Theo Härder
Erhard Rahm

Datenbanksysteme - Konzepte und Techniken der Implementierung

Mit 216 Abbildungen und 15 Tabellen

Springer

Prof. Dr.-Ing. Theo Härder
Universität Kaiserslautern
Fachbereich Informatik
D-67653 Kaiserslautern

Prof. Dr.-Ing. Erhard Rahm
Universität Leipzig
Institut für Informatik
Augustusplatz 10-11
D-04109 Leipzig

Springer-Verlag 1999

Vorwort

Datenbanksysteme gehören zu den wichtigsten Produkten der Software-Industrie; kaum eine größere Informatikanwendung ist ohne Datenbankunterstützung denkbar. Ausgehend von der Nutzung im Rahmen betrieblicher Transaktions- und Informationssysteme hat sich ihr Einsatzbereich ständig ausgeweitet, insbesondere in zahlreichen anspruchsvollen Anwendungsfeldern. Ausdruck dafür ist die Vielzahl spezifischer Informationssysteme (Krankenhaus-Informationssysteme, Geographische Informationssysteme, Dokumenten-Verwaltungssysteme, Entscheidungsunterstützende Systeme, ...) sowie rechnergestützter Entwurfs- und Fertigungssysteme (CAD/CAM, CASE, CIM, ...), deren Datenhaltung und -verwaltung typischerweise über ein Datenbanksystem als Kernkomponente erfolgt. Eine starke Zunahme an Datenbanken ergibt sich ferner durch die enorm wachsenden Datenmengen, die im Internet weltweit zugänglich bereitgestellt werden, verbunden mit einer sehr großen Benutzerzahl und entsprechenden Leistungserfordernissen. Schließlich werden Datenbanksysteme millionenfach als PC-Einzelplatzsysteme zur einfachen und flexiblen Verwaltung privater Datenbestände eingesetzt.

Aufgrund von rund 30 Jahren intensiver Entwicklung und Forschung sind Datenbanksysteme eine klassische Domäne der Informatik, zu der eine Fülle abgesicherten, in der Praxis erprobten Wissens vorliegt. Von primärer Bedeutung für den Nutzer von Datenbanksystemen ist hierbei das zugrundeliegende Datenmodell mit seinen Möglichkeiten der Strukturierung und Manipulation der für eine Anwendungsumgebung zu verwaltenden Datenbestände. Die zugehörigen Fragestellungen und Aspekte des konzeptionellen Datenbankentwurfs, der logischen Datenbankdefinition und der Verwendung von Anfragesprachen wie SQL sind daher auch Gegenstand einführender Bücher und Lehrveranstaltungen zu Datenbanksystemen. Dem Nutzer, für den vor allem einfache Handhabung und hohe Datenunabhängigkeit zu gewährleisten sind, bleibt dagegen weitgehend verborgen, wie die sehr umfangreiche Funktionalität von Datenbanksystemen realisiert wird. Die hierzu verfügbaren Implementierungstechniken und -konzepte sind in ihrer vollen Breite somit auch nur vergleichsweise wenigen Fachleuten bekannt. Die Kenntnis dieser Verfahren ist jedoch Voraussetzung für ein tieferes Verständnis von Datenbanksystemen und somit für alle Informatiker unentbehrlich, die große Systeme entwikkeln oder administrieren.

Das vorliegende Buch behandelt die Implementierung von Datenbanksystemen mit der dabei zu wählenden Vorgehensweise, den bereitzustellenden Funktionen sowie den verfügbaren Algorithmen und Datenstrukturen. Das gilt für alle Aspekte der Datenabbildung, also vor allem der Speicherungsstrukturen und Zugriffspfade sowie der Bereitstellung von Datenstrukturen gemäß logischer Datenbankmodelle. Berücksichtigung finden ferner Aufgaben wie die Externspeicherverwaltung (Realisierung einer Speicherhierarchie, Datei- und Segmentkonzept) sowie die Pufferverwaltung von Datenbanksystemen. Einen weiteren Schwerpunkt bildet die Transaktionsverwaltung, die insbesondere Funktionen zur Synchronisation des Mehrbenutzerbetriebs und für die Wiederherstellung der Datenbank im Fehlerfall (Recovery-Funktionen) umfaßt. Die Darstellung orientiert sich an einem Schichtenmodell für datenunabhängige Datenbanksysteme, das durchgängig zur Erklärung aller Abbildungen und Abläufe herangezogen wird. Dieses als unser Architekturrahmen dienende Modell wird in Kapitel 1 vorgestellt, bevor in Abschnitt 1.5 der weitere Aufbau des Buches näher erläutert wird.

Die ausgewählten Techniken eignen sich in erster Linie zur Realisierung zentralisierter Datenbanksysteme nach den „klassischen“ Datenbankmodellen wie dem Relationen- und dem Netzwerkmodell. Jedoch wurden diese bewährten Techniken um spezielle Algorithmen und Strukturen ergänzt, die bei der Implementierung neuerer Datenbankmodelle (mit komplexen Objekten oder Objektorientierung) große Vorteile versprechen. Die vorgestellten Implementierungstechniken bilden auch die Grundlage zur Realisierung verteilter Architekturen wie Verteilte Datenbanksysteme, Parallele Datenbanksysteme und Föderierte Datenbanksysteme. Die nähere Behandlung solcher Mehrrechner-Datenbanksysteme hätte jedoch den Rahmen dieses Buches gesprengt, zumal sie bereits Gegenstand von Büchern wie [RAHM94] und [CONR97] sind.

Das Buch ist eine vollständig überarbeitete, erweiterte und aktualisierte Version der Kapitel 3 und 4 des 1987 erschienenen Datenbank-Handbuchs [LOCK87]. Schon damals war es aus Platzgründen unmöglich, alle in der Literatur vorgeschlagenen oder in existierenden Datenbanksystemen eingesetzten Konzepte und Techniken zu beschreiben. Heute gilt diese Einschränkung trotz der annähernden Verdopplung des verfügbaren Seitenumfangs in verschärftem Maße; sie wird besonders deutlich bei Themenbereichen wie „Mehrdimensionale Zugriffspfade“ oder „Erweiterungen des Transaktionskonzeptes“, in denen in den letzten Jahren sehr viele Vorschläge, allerdings oft ohne jede praktische Überprüfung, publiziert wurden. Unsere Stoffauswahl, die natürlich subjektiv gefärbt ist, richtete sich in erster Linie nach der praktischen Tauglichkeit der Konzepte und Techniken. Allerdings wurden auch Vorschläge aufgenommen und exemplarisch diskutiert, bei denen ein solcher Nachweis noch nicht erbracht werden konnte, insbesondere um neue und interessante Entwicklungslinien aufzuzeigen.

Die Konzeption des Buches wurde auch durch mehrere an den Universitäten Kaiserslautern und Leipzig gehaltene weiterführende Vorlesungen über Datenbanksysteme geprägt, da die Diskussion mit den Studenten und deren Rückmeldungen Stoffauswahl und Darstellung beeinflußten. Das Buch richtet sich somit zum einen an Studenten und Dozenten im Informatik-Hauptstudium. Zum anderen sind Forscher sowie in Entwicklung, Anwendung oder System-

verwaltung stehende Praktiker angesprochen, die auf fundierte Datenbankkenntnisse angewiesen sind. Vorausgesetzt werden ein solides Informatikgrundwissen sowie Grundkenntnisse von Datenbanksystemen.

Wir sind zahlreichen Fachkollegen, Mitarbeitern und Studenten, die wesentlich zum Gelingen des Buches beigetragen haben, zu Dank verpflichtet. Besonders möchten wir uns bedanken bei Michael Gesmann, Axel Herbst, Michael Jaedicke, Henrik Loeser, Holger Märtens, Bernhard Mitschang, Udo Nink, Norbert Ritter, Dieter Sosna, Hans-Peter Steiert, Thomas Stöhr, Nan Zhang und Jürgen Zimmermann für die vielen Anregungen und Verbesserungsvorschläge, die wir während verschiedener Phasen der Entstehungsgeschichte des Buches von ihnen bekommen haben. Unsere Sekretärinnen Manuela Burkart, Heike Neu und Andrea Hesse haben uns bei der Erstellung des Manuskriptes und seiner zahlreichen Abbildungen wesentlich unterstützt. Hans Wössner und seine Mitarbeiter vom Springer-Verlag waren stets hilfreich und haben für eine schnelle und reibungslose Veröffentlichung des Buches gesorgt.

Die Fertigstellung des Buches beanspruchte, insbesondere in den letzten Monaten, einen Großteil unserer Zeit. Natürlich kam dabei, wie auch sonst oft, das gemeinsame Leben mit unseren Familien zu kurz. Deshalb möchten wir uns an dieser Stelle ganz besonders bei unseren Familien für ihre Geduld und den Verzicht auf gemeinsame Zeit bedanken.

Ergänzende Informationen und Unterlagen zu dem Buch (Folien zugehöriger Vorlesungen usw.) können im Internet unter

http://www.uni-kl.de/AG-Haerder/Buecher/ oder unter
http://www.informatik.uni-leipzig.de/DBSI-Buch

abgerufen werden.

Kaiserslautern und Leipzig, im Januar 1999

Theo Härder
Erhard Rahm

Inhaltsverzeichnis

Teil I

Architektur von Datenbanksystemen

1 Architektur von Datenbanksystemen

1.1 Anforderungen an Datenbanksysteme

Informationssysteme werden für bestimmte Anwendungsbereiche, auch Miniwelten genannt, eingeführt und sollen über alle Sachverhalte und Vorgänge dieser Miniwelten möglichst aktuelle Auskunft geben. Dazu ist es erforderlich, die betrachteten Realitätsausschnitte rechnerseitig durch ein Modell zu repräsentieren und die Vorgänge (Ereignisse) der Miniwelt, die zu neuen oder geänderten Sachverhalten führen, in diesem Modell zeitgerecht und genau nachzuvollziehen. Die modellhafte Nachbildung der Miniwelt erzwingt Abstraktion bei der Beschreibung ihrer Objekte und Beziehungen, die durch das Modell abgebildet werden. Vorgänge in der Realität überführen diese in neue Zustände. Da relevante Zustände vom Modell erfaßt werden müssen, sind auch die Zustandsänderungen durch Folgen von Operationen des Modells möglichst genau nachzubilden, so daß eine möglichst gute Übereinstimmung der Folgezustände in Realität und Modell erreicht werden kann. Darüber hinaus müssen Zustandsübergänge systemseitig auch hinsichtlich des Auftretens von Fehlern ununterbrechbar sein. Solche Anforderungen werden technisch durch Transaktionen umgesetzt, wobei das ACID-Paradigma [GRAY81c] weitreichende Zusicherungen für die Qualität der Modellzustände übernimmt.

1.1.1 Entwicklung von Anwendungssystemen

Die Entwicklung von großen Anwendungs- und Informationssystemen erfordert vielfältige Maßnahmen zur Gewährleistung aktueller, konsistenter und persistenter Daten. Systemrealisierungen, die dazu isolierte Dateien einsetzen, weisen eine Reihe schwerwiegender Nachteile auf. Mit solchen losen Sammlungen von Dateien lassen sich Ausschnitte aus einem Anwendungsbereich mit ihren Sachverhalten, Abhängigkeiten und Beziehungen nur sehr grob modellieren. Die Speicherung und Aktualisierung der Daten erfolgt ohne zentralisierte Kontrolle, so daß fehlerhafte, widersprüchliche oder unvollständige Informationen nur sehr schwer oder oft gar nicht aufgedeckt werden. In der Regel werden Dateien im Hinblick auf konkrete Anwendungen konzipiert und in ihren Speicherungsstrukturen auf die speziellen Verarbeitungsanforderungen optimiert. Diese Anwendungsanbindung erzeugt in hohem Maße Datenabhängigkeiten, schränkt die flexible Nutzung von Dateien durch andere Anwendungsprogramme

ein und ist ein wesentlicher Grund für die mangelnde Erweiterbarkeit und Offenheit des gesamten Anwendungssystems. Ergebnis einer solchen Vorgehensweise ist eine häufig redundante Speicherung gleicher Daten in verschiedenen Dateien, wodurch eine zeitgerechte und alle Kopien umfassende Änderung verhindert wird. Zusätzlicher Informationsbedarf und Änderungen in betrieblichen Abläufen erzwingen ständig eine Evolution des Anwendungssystems, wobei wegen der „eingefrorenen“ Datenstrukturen und -beziehungen oft nur die Einführung zusätzlicher Datenredundanz weiterhilft und so die Situation immer weiter verschlimmert wird.

Bei einer sich schnell verändernden Anwendungswelt und der Forderung nach kurzfristigen und flexiblen Anpassungen der Anwendungssysteme an die sich ständig ändernde Systemumgebung sind bei Einsatz isolierter Dateien zumindest die Entwurfsziele „Aktualität und Konsistenz der Daten“ nicht zu erreichen. Seit mehr als 30 Jahren setzen sich deshalb für alle Aufgaben der Datenhaltung mit zunehmendem Erfolg Datenbanksysteme durch, die eine integrierte Sichtweise auf alle Daten anbieten und alle Fragen der Konsistenzerhaltung, Systemevolution, Anpassung an geänderte Umgebungsbedingungen usw. unabhängig von den Anwendungsprogrammen regeln.

1.1.2 Entwurfsziele

Die Entwicklung von Datenbanksystemen wird durch ein breites Spektrum an Anforderungen begleitet, die in geeignete Systemarchitekturen sowie Implementierungskonzepte und -techniken umzusetzen sind. Dabei sind vor allem folgende Entwurfsziele zu realisieren oder zumindest vorrangig anzustreben:

- Integration der Daten und ihre unabhängige sowie logisch zentralisierte Verwaltung,
- Datenunabhängigkeit und Anwendungsneutralität beim logischen und physischen Datenbankentwurf,
- Einfache und flexible Benutzung der Daten durch geeignete Anwendungsprogrammierschnittstellen,
- Zentralisierung aller Maßnahmen zur Integritätskontrolle,
- Transaktionsschutz für die gesamte Datenbankverarbeitung,
- Effiziente und parallele Verarbeitung von großen Datenbasen,
- Hohe Verfügbarkeit und Fehlertoleranz,
- Skalierbarkeit bei Wachstum der Transaktionslast und der Datenvolumina.

Integration der Daten und Unabhängigkeit der Datenverwaltung erfordern das Herauslösen aller Aufgaben der Datenverwaltung und Konsistenzkontrolle aus den Anwendungsprogrammen sowie ihre Standardisierung und Übernahme in ein logisch zentralisiertes System, das Zuverlässigkeit, Widerspruchsfreiheit und Vollständigkeit der Operationen auf allen Daten gewährleisten kann. Langfristigkeit der Datenspeicherung und Konsistenzzusicherungen, auch im Fehlerfall, können bei einer integrierten Datenbasis ohnehin nicht gemeinsam durch eine Menge individuell entworfener Anwendungsprogramme erbracht werden. Die *Zentralisierung dieser Aufgabe* erfolgte deshalb durch große, unabhängige und generische Software-Systeme,

die als Datenbankverwaltungssysteme (DBVS) bezeichnet werden. Zusammen mit den gespeicherten Daten der Datenbasis, kurz Datenbank (DB) genannt, bildet das DBVS ein Datenbanksystem (DBS). In der Regel werden wir auf die feine Unterscheidung der Begriffe DBVS und DBS verzichten und DBS als einheitliche Bezeichnung benutzen.

Datenunabhängigkeit ermöglicht den Anwendungsprogrammen eine Benutzung der DB-Daten, ohne Details der systemtechnischen Realisierung zu kennen. Dies läßt sich vor allem durch logische Datenmodelle und deklarative Anfragesprachen erzielen. Datenunabhängigkeit der Anwendungen besitzt als Entwurfsziel von DBS eine herausragende Rolle, da durch zusätzlichen Informationsbedarf und Strukturänderungen in der Miniwelt die DB-Strukturen einer ständigen Evolution unterworfen sind. Idealerweise sollten Anwendungsprogramme und DBS so stark voneinander isoliert sein, daß sie eine wechselseitige Änderungsimmunität aufweisen. Neben der Datenunabhängigkeit ist beim logischen DB-Entwurf *Anwendungsneutralität* und damit auch Offenheit für neue Anwendungen anzustreben. Die gemeinsame Benutzung von großen und integrierten Datenbeständen läßt einen Zuschnitt auf die Anforderungen bestimmter Anwendungen nicht sinnvoll erscheinen; außerdem könnte eine einseitige Ausrichtung des DB-Entwurfs wegen der Notwendigkeit ständiger Systemevolution schnell obsolet werden. Eine solche Anwendungsneutralität beim Entwurf von logischen Datenstrukturen bedeutet die Wahl redundanzfreier und symmetrischer Organisationsformen und verbietet die explizite Bevorzugung einzelner Anwendungsprogramme durch zugeschnittene Verarbeitungs- und Auswertungsrichtungen. Aus der Sicht der Anwendungen ist Redundanzfreiheit auch für die physische Repräsentation der Daten (physischer DB-Entwurf) gefordert. Da diese sich jedoch nicht unmittelbar auf Speicherungsstrukturen (siehe nachfolgende Diskussion der Anwendungsprogrammierschnittstellen) beziehen können, kann hier selektive und DBS-kontrollierte Redundanz zugelassen werden, um beispielsweise das Leistungsverhalten wichtiger Anwendungen zu verbessern.

Wesentliche Voraussetzung für Datenunabhängigkeit und zugleich einfache Benutzung von DBS sind „hohe“ *Anwendungsprogrammierschnittstellen* (API, application programming interface). Hauptkennzeichen sind logische Datenmodelle und deklarative Anfragesprachen, da sich DB-Operationen dadurch nur auf abstrakte Objektrepräsentationen, ohne Hinweis auf die erforderliche Zugriffslogik, beziehen können. Da nur das „was wird benötigt?“ und nicht das „wie wird es aufgesucht?“ zu spezifizieren ist, ergibt sich zugleich eine einfache Handhabung der Daten durch den Benutzer. Deklarative Anfragesprachen sind mengenorientiert und geben keine Auswertungsrichtung vor, was dem DBS Optimierungsmöglichkeiten bei komplexen DB-Operationen einräumt. Neben der Einbettung von Programmierschnittstellen in Wirtssprachen, die eine anwendungsbezogene Weiterverarbeitung von DB-Daten zulassen, ist ein flexibler DB-Zugang über Ad-hoc-Anfragesprachen eine wichtige Schnittstellenanforderung.

