Prof. Dr. Markus Schneider
Prof. Dr. Ralf Hartmut Güting
Prof. Dr. Uta Störl

Modul 63122 Architektur und Implementierung von Datenbanksystemen

LESEPROBE

Fakultät für
Mathematik und Informatik

FernUniversität in Hagen

Vorwort

Liebe Fernstudent:innen,

wir begrüßen Sie herzlich zum Kurs 01614 „Architektur und Implementierung von Datenbanksystemen“ und hoffen, dass Sie die Kurseinheiten motiviert und mit Erfolg bearbeiten.

Datenbanksysteme sind in der heutigen Zeit als Systeme zur effizienten Speicherung und Verwaltung von Daten nicht mehr wegzudenken und in allen Anwendungsbereichen vom Finanzwesen über Web-Anwendungen, Anwendungssystemen in der Automobilindustrie, dem Maschinenbau und in der Medizin bis zu Geo-Informationssystemen vertreten.

Dabei ist es sehr wichtig, dass Datenbanksysteme auch für sehr große Datenbestände und hohe Transaktionslasten effizient funktionieren. Dies sicherzustellen, ist Aufgabe von Datenbankentwickler:innen, Datenbankadminstrator:innen und Software-Architekt:innen. Dafür ist ein tiefes Verständnis des internen Aufbaus eines Datenbanksystems Voraussetzung. Die grundlegenden Prinzipien der Architektur und Implementierung eines Datenbanksystems gelten dabei relativ unabhängig vom gewählten Datenbankmodell – also sowohl für relationale Datenbanksysteme als auch nicht-relationale Datenbanksysteme wie NoSQL-Datenbanksysteme oder Geo-Informationssysteme.

Das Lernziel dieses Kurses ist es deshalb, ein grundlegendes Verständnis für die Architektur und Implementierung von Datenbanksystemen zu entwickeln.

Gliederung des Kurses

Der Kurs ist in neun Kapitel unterteilt:

Kapitel 1 beschäftigt sich mit der Architektur von Datenbanksystemen und den Anforderungen an ihre Implementierung. Es werden verschiedene Modelle und Systemarchitekturen für Datenbanksysteme vorgestellt. Diese bilden die Grundlage für die Beschreibung der einzelnen Komponenten in den nachfolgenden Kapiteln.

Kapitel 2 behandelt das Speichersystem eines Datenbanksystems. Das Speichersystem untergliedert sich in die beiden Bereiche der Externspeicherverwaltung und der Systempufferverwaltung. Diese beiden Komponenten sind verantwortlich für das effiziente Lesen und Schreiben der Daten von externen Speichermedien in den Hauptspeicher und wieder zurück.

Kapitel 3 führt Indexstrukturen als spezialisierte Datenstrukturen zum effizienten Zugriff auf die Daten einer Datenbank ein. Dabei werden nach einer Klassifikation die wichtigsten Vertreter für Indexstrukturen vorgestellt.

Kapitel 4 befasst sich mit dem effizienten externen Sortieren großer Datenbestände auf Sekundärspeichern. Sortieren ist eine wichtige Funktion eines Datenbanksystems und wird beispielsweise zur Beantwortung von Benutzeranfragen in einer gewünschten Sortierreihenfolge, zur Eliminierung von Duplikaten in einer Menge von Datensätzen oder zur Unterstützung der Implementierung bestimmter relationaler Algebraoperationen benötigt.

Kapitel 5 und 6 behandeln die Verarbeitung von Anfragen. Die Möglichkeit, mit Hilfe einer Anfragesprache Anfragen an eine Datenbank zu stellen, sei es durch ein Anwendungsprogramm oder ad hoc durch den Endbenutzer am Computer, gehört zu den herausragenden Eigenschaften eines Datenbanksystems. Ziel der Anfrageverarbeitung ist es, eine gegebene Anfrage unter Ausnutzung logischer Gesetzmäßigkeiten und externer physischer Speicherungsstrukturen möglichst effizient auszuführen.

Kapitel 7 hat Transaktionen und Concurrency Control zum Thema. Aufgabe des Concurrency Control ist es, den nebenläufigen Zugriff auf geteilte Daten von vielen verschiedenen Anwendungsprogrammen und interaktiven Benutzern zu steuern und die beteiligten Prozesse zu

synchronisieren, um inkonsistente Datenbankzustände zu verhindern. Das grundlegende Konzept für Concurrency Control (und Recovery) ist dabei die Transaktion.

Kapitel 8 befasst sich mit dem Thema Recovery, also mit denjenigen Funktionen eines Datenbanksystems, welche die Wiederherstellung eines korrekten Datenbankzustands nach dem Auftreten eines Datenbankfehlers ermöglichen und die Atomarität (Unteilbarkeit) und Dauerhaftigkeit von Transaktionen sicherstellen.

Nachdem die Kapitel 1 bis 8 den grundlegenden Aufbau eines Datenbanksystems beschrieben haben und dabei primär von nicht-verteilten Architekturen ausgegangen wurde, widmet sich **Kapitel 9** verteilten Datenbankarchitekturen. Diese Architekturen haben in den letzten Jahren aufgrund der riesigen zu verarbeitenden Datenmengen (Big Data) in Internet-Anwendungen stark an Bedeutung gewonnen und viele Techniken wurden hierfür weiter- oder neu entwickelt. In diesem Kapitel werden zum einen spezifische Aspekte der verteilten Datenhaltung wie Verteilung und Replikation behandelt. Zum anderen werden verschiedene Aspekte aus den Kapiteln 1 bis 8 wie Anfrageverarbeitung, Transaktionsverarbeitung und Konsistenz unter dem Blickwinkel der Auswirkungen der verteilten Datenbankarchitektur betrachtet.

Die nachfolgende Tabelle zeigt die Zuordnung der neun Kapitel zu den sieben Kurseinheiten:

Kurseinheit	Kapitel	Inhalt
1	1, 2	Architektur eines Datenbanksystems, Externspeicherverwaltung
2	2, 3	Systempufferverwaltung, Indexstrukturen
3	3, 4, 5	Indexstrukturen, Externes Sortieren, Auswertung relationaler Pläne
4	6	Anfrageoptimierung: Berechnung eines effizienten Plans
5	7	Transaktionen und Concurrency Control
6	8	Recovery
7	9	Verteilte Datenbankarchitekturen

Voraussetzungen

Dieser Kurs setzt grundlegende Kenntnisse im Bereich Datenbanken, wie sie beispielsweise im Kurs 01671 vermittelt werden, voraus.

Übungen

Zur Vertiefung des Verständnisses der in diesem Kurs vorgestellten Inhalte empfehlen wir die Bearbeitung der Einsende- und Selbsttestaufgaben. Weitere Informationen zum Ablauf der Übungen etc. finden Sie in einem gesonderten Anschreiben.

Prüfung

Dieses Modul wird in Form einer mündlichen Prüfung geprüft. Alle Kapitel sind prüfungsrelevant.

Literatur

Auf relevante und hilfreiche Literatur wird in den Literaturhinweisen am Ende jedes Kapitels hingewiesen.

