelegung
- physische Cluster-Bildung logisch zusammengehöriger Blöcken gehen verloren
  - alternativ: Einteilung der Datei in physische Cluster der Größe p (typischerweise Zylinder einer Festplatte) und der Segmente in logische Cluster der Größe l.
  - Seiten werden nun zumindest innerhalb des Clusters auf dem Medium abgelegt.

(weiteres Beispiel einer indirekten Seitenzuordnung)

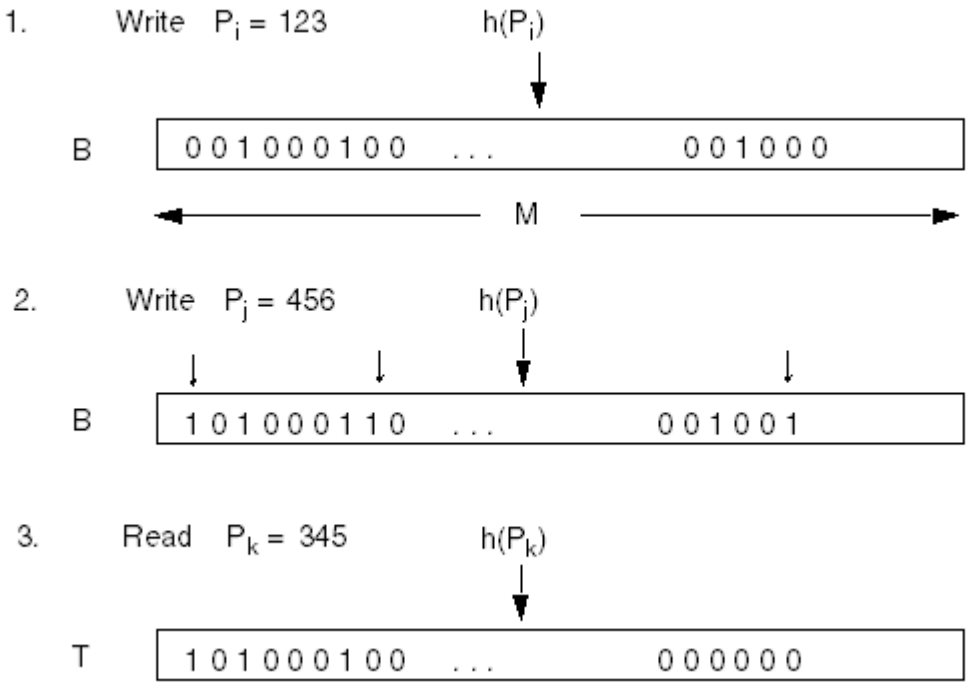
Grundidee

- bei Öffnung eines Änderungsbetriebs wird eine zusätzliche temporäre Datei angelegt
- alle modifizierten Seiten werden auf Blöcke dieser Zusatzdatei abgebildet, die Originaldatei wird nicht verändert
- Änderungen werden am Ende eines Änderungsintervalls in die Orginaldatei verzögert eingebracht

Problem

- wie kann entschieden werden, ob die aktuelle Version einer Seite in der Zusatz- oder Originaldatei steht?
- Bitliste zeigt an, ob eine Seite in dem aktuellen Änderungsintervall bereits verändert wurde und sich somit die gültige Version in der Zusatzdatei befindet
  - Bloom-Filter
    verlässliche Aussagen darüber, dass eine Seite nicht verändert wurde

- Einschränkung des Hauptspeicherbedarfs für die Bitliste B der Länge M
- notwendig: Hashfunktion h() über Seitennummern S_i



Beim Schreiben einer Seite

- Bitstring T der Länge M ist Ergebnis der Hashfunktion h() angewandt auf die Seitennummer S_i.
- Der Bitstring B wird ersetzt durch (B OR T)

Beim Lesen einer Seite

- T := h(S_k)
- Falls **(B AND T) != T**
  ist, dann ist die Seite nicht verändert worden und befindet sich in der Originaldatei
- Anderfalls, d.h. falls (B AND T) == T
  ist, dann ist die Seite **VIELLEICHT** geändert worden.

Zweistufige Abbildung

- von Segment/Seite auf Datei/Block erlaubt Einführung von Abbildungsredundanz durch verzögertes Einbringen

Verzögerte Einbringstrategien

- sind teurer als direkte, besitzen jedoch implizite Fehlertoleranz
- geringe Kosten für Protokollieren und Recovery
- sie belasten den Normalbetrieb zugunsten der Recovery (Verwaltung der Seitentabellen)
- schwierige Implementierung

Direkte Einbringstrategie (update-in-place)

- einfach zu implementieren
- keine zusätzlichen Kosten zur Ausführungszeit für die Seitenzuordnung
- Fehlertoleranz nur durch explizite Logging- u. Recovery-Funktionen