Die gemeinsame Datenbenutzung erfordert zwingend eine *Zentralisierung aller Maßnahmen zur Integritätskontrolle*. Das betrifft vier verschiedene Arten von Kontrollaufgaben mit entsprechenden Reaktionen bei erkannten Integritätsverletzungen. Die erste Kontrollaufgabe regelt die Zulässigkeit des Datenzugriffs aufgrund betrieblicher Regelungen oder gesetzlicher Maßnahmen (Datenschutz). Durch Zugriffskontrolle ist zu gewährleisten, daß nur berechtigte

Benutzer Daten verarbeiten dürfen, und zwar im Rahmen der für sie definierten Zugriffsrechte und Datengranulate. Um die Datenbank konsistent zu halten, muß mit Hilfe von semantischen Integritätsbedingungen (constraints) bei allen Änderungen oder Aktualisierungen (siehe Transaktionskonzept) geprüft werden, ob der neue DB-Zustand akzeptabel ist.[1] Solche Bedingungen (Prädikate) sind explizit zu spezifizieren und durch das DBS zu überwachen, was bei Systemevolution auch die Durchführung von Änderungen oder Ergänzungen der Integritätsbedingungen erleichtert. Weiterhin muß das DBS durch Aufzeichnung redundanter Daten (Logging) im Normalbetrieb Vorsorge für den Fehlerfall treffen, um beispielsweise nach Auftreten eines Systemausfalls oder eines Gerätefehlers einen konsistenten DB-Zustand (genauer, den jüngsten transaktionskonsistenten DB-Zustand [HÄRD83b]) wiederherstellen zu können (Recovery). Schließlich muß das DBS für Synchronisation, d. h. für korrekte Abwicklung konkurrierender Benutzeroperationen auf den gemeinsamen DB-Daten sorgen. Hierbei handelt es sich vor allem um die Vermeidung von Fehlern, die im unkontrollierten Fall durch wechselseitige Beeinflussung provoziert werden. Es ist offensichtlich, daß alle genannten Kontrollaufgaben nicht von einzelnen Anwendungsprogrammen ausgeübt werden können; sie verlangen vielmehr zentralisierte Überprüfung und Abhilfemaßnahmen.

Zentralisierte Integritätskontrolle ist wiederum Voraussetzung für den *Transaktionsschutz*, durch den das DBS jeder Anwendung weitreichende Zusicherungen für die Ausführung ihrer Aufsuch- und Aktualisierungsoperationen garantiert. Durch das Transaktionskonzept mit seinen ACID-Eigenschaften [GRAY81c] werden Korrektheit und Ablauf der gesamten DB-Verarbeitung, insbesondere bei konkurrierender Benutzung und im Fehlerfall, in wesentlichen Aspekten festgelegt. Als dynamische Kontrollstruktur bietet eine Transaktion Atomarität (atomicity) für alle ihre DB-Operationen, Konsistenz (consistency) der geänderten DB, isolierte Ausführung (isolation) der Operationen im Mehrbenutzerbetrieb sowie Dauerhaftigkeit (durability) der in die DB eingebrachten Änderungen [HÄRD83b][2]. Im Kern verkörpert das Transaktionskonzept die „Alles-oder-Nichts“-Eigenschaft (Atomarität) jeglicher DBS-Verarbeitung, was ein einfaches Fehlermodell für die Abwicklung von DB-Operationen zuläßt. Auch im Fehlerfall wird eine Transaktion entweder vollständig unter Beachtung der ACID-Eigenschaften ausgeführt oder alle ihre Auswirkungen in der DB werden so getilgt, als sei sie nie gestartet worden. Diese weitreichenden Garantien erlauben die Implementierung einer Art Vertragsrecht durch DBS-Anwendungen. Sie machen es einsichtig, daß heute alle unternehmenskritischen Vorgänge und Abläufe DB-seitig transaktionsgeschützt abgewickelt werden. Das betrifft sowohl zentralisierte als auch verteilte Anwendungen. In [GRAY93] wird gezeigt, wie durch sog. Ressourcen-Manager-Architekturen neben den DB-Daten auch andere Betriebsmittel wie Nachrichten, Dateien, Programmiersprachenobjekte usw. sich in den Transaktionsschutz einbeziehen lassen.

---

[1] Akzeptabel heißt hier, daß eine Transaktion einen DB-Zustand hinterläßt, der alle definierten Integritätsbedingungen erfüllt. Das bedeutet nicht, daß der DB-Zustand auch korrekt ist, d. h. mit der abgebildeten Miniwelt übereinstimmt.

[2] Daß sich diese weitreichenden Zusicherungen des Transaktionsmodells auch lückenlos auf reale Anwendungen übertragen lassen, wünscht Jim Gray allen Betroffenen: „May all your transactions commit and never leave you in doubt“.

Historisch gesehen wurde der Begriff „Transaktion" Mitte der siebziger Jahre durch DBS-Forscher und -Entwickler geprägt [GRAY76]. Seit dieser Zeit führte das Transaktionskonzept in DBS zu einem Paradigmenwechsel bei der Anwendungsprogrammierung. Seine Hauptvorteile liegen vor allem darin, daß dem Anwendungsprogramm eine fehlerfreie Sicht auf die Datenbank gewährt wird und daß es von allen Aspekten des Mehrbenutzerbetriebs isoliert ist. Weiterhin hat sich die Transaktionsorientierung als wesentliches Systemmerkmal in rasanter Weise durchgesetzt. Für DBS gilt sie heute bereits als unverzichtbare Eigenschaft, ja sogar als Definitionskriterium: ein DBS ohne Transaktionskonzept verdient seine Bezeichnung nicht. Auch in anderen Systemen, wie z. B. Betriebssystemen, findet es zunehmend Eingang.

Da Speichermedien immer billiger und zuverlässiger werden, wachsen heute die betrieblichen Datenbestände in enorme Größenordnungen, so daß *effiziente und parallele Verfahren der DB-Verarbeitung* einen hohen Stellenwert gewinnen. Zugriffsmethoden sollten sehr schnell und idealerweise unabhängig von der Größe des Datenbestandes sein, jedoch höchstens logarithmisch in der Zahl der Externspeicherzugriffe mit den Datenvolumina wachsen. Auswertungen, die durch neue Anforderungen auch immer komplexer werden, sollten durch zugeschnittene Algorithmen, verbesserte Vorplanung, vermehrten Hauptspeichereinsatz zur Datenpufferung usw. sowie durch Nutzung inhärenter Zugriffsparallelität optimiert werden können, um Leistungseinbußen beim Größenwachstum so weit wie möglich abzufangen.

Die Ubiquität von DBS in betrieblichen Anwendungs- und Informationssystemen verlangt quasi ständige Betriebsbereitschaft, was einerseits *hohe Verfügbarkeit und Fehlertoleranz* (*continuous operation*) erzwingt und andererseits keine separaten Zeiträume für die Reorganisation der Datenbestände, das Erstellen von Sicherungskopien usw. offenläßt. Folglich müssen Implementierungstechniken für Speicherungsstrukturen und Sicherungsmaßnahmen so ausgelegt sein, daß laufend dynamisch reorganisiert und inkrementell vorgegangen werden kann. Natürlich verlangt die Realisierung solcher Anforderungen redundante Auslegung und gekapselste Hardware- und Software-Entwurfseinheiten, um Fehler lokal isolieren und den Betrieb aufrechterhalten zu können. Techniken zur Erzielung sehr hoher Zuverlässigkeit sind jedoch nicht Inhalt dieses Buchs.

Ein DBVS ist als generisches System zu sehen, das in verschiedensten Anwendungsbereichen und mit unterschiedlichsten Leistungsanforderungen in bezug auf Durchsatz und Antwortzeit eingesetzt werden soll. Deshalb ist es außerordentlich wichtig, daß es *Skalierbarkeit* als wesentliche und durchgängige Systemeigenschaft besitzt, um bei unterschiedlichen Anwendungsprofilen mit weit variierenden Leistungsbereichen genutzt werden zu können. Ausschließlich durch Installation von mehr oder leistungsstärkeren Prozessoren sowie durch Vergrößerung von Haupt- und Externspeicher sollte eine idealerweise lineare Leistungssteigerung erzielbar sein, die sich in Einheiten der Anwendungsverarbeitung (Transaktionen) messen läßt. Beispielsweise sollte bei Zukauf von Hardware-Ressourcen der Transaktionsdurchsatz bei gleichbleibender mittlerer Antwortzeit entsprechend anwachsen. Andererseits sollte es, allerdings in gewissen Grenzen, möglich sein, durch vermehrten Hardware-Einsatz bei gleichbleibendem Transaktionsdurchsatz die mittlere Antwortzeit zu senken. Skalierbarkeit bedeutet vor allem, daß Datenvolumina und Anwendungslasten eines DBVS in weiten Bereichen aus-

schließlich durch Einsatz von HW-Ressourcen anwachsen können, ohne daß seine Leistung beeinträchtigt wird. Skalierbarkeit eines DBVS ist deshalb die Voraussetzung, daß es sich einerseits für Anwendungen unterschiedlichster Größe und Leistungsanforderungen heranziehen läßt und daß andererseits ein DBS mit dem Unternehmen wachsen kann.

1.2 DB-Schemaarchitektur nach ANSI/SPARC

Der Wunsch, ein DBS einzusetzen, um damit Kontrolle über alle Aspekte der Speicherung, Konsistenz und Nutzung der Daten ausüben zu können, setzt planmäßiges Vorgehen und Berücksichtigung vieler Randbedingungen voraus. Integration aller Daten, die vorher oft in verschiedenen Dateien und Dateisystemen nur getrennt zugänglich waren, und Datenunabhängigkeit bei möglichst allen Arten von Nutzungen sind dabei zweifellos Hauptziele.

Integration bedeutet einerseits, daß alle Daten einheitlich modelliert und beschrieben werden müssen, um typübergreifende Operationen und flexible Auswertungen auf den Daten durchführen zu können. Da es sich zunächst um eine gemeinschaftliche Sicht der Daten handeln soll, ist dabei auch Anwendungsneutralität von großer Wichtigkeit. Andererseits dürfen einzelne Benutzer nur auf die Teile der Datenbasis zugreifen, die zur Lösung ihrer Aufgaben erforderlich sind. Dabei sollen individuelle Sichten der benötigten Daten eine angemessene Unterstützung für anwendungsspezifische Problemlösungen liefern, ohne wiederum Abhängigkeiten zwischen Daten und Programmen einzuführen. Weiterhin können Leistungsanforderungen, durch die Mehrfachbenutzung der integrierten Daten bedingt, den Einsatz verschiedenartiger Speicherungsstrukturen für bestimmte Teile der Datenbasis (Repräsentationssicht) diktieren, so daß systemkontrollierte Redundanz für ausgewählte Anwendungen durch die physische Organisation der DB bereitzustellen ist. Auch hier dürfen Speicherungsredundanz und andere leistungsbezogene Maßnahmen die geforderte Datenunabhängigkeit nicht abschwächen. Durch eine geeignete Systemarchitektur und vorgegebene Zugriffsschnittstellen sollen die DB-Nutzer von den Leistungsaspekten des DBS abgeschirmt werden. Auf keinen Fall sind Beeinflussungen der logischen Organisation der integrierten Daten und der Benutzersichten oder gar Funktionsveränderungen zulässig.

1.2.1 Beschreibungsrahmen für DBS-Schnittstellen

Die hier exemplarisch aufgezeigten Abhängigkeiten zwischen Integration der Daten und Datenunabhängigkeit der Anwendungen machen deutlich, daß eine Lösung des Entwurfsproblems durch die Einführung eines Datenmodells alleine nicht zu erreichen ist. Es ist dafür vielmehr eine gesamtheitliche Analyse aller DBS-Aspekte und eine abgestimmte Vorgehensweise erforderlich. Dies wurde jedoch in den siebziger Jahren an der „Front“ der DBS-Entwicklung nicht so deutlich erkannt. Im Expertenstreit ging es ausschließlich um Eigenschaften von Datenmodellen; dabei konkurrierten das Hierarchie-, das Netzwerk- und das Relationenmodell bei der Suche nach einem Beschreibungsmodell für die Datenintegration und einem

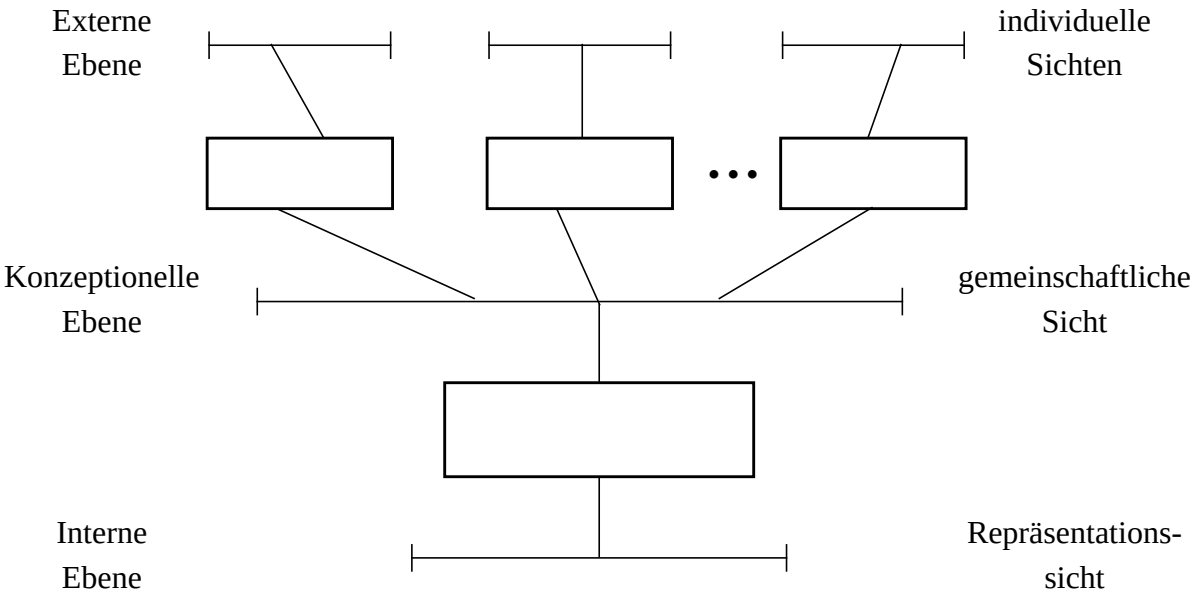


Abb. 1.1: Grobarchitektur für Schnittstellen nach ANSI/SPARC

DB-Verarbeitungsmodell für Anwendungsprogramme. Ohne konkretes Ergebnis wurden heftige wissenschaftliche Debatten um „das beste konzeptionelle Modell“ geführt, was gelegentlich als Religionskrieg apostrophiert wurde.

In einer mehrjährigen Studie hat die „ANSI/X3/SPARC Study Group on Database Management Systems“ (ANSI steht für American National Standards Institute) die Frage nach einer allgemeinen Beschreibungsarchitektur für DB-Funktionen, -Schnittstellen und -Nutzungen untersucht, wobei Datenintegration bei gleichzeitiger Datenunabhängigkeit herausragendes Entwurfsziel war. Ein Überblick zum ANSI/SPARC-Architekturvorschlag findet sich in [TSIC78], wo durch Graphiken und Schnittstellenbeschreibungen dessen Reichweite verdeutlicht wird. Durch die Festlegung von 40 Schnittstellen (unterschiedlicher Wichtigkeit und Komplexität) will der Architekturvorschlag einen Beschreibungsrahmen für alle Probleme und Aufgaben der Modellierung, Realisierung, Verwaltung und Nutzung von DB-Daten definieren. Aus dieser „Schnittstellenarchitektur“ wählen wir nur solche Schnittstellen aus, die für unsere Diskussion der DBS-Realisierung wichtig sind. In Abb. 1.1 ist ein Ausschnitt der Schnittstellenarchitektur skizziert, der oft als ANSI/SPARC-Grobarchitektur bezeichnet wird. Schnittstellen bieten Beschreibungsebenen und damit gewisse Sichten auf die Daten. Zentral ist danach die integrative oder gemeinschaftliche Datensicht, die auch als *Konzeptionelle Ebene* bezeichnet wird. Benutzerspezifische Sichten werden auf der *Externen Ebene* festgelegt, während die *Interne Ebene* bestimmte Aspekte der physischen DB-Organisation beschreibt. Weitere Schnittstellen, welche die Spezifikation der Abbildung und Zuordnung der Daten auf Externspeicher erlauben, werden bei dieser Großcharakterisierung gewöhnlich weggelassen, sind im ANSI/SPARC-Vorschlag jedoch vorgesehen. Zu jeder Beschreibungsebene (Schnittstelle) gehört eine Sprache zur Spezifikation konkreter Objekte; das Ergebnis einer solchen Beschreibung wird, wie im DB-Bereich üblich, als Schema (und nicht als Modell) bezeichnet.

1.2.2 Die wichtigsten DBS-Schnittstellen

In Abb. 1.2 wird der ANSI/SPARC-Vorschlag noch etwas weiter detailliert. Insbesondere soll dadurch der Kern des Vorschlages, die Definition einer DB-Schemaarchitektur, nochmals verdeutlicht werden. Die Aufgaben des DBVS erstrecken sich von der Verwaltung der gespeicherten Datenbank, über die Abbildungen, die durch die verschiedenen Schemata festgelegt werden, bis zu den Arbeitsbereichen der Programme, in denen die gelesenen und die zu schreibenden Datenobjekte abzuholen bzw. abzulegen sind.

Bei der Entwicklung eines Anwendungssystems liefert eine Informationsbedarfsanalyse in der betrachteten Miniwelt die Informationen über Sachverhalte, Beziehungen und Vorgänge, die im DBS durch Daten repräsentiert werden sollen. Der DB-Entwurf zielt zunächst auf eine gemeinschaftliche Sicht der integrierten Daten ab. Die Aufbereitung dieser Daten führt dann zur Definition des *Konzeptionellen Schemas*. Die Festlegung der physischen DB-Struktur erfolgt im *Internen Schema*. Sie ist zu ergänzen durch Angaben der Speicher- und Gerätezuordnung, für deren Spezifikation spezielle Sprachen[3] oder fallweise Dienstprogramme eines konkreten DBS eingesetzt werden.