Die Autoren

Prof. Dr. Markus Schneider Markus Schneider ist seit 2014 als Professor am Department of Computer and Information Science and Engineering (CISE) der University of Florida, USA tätig. Er hat an der Universität Dortmund studiert und an der FernUniversität in Hagen promoviert. Sein Forschungsschwerpunkt sind neue Datenbanktechnologien, insbesondere erweiterbare und spatial-temporale Datenbanksysteme. Während seiner Tätigkeit an der FernUniversität hat er die Kapitel 1–6 des vorliegenden Kurses erstellt.

Prof. Dr. Ralf Hartmut Güting Ralf Harmut Güting war von 1989 bis 2021 als Professor an der FernUniversität in Hagen tätig und leitete dort das Lehrgebiet „Datenbanksysteme für neue Anwendungen“. Er hat an der Universität Dortmund studiert und promoviert. Nach einem Forschungsaufenthalt im IBM Almaden Research Center wurden erweiterbare und spatial Datenbanksysteme sein Forschungsschwerpunkt. In seinem Lehrgebiet an der FernUniversität in Hagen wurden Prototpyen erweiterbarer und spatio-temporaler Datenbanksysteme, das Gral- und das SECONDO-System entwickelt. Die Kapitel 3–6 wurden von ihm überarbeitet.

Prof. Dr. Uta Störl Uta Störl ist seit April 2021 an der FernUniversität in Hagen tätig und leitet das Lehrgebiet „Datenbanken und Informationssysteme“. Sie hat an der Friedrich-Schiller-Universität in Jena studiert und promoviert. Danach war sie mehrere Jahre bei der Dresdner Bank in Frankfurt am Main tätig. Von 2005 bis 2021 war sie Professorin für Datenbanken an der Hochschule Darmstadt und hat dort das Big Data Competence Center aufgebaut und geleitet. Ihr Forschungsschwerpunkt sind Big-Data-Technologien, insbesondere NoSQL-Datenbanksysteme und Data Engineering für Data Science. Die Kapitel 7–9 des vorliegenden Kurs wurden von ihr erstellt.

2.9.4 Indirekte Einbringstrategien für Änderungen

Die im letzten Abschnitt behandelten Verfahren zur Abbildung von Segmenten in Dateien und von Seiten auf Blöcke gehen davon aus, dass eine geänderte Seite in den ihr einmal zugeordneten Block zurückgeschrieben wird (*update in place*). Tritt innerhalb einer Transaktion ein Fehler auf, so muss der Recovery-Manager, bedingt durch das direkte Einbringen von Änderungen, zur Wiederherstellung des alten Zustands der Seite genügend Log-Informationen (*Undo*-Information) bereitstellen, die vor dem Zurückschreiben einer Seite in einem sicheren Speicherplatzbereich stehen müssen. Da das Schreiben großer Mengen von Log-Daten aber mit beträchtlichem Aufwand verbunden ist, ist es häufig vorteilhaft, das Einbringen von Änderungen in einer Seite so durchzuführen, dass ihr alter Zustand bis zum Ende der Transaktion verfügbar bleibt. In diesem Fall ist die Verwaltung von Undo-Informationen in der Regel nicht erforderlich. Im Folgenden werden Konzepte vorgestellt, die die im letzten Abschnitt behandelten Verfahren derart modifizieren, dass durch das indirekte Einbringen von Änderungen eine erhebliche Unterstützung der Recovery-Verfahren erreicht wird. Dabei wird der durch diese Indirektion erforderliche Mehraufwand durch den geringeren Aufwand für Recovery-Aufgaben und durch Effizienzgewinn bei Weitem aufgewogen.

Twin Slot-Verfahren

Das *Twin Slot-Verfahren* kann als Modifikation der direkten Seitenadressierung aufgefasst werden. Es ist insbesondere für den Einsatz auf Festplatten konzipiert, verursacht sehr geringe Kosten für Recovery-Maßnahmen und kompensiert teilweise diesen Vorteil jedoch durch doppelten Speicherplatzaufwand. Für eine Seite P_{ki} eines Segmentes S_k werden jeweils zwei aufeinanderfolgende Blöcke B_{jl} und $B_{j,l\text{-}1}$ einer Datei D_j mit $l = K_j - 1 + 2 \cdot i$ reserviert. Im Wechsel hält einer der beiden Blöcke zu Beginn einer Transaktion den aktuellen Zustand der Seite fest, während Änderungen in den anderen Block zurückgeschrieben werden. Die beiden einer Seite zugeordneten Blöcke enthalten also stets die beiden jüngsten, auf Transaktionen bezogenen Änderungszustände dieser Seite. Durch eine zusätzliche Kennung in der Seite (Transaktionskennung) ist es der Systempufferverwaltung möglich, diese beiden Zustände zu unterscheiden. Bei einer Seitenanforderung werden beide Blöcke gelesen, und der Block mit dem jüngeren Zustand wird im Systempuffer als aktuelle Seite zur Verfügung gestellt. Der Block mit dem älteren Zustand nimmt dann die geänderte Seite wieder auf. Um den Aufwand beim Lesen einer Seite gering zu halten, werden die zu einer Seite zugehörigen beiden Blöcke physisch benachbart abgespeichert (hierauf weist auch der Name des Verfahrens hin), so dass sie mit einem Zugriff gelesen werden können. Mittels Seitensperren kann mit dem Twin Slot-

Verfahren ohne explizite Verwaltung von Log-Daten ein transaktionsorientiertes Recovery-Konzept realisiert werden.

Schattenspeicher-Konzept

Das sogenannte *Schattenspeicher-Konzept* (Bild 2.9) stellt die Erweiterung der indirekten Seitenadressierung dar und unterstützt wie das Twin Slot-Verfahren ein indirektes Einbringen von Änderungen. Die Grundidee dieses Konzepts besteht darin, vor Beginn eines neuen, durch zwei *Sicherungspunkte* gegebenen Sicherungsintervalls den Inhalt aller gerade aktuellen Seiten eines Segments als sogenannte *Schattenseiten* zu duplizieren und somit unverändert und konsistent zu erhalten. Genauer bedeutet dies, dass bei Erzeugung eines neuen Sicherungspunktes alle Datenstrukturen, die zur Darstellung eines Segments S_k gehören (d. h., alle belegten Seiten, die Seitentabelle T_k, die Bittabelle M), als konsistente Momentaufnahme des Segments auf einem Externspeicher gesichert werden. Alle Änderungen während des Sicherungsintervalls werden auf Kopien T_k' und M' von T_k und M vollzogen. Geänderte Seiten werden nicht in ihre ursprünglichen, sondern in freie Blöcke zurückgeschrieben. Bei der Erzeugung des nächsten Sicherungspunktes, die als atomare Operation realisiert sein muss, werden die Tabellen T_k' und M' und alle zu diesem Zustand gehörenden und geänderten Seiten zurückgeschrieben. Ferner werden diejenigen Blöcke freigegeben, deren Seiten während des letzten Sicherungsintervalls einer Änderung unterworfen waren; dies betrifft also gerade diejenigen Schattenseiten, für die eine jüngere Version existiert. Zu Beginn des nächsten Sicherungsintervalls muss