Aus der gemeinschaftlichen Sicht werden i. allg. eine Reihe verschiedener individuellen Sichten abgeleitet, um Anwendungsprogrammen für ihre Problemlösung angemessene Datenstrukturen bieten zu können. Solche speziellen Benutzersichten sind jeweils durch ein separates *Externes Schema* zu spezifizieren. Mit der benutzerspezifischen Sichtenbildung werden zugleich wichtige Aufgaben, die einfache Nutzung und Zugriffsschutz der Daten betreffen, erfüllt. Durch eine explizite Abbildung (durch Projektion, Selektion und Verbund) ausgehend vom konzeptionellen Schema lassen sich externe Schemata an die Erfordernisse der Anwendung anpassen. Da außerdem nur die im externen Schema spezifizierten Daten für den Benutzer (Programm) sichtbar sind, wird eine Reduktion der Komplexität und eine vereinfachte Verarbeitung erreicht. Zugleich sind alle nicht in der Sicht definierten Datenobjekte dem Benutzer verborgen. Da diese nicht adressiert werden können, ergibt sich automatisch eine starke Isolation dieser Daten, die dem Zugriffsschutz dient. Schließlich wird eine explizite Abbildung der Datentypen (Datentypkonversion) auf die Datentypen der Wirtssprache vorgenommen. Dies ist eine Voraussetzung dafür, daß das DBS zusammen mit Anwendungsprogrammen, die in verschiedenen Programmiersprachen geschrieben sind, eingesetzt werden kann.

Solche mehrsprachenfähige DBS entsprechen einer wichtigen Anforderung aus der Praxis, da in einem Unternehmen nicht davon auszugehen ist, daß über einen langen Zeitraum nur eine einzige Programmiersprache eingesetzt wird. Die explizite Abbildung der Daten durch das externe Schema und ihre räumlich getrennte Verarbeitung im Arbeitsbereich des Benutzers erzielen auch eine Isolationswirkung bei Programmfehlern (oder beabsichtigten Manipulationen) in der Anwendung, die eine Zerstörung oder Korruption der DB-Daten verhindert.

---

[3] Beispielsweise wurde DMCL (Data Media Control Language) als entsprechende Spezifikationssprache für Speicherzuordnungsschemata von ANSI/SPARC vorgeschlagen.

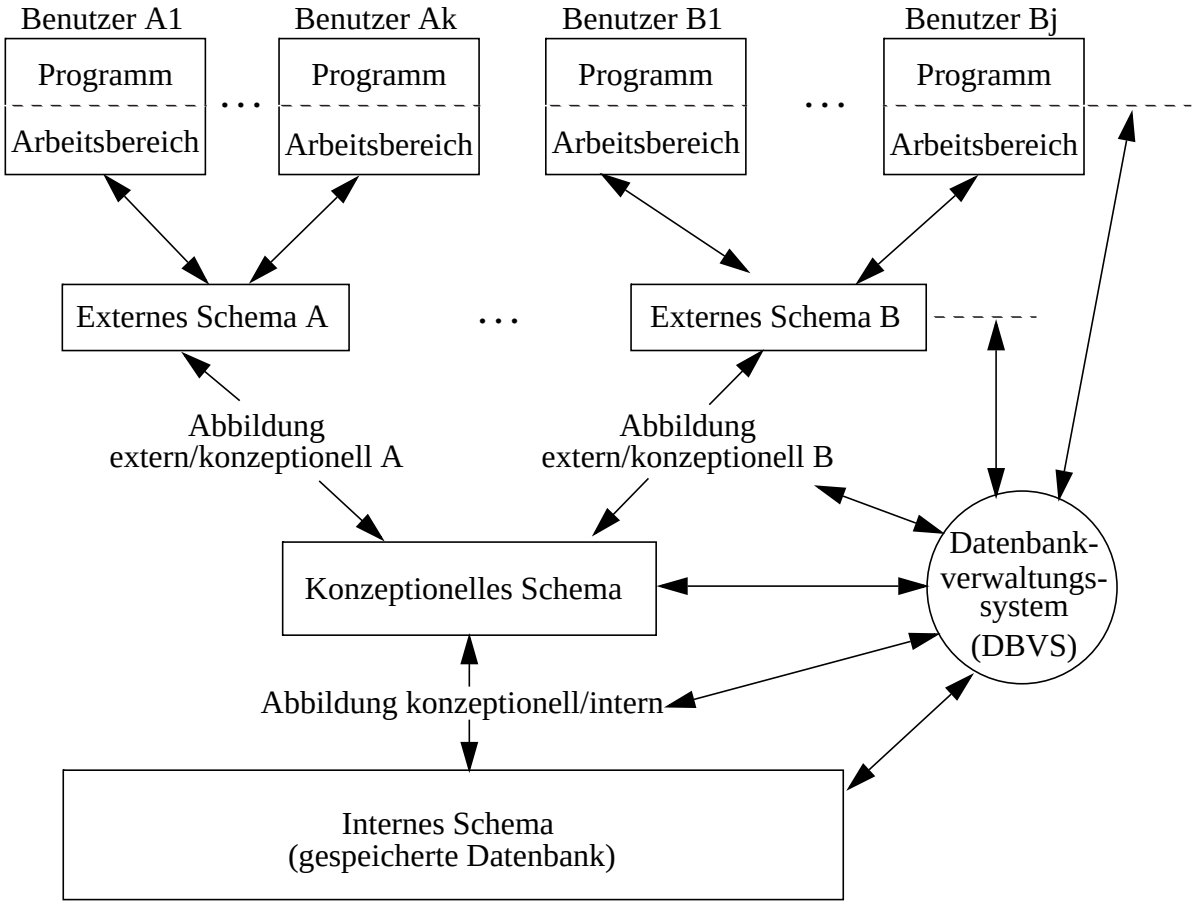


Abb. 1.2: Schnittstellen und ihre Abbildung im ANSI/SPARC-Vorschlag

Alle Schemata müssen vollständig definiert sein, bevor mit ihnen und einem generischen Datenbankverwaltungssystem (DBVS) ein konkretes DBS erzeugt werden kann. Im Mittelpunkt unserer Betrachtungen steht nachfolgend der Aufbau und die Realisierung solcher generischer DBVS, für die wir zunächst geeignete Schichtenarchitekturen einführen. Im Gegensatz zur ANSI/SPARC-Architektur, die im wesentlichen Schnittstellen beschreibt, dienen die nachfolgend eingeführten Schichtenmodelle zur Beschreibung und Erklärung der Realisierung von DBVS.

1.3 Schichtenmodell-Architektur

1.3.1 Anforderungen an den Systementwurf

Obwohl das Entwicklungsziel eines DBS durch Datenbankmodell und externe Benutzerschnittstellen klar vorgegeben ist, ist seine Realisierung eine vielschichtige Aufgabe, da es eine Reihe von zusätzlichen nichtfunktionalen Anforderungen wie Leistungsverhalten, Datenunabhängigkeit, Zuverlässigkeit, Robustheit u. a. erfüllen muß, um seine Praxistauglichkeit

nachweisen zu können. Über die generelle Vorgehensweise beim Entwurf großer Software-Systeme und insbesondere über die Anwendungen bewährter Prinzipien des Software-Engineering herrscht in der Fachwelt weitgehend Übereinstimmung. Zur Reduktion der Komplexität sollte der Systementwurf auf eine durch Schichten von Abstraktionen[4] gekennzeichnete Architektur mit klar definierten Spezifikationen führen, die Voraussetzung für wesentliche Systemeigenschaften wie Modularität, Anpaßbarkeit, Erweiterbarkeit und Portabilität ist. Diese Eigenschaften werden weiter gefördert, wenn bei der Realisierung der einzelnen Schichten strukturierte Methoden des Programmentwurfs und Techniken der Kapselung von Datenstrukturen angewendet werden.

Neben diesen allgemeinen Entwurfseigenschaften sind bei der DBS-Realisierung, wie erwähnt, eine Reihe zusätzlicher Anforderungen einzuhalten, die eine effiziente und zuverlässige Systemnutzung gewährleisten sollen. Bei diesen Nebenbedingungen wollen wir zwei besonders betonen. Damit ein DBS im praktischen Einsatz akzeptiert wird, muß es die angebotenen Operationen ausreichend effizient ausführen können.[5] Dazu hat es einen Vorrat an geeigneten Speicherungsstrukturen und Zugriffspfaden sowie spezielle, darauf zugeschnittene Verfahren der Anfrageoptimierung bereitzuhalten. Neben dem Leistungsaspekt gilt ein hoher Grad an Datenunabhängigkeit als wichtigste Nebenbedingung des Systementwurfs. Datenunabhängigkeit soll einerseits eine möglichst große Isolation von Anwendungsprogramm und DBS gewährleisten und andererseits auch innerhalb des DBS eine möglichst starke Kapselung der einzelnen Komponenten bewerkstelligen. Datenunabhängigkeit muß deshalb durch eine geeignete DBS-Architektur unterstützt werden.

Da beim Einsatz von DBS bei der Vielfalt kommerzieller Anwendungen eine große Variationsbreite und Änderungswahrscheinlichkeit in der Darstellung und Menge der gespeicherten Daten, in der Häufigkeit und Art der Zugriffe sowie in der Verwendung von Speicherungsstrukturen und Gerätetypen zu erwarten ist, muß beim Systementwurf Vorsorge dafür getroffen werden. Erweiterbarkeit und Anpaßbarkeit sind beispielsweise nur zu erreichen, wenn die Auswirkungen von später einzubringenden Änderungen und Ergänzungen in der DBS-Software lokal begrenzt werden können. Diese Forderungen lassen sich am besten durch eine DBS-Architektur realisieren, die sich durch mehrere hierarchisch angeordnete Systemschichten auszeichnet. Aus diesem Grund verwenden wir hier als methodischen Ansatz ein hierarchisches Schichtenmodell zur Beschreibung des Systemaufbaus. Durch die einzelnen Abbildungen (Schichten) werden die wesentlichen Abstraktionsschritte von der Externspeicherebene bis zur Benutzerschnittstelle charakterisiert, die das DBS dynamisch vorzunehmen hat, um aus einer auf Magnetplatten gespeicherten Bitfolge abstrakte Objekte an der Benutzerschnittstelle abzuleiten.

---

[4] „Eine Hauptaufgabe der Informatik ist systematische Abstraktion“ (H. Wedekind).

[5] Denn für DBS und ihre Anwendungen gilt folgender populärer Spruch in besonderer Weise: „Leistung ist nicht alles, aber ohne Leistung ist alles nichts“.

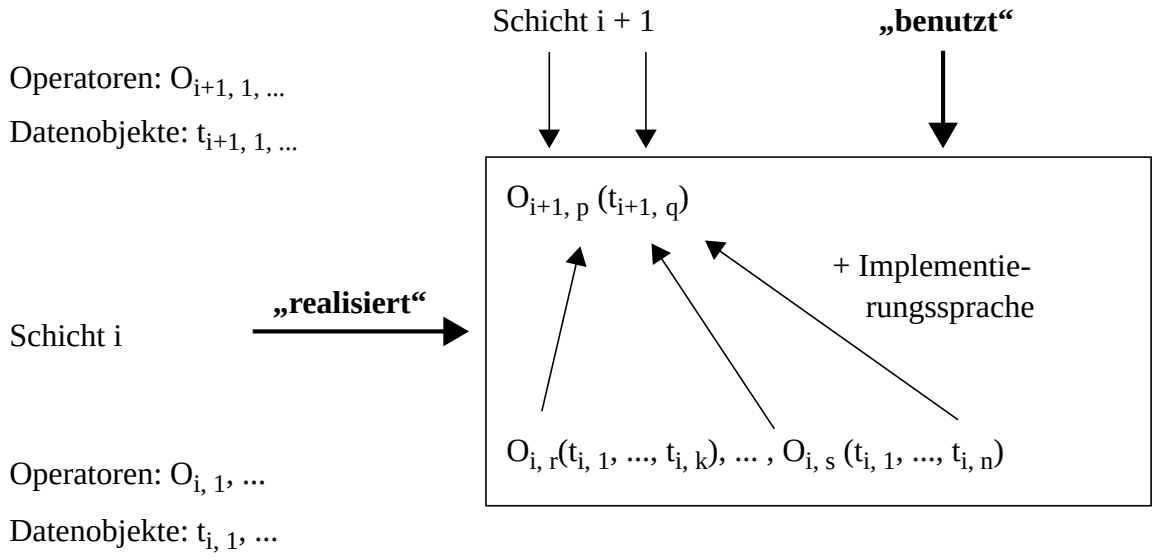


Abb. 1.3: Aufbauprinzip für eine Schicht

1.3.2 Architekturprinzipien

Ziel unserer Überlegungen ist die Entwicklung einer Systemarchitektur für ein datenunabhängiges DBS. Da es keine Architekturlehre für den Aufbau großer Software-Systeme gibt, können wir keine konkreten Strukturierungsvorschläge heranziehen. Es existieren aus dem Bereich Software Engineering lediglich Empfehlungen, allgemeine Konzepte wie das Geheimnisprinzip (*information hiding* nach Parnas [PARN72]) und eine hierarchische Strukturierung [PARN75] zu nutzen. Daraus lassen sich wichtige Hinweise ableiten, große SW-Systeme aus hierarchisch angeordneten Schichten aufzubauen, wobei Schicht i+1 die Operatoren und Datenobjekte „benutzt“[6], die Schicht i „realisiert“. Dieses Aufbauprinzip ist in Abb. 1.3 veranschaulicht.

Nach Parnas ergeben sich unmittelbar eine Reihe von Vorteilen für die Entwicklung des SW-Systems, die als Konsequenzen der Nutzung hierarchischer Strukturen und durch die Benutzt-Relation erzielte Kapselung angesehen werden können:

- Höhere Ebenen (Systemkomponenten) werden einfacher, weil sie tiefere Ebenen (Systemkomponenten) benutzen können.
- Änderungen auf höheren Ebenen sind ohne Einfluß auf tieferen Ebenen.
- Höhere Ebenen lassen sich abtrennen, tiefere Ebenen bleiben trotzdem funktionsfähig.
- Tiefere Ebenen können getestet werden, bevor die höheren Ebenen lauffähig sind.

Weiterhin läßt sich jede Hierarchieebene als abstrakte oder virtuelle Maschine auffassen. Solche Abstraktionsebenen erlauben, daß

---

[6] Definition der Benutzt-Relation nach [PARN72]: A benutzt B, wenn A B aufruft und die korrekte Ausführung von B für die vollständige Ausführung von A notwendig ist.

- Programme (Module) der Schicht i als abstrakte Maschine die Programme der Schicht i-1, die als Basismaschine dienen, benutzen und
- die abstrakte Maschine der Schicht i wiederum als Basismaschine für die Implementierung der abstrakten Maschine der Schicht i+1 dient.

Eine abstrakte Maschine entsteht aus der Basismaschine durch Abstraktion. Dabei werden einige Eigenschaften der Basismaschine verborgen. Zusätzliche Fähigkeiten werden durch Implementierung höherer Operationen für die abstrakte Maschine bereitgestellt.

Die eingeführten Prinzipien beantworten noch nicht die Frage nach der Anzahl n der Schichten, die eine „optimale“ DBS-Architektur aufweisen sollte. Offensichtlich ist die Wahl von n = 1 nicht geeignet, die oben eingeführten Anforderungen zu erfüllen, da die resultierende monolithische Systemstruktur keine Reduktion der Systemkomplexität erzielt (fehlende Aufteilung) und auch keine Kapselung von Aufgaben und Abbildungsvorgängen im DBS-Code erzwingt. Zur Diskussion einer geeigneteren Architektur wollen wir zunächst die Rolle von n, d. h. der Schichtenanzahl, ausloten. Zwei divergierende Einflußfaktoren lassen sich für n > 1 (bis zu einer vernünftigen Obergrenze bis etwa n < 10) ausmachen. Zunächst führt ein größeres n auf eine Reduktion der Komplexität der einzelnen Schichten. Schritthaltend damit werden wegen der Kapselung die Auswirkungen von Systemänderungen und -ergänzungen eingegrenzt. Also wird damit eine Evolution der Systemfunktionalität oder ihre Anpassung an Umgebungsänderungen einfacher und weniger fehleranfällig. Dagegen ist bei steigender Anzahl der Schnittstellen vor allem mit Leistungseinbußen zu rechnen. Die unterstellte strikte Kapselungseigenschaften der Schichten erzwingt bei jedem Schnittstellenübergang eine Parameterprüfung der Operationsaufrufe und einen Datentransport von Schicht zu Schicht mit einer schichtenspezifischen Konversion der Datentypen und Übertragung der angeforderten Granulate. Das impliziert Kopiervorgänge beim Lesen (nach oben) und Propagieren von Änderungen (nach unten). Zudem wird die nichtlokale Fehlerbehandlung schwieriger, da Fehlermeldungen von Schicht zu Schicht (nach oben) gereicht und dabei „schichtenspezifisch erklärt“ werden müssen. Wenn nur innerhalb einer Schicht Annahmen über die Operationsausführung getroffen werden können, werden offensichtlich mit der Verkleinerung der Softwareschicht (Abbildung) auch die Optimierungsmaßnahmen und ihre Reichweiten reduziert. Mit diesen Überlegungen als Entwurfsinformation führen wir nun zuerst ein Drei-Schichten-Modell für den statischen DBS-Aufbau ein. Dieses wird später zur besseren Erklärung und Separierung der Realisierungskonzepte und -techniken zu einem Fünf-Schichten-Modell erweitert. Um die oben erwähnte Problematik der Rolle von n zu entschärfen, wird anschließend gezeigt, wie zur Laufzeit, also bei der Abwicklung von DB-Operationen, einige Schichten „wegoptimiert“ werden können, um so ein besseres Leistungsverhalten zu gewährleisten.