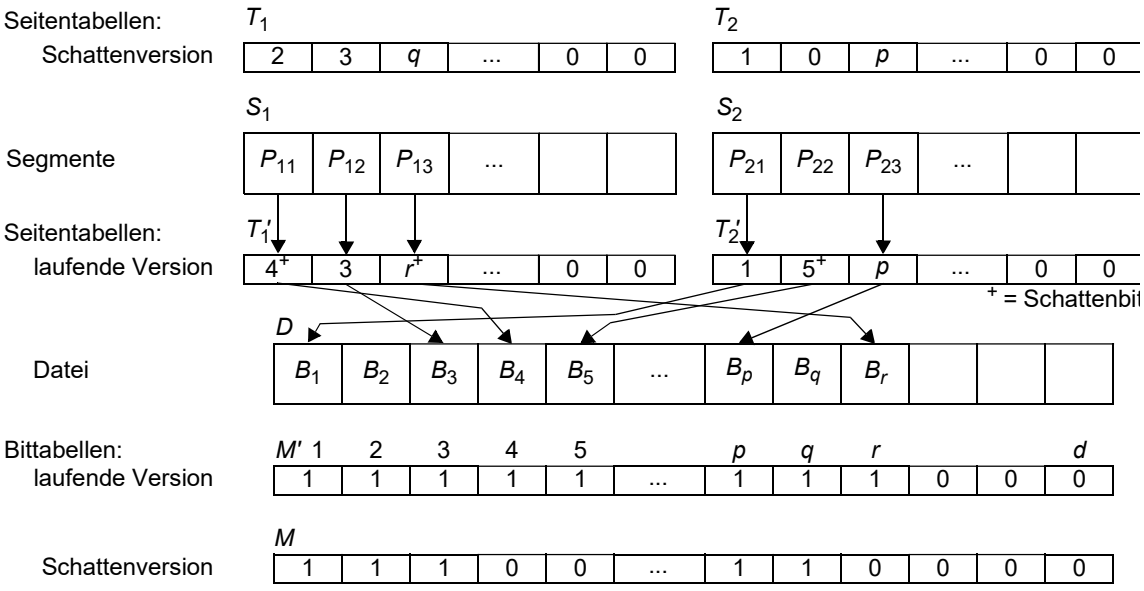


Bild 2.9: Schattenspeicher-Konzept (Segmente S_1 und S_2 in Bearbeitung)

der aktuelle Inhalt von T_k' und M' wieder nach T_k und M kopiert werden. Bei Auftreten eines Fehlers innerhalb eines Sicherungsintervalls kann sofort auf den vorherigen, durch T_k und M repräsentierten, konsistenten Zustand zugegriffen werden.

Bild 2.9 zeigt beispielhaft mehrere Seitenänderungen in den Segmenten S_1 und S_2, die durch sogenannte *Schattenbits* in den Seitentabellen markiert sind. Schattenbits werden bei der Erzeugung neuer Sicherungspunkte zur Freigabe der Schattenseiten verwendet. Besteht ein Segment aus s Seiten, so muss die zugehörige Datei weitere s Blöcke als Reserve vorsehen, da jede geänderte Seite während eines Sicherungsintervalls zwei Blöcke belegt. Außerdem lässt sich für Segmente keine physische Clusterung erreichen, da eine geänderte Seite auf einen neuen, möglicherweise physisch weiter entfernten Block abgebildet wird.

Die beim Schattenspeicherkonzept erzeugten Sicherungspunkte orientieren sich an Segmenten und richten sich nicht nach Transaktionsgrenzen aus. Daher wird im Fehlerfall eine *segmentorientierte Recovery* durchgeführt. Für eine *transaktionsorientierte Recovery* sind zusätzliche Log-Daten zu sammeln. Die Einzelheiten und genauen Zusammenhänge hierzu werden wir später bei der Besprechung der Recovery-Komponente erörtern.

Wie wir gesehen haben, haben indirekte Einbringstrategien gegenüber direkten Strategien den Vorteil, dass sie Recovery-Maßnahmen unmittelbar unterstützen. Wegen des doppelten Speicherplatzbedarfs lässt sich das Twin Slot-Verfahren aber nur in Sonderfällen verwenden, während sich das Schattenspeicherkonzept für ein breites Anwendungsspektrum eignet. Das Rücksetzen des Datenbanksystems auf einen konsistenten Zustand im letzten Sicherungspunkt ist sehr effizient, da es in das Verfahren integriert ist. Da nach einem Systemausfall beim Wiederanlauf mindestens eine speicherkonsistente Datenbank vorliegt, kann ein platzsparendes Übergangslogging benutzt werden, das nur die durchgeführten Aktionen seit dem letzten Sicherungspunkt beschreibt. Bei einem katastrophalen Fehler, bei dem die Log-Daten zerstört worden sind, ist es wahrscheinlicher, mit Hilfe der Schattenspeicherseiten einen brauchbaren Zustand der Datenbank wiederherzustellen als mittels direkt modifizierter, aber zum Zeitpunkt des Fehlers undefinierter Seiten.

Aber auch einige Nachteile fallen beim Schattenspeicherkonzept auf. Da die Zuordnung von Seiten zu Blöcken dynamisch und beliebig verteilt erfolgt, können logisch zusammengehörende Seiten nicht mehr zur Minimierung der Zugriffszeit geclustert werden, wodurch die Suchzeit bei sequentiellem Zugriff etwas zunimmt. Die Clustereigenschaft von Datensätzen innerhalb einer Seite bleibt aber erhalten. Werden die Datenbanken größer, wachsen auch die Hilfsdatenstrukturen, so dass sie nicht mehr in den Hauptspeicher passen und in Blöcke unterteilt werden müssen, die in einem speziellen Puffer mittels eines geeigneten Ersetzungsalgorithmus zu verwalten sind. Ferner erfordert der Übergang

von einem zum nächsten Sicherungsintervall an einem Sicherungspunkt das Herausschreiben aller geänderten Seiten und der Hilfsstrukturen aus den Puffern. Dies benötigt einen beträchtlichen Anteil an CPU-Zeit und Ein-/Ausgabe-Aktivität, so dass als Folge ungewöhnlich lange Antwortzeiten auftreten. Nicht zuletzt muss durch die Doppeltbelegung von geänderten Seiten zwischen zwei Sicherungspunkten zusätzlicher Speicherplatz aufgewendet werden. Bei Untersuchungen mit eingesetzten Datenbanksystemen wurde ermittelt, dass bei großen Datenbanken (≥ 100 MB) ein Speichersystem, das auf dem Schattenspeicherkonzept beruht, weniger leistungsfähig ist als eines, das auf einer direkten Einbringstrategie und dem Erzeugen von Log-Informationen beruht. Die Stärken des Schattenspeicherkonzepts liegen bei kleineren Datenbanken (≤ 10 MB).