1.3.3 Ein einfaches Schichtenmodell

Das mehrstufige Schichtenmodell, das von uns als allgemeiner Beschreibungsrahmen für die DBS-Architektur herangezogen wird, begünstigt eine saubere hierarchische Systemstrukturierung. Wie eben erörtert, hat eine solche DBS-Architektur weitreichende Folgen für Komplexi-

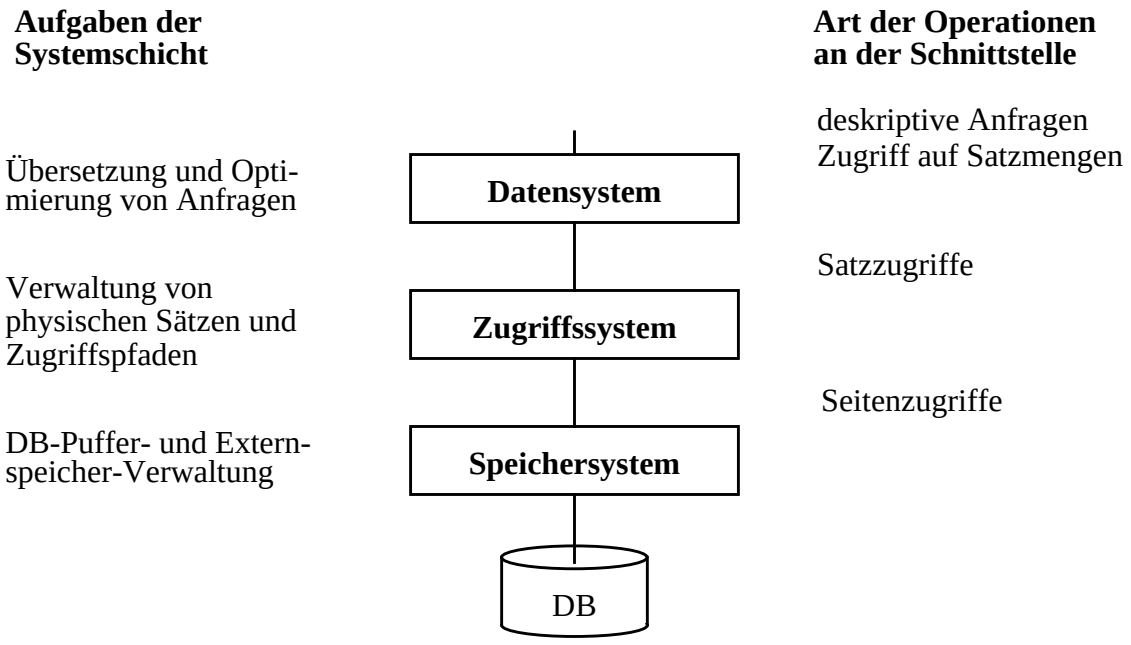


Abb. 1.4: Vereinfachtes Schichtenmodell

tät und Leistungsverhalten des zu realisierenden Systems. Es ist jedoch nicht trivial, eine gute Schichtenbildung zu finden.[7] Ganz allgemein sollten dabei drei wichtige Kriterien erfüllt werden:

- „günstige“ Zerlegung des DBS in „nicht beliebig viele“ Schichten,
- optimale Bedienung der darüberliegenden Schicht mit ihren Aufgaben,
- implementierungsunabhängige und möglichst allgemeine Beschreibung der Funktionen jeder Schnittstelle (Ebene).

Für die Zerlegung und für die Wahl der Objekte/Operatoren läßt sich kein Algorithmus angeben. Deshalb muß sich der Entwurf des Schichtenmodells in der Regel auf „Erfahrung“ abstützen.

Ein vereinfachtes Schichtenmodell soll zunächst zur Beschreibung der Datenabbildungen von der mengenorientierten DB-Schnittstelle bis zur Externspeicherrepräsentation dienen. Es wurde schon in [ASTR76] zur Darstellung der Architektur eines relationalen DBS in ähnlicher Weise eingeführt. Abb. 1.4 veranschaulicht die Aufgaben der jeweiligen Systemschicht und den Abstraktionsgewinn, den die Operationen auf den jeweiligen Schichten nutzen können. Durch explizite Separierung des Speichersystems heben wir in diesem Modell die Aufgaben der Extern- und Hauptspeicherverwaltung (in Form des DB-Puffers) hervor. So stellt das Speichersystem die Zugriffseinheit „Seite“ im DB-Puffer zur Verfügung. Es verbirgt auf diese

[7] „Die durch Abstraktion entstandenen Konstrukte der Informatik als Bedingungen möglicher Information sind zugleich die Bedingungen der möglichen Gegenstände der Information in den Anwendungen“ (H. Wedekind in Anlehnung an eine Aussage Kants aus der „Kritik der reinen Vernunft“). Vereinfacht ausgedrückt: Informatiker erfinden (konstruieren) abstrakte Konzepte; diese ermöglichen (oder begrenzen) wiederum die spezifischen Anwendungen.

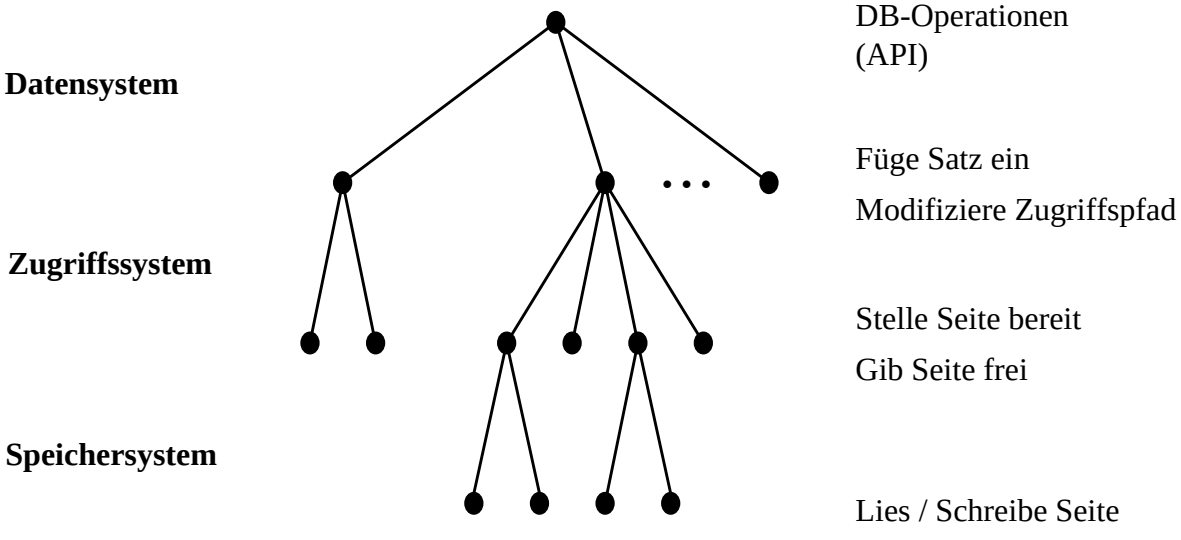


Abb. 1.5: Dynamischer Kontrollfluß durch das Schichtenmodell - Beispiel

Weise alle Aspekte der Externspeicherabbildung und -anbindung und bietet somit für höhere Systemschichten die Illusion einer seitenstrukturierten Hauptspeicher-DB. Die Abbildung von Sätzen und Zugriffspfaden auf Seiten erfolgt durch das Zugriffssystem, wobei nach oben ein satzweiser Zugriff auf die physischen Objekte und Speicherungsstrukturen (interne Sätze) angeboten wird. Das Datensystem schließlich überbrückt die Kluft von der mengenorientierten Tupelschnittstelle (für Relationen und Sichten) zur satzweisen Verarbeitung interner Sätze. Dabei ist die Übersetzung und Optimierung von deskriptiven Anfragen eine der Hauptaufgaben dieser Schicht.

In Abb. 1.5 ist der dynamische Kontrollfluß durch das Schichtenmodell skizziert, um das Zusammenspiel der einzelnen Schichten etwas zu beleuchten. Eine mengenorientierte DB-Operation wird im Datensystem mit Hilfe von satzorientierten Operationen des Zugriffssystems abgewickelt. Manche dieser Operationen können auf bereits im DB-Puffer vorhandenen Seiten ausgeführt werden. Andere Operationen erfordern dagegen das Nachladen (Lesen) oder Ausschreiben von Seiten, was den Aufruf des Speichersystems impliziert. Bei einem sequentiellen DBS kann man sich die Abwicklung des dynamischen Ablaufs nach dem Prinzip der Tiefensuche (*left-most, depth-first*) vorstellen. Die Ausführung mengenorientierter DB-Operationen erzeugt in vielen Fällen sehr breite Kontrollflußgraphen (oder Mehrwegbäume mit einem großen Verzweigungsgrad, ggf. in allen drei Systemschichten). Deshalb soll Abb. 1.5 auch die Möglichkeit verdeutlichen, daß mengenorientierte DB-Operationen sich neben der Anfrageoptimierung auch zur Parallelisierung eignen. Im Beispiel bedeutet dies, daß bei geeigneten Systemvoraussetzungen mehrere Aufrufe des Zugriffssystems und dort möglicherweise mehrere Aufrufe des Speichersystems parallel abgesetzt und ausgeführt werden können.

1.3.4 Integration von Metadaten- und Transaktionsverwaltung

DBVS sind von ihrem Aufbau und ihrer Einsatzorientierung her in hohem Maße generische Systeme. Sie sind so entworfen, daß sie flexibel durch Parameterwahl und ggf. durch Einbindung spezieller Komponenten für eine vorgegebene Anwendungsumgebung zu konfigurieren sind. Das resultierende DBVS zusammen mit seiner konkreten Datenbasis wird zur besseren Unterscheidung vom generischen System auch als DBS bezeichnet. In diesem Buch diskutieren wir in der Regel nur die generischen Aspekte im Rahmen unserer Schichtenarchitektur und bezeichnen das generische Laufzeitsystem zur Datenbankverwaltung verkürzt mit DBS.

Die Datenstrukturen und Operatoren unseres Schichtenmodells sind also generisch, d. h., an jeder Schnittstelle sind nur bestimmte Objekttypen und ihre charakteristischen Operationen vorgegeben, jedoch nicht ihre anwendungsbezogene Spezifikation und Semantik (z. B. durch Angabe spezieller Einschränkungen). Beispielsweise bietet unser Schichtenmodell (Abb. 1.4) an der Schnittstelle des Datensystems nur generische „Relationen“ (Satzmengen) mit ihren Operationen. Ihre Anzahl oder ihre konkrete Anwendungsanbindung mit Angabe von Wertebereichen, Attributen, Integritätsbedingungen usw. ist jedoch nicht festgelegt. Ebenso wird die Externspeicherabbildung durch ein generisches Dateikonzept vorgenommen. Die Beschreibung der konkret benötigten Dateien, ihre Anzahl und ihre Verteilung auf Speichermedien sowie andere anwendungsspezifische Parameter sind nicht berücksichtigt.

Diese Beispiele sollen zeigen, daß in jeder Schicht anwendungsbezogene Beschreibungsinformation benötigt wird, um aus dem generischen System ein DBS zu konfigurieren. Diese Beschreibungsinformation wird beim DB-Entwurf gewonnen und z. B. nach dem ANSI/SPARC-Ansatz systematisch geordnet und vervollständigt. Das DB-Schema (mit seiner Unterteilung in externes, konzeptionelles und internes Schema) dokumentiert also den DB-Entwurf und enthält die zugehörige Beschreibungsinformation für ein konkretes DBS in lesbarer Form. Bei der Konfigurierung des DBS muß sie dem System zugänglich gemacht werden, um aus den generischen Objekten konkrete Objekte mit ihren Operationen ableiten zu können. Diese dem DBS zugänglichen Beschreibungsinformationen heißen auch Metadaten. Sie werden zur Laufzeit eines DBS gelesen, interpretiert und ständig aktualisiert. Während der Lebenszeit eines DBS fallen häufig auch sog. Schemaevolutionen an, die Änderungen von Struktur und Beschreibung der Miniwelt im DBS nachvollziehen. Hierbei sind persistente und konsistente Änderungen der Metadaten zu gewährleisten. Deshalb wird eine eigene Metadatenverwaltung bereitgestellt.

Da dafür im Prinzip alle Aufgaben der Datenverwaltung (eine DB zur Beschreibung der DB) durchzuführen sind, entspricht ihre Funktionalität der des eigentlichen DBS. Häufig werden deshalb solche Systeme als eigenständige Systeme entwickelt und am Mark angeboten (Data Dictionary, Data Repository u. a.). In unserer Vorstellung wollen wir jedoch die entsprechende Funktionalität im Rahmen unseres Schichtenmodells beschreiben und integrieren, da dazu prinzipiell dieselben Verfahren und Implementierungstechniken wie für die Verwaltung der eigentlichen DB-Daten erforderlich sind.

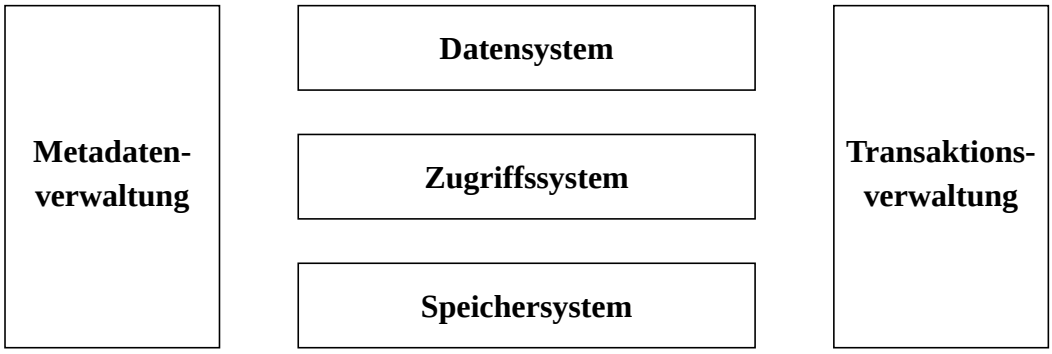


Abb. 1.6: Grobarchitektur eines DBS

Beschreibungsdaten werden zur Realisierung der Aufgaben in jeder Schicht benötigt. Um dies zu unterstreichen, illustrieren wir in Abb. 1.6 die Metadatenverwaltung als konzeptionell eigenständige Komponente, welche die Abstraktionsebenen aller unserer Modellschichten überdeckt. Das soll nicht bedeuten, daß sie immer auch als eigenständige Komponente realisiert ist. Später werden wir die Beschreibungs- und Abbildungsinformationen wiederum den einzelnen Schichten zuordnen und im Rahmen der Konkretisierung der jeweiligen Schicht beschreiben (siehe Abb. 1.7).

Eine weitere wichtige Aufgabe eines DBS ist die Bereitstellung eines Transaktionskonzeptes (siehe Kapitel 13), was durch die Komponente der Transaktionsverwaltung übernommen wird. Auch hier gibt es verschiedene Möglichkeiten der Realisierung und der Schichtenzuordnung, die erst später detailliert werden können. In Abb. 1.6, in dem nur grobe Zusammenhänge veranschaulicht werden, stellen wir die Transaktionsverwaltung deshalb ebenfalls als schichtenübergreifende Komponente dar.

1.3.5 Schichtenmodell eines datenunabhängigen DBS

Nach der Illustration der Schichtenbildung anhand eines vereinfachten Modells wollen wir nun zur besseren Erklärung der Konzepte und Verfahren ein verfeinertes Schichtenmodell zur Implementierung eines datenunabhängigen DBS [HÄRD83a] einführen, dessen fünf Schichten in Abb. 1.7 gezeigt sind. Im Vergleich zum vereinfachten Schichtenmodell in Abb. 1.4 wurden sowohl das Datensystem als auch das Speichersystem aufgespalten und jeweils durch zwei Schichten dargestellt.

Jede Systemschicht realisiert die entsprechenden Strukturen der zugeordneten Entwurfsebene. Die Schnittstellen zwischen den einzelnen Schichten werden jeweils durch einige typische Operationen charakterisiert. Die zugehörigen Objekte sind beispielhaft als Adressierungseinheiten für jede Abbildungsschicht dargestellt, wobei die Objekte an der oberen Schnittstelle den Objekten der unteren Schnittstelle der nächsthöheren Schicht entsprechen. Die an einer Schnittstelle verfügbaren Objekte und Operationen werden von den direkt übergeordneten Komponenten wiederum zur Realisierung ihrer spezifischen Strukturen und Funk-

tionen benutzt. Daneben sind noch einige wichtige Hilfsstrukturen und Beschreibungsdaten, die zur Realisierung der Schicht herangezogen werden, angegeben. Das Geheimnisprinzip verlangt, daß diese Hilfsstrukturen an den jeweiligen Schnittstellen nicht sichtbar sind. So lassen sich Änderungen von Implementierungstechniken und (bis zu einem gewissen Grad) ihre Ergänzungen oder Erweiterungen „nach außen“ verbergen. Das Geheimnisprinzip ist somit Voraussetzung für die geforderte Anpaßbarkeit und die Erweiterbarkeit des DBS.

Die *Geräteschnittstelle* ist durch die verwendeten externen Speichermedien vorgegeben. Durch die Systemschicht, welche die Speicherzuordnungsstrukturen verkörpert, wird eine *Dateischnittstelle* erzeugt, auf der von Gerätecharakteristika wie Speichertyp, Zylinder- und Spuranzahl, Spurlänge usw. abstrahiert werden kann. Diese explizite Abbildung erzielt eine Trennung von Block und Slot sowie von Datei und Speichermedium.

Die nächste Systemschicht – die Ebene der sog. Seitenzuordnungsstrukturen – realisiert die *DB-Pufferschnittstelle* und stellt Segmente mit sichtbaren Seitengrenzen als lineare Adreßräume im DB-Puffer zur Verfügung. Dadurch erfolgt eine konzeptionelle Trennung von Segment und Datei sowie Seite und Block. Als Folge davon lassen sich, für höhere Systemschichten verborgen, verschiedenartige Einbringstrategien für geänderte Seiten einführen, die vor allem die Recovery-Funktionen vereinfachen.