2.9.5 Verwaltung des Systempuffers

Der *Systempuffer* dient in einem Datenbanksystem zur Ausführung von Lese- und Schreibvorgängen aller parallelen Transaktionen. Er ist aus *n* zusammenhängenden Rahmen im Hauptspeicher aufgebaut, so dass also zu jedem Zeitpunkt bis zu *n* Datenbankseiten zeitweilig zwischengespeichert werden können. Da wesentlich mehr Datenbankseiten als Systempufferseiten existieren, entsteht um die Systempufferseiten eine Konkurrenzsituation; im Systempuffer befindliche Seiten müssen ersetzt und freigewordene Pufferrahmen neuen Seiten zugewiesen werden. Für die Performance eines Datenbanksystems sind also die Verfügbarkeit eines hinreichend großen Systempuffers (die Größe liegt bei existierenden Systemen etwa zwischen 20 KB und 20 MB) sowie seine Verwaltung durch geeignete Such- und Ersetzungsalgorithmen von ausschlaggebender Bedeutung. Vordringliches Ziel der Systempufferverwaltung ist eine möglichst große Reduzierung der physischen Ein-/Ausgabe-Vorgänge, d. h., häufig benutzte Datenbankseiten müssen im Systempuffer gehalten werden.

Allgemeine Arbeitsweise

Die Systempufferverwaltung stellt Komponenten höherer Systemschichten seitenstrukturierte Segmente als Adressräume zur Verfügung. Diese Komponenten sind sich also der Seitengrenzen bewusst und ermitteln zum Zugriff auf die gewünschten Objekte beispielsweise durch Nachschauen im Katalog oder durch Verwendung von Daten über Zugriffspfade die entsprechenden Seitennummern und fordern die zugehörigen Seiten an. Zunächst wird geprüft, ob die angeforderte Seite sich nicht schon im Systempuffer befindet. Ist dies nicht der Fall, wird ein Rahmen gemäß der Ersetzungsstrategie des Systempuffer-Managers ausgewählt, dessen Seite ersetzt wird. Wurde die im ausgewählten Rahmen enthaltene Seite geändert, so wird sie zunächst auf den Externspeicher zurückge-

schrieben, bevor die angeforderte Seite in den ausgewählten Rahmen eingelesen wird. Ansonsten kann sie einfach von der neuen Seite überschrieben werden. Es sind also gegebenenfalls zwei physische Seitenzugriffe erforderlich. Soll die einzulagernde Seite nicht nur gelesen, sondern auch geändert werden, erhält sie einen Änderungsvermerk. Zuguterletzt wird die Seite implizit fixiert, und ihre Adresse wird an die aufrufende Komponente übergeben. Die Fixierung der Seite stellt sicher, dass die Seite nicht vom Systempuffer-Manager aus dem Systempuffer ausgelagert wird und dass die Seite für die Dauer ihrer Bearbeitung im zugewiesenen Rahmen bleibt, so dass Hauptspeicheroperationen mittels direkter Adressierung auf ihr ausgeführt werden können. Nach Beendigung der Bearbeitung liegt es in der Verantwortung der anfordernden Komponente, die Seite explizit durch den Systempuffer-Manager freigeben zu lassen, so dass sie nun wieder zur Ersetzung ausgewählt werden kann.

Wenn die gleiche Seite von mehreren Transaktionen angefordert wird, stellt das Sperrprotokoll des Transaktions-Managers sicher, dass jede Transaktion eine teilweise oder exklusive Sperre erhält, bevor eine Seite zum Lesen oder Ändern bereitgestellt wird. Wenn eine neue Seite angefordert wird und kein unbesetzter Rahmen zur Verfügung steht, sind die unfixierten Seiten erste Wahl für eine Ersetzung. Es ist dann die Aufgabe der Ersetzungsstrategie des Systempuffer-Managers, einen geeigneten Rahmen zur Ersetzung auszuwählen. Verschiedene, mögliche Ersetzungsstrategien, wie wir sie auch von Betriebssystemen her kennen, werden weiter unten besprochen. Wird ein Rahmen zur Ersetzung ausgewählt und ist die darin befindliche Seite geändert worden, so muss die ältere Seitenversion auf dem Externspeicher durch sie überschrieben werden. Die angeforderte Seite wird dann in den Pufferrahmen gelesen. Gibt es keine unfixierte Seite im Systempuffer und befindet sich die angeforderte Seite nicht im Systempuffer, so ist der Systempuffer-Manager gezwungen zu warten, bis irgendeine Seite unfixiert ist. Erst dann kann die Seitenanforderung erfüllt werden. Transaktionen sollten also so schnell wie möglich die Fixierung von Seiten aufheben.

Auffinden einer Seite

Bei Anforderung einer Seite hat der Systempuffer-Manager zunächst festzustellen, ob sich die angeforderte Seite bereits im Systempuffer befindet. Da diese Situation sehr häufig eintritt, ist eine effiziente Suchstrategie von Nutzen. Unterschieden werden hierzu direkte Suchen in Pufferrahmen und indirekte Suchen unter Verwendung von Hilfsstrukturen wie Zuordnungstabellen, unsortierte Tabellen, sortierte Tabellen, verkettete Tabellen und Hash-Tabellen. Die direkte Suche erfolgt sequentiell über alle Rahmen des Puffers. Im Erfolgsfall sind durchschnittlich die Hälfte, bei Misserfolg alle Pufferrahmen zu durchsuchen. Dieser Aufwand ist insbesondere bei großen Puffern nicht zu unterschätzen.

Hilfsstrukturen können verwendet werden, wenn die Verwaltungsinformationen nicht innerhalb sondern getrennt von den Seiten abgespeichert sind. Eine Zuordnungstabelle, die für jede Seite der Datenbank anzeigt, ob und in welchem Rahmen sich die Seite im Puffer befindet, ist nur für kleine Datenbanken praktikabel, da d Einträge (d = DB-Größe in Seiten) zu verwalten sind. Die anderen Tabellenarten benötigen nur n Einträge für einen Puffer der Größe n. Die unsortierte Tabelle erfordert im Durchschnitt $n/2$ Zugriffe bei erfolgreicher Suche; die sortierte Tabelle benötigt mittels binärer Suche $\log_2 n$ Zugriffe. Sortierte Tabellen erfordern aber einen hohen Wartungsaufwand, da beim Einfügen Einträge verschoben werden müssen. Tabellen mit verketteten Einträgen können zur Darstellung von bestimmten Seitenreihenfolgen, auch *Seitenreferenzfolgen* genannt, verwendet werden, die für Ersetzungsstrategien (z. B. LRU) von Bedeutung sein können. Änderungen sind weniger aufwendig als bei sortierten Tabellen, da keine Einträge verschoben werden müssen. Mit Hash-Verfahren lassen sich effiziente Suchen realisieren. Eine Hash-Funktion ordnet jeder vorhandenen Seitennummer einen Eintrag in der Hash-Tabelle zu, der Seitennummer und Pufferrahmenadresse enthält. Dem Laden/Auslagern einer Seite entspricht das Einfügen/Löschen eines entsprechenden Eintrages in der zugehörigen Hash-Kategorie. Alle Seitennummern, die der gleichen Hash-Kategorie angehören, werden in einer Überlaufkette miteinander verknüpft. Nach erfolglosem Suchen in der Überlaufkette stellt sich dann heraus, dass sich die gesuchte Seite nicht im Systempuffer befindet.

Speicherzuteilung im Systempuffer

Der *Systempuffer* wird als globale Datenstruktur von allen Transaktionen des Datenbanksystems benutzt. Parallele Transaktionen konkurrieren daher in ihrem Bestreben, möglichst viele ihrer Seiten in die Rahmen des Systempuffers einlagern zu lassen. Die *Speicherzuteilungsstrategie* hat die Aufgabe, jeder Transaktion eine Menge von Rahmen zur Aufnahme eines Teils ihrer Seiten zuzuordnen. Das Speicherzuteilungsproblem ist ähnlich dem in Betriebssystemen. In beiden Fällen handelt es sich um die Verwaltung einer beschränkten Anzahl von Rahmen für den Seitenzugriff mehrerer Transaktionen, wobei die Anzahl der Ein-/Ausgabevorgänge minimiert werden soll, ohne dass die Kosten für eine Transaktion eine vorgegebene Schranke überschreiten. Neben grundsätzlichen Konzepten aus dem Betriebssystembereich zur Optimierung der Speicherzuteilung sind jedoch bei Datenbanksystemen eine Reihe von wesentlichen Unterschieden zu beachten.

1. Infolge der gemeinsamen Benutzung von Datenbankseiten kann ein Zugriff auf dieselbe Seite im Puffer durch mehrere Transaktionen erfolgen.

2. Die Zugriffe einzelner Transaktionen sind überwiegend sequentiell. Die Lokalität von Datenbankzugriffen entsteht daher nicht durch das Zugriffsverhalten einzelner Transaktionen, sondern vielmehr durch die gemeinsame Benutzung von Seiten durch verschiedene parallele Transaktionen.

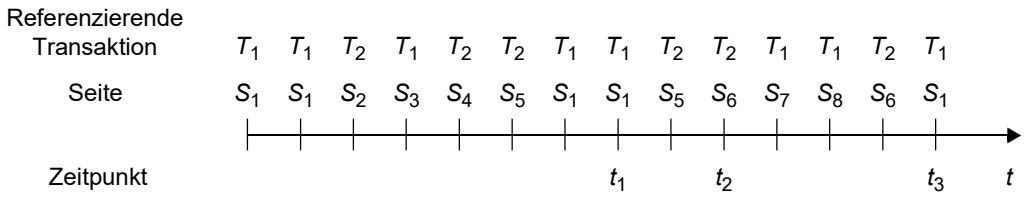
3. Da die Zugriffe auf die Datenseiten von den eingesetzten Zugriffspfaden und Speicherungsstrukturen abhängen, lässt sich eine Vorhersage der Zugriffswahrscheinlichkeit machen. So weisen Seiten mit Verwaltungs- und Zugriffspfadinformationen eine wesentlich höhere Wiederbenutzungswahrscheinlichkeit als normale Datenseiten auf.

Speicherzuteilungsstrategien lassen sich in lokale, globale und seitentypbezogene Strategien unterscheiden. *Lokale Speicherzuteilungsstrategien* ordnen jeder Transaktion ohne Einbeziehung des Verhaltens paralleler Transaktionen eine Menge reservierter Rahmen im Systempuffer zu. Von mehreren Transaktionen gemeinsam benutzte Seiten müssen gesondert verwaltet werden (Eigenschaft 1). Lokale Strategien lassen sich weiter unterteilen in *dynamische Strategien*, d. h., in solche, die abhängig vom aktuellen Bedarf einer Transaktion dynamisch Speicherplatz zuteilen, und in *statische Strategien*, d. h., in solche, bei denen der einmal zugeteilte Speicherplatz für die Dauer der Transaktion unverändert bleibt. *Globale Speicherzuteilungsstrategien* teilen die verfügbaren Systempufferrahmen auf alle parallelen Transaktionen unter Einbeziehung ihres gesamten Seitenreferenzverhaltens auf. Zuteilungs- und Ersetzungsentscheidungen werden global getroffen. Dadurch werden insbesondere die Eigenschaften 1 und 2 unterstützt. *Seitentypbezogene Speicherzuteilungsstrategien*, die insbesondere die Eigenschaften 2 und 3 unterstützen, partitionieren den Systempuffer nach bestimmten Seitentypen wie Datenseiten, Zugriffspfadseiten, Systemseiten, usw. Auch hier lassen sich statische Strategien mit unveränderlicher Partitionierung und dynamische Strategien unterscheiden. Bei letzteren variieren die Partitionsgrößen abhängig vom aktuellen Bedarf an Seiten des jeweiligen Typs. Zuteilungs- und Ersetzungsentscheidungen sind insofern global, als dass sie sich auf eine bestimmte Partition beziehen.

Besonders bei den statischen Strategien ist die Aktivierung von Transaktionen einfach. Sobald die erforderliche Rahmenmenge im Systempuffer verfügbar ist, kann eine neue Transaktion gestartet werden. Die Zuteilung feststehender Partitionen erweist sich aber bei stark variierendem Bedarf an Seitenrahmen als sehr ineffizient und unflexibel, da keine transaktionsübergreifende Anpassung möglich ist. Diese Strategien sind daher als wenig geeignet anzusehen. Die neben den globalen verbleibenden dynamischen, lokalen und seitentypbezogenen Strategien hingegen ermöglichen eine optimale Speicherausnutzung und sind insbesondere auch in ihrem Zusammenspiel mit Seitenersetzungsstrategien von Interesse. Als einziges Problem bei der Speicherplatzzuteilung erweist sich die

Bestimmung der dynamischen Partitionen. Ziel muss es sein, die Partition jeder Transaktion bzw. jedes Seitentyps entsprechend dem tatsächlichen Rahmenbedarf dynamisch wachsen und schrumpfen zu lassen. Bei einer lokalen Strategie wird dabei allerdings nur das aktuelle Referenzverhalten der gerade betrachteten Transaktion berücksichtigt. Um eine dynamische Partitionierung zu realisieren, gibt eine Transaktion, sobald sie in einem bestimmten Zeitabschnitt mit weniger Rahmen auskommt, den „Überschuss" an Rahmen ab. Andererseits werden ihr bei zusätzlichem Bedarf weitere Rahmen zugewiesen. Bei globalen Strategien steht der gesamte Systempuffer allen aktiven Transaktionen zur Verfügung. Eine Schätzung des Rahmenbedarfs einer Transaktion erfolgt nicht allein in Abhängigkeit von ihrem eigenen Referenzverhalten, sondern hängt insbesondere vom Referenzverhalten der parallel ablaufenden Transaktionen ab. Ähnliches gilt für seitentypbezogene Strategien. Die Rahmenzuteilung für aktivierte Transaktionen wird bei globalen Strategien der Verantwortlichkeit der gewählten Ersetzungsstrategie überlassen.