Die nächste Systemschicht realisiert eine Menge von Speicherungsstrukturen wie interne Sätze und physische Zugriffspfade, die sie auf Seiten von Segmenten abbildet. Durch ein reichhaltiges Angebot von verschiedenartigen Zugriffspfaden und Speicherungsoptionen, mit denen das Systemverhalten auf die Leistungsanforderungen der konkreten Anwendungen hin optimiert werden kann, trägt diese Systemschicht in besonderer Weise zur Einhaltung von Performance-Zielen bei. Sie implementiert die *interne Satzschnittstelle*, deren Zweck die Trennung von Sätzen, Einträgen in Zugriffspfaden usw. und Seiten sowie ihrer Zuordnung zu Segmenten ist. Weiterhin erlaubt sie eine Abstraktion von Implementierungsdetails der Sätze und Zugriffspfade.

Die Ebene der Logischen Zugriffspfade stellt eine *satzorientierte DB-Schnittstelle* zur Verfügung, die eine Bezugnahme auf externe Sätze und auf Funktionen von Zugriffspfaden gestattet. Sie verbirgt die gewählten Implementierungskonzepte für Sätze und Zugriffspfade und erzielt damit eine Unabhängigkeit von den Speicherungsstrukturen. Ihre Funktionsmächtigkeit entspricht der eines zugriffspfadbezogenen Datenmodells. Die satzorientierte DB-Schnittstelle dient in DBS nach dem Hierarchie- oder Netzwerkmodell sowie in objektorientierten DBS (OODBS) als Anwendungsprogrammierschnittstelle. Auch aus diesem Grund wird sie in unserem Schichtenmodell explizit ausgewiesen.

Als oberste Schicht der Abbildungshierarchie wird durch die Ebene der Logischen Datenstrukturen eine *mengenorientierte DB-Schnittstelle* (zugriffspfadunabhängiges Datenbankmodell) realisiert, die Zugriffsmöglichkeiten in deskriptiven Sprachen bietet. Der Benutzer kommt auf ihr ohne Navigieren auf logischen Zugriffspfaden aus. Neben der Zugriffspfadunabhängigkeit läßt sich auf dieser Schicht mit Hilfe des Sicht-Konzeptes [CHAM80] ein gewisser Grad an logischer Datenstrukturunabhängigkeit erreichen. Ein wichtiges Beispiel für die mengenorientierte DB-Schnittstelle verkörpert das Relationenmodell mit der Sprache SQL.

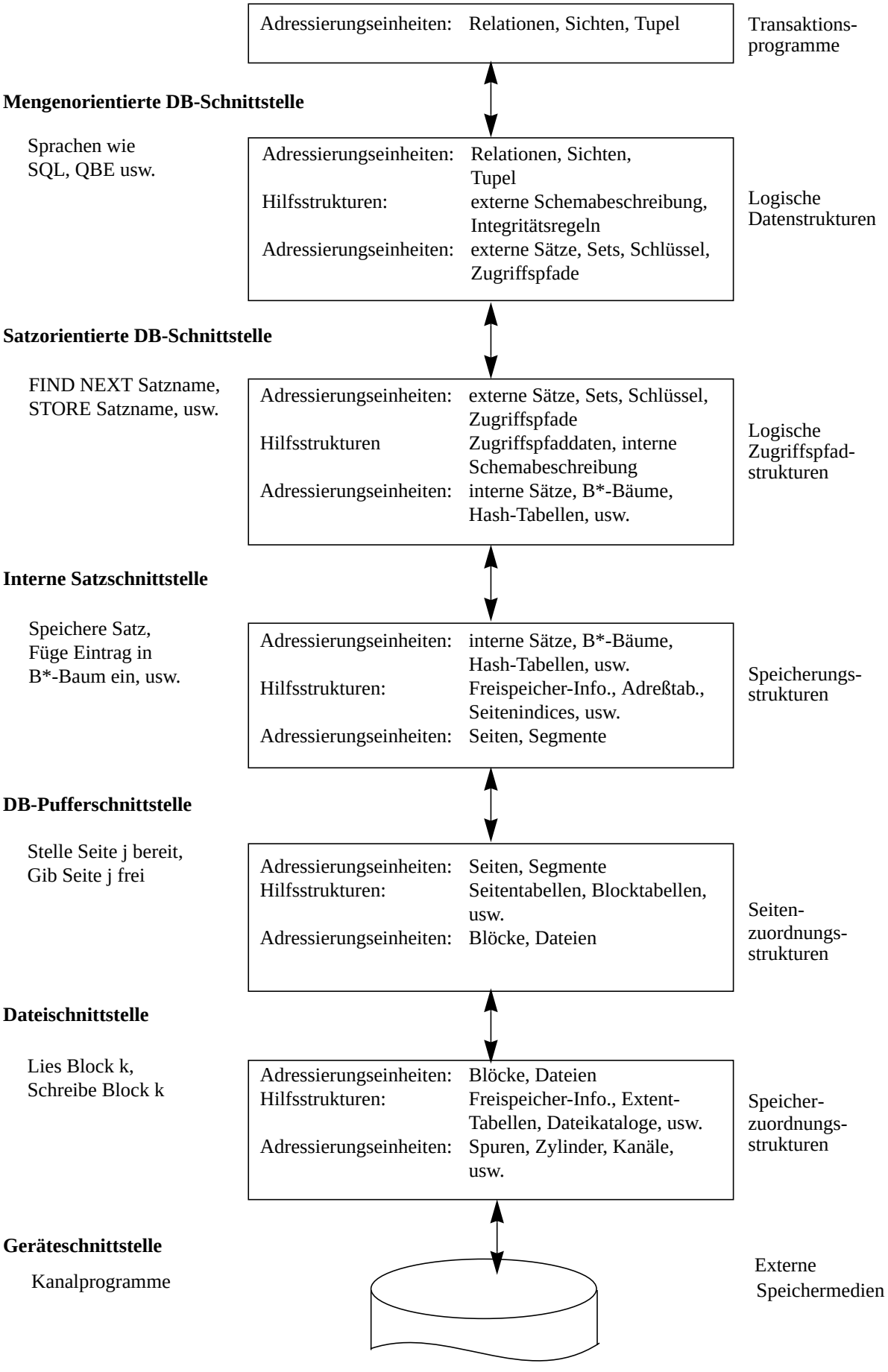


Abb. 1.7: Schichtenmodell für ein datenunabhängiges Datenbanksystem

Während die Systematik der Schichtenbildung sich an den Notwendigkeiten einer geeigneten Objektabbildung orientiert und dadurch in ihrer vorliegenden Form festgelegt ist, ergeben sich für die Einordnung der Datensicherungs- und Recovery-Funktionen eine Reihe von Freiheitsgraden. Zugriffs- und Integritätskontrolle sind jeweils an die an der externen Benutzerschnittstelle (Programmierschnittstelle) sichtbaren Objekte und Operationen gebunden. Deshalb sind die zugehörigen Maßnahmen in der entsprechenden Abbildungsschicht (also entweder in der Schicht der Logischen Datenstrukturen oder in der Schicht der Logischen Zugriffspfade) zu bewerkstelligen. Auch die Transaktionsverwaltung ist einer Schicht zuzuordnen, in der die Zusammengehörigkeit von externen Operationsfolgen eines Benutzer noch erkannt und kontrolliert werden kann. Wie in Kapitel 14 und 15 ausführlich gezeigt wird, können die Synchronisationsverfahren und die Recovery-Funktionen in verschiedenen Abbildungsschichten mit unterschiedlicher Effizienz angesiedelt werden.

1.3.6 Optimierungsüberlegungen

Wir betrachten unser Schichtenmodell, das als statisches Modell die schrittweise Realisierung immer komplexerer DB-Objekte zu beschreiben gestattet, eher als ein Erklärungsmodell, bei dem die strikte Isolation der einzelnen Abbildungsschichten gewahrt wird. Es wurde bereits erörtert, daß bei strikter Schichtenkapselung die Überquerung einer Schnittstelle gewisse Kosten verursacht. Deshalb ist es zur Laufzeitoptimierung wichtig, die Anzahl der Schnittstellenüberquerungen und insbesondere die dabei erforderlichen Kopiervorgänge der Daten zu minimieren.

In den unteren beiden Schichten bietet ein großer DB-Puffer mit effektiven Verwaltungsalgorithmen die wirksamste Maßnahme zur Laufzeitoptimierung, da so erreicht werden kann, daß nur für einen Bruchteil der logischen Seitenreferenzen tatsächlich physische E/A-Vorgänge erforderlich werden. Außerdem kann man bei der Abbildung von Seiten auf Blöcke (bei gleicher Granulatgröße) sehr effizient verfahren und insbesondere explizite Zwischenkopien der Daten einsparen. Bei den oberen Systemschichten lassen sich erhebliche Einsparungen erzielen, wenn die Schicht der Logischen Datenstrukturen nicht als Interpreter für SQL-Anweisungen fungieren muß, sondern zur Laufzeit durch sog. Zugriffsmodule (pro SQL-Anweisung oder Anwendungsprogramm) ersetzt wird, wobei bereits eine frühe Bindung der Zugriffsoperationen an interne Schnittstellen erfolgt (siehe Kapitel 12). Auf diese Weise läßt sich die oberste Schicht (oder gar die beiden obersten Schichten) „wegoptimieren“, d. h., Anweisungen der mengenorientierten DB-Schnittstelle werden direkt auf die satzorientierte DB-Schnittstelle (oder gar die interne Satzschnittstelle) abgebildet und mit Hilfe von generierten Zugriffsmodulen als Operationsfolgen auf dieser Schnittstelle ausgeführt.

Zusätzlich ist es in einer konkreten Implementierung häufig erforderlich, die Reichweite von Optimierungsmaßnahmen auszudehnen oder globale Information zentralisiert zur Verfügung zu stellen. Als besonders wirksam hat es sich in diesem Zusammenhang erwiesen, Informationsflüsse selektiv auch über mehrere Schichten hinweg zu erlauben, in erster Linie, um eine globale Optimierung von leistungskritischen Verfahren durchführen zu können. Ein Musterbeispiel dafür ist die Kooperation von Anfrageoptimierung und DB-Pufferverwaltung.

Wenn beispielsweise nach der Übersetzung und Optimierung einer SQL-Anfrage in Schicht 5 bekannt ist, welcher Operator auf welche Daten (wiederholt) zugreift, kann die DB-Pufferverwaltung in Schicht 2 mit dieser Information eine gezielte Speicherplatzallokation vornehmen und den Auswertungsaufwand ggf. durch Prefetching noch weiter reduzieren. Auch bei der zur Lastkontrolle erforderliche Abstimmung mehrerer Komponenten sowie bei bestimmten Synchronisations- und Recovery-Funktionen ist häufig eine schichtenübergreifende Kooperation angezeigt, um effiziente Protokolle anbieten zu können.

1.4 Erweiterungen der DBS-Architektur

Das Schichtenmodell beschreibt im Detail die erforderlichen Schritte der Datenabbildung von mengen- oder satzorientierten DB-Schnittstellen bis zur persistenten Speicherung der Daten auf Externspeicher. Es läßt sich deshalb nicht nur für zentralisierte Datenbanksysteme nach dem Hierarchie-, dem Netzwerk- oder dem Relationenmodell heranziehen[8], sondern es kann auch zu Beschreibung komplexerer Datenhaltungssysteme dienen, seien sie nun zentralisiert oder verteilt. Unser bisher eingeführtes Schichtenmodell verkörpert dann das Kernsystem, das allgemeine und anwendungsneutrale DBS-Funktionen über satz- oder mengenorientierte Schnittstellen anbietet, die wiederum in speziellen Subsystemen und Komponenten für zugeschnittene Dienste oder in zusätzlichen Schichten zur Abbildung anwendungsspezifischer Objekte und Operationen genutzt werden können.

1.4.1 DBS-Kern-Architektur

In Anwendungsbereichen, die beispielsweise eine Unterstützung der Entwurfs-, Planungs- und Verwaltungsarbeit durch DBS erfordern und oft als Non-Standard-Anwendungen (z. B. CAD, CAM oder VLSI-Entwurf) bezeichnet werden, reichen die Modellierungs- und Verarbeitungskonzepte der klassischen Datenmodelle nicht mehr aus, um darauf aufbauend unmittelbar angemessene Problemlösungen zur Verfügung stellen zu können. Deshalb wurden schon Anfang der achtziger Jahre verbesserte Modellierungs- und Verarbeitungskonzepte (für sog. komplexe Objekte) als DBS-Angebot verlangt.

Wegen der komplexen Datenstrukturen und Beziehungen sowie der zugeschnittenen Operationen kann keine allgemeingültige Lösung im Sinne unserer mengenorientierten DB-Schnittstelle (Datenmodellschnittstelle in Abb. 1.8) erwartet werden. Deshalb wurden erweiterte DBS-Architekturen entwickelt, bei denen die gewünschte anwendungsspezifische DBS-Funktionalität durch Zusatzschichten bereitgestellt wurde [HÄRD87a, PAUL87]. Wegen der starken Differenzierung dieser Funktionalität mußte pro Anwendungsbereich eine separate Zusatzschicht (Modellabbildung) entwickelt werden, deren Dienste dem Benutzer an der An-

[8] Eine Variation dieses Schichtenmodells läßt sich auch zur Realisierung eines datenmodellunabhängigen DBS-Kern-Systems heranziehen. In [ARUN98] wird eine Kern-Architektur beschrieben, die gleichzeitig zwei verschiedene Datenmodelle unterstützt und mit relationalen (Oracle/Rdb) und netzwerkartigen DB-Schnittstellen (Oracle CODASYL DBMS) ausgestattet wurde.

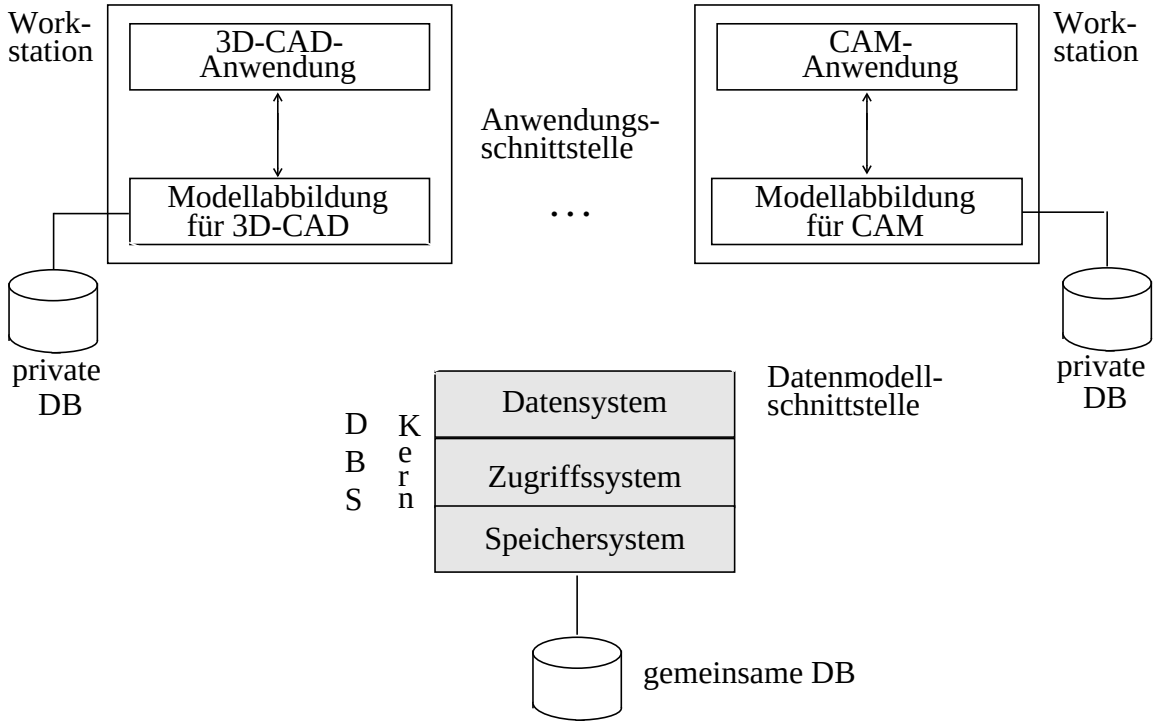


Abb. 1.8: DBS-Kern-Architektur mit anwendungsspezifischen Zusatzschichten

wendungsschnittstelle angeboten werden. Außerdem erzwangen die interaktive Systemnutzung, die häufige Ausführung berechnungsintensiver Funktionen sowie die aus Leistungsgründen erforderliche Nutzung von Referenzlokalität auf den Daten die Verfügbarkeit solcher Zusatzschichten „in der Nähe der Anwendung“, was ihre client-seitige Zuordnung (in einer Workstation) begründet.

In Abb. 1.8 ist eine solche Grobarchitektur mit einem allgemein einsetzbaren DBS-Kern-System skizziert. Insbesondere ergeben sich bei der Realisierung solcher DBS-Erweiterungen client-seitig spezielle Probleme, deren Lösung hier nicht weiterverfolgt werden kann. Vertiefende Darstellungen sind in der aktuellen Literatur zur Architektur von DBS-Erweiterungen für Non-Standard-Anwendungen veröffentlicht [CARE86b, DADA86, HAAS90, HÄRD88, HÜBE92, MITS88].

1.4.2 Client/Server-Architekturen

Das Client/Server-Paradigma verkörpert eine geeignete Vorgehensweise zur Strukturierung von Systemen und zur Entwicklung von Anwendungen [GRAY93]. Auf den DBS-Bereich angewendet erlaubt es eine vereinfachte Verteilung von DBS-Funktionalität über Rechnergrenzen hinweg. Deshalb stand es insbesondere bei der Entwicklung von objektorientierten DBS Pate.

Im Gegensatz zu den DBS, die klassische Datenmodelle realisieren, wollen objektorientierte DBS einen nahtlosen Übergang und eine stärkere Verzahnung zu/mit dem Anwendungsprogramm erzielen. OODBS besitzen in der Regel satzorientierte DB-Schnittstellen als API,

mit denen die Anwendung direkt und typischerweise navigierend Bezug auf die DBS-Datenstrukturen nehmen kann. Ein nahtloser Übergang zur Anwendung wird vor allem dadurch ermöglicht, daß das Typsystem eines OODBS mit dem der eingesetzten Programmiersprache übereinstimmt und daß die Anwendung in einem anwendungsnahen Objektpuffer direkt Objekte durchsuchen und manipulieren kann. Deshalb muß die DBS-seitige Verwaltung auf Client/Server-Architekturen ausgelegt sein,[9] wobei in komplexen Anwendungen häufig leistungsfähige Workstations als Client-Rechner herangezogen werden.

DB-gestützte Client/Server-Architekturen, die vorwiegend für Non-Standard-Anwendungen eingesetzt werden, lassen sich danach unterscheiden, wie die Datenversorgung und damit maßgeblich die Aufgabenverteilung zwischen Client und Server organisiert ist [HÄRD95]. Die drei wichtigsten Grundformen (Page-Server, Object-Server und Query-Server) sind in Abb. 1.9 veranschaulicht. Oft wird zusätzlich der File-Server-Ansatz (z. B. mit NFS) als eigenständiger Architekturtyp diskutiert. Dieser läßt sich als spezieller Page-Server auffassen, der NFS als Transportmechanismus für die zu übertragenden Seiten benutzt. Solche File-Server weisen allerdings ein schlechteres Leistungsverhalten auf, da sie i. allg. weder lokal und noch an die typischen Verarbeitungscharakteristika angepaßt sind [DEWI90].

Wie Abb. 1.9 zeigt, stellt jeder Architekturansatz durch den Object-Manager den verschiedenen Anwendungen die gleiche navigierende und objektorientierte Schnittstelle zur Verfügung. Die gezeigten Server-Typen besitzen als minimale Funktionalität ein Speichersystem zur Verwaltung von Dateien auf Externspeicher (DB) und einen Seitenpuffer zur Minimierung der physischen Ein/Ausgabe auf die DB. Weiterhin muß der Server als zentrale Komponente die Synchronisation der client-seitigen DB-Anforderungen übernehmen, was bei langen Entwurfstransaktionen mit Hilfe von persistenten Datenstrukturen (persistente Sperren) zu geschehen hat. Vorsorgemaßnahmen für DB-Fehler und Server-Crash (Logging) und die Behandlung dieser Situationen (Recovery) gehören ebenfalls zu den zentralen Aufgaben. In Anlehnung an die Originalliteratur [DEWI90] erhalten die einzelnen Architekturtypen ihren Namen vom Granulat der Datenanforderung durch den Client.

Beim *Page-Server* wird jede Seitenanforderung client-seitig durch eine Fehlseitenbedingung (Page-Fault) ausgelöst; daraufhin wird die betreffende Seite vom Server zur Verfügung gestellt. Der Server hat dabei keine Kenntnis des Objektbegriffs: er verwaltet nur Seiten, die (normalerweise) auch das Granulat der Synchronisation und des Logging darstellen. Die Pufferung der Objekte in der Workstation kann nur über die Pufferung der zugehörigen Seiten geschehen, in denen auch die DB-bezogene Verarbeitung (Object-Manager) stattfindet. Alle objektbezogenen Zugriffe und Auswertungen erfolgen demnach in einem client-seitigen Puffer und mit Hilfe eines entsprechenden Zugriffssystems (Abbildung von Objekten auf Seiten).

---

[9] Durch diese starke Anwendungsverflechtung wird die Mehrsprachenfähigkeit von OODBS eingeschränkt. Sie bedingt zugleich einen geringeren Grad an Anwendungsisolation, so daß beispielsweise Fehler in Anwendungsprogrammen zu unbemerkter Verfälschung von im Arbeitsbereich gepufferten DB-Daten führen können. Außerdem ist die client-seitige DBS-Verarbeitung, die eine Übertragung aller benötigten Daten (*data shipping*) und ein Zurückschreiben aller Änderungen impliziert, ein wesentlicher Grund für die mangelnde Skalierbarkeit von OODBS.

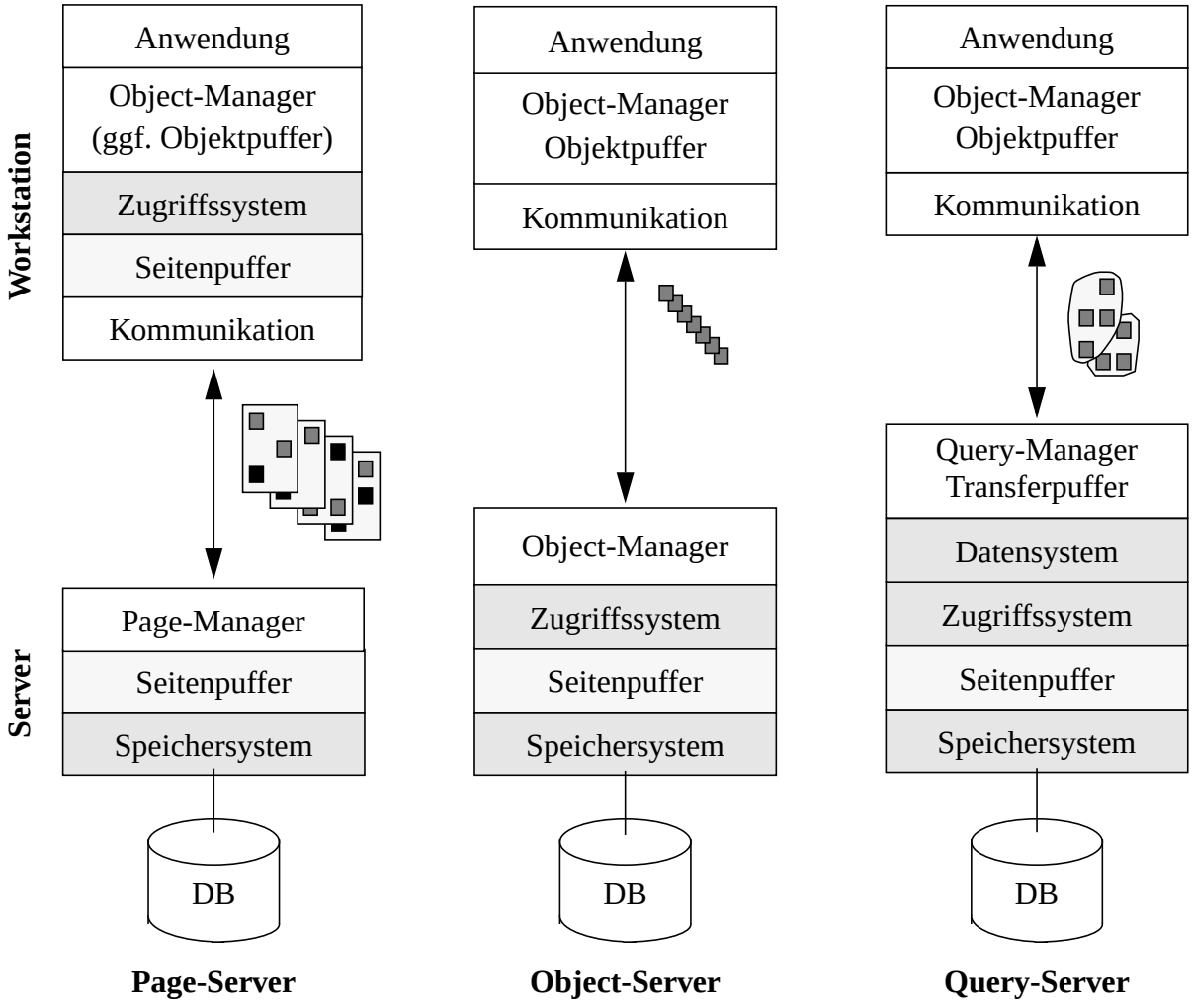


Abb. 1.9: Client//Server-Architekturen für objektorientierte DBS

Ein *Object-Server* liefert in der Grundform auf Anforderung (über eine OID) ein einzelnes Objekt. Erweiterungen dieses Ansatzes erlauben eingeschränkte Anfragemöglichkeiten über inhaltsorientierte Suche mit Hilfe einfacher Suchausdrücke (SSA: simple search arguments), die man auch als „1-mengenorientiert“ [NINK98] bezeichnet. Auch bei diesen erweiterten Anforderungsmöglichkeiten wird ein objektweiser Transfer zur Workstation unterstellt. Die übertragenen Objekte können in den Objektpuffer so eingelagert werden, daß ihre Speicherungs- und Zugriffsstrukturen an die Bedürfnisse der navigierenden Anwendung angepaßt sind. Workstation wie auch Server kennen „Objekte“, besitzen die entsprechende Verarbeitungsfunktionalität und sind in der Lage, objektbezogene Manipulationen und Auswertungen (Selektion, Projektion, Methoden) durchzuführen. Im Gegensatz zum Page-Server kann hier (wie auch beim nachfolgend beschriebenen Query-Server) das Objekt sowohl als Synchronisations- als auch als Logging-Granulat herangezogen werden.

Query-Server beliefern die Anwendung auf eine „n-mengenorientierte“ Anfrage hin mit einer Menge von (komplexen) Objekten, die auf einmal übertragen und auf die Verarbeitungsanforderungen zugeschnitten im Objektpuffer gespeichert wird. Voraussetzung ist eine mäch-

tige Anfrageschnittstelle, die der Object-Manager der Anwendung verfügbar macht. Die Verarbeitung im Objektpuffer geschieht dann über eine navigierende Schnittstelle.

Abb. 1.9 veranschaulicht die Nutzung unseres Schichtenmodells in allen drei Architekturtypen. Im Prinzip wird die Schichtenarchitektur vertikal über Client und Server verteilt. Deshalb sind für die Abwicklung von Client-Anforderungen und für die Ergebnisübertragung zusätzlich client-seitig Kommunikationskomponenten und server-seitig Page-, Object-, oder Query-Manager vorgesehen. Im Gegensatz zur Schichtenmodelldarstellung nach Abb. 1.4 wird beim Speichersystem der server-seitige Seitenpuffer explizit dargestellt, um deutlich zu machen, daß die zu verarbeitenden Daten mehrfach kopiert und client-seitig in Seiten- und Objektpuffern gehalten werden. Beim Page-Server ist das Zugriffssystem client-seitig angesiedelt, da auch die Objektauswahl im Client erfolgt. Die volle Funktionalität des Datensystems wird nur beim Query-Server benötigt, da die beiden anderen Server-Typen in ihrer Reinform keinen mengenorientierten DB-Zugriff gestatten.

In Abb. 1.9 sind zweistufige Client/Server-Architekturen (2-tier architectures) illustriert, die sowohl server- als auch client-seitige DBS-Funktionalität bereitstellen. Künftige Systeme können zudem mehrstufige Client/Server-Architekturen (n-tier architectures) umfassen [ORFA96], in denen die DBS-Funktionalität auf mehrere Ausführungsorte verteilt ist und dynamisch der Anwendungsabwicklung zugeordnet werden kann. Neben der Kernfunktionalität, die unser Schichtenmodell beschreibt, sind zur Realisierung aller DB-basierten Client/Server-Architekturen spezielle Zusatzfunktionen und -protokolle erforderlich, die den Rahmen unserer Betrachtungen allerdings sprengen [DEWI90, FRAN97, HÄRD95].

1.4.3 Verteilte und parallele DBS-Architekturen

Oft legen in einem Unternehmen, ob lokal oder ortsverteilt angesiedelt, wirtschaftliche, organisatorische und technische Gründe eine verteilte Speicherung und Verwaltung der Daten nahe. Kleinere Rechner als Knoten eines verteilten Systems versprechen reduzierte Hardware-Kosten und zusammen mit verfügbaren oder einfach ausbaubaren Kommunikationseinrichtungen kosteneffektive Lösungen. Durch Rechnerleistung „vor Ort“ lassen sich verteilte Organisationsformen besser unterstützen; beispielsweise werden Sonderwünsche „freundlicher“ akzeptiert oder auftretende Fehler schneller behoben. Bei lokaler Autonomie können Teilsysteme flexibler nach den Erfordernissen der Organisation ausgebaut und existierende Datenquellen leichter integriert werden. Technische Gründe beziehen sich vor allem auf eine Erhöhung der Systemleistung, eine verbesserte Verfügbarkeit und Zuverlässigkeit sowie eine modulare Wachstumsfähigkeit des Gesamtsystems. Einzelne Systemknoten können unabhängig voneinander ausgebaut, neue können dynamisch hinzugefügt werden. Nach Möglichkeit sollten die Daten dort gespeichert sein, wo sie am häufigsten benötigt werden. Eine solche Lokalität der Verarbeitung reduziert Zugriffs- und Kommunikationskosten und erlaubt die Realisierung effizienterer Anwendungen. Andererseits ist es bei geeigneter Datenverteilung möglich, parallele Problemlösungen anzustreben und dabei verteilte Systemressourcen zur Beschleunigung der Anwendung einzusetzen. Geeignete Verteilung und Replikation der Daten zusammen mit einer Kapselung der Teilsysteme fördern viele Aspekte der Verfügbarkeit

und Zuverlässigkeit. Beispielsweise minimieren sie die Auswirkungen von „entfernten“ Fehlern oder helfen Fehler durch Redundanzen verschiedenster Art zu maskieren. Diese Gründe lassen den Einsatz verteilter DBS äußert attraktiv erscheinen, so daß zur Nutzung des skizzierten Anforderungsspektrums seit etwa zwanzig Jahren verteilte DBS entwickelt werden. Kritischerweise sei hier jedoch angemerkt, daß verteilte DBS kein Allheilmittel darstellen. Die praktische Erfahrung beim Einsatz solcher Systeme hat nämlich gezeigt, daß i. allg. große Administrationsprobleme zu erwarten sind, welche vor allem

- die globale Konsistenzkontrolle aller Daten,
- die Aktualität des Konzeptionellen Schemas,
- die konsistente Modifikation verteilter Schemainformation usw.

betreffen [GRAY86, HÄRD90a].

Verteilte DBS bestehen aus autonomen Teilsystemen, die koordiniert zusammenarbeiten, um eine logisch integrierte DB bei physischer Verteilung von möglicherweise redundanten oder replizierten Daten zu verwalten. Im Hinblick auf Autonomie und Funktionszuordnung sind alle Teilsysteme (Knoten) Partnersysteme (*peer system*). Sie sind mit allen DBS-Funktionen ausgestattet und sind einander völlig gleichgestellt, d. h., jeder Knoten verfügt über alle Funktionen eines zentralisierten DBS und kann somit durch unser Schichtenmodell beschrieben werden (siehe Abb. 1.10). Eine Partitionierung von DBS-Funktionen dagegen resultiert in Client/Server-DBS, wie sie in Abschnitt 1.4.2 gestreift wurden. Als Prinzip der Datenzuordnung können Partitionierung (DB-Pi in Abb. 1.10), teilweise oder vollständige Replikation oder DB-Sharing [RAHM94] herangezogen werden, wobei zu berücksichtigen ist, daß DB-Sharing eine lokale Rechneranordnung erfordert, da jeder Rechner alle Externspeicher der DB direkt erreichen muß.[10] Abhängig von der gewählten Datenallokation ergeben sich verschiedene Verteilgranulate, mit denen sich die Transaktionslast den Knoten des verteilten DBS zur Verarbeitung zuteilen läßt.

Das zentrale Problem, das ein verteiltes DBS zu lösen hat, ist die Bereitstellung von lokalen Benutzersichten trotz einer beliebigen physischen Verteilung und ggf. Replikation der Daten, so daß jedes Anwendungsprogramm den Eindruck hat, die Daten würden an einer Stelle, in einem DBS und auf dem Rechner gehalten, wo das jeweilige Anwendungsprogramm abläuft. Da jeder Knoten wie ein zentralisiertes DBS zwar volle Funktionalität, aber nur eine lokale Systemsicht auf „seine“ Daten besitzt, ist die Gesamtarchitektur nach Abb. 1.10 um eine Komponente zu erweitern, mit der die globale Systemsicht hergestellt werden kann. Ein separates, zentralisiertes Teilsystem dafür einzurichten, würde einen „single point of failure“ implizieren und außerdem einen Flaschenhals für die Systemleistung darstellen, was beides höchst unerwünschte Konsequenzen wären. Deshalb wird jeder Knoten um die erforderliche Funktionalität erweitert, wobei diese zusammen mit dem globalen Schema, der Verteilinformation sowie den systemweiten Koordinations- und Kommunikationsaufgaben vorteilhafterweise oberhalb der mengenorientierten DB-Schnittstelle anzusiedeln ist. Wegen der abstrak-

---

[10] Nur bei DB-Sharing (Shared-Disk-Architekturen) oder vollständiger Replikation kann eine Transaktion immer vollständig in einem Knoten abgewickelt werden. In den anderen Fällen „folgt die Last den Daten“, d. h., es sind Funktionsaufrufe oder Teiltransaktionen zu verschicken.

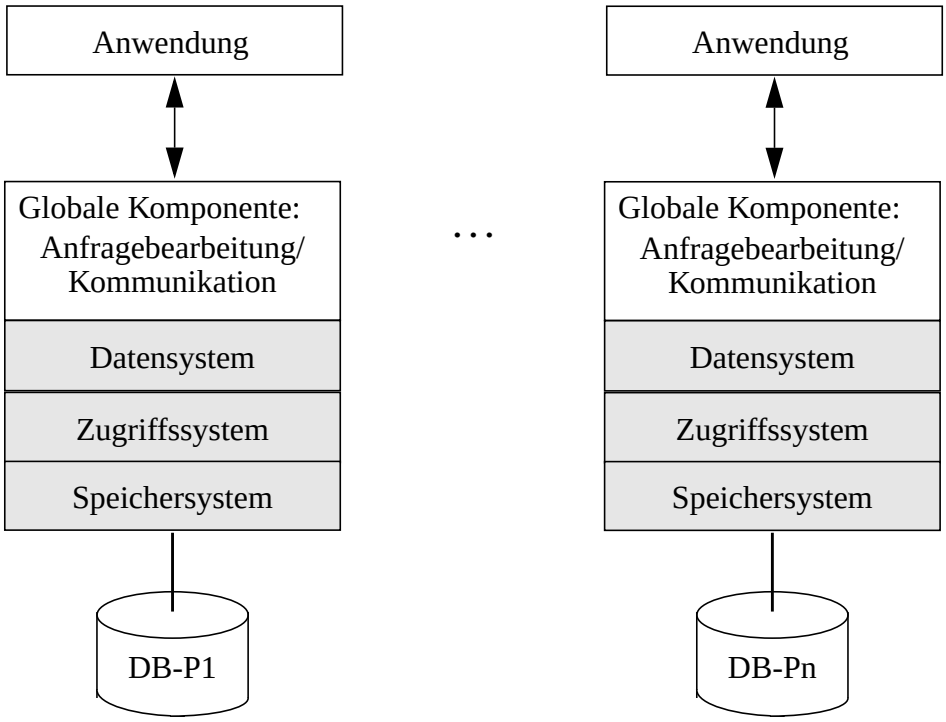


Abb. 1.10: Einsatz des Schichtenmodells in verteilten DBS

ten Objektsicht und der Mengenorientierung dieser Schnittstelle lassen sich mit minimalem Kommunikationsaufwand Teilanfragen an Knoten verschicken und Ergebnismengen einsammeln. Die globale Komponente ist deshalb in Abb. 1.10 als eigenständige Schicht oberhalb des Datensystems illustriert.