Ein bekanntes dynamisches Speicherzuteilungsverfahren ist die *Working-Set-Strategie*, die auf dem sogenannten *Working-Set-Modell* basiert. Anhand dieses Modells lässt sich Lokalität im Referenzverhalten von Transaktionen beschreiben. Der *Working-Set* $WS_{T_i}(t, \tau)$ einer Transaktion T_i ist die Menge derjenigen Seiten, die zum Zeitpunkt t von der betrachteten Transaktion innerhalb ihrer letzten τ Seitenzugriffe angesprochen worden sind. τ wird als *Fenstergröße* bezeichnet; $ws_{T_i}(t, \tau) = | WS_{T_i}(t, \tau) |$ heißt *Working-Set-Größe*. Diese Größe kann als Maß für die Lokalität des Referenzverhaltens einer Transaktion T_i dienen. Je kleiner $ws_{T_i}(t, \tau)$ bei festem τ ist, desto häufiger benötigte T_i erst kürzlich adressierte Seiten erneut. Um so höher war also die Lokalität ihres Referenzverhaltens. Beispiele für Working-Sets zeigt Bild 2.10.



$WS_{T_i}(t, \tau)$ mit $\tau = 5$

$WS_{T_1}(t_1, 5) = \{S_1, S_3\}$, $ws_{T_1}(t_1, 5) = 2$
$WS_{T_2}(t_1, 5) = \{S_2, S_4, S_5\}$, $ws_{T_2}(t_1, 5) = 3$
$WS_{T_1}(t_2, 5) = \{S_1, S_3\}$, $ws_{T_1}(t_2, 5) = 2$
$WS_{T_2}(t_2, 5) = \{S_2, S_4, S_5, S_6\}$, $ws_{T_2}(t_2, 5) = 4$
$WS_{T_1}(t_3, 5) = \{S_1, S_7, S_8\}$, $ws_{T_1}(t_3, 5) = 3$
$WS_{T_2}(t_3, 5) = \{S_4, S_5, S_6\}$, $ws_{T_2}(t_3, 5) = 3$

Bild 2.10: Beispiele für Working-Sets

Ziel der Working-Set-Strategie ist es, einer Transaktion T_i ihren Working-Set verfügbar zu halten. Hierbei wird davon ausgegangen, dass der Working-Set durch geschickte Wahl von τ gerade so groß ist, dass T_i ihre Seiten effizient bearbeiten kann und dass mit hoher Wahrscheinlichkeit gerade die Seiten zum Working-Set von T_i zum Zeitpunkt t gehören, die auch zum Zeitpunkt $t+\Delta t$ benötigt werden. Phasen hoher Lokalität bei konstanter Fenstergröße τ zeichnen sich durch eine Verkleinerung des Working-Sets einer Transaktion aus. Rahmen für Seiten, die den Working-Set verlassen, werden wieder für eine erneute Zuteilung freigegeben. Hierdurch wird versucht, für alle miteinander konkurrierenden Transaktionen eine optimale Speicherzuteilung zu erreichen. Die Working-Set-Strategie entscheidet also, welche Seite zu einem gegebenen Zeitpunkt ersetzbar und welche nicht ersetzbar ist. Sie beinhaltet aber keine Konzepte zur optimalen Ersetzung von Seiten, da sie nur die Lokalität der Transaktionen innerhalb ihrer Fenstergrößen τ betrachtet. Daher muss mittels einer Seitenersetzungsstrategie (siehe unten) eine Seite aus der Menge der ersetzbaren Seiten bestimmt werden.

Bei jeder vom Systempuffer nicht erfüllbaren Seitenanforderung muss die Working-Set-Strategie die aktuellen Working-Sets der Transaktionen ermitteln, d. h., für jede Transaktion muss die Menge der innerhalb der letzten τ Referenzen angesprochenen Seiten festgestellt werden. Realisiert werden kann dies durch die Einführung von transaktionsbezogenen Referenzzählern $trz(T_i)$ und seitenbezogenen Referenzzählern $srz(T_i, S)$. Wird Seite S von Transaktion T_i referenziert, wird $trz(T_i)$ zunächst inkrementiert, bevor dessen Wert nach $srz(T_i, S)$ kopiert wird. Zu einem bestimmten Zeitpunkt sind alle diejenigen einer Transaktion zugeordneten Seiten ersetzbar, für die $trz(T_i) - srz(T_i, S) \geq \tau$ gilt. Für in mehreren Working-Sets gleichzeitig vorkommende und gemeinsam benutzte Seiten sind zusätzliche, hier nicht beschriebene Maßnahmen erforderlich. Bezüglich der in Bild 2.10 dargestellten Seitenreferenzfolge erhält man mittels dieser Methode folgende Werte für die Referenzzähler:

Zustand zum Zeitpunkt t_3:	$trz(T_1) = 8$	$trz(T_2) = 6$	
	$srz(T_1, S_1) = 8$	$srz(T_2, S_2) = 1$	Für $\tau = 5$ sind
	$srz(T_1, S_3) = 3$	$srz(T_2, S_4) = 2$	S_3 und S_2 ersetzbar.
	$srz(T_1, S_7) = 6$	$srz(T_2, S_5) = 4$	
	$srz(T_1, S_8) = 7$	$srz(T_2, S_6) = 6$	

Ersetzungsstrategien für Seiten

Ein wichtiger Aspekt der Systempufferverwaltung ist die *Seitenersetzungsstrategie* (*replacement strategy*), die benutzt wird, um eine unfixierte Seite zur Ersetzung auszuwählen, falls eine Seitenanforderung im Systempuffer nicht befriedigt werden kann. Das Ziel einer Ersetzungsstrategie ist die Minimierung der Anzahl der Seitenzugriffe. Es gibt

viele verschiedene Strategien, und jede ist für bestimmte Situationen besonders geeignet. Näherungsweise kann das Problem mit demjenigen in Betriebssystemen verglichen werden. Es gibt jedoch auch einige Unterschiede zu berücksichtigen. Im virtuellen Speicher eines Betriebssystems ist prinzipiell jede Seite zu jedem Zeitpunkt ersetzbar. Allerdings ist es nicht allgemein möglich, die zukünftige Seitenreferenz präzise vorherzusehen. Daher verwenden Betriebssysteme vergangene Seitenreferenzen als Bezugspunkte für zukünftige Seitenanforderungen. Im Gegensatz dazu sind im Systempuffer fixierte Seiten zu berücksichtigen, die von der Ersetzung ausgenommen sind. Ferner ist ein Datenbanksystem häufig in der Lage, aufgrund zusätzlicher Informationen zukünftige Seitenreferenzen genauer als ein Betriebssystem vorauszusehen.

Der Wirkungskreis einer Ersetzungsstrategie hängt von der gewählten Speicherzuteilungsstrategie ab und wird zusätzlich durch fixierte Seiten reduziert. Eine Ersetzung kann erfolgen bei globaler Speicherzuteilung im gesamten Systempuffer, bei seitentypbezogener und lokaler Speicherzuteilung mit statischen Bereichen in der betreffenden Partition, der die auslösende Seitenreferenz zuzuordnen ist, und bei dynamischer Speicherzuteilung in der Menge der ersetzbaren Seiten, die augenblicklich zu keinem Working-Set einer Transaktion oder eines Seitentyps gehören.

Im Folgenden stellen wir einige Beispiele für Ersetzungsstrategien vor. Diese sind angesiedelt zwischen einer nicht realisierbaren Strategie *Optimal*, die jeweils diejenige Seite im Systempuffer ersetzt, deren zeitlicher Abstand bis zur nächsten Seitenanforderung maximal ist, und einer realisierbaren Strategie *Random*, die keine Kenntnisse des Referenzverhaltens ausnutzt und davon ausgeht, dass alle Seiten im Systempuffer denselben Erwartungswert für ihre erneute Benutzung haben. Realisierbare Strategien mit einem besseren Verhalten als *Random* ersetzen diejenige Seite im Systempuffer, deren Erwartungswert für ihre erneute Benutzung minimal ist. Zur Vorhersage des zukünftigen Referenzverhaltens nutzen sie dazu Kenntnisse des bisherigen Verhaltens aus. Wegen der in Seitenreferenzfolgen beobachteten Lokalität gilt das jüngste Referenzverhalten als ein guter Indikator für die nähere Zukunft. Als Kriterien für das zukünftige Referenzverhalten eignen sich vor allem das Alter und die Referenzen einer Seite im Systempuffer. Dabei ist zu differenzieren, ob das Alter seit der Einlagerung, seit dem letzten Referenzzeitpunkt oder überhaupt nicht, und ob alle Referenzen, die letzte Referenz oder keine bei der Festlegungsentscheidung einer Strategie zum Tragen kommen.

Die Strategie *FIFO* (*First-In*, *First-Out*) ersetzt diejenige Seite, die sich am längsten im Systempuffer befindet. Es entscheidet also allein das Alter einer Seite seit ihrer Einlagerung. Daher ist die FIFO-Strategie nur bei strikt sequentiellem Zugriffsverhalten anwendbar. Stellen wir uns die Seiten kreisförmig nach Alter angeordnet vor, und nehmen wir an, dass ein Zeiger jeweils auf die älteste Seite im Systempuffer verweist. Beim Laden

einer neuen Seite wird diese Seite ersetzt und der Zeiger auf die nächste Seite fortgeschaltet. Eine andere Strategie, *LFU* (*Least Frequently Used*) genannt, setzt ausschließlich auf die Referenzhäufigkeit, und zwar ersetzt sie diejenige Seite im Systempuffer mit der geringsten Referenzhäufigkeit. Realisiert werden kann diese Strategie durch Einführung eines Referenzzählers für jede Seite im Systempuffer, der bei Seiteneinlagerung mit 1 initialisiert und bei jeder weiteren Referenz um 1 erhöht wird. Muss eine Seite ersetzt werden, wird diejenige mit dem kleinsten Wert im Referenzzähler ausgewählt. Gibt es mehrere Seiten mit kleinstem Wert im Referenzzähler, so muss ein geeigneter Algorithmus eine dieser Seiten auswählen. Diese Strategie hat zur Folge, dass Seiten, die in einem kurzen Zeitraum außerordentlich häufig referenziert wurden, für lange Zeit nicht mehr zu verdrängen sind, selbst dann nicht, wenn sie später nie mehr angefordert werden. Aus diesem Grund erweist sich die Realisierung der LFU-Strategie als sehr ungünstig.

Alle im Folgenden beschriebenen Strategien berücksichtigen sowohl Alter als auch Referenzhäufigkeit. Die wohl bekannteste und verbreitetste Ersetzungsstrategie ist *LRU* (*Least Recently Used*), die diejenige Seite im Systempuffer ersetzt, die am längsten nicht mehr angesprochen wurde. Alle im Systempuffer befindlichen, unfixierten Seiten werden in einer Schlange (Queue) verwaltet. Eine Seite kommt bei jeder Referenz an den Anfang der Schlange. Die Annahme ist, dass eine Seite, die erst kürzlich referenziert worden ist, wahrscheinlich bald wieder referenziert werden wird. Bei einer erforderlichen Seitenersetzung wird die Seite am Ende der Schlange ausgelagert. Eine Seite gelangt an das Ende der Schlange, wenn sie von einem fixierten in einen unfixierten Zustand übergeht und somit ein Kandidat für eine Ersetzung wird. Die LRU-Strategie ist allerdings nicht immer die beste Ersetzungsstrategie für ein Datenbanksystem. Insbesondere können sequentielle Durchläufe durch Dateien ständige Ersetzungen notwendig machen. Betrachten wir beispielsweise einen Systempuffer mit n Rahmen und eine sequentiell zu durchlaufende Datei mit k Seiten. Sei zunächst $k \leq n$, und nehmen wir der Einfachheit halber an, dass es keine konkurrierenden Anforderungen nach Seiten gibt. Nur der erste Durchlauf durch die Datei erfordert externe Seitenzugriffe; Seitenanforderungen in weiteren Durchläufen können auf die gewünschten Seiten im Systempuffer zugreifen. Nehmen wir nun an, dass die zu durchlaufende Datei $k = n + 1$ Seiten besitzt, so wird jeder Durchlauf der Datei ein Lesen jeder Seite der Datei zur Folge haben. In dieser Situation ist die LRU-Strategie die schlechteste Ersetzungsstrategie.

Eine Variante, *Clock*-Strategie genannt, zeigt ein ähnliches Verhalten wie die LRU-Strategie, ist aber einfacher zu realisieren. Jeder Seite wird ein Benutzt-Bit zugeordnet, dass bei jeder Seitenreferenz auf 1 gesetzt wird. Bei Einlagerung einer neuen Seite wird mittels einer zyklischen Suche eine geeignete, ersetzbare Systempufferseite zur Ersetzung bestimmt. Dabei werden nicht ersetzbare, also fixierte, Seiten übersprungen. Bezüglich jeder ersetzbaren Seite wird das Benutzt-Bit überprüft. Steht es auf 1, wird es auf 0

gesetzt, und die Suche schreitet mit der nächsten Seite voran. Die erste ersetzbare Seite, deren Benutzt-Bit auf 0 steht, wird zur Ersetzung ausgewählt. Jede Seite überlebt also mindestens zwei zyklische Umläufe durch die Systempufferseiten. Letztendlich überlebt eine Seite also nur dann, wenn sie während eines Umlauf erneut referenziert wird. Sollen Seiten, die nur einmal referenziert werden, schneller verdrängt werden, so wird das Benutzt-Bit bei der ersten Referenz mit 0 initialisiert und nur bei jeder weiteren Referenz auf 1 gesetzt.

Zuguterletzt erwähnen wir noch die *MRU* (*Most Recently Used*)-Strategie. Sie beruht auf den genau entgegengesetzten Annahmen wie LRU. Hier wird diejenige Seite im Systempuffer ersetzt, die zuletzt referenziert worden ist. Weitere, teilweise sehr ausgefeilte Strategien sind der (Betriebssystem-)Literatur zu entnehmen.