Das Grundproblem verteilter Systeme, und insbesondere verteilter DBS, deren Teilsysteme gemeinsam einen globalen konsistenten DB-Zustand verwalten sollen, ist der Mangel an globalem (zentralisiertem) Wissen. Dieses Kernproblem läßt sich durch die „Coordinated Attack“-Aufgabe (Generals-Paradoxon) illustrieren, wo zwei (oder n) Generäle gemeinsam einen Angriff bei unsicherer Kommunikation koordinieren müssen. Symmetrische Protokolle zur Abstimmung hinterlassen immer einen Rest an Entscheidungsunsicherheit, der auch durch beliebig häufige Wiederholung solcher Abstimmungsrunden nicht beseitigt werden kann. Durch fallweise Zuordnung der Kontrolle und Koordination (Entscheidungsbefugnis) läßt sich dieses Kernproblem jedoch relativ leicht beheben. Da jeder Knoten volle Funktionalität besitzt, kann er auch fallweise die Koordination der verteilten Verarbeitung übernehmen. Das wichtigste Einsatzbeispiel hierfür ist das verteilte Zwei-Phasen-Commit-Protokoll, mit dessen Hilfe sich verteilte Transaktionen atomar abschließen lassen (siehe Abschnitt 15.8).

Verteilte DBS sind hardware-seitig durch mehrere unabhängige Rechner realisiert, sind also Mehrrechner-DBS. Als Architekturklassen unterscheidet man Shared-Disk- und Shared-Nothing-Architekturen[11] [BHID88, RAHM94]. Während das Shared-Disk-Prinzip die Er-

[11] Daneben gibt es noch Shared-Everything-Architekturen, die jedoch (hier) nicht der Klasse der Mehrrechner-DBS zugerechnet werden, da sie sich durch einen gemeinsamen Hauptspeicher und gemeinsam genutzte Externspeicher auszeichnen.

reichbarkeit aller Magnetplatten und damit lokale Verteilung (z. B. in einem Raum) impliziert, erlaubt der Shared-Nothing Ortsverteilung mit Partitionierung und Replikation der Daten.

Parallele DBS sind Mehrrechner- oder Mehrprozessor-DBS. Deshalb können alle Shared-Nothing- und Shared-Disk-, aber auch Shared-Everything-Ansätze zur Realisierung paralleler DBS eingesetzt werden. Bei Shared-Everything-Architekturen sind ein gemeinsamer Hauptspeicher und gemeinsam genutzte Externspeicher vorhanden. Sie lassen sich als eng gekoppelte Mehrprozessor-Systeme realisieren, bei denen das Betriebssystem die Abstraktion eines Rechners bietet. Deshalb können zentralisierte DBS ablaufen, wobei mehrere Prozessoren zugleich DB-Operationen abwickeln können und damit die Realisierung von „echter“ Parallelität ermöglichen. Was unser Schichtenmodell und seine Rolle bei der Modellbildung paralleler DBS anbelangt, so läßt sich wiederum der Aufbau jedes beteiligten Knotens damit beschreiben. Im Vergleich zu verteilten DBS, wo in der Regel eine Benutzertransaktion synchron (sequentiell) und verteilt (an den Knoten der benötigten DB-Partitionen) ausgeführt wird, versuchen parallele DBS typischerweise die gleiche Operation oder Funktion auf partitionierten Daten zu parallelisieren, um so die Antwortzeit für eine Anfrage oder Transaktion zu minimieren. Wegen dieser Antwortzeitminimierung kommen in der Regel nur lokale Rechneranordnungen mit breitbandigem Kommunikationsnetz bei der Realisierung von parallelen DBS zum Einsatz.

Es können hier weder die Eigenschaften von Mehrrechner-DBS vertieft noch spezielle Probleme, die sich bei ihrer Realisierung ergeben, berücksichtigt werden. Das betrifft vor allem Architekturen zur Unterstützung von Hochleistungssystemen. Überblicksartige oder die besonderen Eigenschaften dieser Architekturen herausstellende Darstellungen finden sich in [GRAY93, HÄRD86a, RAHM89, RAHM94, RAMA98, STON86a, STON96b, WILL82].

1.4.4 Architekturen von Transaktionssystemen

Der Einsatz von Transaktionssystemen (TAS) gestattet eine dialogorientierte Sachbearbeitung direkt am Arbeitsplatz, wobei eine Benutzerführung über Bildschirmformulare bewerkstelligt wird. Für die Arbeitsvorgänge stehen eine Menge von anwendungsbezogenen Funktionen zur Verfügung, die über spezielle Namen, auch Transaktionscodes (TAC) genannt, aufgerufen oder über Menüs ausgewählt werden. Zur Abwicklung dieser Funktionen verwaltet das Transaktionssystem eine Menge von auf die jeweilige Anwendung zugeschnittenen Programmen, die Arbeitsvorgänge oder einzelne Teilschritte in ihnen verkörpern. Über diese sog. Transaktionsprogramme (TAPs) sind alle Funktionen der Anwendung bis ins Detail rechnerintern vorgeplant (*canned transactions*); die „parametrischen Benutzer" versorgen sie nur noch mit aktuellen Parametern, die zusammen mit dem TAC der Eingabenachricht mitgegeben werden [HÄRD86b].

Beim betrieblichen Einsatz wickeln eine Vielzahl von Benutzern (oft $>10^4$) gleichzeitig Transaktionen ab, wobei natürlich auch Denkzeiten anfallen. Ihre PCs oder Terminals sind über unterschiedliche Leitungen, Netze und Kommunikationsprotokolle mit dem Transaktionssystem verknüpft; das Internet erlaubt sogar weltweit einfache und plattformunabhängige

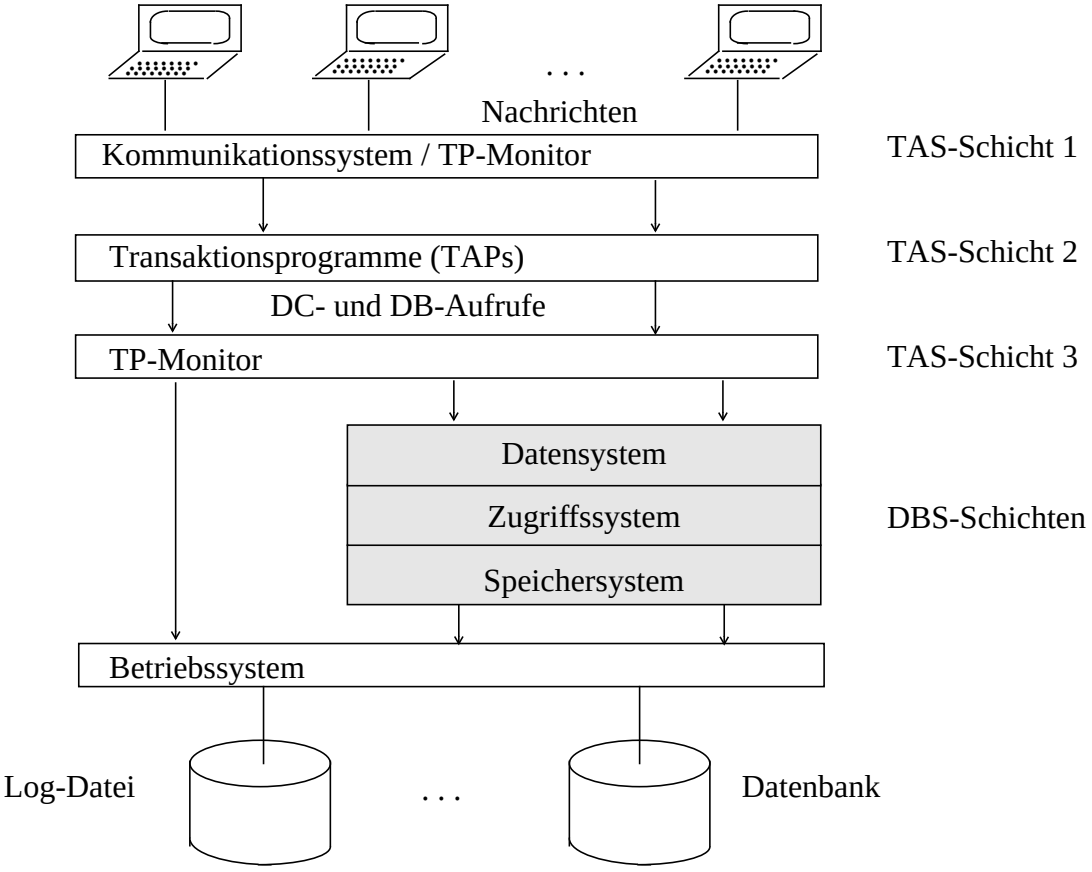


Abb. 1.11: Schichtenmodell eines Transaktionssystems

Zugriffsmöglichkeiten. Oft rufen viele Benutzer gleichzeitig dieselbe Funktion (mit jeweils eigenen aktuellen Parametern) auf, so daß im Prinzip ein TAP „hochgradig parallel" benutzt werden kann [GRAY93]. Solche Anwendungen veranlaßten die Entwicklung von Transaktionssystemen (früher auch DB/DC-Systeme genannt). Während die DB-Komponente für alle Aspekte der Datenhaltung und insbesondere für Datenunabhängigkeit zuständig ist, sorgt der TP-Monitor (DC-Komponente) für die Nachrichten- und TAP-Verwaltung und gewährleistet dabei Kommunikationsunabhängigkeit und isolierte Mehrfachbenutzbarkeit der TAPs [MEYE88].

1.4.4.1 Zentralisierte Verarbeitung und zentralisierte Datenhaltung

Im Rahmen unserer Betrachtungen wollen wir wiederum nur deutlich machen, daß ein DBS das Kernstück eines Transaktionssystems bildet, so daß hier unser Schichtenmodell im Rahmen eines umfassenderen Architekturmodells seinen Einsatz findet. In Abb. 1.11 ist ein solches Architekturmodell skizziert, für das wir nur kurz die Aufgaben der einzelnen TAS-Schichten einführen wollen. Es veranschaulicht mit den TAS-Schichten, daß die TAPs quasi „sandwich-artig“ vom TP-Monitor umschlossen werden, um sie so von der Systemumgebung zu isolieren und insbesondere Unabhängigkeit von allen Kommunikationsaspekten und der Mehrfachnutzung zu erzielen.

Die oberste Systemschicht (TAS-Schicht 1) ist für alle Aufgaben der Ein-/Ausgabe von/zu den Benutzern verantwortlich, wobei eine Reihe von Betriebssystemdiensten genutzt werden. Der Begriff Kommunikationssystem deutet darauf hin, daß hier alle Aufgaben der externen Kommunikation abgewickelt werden. Als Komponente des TP-Monitors organisiert es neben der Datenübertragung die Warteschlangen für die Eingabe- und Ergebnisnachrichten. Eingabenachrichten werden vom TP-Monitor aufbereitet, analysiert und der eigentlichen Verarbeitung zugeführt. Über den TAC wird erkannt, welche Funktion auszuführen ist. Dazu ist , falls erforderlich, für das entsprechende TAP eine Laufzeitumgebung (laden, initialisieren, Speicherplatz zuordnen) zu schaffen, bevor es die Ablaufkontrolle erhält.

TAS-Schicht 2 wird von der Menge der zur Anwendung gehörenden TAPs und den entsprechenden Verwaltungsfunktionen gebildet. In der Aufrufhierarchie der Transaktionsverarbeitung übernimmt das ausgewählte TAP die Eingabenachricht und beginnt nach Routineprüfungen die eigentliche Verarbeitung, die durch eine Reihe von Datei- oder DB-Zugriffen unterstützt wird. Mit Beendigung der Verarbeitung erzeugt das TAP eine Ausgabenachricht, die über den TP-Monitor an den Benutzer weitergeleitet wird.

Der TP-Monitor in TAS-Schicht 3 ist vor allem verantwortlich für die Abwicklung des Mehrbenutzerbetriebs (z. B. Multi-Tasking) und der Verwaltung der TAP-Aktivierungen, was auch eine Beteiligung an der Transaktionskoordination verlangt. Weiterhin registriert der TP-Monitor alle Aufrufe an die Datenhaltung, damit auch Transaktionsbeginn und -ende, und reicht sie entweder an das Dateisystem oder an das DBS weiter. Diese Aufrufregistrierung ist vor allem aus Gründen der Fehlerbehandlung wichtig, da der TP-Monitor bei Ausfall eines TAP die Transaktion DB-seitig zu schließen und den Benutzer zu benachrichtigen hat.

1.4.4.2 Verteilte Verarbeitung und verteilte Datenhaltung

Das eben skizzierte Architekturmodell ist grundsätzlich geeignet, auf verteilte Anwendungen hin verallgemeinert zu werden; d. h., verteilte Verarbeitung und verteilte Datenhaltung sind zu unterstützen. In einem verteilten Transaktionssystem ist die Menge der TAPs und der Daten über mehrere Knoten verteilt, wobei als Betriebsformen Partitionierung, partielle Redundanz sowie vollständige Replikation sowohl von Programmen als auch Daten denkbar sind. Prinzipiell kann nun die Transaktionsverarbeitung in jedem Knoten mit Hilfe des Architekturmodells nach Abb. 1.11 beschrieben werden. Ohne Zusatzmaßnahmen ist zunächst keine globale Systemsicht vorhanden, denn jeder Knoten hat nur eine lokale Sicht auf seine eigenen Betriebsmittel. In verteilten Anwendungen ist jedoch zumindest dem Benutzer gegenüber die Verteilung der TAPs und Daten zu verbergen. Das bedeutet aber, daß eine Schicht des Modells (wie beim Modell für verteilte DBS in Abb. 1.10) eine globale Systemsicht durch Kommunikation mit den Teilsystemen erzeugen muß, um dem Benutzer die Illusion eines „logisch zentralisierten“ Systems geben zu können.

In [MEYE87] werden verschiedene Systemlösungen für verteilte Transaktionssysteme diskutiert, wobei gezeigt wird, daß grundsätzlich jede Modellschicht zur Bildung der globalen Systemsicht herangezogen werden kann. Wie in Abschnitt 1.4.3 erläutert, kann für das Herstellen der globalen Systemsicht ein verteiltes DBS herangezogen werden. In diesem Fall sind

die Verteilungs-/Kommunikationsaufgaben im Datensystem anzusiedeln (Minimierung der Kommunikation, mengenorientierte Anforderungen). Die Funktionalität des verteilten DBS bestimmt die Art und Flexibilität der verteilten Datenhaltung, d. h., in der Regel sind nur homogene DB-Lösungen möglich. Unabhängig von der verteilten Datenhaltung kann jedoch eine Verteilung der Anwendungslogik auf ein oder mehrere (verteilte) TAPs erfolgen.

Die weiteren Lösungsvorschläge zielen darauf ab, die globale Systemsicht durch TP-Monitor-Funktionalität herzustellen. In allen Ansätzen ist es deshalb nicht erforderlich, Annahmen über die Datenhaltung zu treffen. Insbesondere ist in den folgenden drei Fällen die Einbindung heterogener Datenquellen oder verschiedenartiger DBS prinzipiell möglich.

- Die Bereitstellung der globalen Systemsicht in TAS-Schicht 1 heißt „Transaction Routing", d. h., die globale Systemschicht bewirkt lediglich eine Weiterleitung von TACs zu einem Knoten, der das betreffende TAP ausführen kann. Deshalb sind ganze Transaktionen Einheiten der Verteilung, was bei gut partitionierbaren Betriebsmitteln und gleichmäßigem Lastaufkommen mit ausgeprägtem Lokalitätsverhalten (ein hoher Anteil der Aufrufe wird lokal verarbeitet) eine befriedigende Lösung sein kann. Es sind jedoch in einer Transaktion keine Funktionen realisierbar, die Betriebsmittel mehrerer Knoten benutzen. Übergreifende Auswertungen muß der Benutzer selbst vornehmen.
- Soll auf der Ebene der TAPs, also in TAS-Schicht 2, die globale Systemsicht hergestellt werden, so ist es sehr wichtig, daß die TAPs von den Verteilungsaspekten der Daten und den Charakteristika der (verschiedenen) DBS isoliert werden. Zugriffe auf lokale und entfernte Daten müssen in einheitlicher Weise ausgeführt werden. Restrukturierung und Umverteilung der Daten schlägt sonst bis auf den Programmtext des TAP durch. Deshalb wurde hierfür das Prinzip der „Programmierten Verteilung“ entwickelt, wobei DB-Operationen in Funktionen (stored procedures) gekapselt werden, die, vorab geplant und realisiert, von den beteiligten DB-Servern den TAPs zur Ausführung angeboten werden. Aufrufe solcher Funktionen und Ergebnisübermittlung erfolgen durch den TP-Monitor, was Ortsunabhängigkeit gewährleistet.
- Der entsprechende Lösungsansatz in TAS-Schicht 3 heißt „Aufrufweiterleitung“ (function request shipping). Der TP-Monitor muß dafür in die Lage versetzt werden, die globale Systemsicht zu bilden. Er bestimmt den Knoten, der eine Datenanforderung bearbeiten kann, leitet den Auftrag weiter und empfängt das Ergebnis, das er dann dem TAP zur Verfügung stellt.