Probleme bei der Verwaltung des Systempuffers

Ein Datenbankmanagementsystem wird von einem Betriebssystem gewöhnlich wie ein normales Anwendungsprogramm behandelt. Je nach Art der Einbettung des DBMS in eine Betriebssystemumgebung können sich hieraus entscheidende Auswirkungen für die Systempufferverwaltung ergeben. Wenn das DBMS in einer virtuellen Betriebssystemumgebung abläuft, sind sowohl der Programm-Code als auch der Systempuffer dem Seitenersetzungsmechanismus des Betriebssystems unterworfen, ausgenommen, sie können durch spezielle Maßnahmen dauerhaft im Hauptspeicher aufbewahrt werden. Das heißt, den Rahmen des Systempuffers, die Seiten der Datenbank aufnehmen, werden durch den Seitenersetzungsalgorithmus des Betriebssystems eine gewisse Anzahl von Hauptspeicherrahmen zugeordnet. Wird eine Seite benötigt, die sich nicht im Hauptspeicher befindet, sind im Wesentlichen folgende drei Fälle zu unterscheiden:

- *Page Fault*. Die angeforderte Seite befindet sich zwar im Systempuffer, sie ist aber durch den Seitenersetzungsalgorithmus des Betriebssystems ausgelagert worden. Hier muss das Betriebssystem die referenzierte Seite wieder einlagern.
- *Database Fault*. Die angeforderte Seite befindet sich nicht im Systempuffer. Die zu ersetzende Seite ist jedoch im Hauptspeicher und kann nach eventuellem Zurückschreiben freigegeben werden. Danach wird die referenzierte Seite aus der Datenbank ausgelesen.
- *Double Page Fault*. Die angeforderte Seite befindet sich nicht im Systempuffer; die zur Ersetzung ausgewählte Seite befindet sich nicht im Hauptspeicher. In diesem Fall muss zunächst die zu ersetzende Seite durch das Betriebssystem wieder eingele-

sen werden, bevor nach eventuellem Zurückschreiben ihre Freigabe und das Einlagern der angeforderten Seite geschehen kann.

Ein anderes Problem bezieht sich auf eine mögliche Überlast der Systempufferverwaltung, insbesondere bei kleinem Systempuffer. Da jede Transaktion zu einem Zeitpunkt mehrere Systempufferseiten in einem fixierten Zustand halten kann, kann es bei einer weiteren Seitenanforderung zu einer Deadlocksituation (dies bedeutet hier, zu einer Verknappung an Pufferrahmen), kommen, wenn momentan alle Seiten des Systempuffers fixiert sind und daher nicht ersetzt werden können. Die Anzahl der Rahmen, die einer Transaktion zugeordnet werden können, sinkt bei vorgegebener Puffergröße mit zunehmender Anzahl paralleler Transaktionen. Dadurch steigt die relative Häufigkeit von Seitenanforderungen, die zu externen Seitenzugriffen führen. Obwohl die Kosten für jeden externen Seitenzugriff konstant bleiben, erhöht sich insgesamt der Verwaltungsaufwand infolge der zunehmenden relativen Häufigkeit der Seitenersetzung. In extremen Fällen kann ein *Seitenflattern* (*thrashing*) eintreten, bei dem durch drastische Reduzierung der einer Transaktion zur Verfügung stehenden Seiten die Häufigkeit der notwendigen Seitenersetzung enorm ansteigt, so dass das System fast ausschließlich mit Verwaltungsaufgaben beschäftigt ist, aber ansonsten fast keine sinnvolle Arbeit verrichtet. Maßnahmen, die zu einer Verringerung der Thrashing-Problematik vorgeschlagen worden sind, beinhalten eine Optimierung der Ersetzungsstrategie, die Verringerung der Kosten für eine Seitenersetzung sowie eine Optimierung des Referenzverhaltens von Programmen. Diese Maßnahmen bewirken eine Verringerung des Verwaltungsaufwands und haben eine Erhöhung der maximalen Anzahl paralleler Transaktionen zur Folge. Aber auch dann ist insbesondere bei weiterer Zunahme der Parallelität die Gefahr des Seitenflatterns nicht gebannt. Letztendlich kann Seitenflattern nur durch eine Einschränkung der Parallelität von Transaktionen verhindert werden. Durch Zusammenwirken der Systempuffer- und der Transaktionsverwaltung muss erreicht werden, dass bei einem Seitenmangel im Systempuffer keine weiteren Seitenanforderungen von Transaktionen zugelassen werden.

2.9.6 Unterschiede der Systempufferverwaltung in Datenbanksystemen und in Betriebssystemen

An mehreren Stellen wurden bereits die Ähnlichkeiten zwischen dem virtuellen Speicher in Betriebssystemen und der Systempufferverwaltung in Datenbanksystemen angedeutet. In beiden Fällen besteht das Ziel darin, auf mehr Daten Zugriff zu gewähren als in den Hauptspeicher hineinpassen. Die grundlegende Idee besteht darin, Seiten auf Anforderung vom Externspeicher in den Hauptspeicher zu bringen. Hierzu werden aktuell im

Hauptspeicher nicht mehr benötigte Seiten ersetzt. Es stellt sich die Frage, warum ein DBMS nicht mit Hilfe des virtuellen Speichers des Betriebssystems konstruiert wird. Wie bereits erwähnt, kann ein DBMS Seitenreferenzfolgen wesentlich präziser vorhersagen als dies von einem Betriebssystem möglich ist, und diese Eigenschaft sollte möglichst ausgenutzt werden. Die Möglichkeit der Vorhersage von Seitenreferenzfolgen beruht darauf, dass Seitenreferenzen meist von Operationen übergeordneter Komponenten erzeugt werden und diese Operationen (z. B. relationale Algebraoperationen) bekannte Referenzmuster haben. Dies macht den Einsatz spezieller Ersetzungsstrategien in einer DBMS-Umgebung lohnenswerter. Des Weiteren benötigt ein DBMS mehr als ein Betriebssystem Kontrolle darüber, wann eine Seite auf Externspeicher zurückgeschrieben werden kann (Fixierung von Seiten).

Fast noch wichtiger ist, dass eine Vorhersage von Seitenreferenzen, z. B. in einem sequentiellen Durchlauf, eine sehr einfache und effiziente Strategie ermöglicht, nämlich das vorausschauende Holen von Seiten (*pre-fetching of pages*). Der Systempuffer kann die nächsten Seitenanforderungen vorwegnehmen, indem er die entsprechenden Seiten in den Systempuffer lädt, *bevor* die Seiten angefordert werden. Dies hat zwei Vorteile. Zum einen sind die Seiten verfügbar, wenn sie angefordert werden. Zum anderen ist das Lesen einer zusammenhängenden Folge von Seiten viel schneller als das Lesen der gleichen Seiten zu verschiedenen Zeiten aufgrund verschiedener Anfragen. Für den Fall, dass die im voraus zu holenden Seiten nicht zusammenhängend sind, kann das Wissen, dass mehrere Seiten geladen werden müssen, dennoch zu einem schnelleren Ein-/Ausgabe-Verhalten führen, weil die Seiten in einer bestimmten Reihenfolge geladen werden können, die Suchzeiten und Rotationsverzögerungen minimiert.

Ferner muss ein DBMS in der Lage sein, ein explizites Hinausschreiben einer Seite auf einen Externspeicher zu bewirken. Ein DBMS muss sicherstellen können, dass gewisse Systempufferseiten auf einem Externspeicher gesichert werden, *bevor* andere Seiten weggeschrieben werden. Man kann sich bei virtuellen Speicherimplementierungen in Betriebssystemen nicht darauf verlassen, dass solch eine Kontrolle beim Wegschreiben von Seiten ausgeübt wird. Benötigt wird diese Eigenschaft zur Implementierung von Recovery-Algorithmen.