Bei den Ansätzen Aufrufweiterleitung und Programmierte Verteilung ist die Entwicklung knotenübergreifender Anwendungen möglich. Um Transaktionsschutz bieten zu können, müssen jedoch der TP-Monitor und alle teilnehmenden DBS kooperieren und gemeinsam verteilte Zwei-Phasen-Commit-Protokolle (2PC-Protokoll) abwickeln. Weitere Anwendungscharakteristika sowie Vor- und Nachteile der verschiedenen Lösungsansätze werden in [HÄRD90b] im Detail diskutiert und evaluiert.

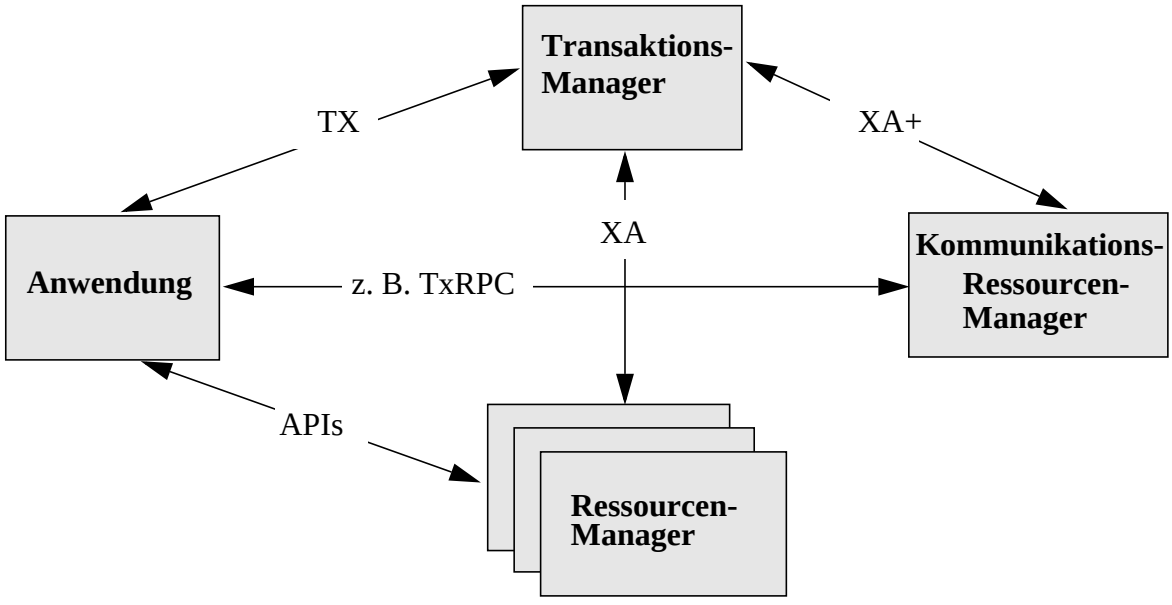


Abb. 1.12: Zusammenspiel der Komponenten nach dem DTP-X/Open-Protokoll

1.4.5 Komponentenbasierte Systemarchitekturen

Die zuletzt diskutierten TAS-Architekturen führen zu komponentenbasierten und offenen Systemen [GRAY93, LAME94], bei denen der (verteilte) TP-Monitor für Kooperation und Koordination verantwortlich ist. Die teilnehmenden DBS oder Datenquellen sind unabhängig, so daß verteilte TAPs heterogene Daten „einbinden“ und verarbeiten können, um knotenübergreifende Anwendungslösungen zu erreichen. Transaktionsschutz verlangt, daß jede beteiligte Datenquelle ihren Zustand sichern kann. Weiterhin ist es erforderlich, daß sie „nach außen“ einen sog. Prepare-Zustand anbieten kann, der Voraussetzung für die Teilnahme an einem 2PC-Protokoll ist. Gray und Reuter bezeichnen eine solche Komponente, die ihre gemeinsam benutzbaren Betriebsmittel stabil machen und (in Kooperation mit anderen verteilte) ACID-Transaktionen gewährleisten kann, als *Ressourcen-Manager* [GRAY93]. In diesem Sinne sind DBS Musterbeispiele für (komplexe) Ressourcen-Manager.

1.4.5.1 Transaktionsverarbeitung in offenen Systemen

Für die Transaktionsverarbeitung in offenen Systemen sind eine Reihe von Komponenten und Protokollen erforderlich, die als Infrastruktur verteilter Systeme mit dem Begriff „Middleware“ bezeichnet werden [ORFA96]. Für Anwendung und Ressourcen-Manager wurden bereits entsprechende Kommunikations- und Kooperationsprotokolle durch DTP-X/Open standardisiert [XOPE93]. In Abb. 1.12 ist das Zusammenspiel der Komponenten mit den zugehörigen Schnittstellen gezeigt. Die Anwendung besitzt eine standardisierte Schnittstelle (TX) zum Transaktions-Manager, um Anwendungen mit Transaktionsschutz zu versorgen (Begin_TA, Commit_TA, Rollback_TA). Alle an einer Transaktion teilnehmenden Ressour-

cen-Manager benutzen mit dem Transaktions-Manager eine standardisierte Schnittstelle (XA), um Aufrufe mehrerer verschiedener Ressourcen-Manager in die (verteilte) Anwendungstransaktion einbinden zu können. Die wichtigsten Aufrufe der XA-Schnittstelle sind Join_TA, Prepare_TA, Commit_TA und Rollback_TA. Die entsprechenden Anwendungsprogrammierschnittstellen (APIs) zu den Ressourcen-Managern, z. B. SQL für relationale DBS, sind nicht im Rahmen von DTP-X/Open standardisiert.

Man kann sich den Ablauf einer Anwendungstransaktion folgendermaßen vorstellen. Nach Anmeldung (Begin_TA) erhält die Anwendung für die begonnene Transaktion eine systemweit eindeutige Transaktions-ID (TRID) durch den Transaktions-Manager zugeteilt. Sie dient der Kontrolle der verteilten Transaktionsverarbeitung und wird jeder Anforderung der Anwendung an einen Ressourcen-Manager mitgegeben. Erhält ein Ressourcen-Manager von einer Transaktion erstmalig eine Dienstanforderung, so meldet er die TRID mit einem Join_TA-Aufruf dem (lokalen) Transaktions-Manager. Ab diesem Zeitpunkt ist der betreffende Ressourcen-Manager in die globale Transaktionskontrolle eingebunden und kann mit Rollback_TA u. a. in die Fehlerbehandlung oder mit Prepare_TA und Commit_TA in das 2PC-Protokoll einbezogen werden.

Für den verteilten Fall werden die Dienste von Kommunikations-Ressourcen-Manager herangezogen, die mit dem Transaktions-Manager eine erweitere Schnittstelle (XA+) zur Transaktionskontrolle und eine Reihe von standardisierten Schnittstellen zur Anwendung besitzen (TxRPC, CPI-C V2, XATMI, [ORFA96]).[12] Das Zusammenspiel der Komponenten ist in jedem Knoten entsprechend Abb. 1.12 organisiert. Bei knotenübergreifender Transaktionsverarbeitung übernehmen die Kommunikations-Ressourcen-Manager der beteiligten Knoten die Abwicklung der Kommunikation. Um verteilten Transaktionsschutz zu gewährleisten, melden sie die erstmalige knotenübergreifende Dienstanforderung einer Transaktion beim jeweiligen lokalen Transaktions-Manager ab/an, so daß auch knotenübergreifend Fehlerbehandlung oder Commit-Verarbeitung ermöglicht wird.

1.4.5.2 Architektur kooperierender Ressourcen-Manager

Heutige Anwendungen fordern jedoch nicht nur für DB-Objekte, sondern auch für andere Betriebsmittel wie Nachrichten, Dateien, Warteschlangen, Programmiersprachenobjekte usw. die Einbeziehung in den Transaktionsschutz. Als Verallgemeinerung bisheriger Systemarchitekturen schlagen Gray und Reuter deshalb eine Architektur kooperierender Ressourcen-Manager vor, die als ausgezeichnete Systemkomponenten (Ressourcen-Manager) den TP-Monitor und den Transaktions-Manager besitzt. Der TP-Monitor verwaltet andere Ressourcen-Manager und Betriebsmittel wie Prozesse, Tasks, Zugriffsrechte, Programme und Kontexte. Er ist insbesondere zuständig für die transaktionsorientierte Betriebsmittelzuteilung. Dem Transaktions-Manager dagegen obliegen alle Aufgaben im Zusammenhang mit der Transaktionskon-

[12] Auf der Anwendungsebene hat X/Open drei Schnittstellen zwischen Anwendungen und Kommunikations-Ressourcen-Managern definiert. TxRPC ist die transaktionsgeschützte Version von DCE RPC. CPI-C V2 ist eine erweiterte Peer-to-Peer-Konversationsschnittstelle mit OSI-TP-Semantik. XATMI (Application/Transaction Management Interface) ist eine Konversationsschnittstelle für Client/Server-Anwendungen.

trolle und der Gewährleistung der ACID-Eigenschaften. Zusammen mit dem TP-Monitor ist er für die Korrektheit der verteilten Transaktionsverarbeitung verantwortlich.

Die beteiligten Ressourcen-Manager sind Systemkomponenten, die in Kooperation Transaktionsschutz für ihre gemeinsam nutzbaren Betriebsmittel übernehmen, d. h., sie gestatten die externe Koordination der Aktualisierung ihrer Betriebsmittel durch 2PC-Protokolle. Der TP-Monitor „orchestriert“ und integriert solche verschiedenartigen Systemkomponenten, um eine gleichförmige Schnittstelle für Anwendungen und Operationen mit demselben Verhalten im Fehlerfall (*failure semantics*) zu bieten. Solche komponentenorientierten Architekturen versprechen einen hohen Grad und Flexibilität, Skalierbarkeit und Offenheit für die Integration neuer Ressourcen-Manager. Sie berücksichtigen zugleich alle Aspekte der Transaktionsverwaltung, der verteilten Verarbeitung sowie der Anpaßbarkeit des Systems an verschiedene Lastcharakteristika [REUT90].

Der Beschreibungsrahmen unseres Schichtenmodells und die einzuführenden Implementierungskonzepte und -techniken sind für alle Ressourcen-Manager geeignet, die Daten verschiedensten Typs verwalten. Das betrifft DBS, Dateisysteme, Archivierungssysteme für Daten oder große Objekte, Systeme zur Verwaltung persistenter Warteschlangen u. a., wenn auch nicht immer alle Schichten und die gesamte Funktionalität benötigt werden. Die Realisierung anderer Arten von Ressourcen-Managern, beispielsweise für die Verwaltung von Ausgabefenstern (Windows) oder persistenten Programmiersprachenobjekten, verlangt dagegen neue Konzepte und Techniken, die im Rahmen komponentenorientierter Systemarchitekturen bereitgestellt werden oder noch zu entwickeln sind.

1.5 Themenüberblick

In den nachfolgenden Kapiteln werden die wichtigsten Implementierungskonzepte und -techniken zentralisierter DBS im Rahmen unseres Schichtenmodells von „unten nach oben“ eingeführt, analysiert und bewertet. Dazu orientieren wir uns an der Grobarchitektur von Abb. 1.6 und gliedern unsere Aufgabe in vier große Teilbereiche – Speichersystem, Zugriffssystem, Datensystem und Transaktionsverwaltung. Die besonderen Aspekte der Metadatenverwaltung werden dabei zusammen mit den Aufgaben, Funktionen und Strukturen der Abbildungsschichten diskutiert, wie es auch das verfeinerte Schichtenmodell nach Abb. 1.7 verdeutlicht.

Teil II beschreibt die Aufgaben des Speichersystems, das eine Menge von Externspeichern zu verwalten und eine abstrakte Zugriffsschnittstelle für sie im Hauptspeicher zur Verfügung zu stellen hat. Dazu werden zunächst in Kapitel 2 die wesentlichen Konzepte zur Realisierung einer Ein-/Ausgabe-Architektur sowie die Eigenschaften der wichtigsten Speichermedien, die in DBS ihren Einsatz finden, skizziert. Zu den wichtigsten Aufgaben der Externspeicherverwaltung zählt die Bereitstellung geeigneter Abstraktionskonzepte, mit denen die Charakteristika externer Geräte für die übrigen DBS-Komponenten verborgen werden. Dazu sind flexible Datei- und Segmentkonzepte sowie adäquate Adressierungsverfahren zu entwickeln. Spezielle Einbringverfahren für Änderungen und Fehlertoleranzmaßnahmen erlauben eine verein-

fachte Fehlerbehandlung in den Abbildungschichten des Speichersystems (Kapitel 3 und 4). Schließlich muß eine leistungsfähige DB-Pufferverwaltung mit zugeschnittenen Such-, Zuordnungs- und Ersetzungsverfahren von Seiten für eine Minimierung des physischen E/A-Operationen sorgen, um damit den Grundstein für ein gutes Leistungsverhalten des Gesamtsystems zu legen (Kapitel 5).

In Teil III werden alle Funktionen des Zugriffssystems mit einer Vielfalt von Implementierungsmethoden für Speicherungsstrukturen und Zugriffspfade dargestellt. Techniken zur Speicherung von einfachen, komplex-strukturierten und langen Objekten sowie verschiedene Verfahren zu ihrer Adressierung werden in Kapitel 6 behandelt. Eindimensionale Zugriffspfade erlauben mit Hilfe von Primär- oder Sekundärschlüssel ein effizientes Aufsuchen von einzelnen Sätzen oder Satzmengen. Eine großes Spektrum solcher Strukturen wird in Kapitel 7 eingeführt; dabei werden die einzelnen Zugriffspfadtypen mit Hilfe eines einheitliches Schemas klassifiziert und auf ihre Tauglichkeit beim DBS-Einsatz hin bewertet. Methoden zum Verknüpfen und Aufsuchen von Sätzen unterschiedlichen Satztyps gestatten die Unterstützung typübergreifender Operationen und komplexerer Auswertungsvorgänge (Kapitel 8). Mehrdimensionale Zugriffspfade, die durch neue Anwendungsklassen wie Data Warehouse, Geographische Informationssysteme usw. heute eine besondere Aktualität gewonnen haben, lassen in effizienter und symmetrischer Weise das Aufsuchen von Sätzen über k Schlüssel (und Teilmengen davon) zu. Die wichtigsten Verfahren für solche Zugriffspfade, die exakte Anfragen, den „räumlichen“ Zugriff und Ähnlichkeitssuche bei punktförmigen und räumlich ausgedehnten Objekten bewerkstelligen sollen, werden in Kapitel 9 dargestellt und systematisiert.

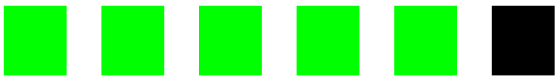
Das Datensystem mit seinen Aufgaben und Konzepten wird in Teil IV behandelt. Zur Abstraktion von physischen Sätzen und Zugriffspfaden wird eine satzorientierte DB-Schnittstelle, die Navigation längs logischer Zugriffspfade und Referenzierung logischer Sätze erlaubt, implementiert. Ihre Aufgaben und Konzepte sind in Kapitel 10 beschrieben. Aufbauend auf der satzorientierten DB-Schnittstelle, die bei DBS nach dem Netzwerk- oder Hierarchiemodell oder auch bei objektorientierten DBS als „externe“ Benutzerschnittstelle dient, wird in Kapitel 11 zunächst die Implementierung relationaler Operatoren betrachtet. Diese höheren Operatoren erleichtern in Kapitel 12 die Realisierungsüberlegungen für eine mengenorientierte DB-Schnittstelle, die der Funktionalität des Relationenmodells und der Sprache SQL entsprechen soll. Bei unserer detaillierten Betrachtung spielen vor allem Fragen der Übersetzung oder Interpretation von mengenorientierten Anfragen und insbesondere ihre Optimierung eine zentrale Rolle.

Teil V ist den Techniken zur Realisierung des Transaktionskonzeptes gewidmet, das zunächst in Kapitel 13 kurz mit seinen ACID-Eigenschaften eingeführt wird. Aspekte der Synchronisation und die vielfältigen Techniken zu ihrer Implementierung werden im Detail in Kapitel 14 diskutiert und bewertet, während Kapitel 15 die Aufgaben der Fehlerbehandlung in DBS aufgreift und wichtige Lösungsverfahren, die mit dem Begriff „Logging- und Recovery-Verfahren“ zusammengefaßt werden, erörtert. Erweiterte Transaktionskonzepte werden in Kapitel 16 behandelt. Diese dienen der Flexibilisierung der Ablaufkontrolle vor allem bei länger

andauernden Transaktionen. Die Erweiterungsvorschläge zielen auf eine Abschwächung einer oder mehrerer ACID-Eigenschaften ab, die ursprünglich nur für kurze Transaktionen definiert worden waren. Zur Verbesserung ihres Ablaufverhaltens bei der Synchronisation und der Fehlerbehandlung wird eine Binnenstruktur eingeführt, die vor allem die „Alles-oder-Nichts“-Eigenschaft von ACID-Transaktionen relativiert.

Abschließend wird in Kapitel 17 ein Ausblick auf die neue Anforderungen und Funktionen, insbesondere für objekt-relationale Konzepte [CHAM96, STON96a] gegeben, die zu Weiterentwicklungen sowohl bei relationalen als auch bei objektorientierten DBS führen. Dabei wird deutlich, daß in solchen künftigen DBS eine Erweiterungsinfrastruktur zur Integration von benutzerdefinierten Datentypen und Funktionen verlangt wird. Die Verarbeitungskonzepte müssen außerdem so flexibilisiert werden, daß DBS-Funktionalität nicht nur auf den DB-Server begrenzt bleibt, sondern auch, dynamisch allokiert, am „günstigsten“ Ort in einer mehrschichtigen Client/Server-Architektur bereitgestellt werden kann.

Wie wir bereits in der bisherigen Diskussion gesehen haben, finden sich diese grundlegenden Konzepte und Techniken auch in DBS-Kern-Systemen, Client-Server-DBS, Mehrrechner-DBS oder Transaktionssystemen wieder, seien sie lokal oder ortsverteilt. Dabei verfügt jeder Knoten über alle Funktionen eines zentralisierten DBS oder über eine Teilmenge (bei funktional spezialisierten Knoten muß die gesamte Funktionalität kooperativ von mehreren Knoten erbracht werden). Selbst in DB-Maschinen lassen sich diese grundlegenden Implementierungskonzepte anwenden, unabhängig davon, was durch spezialisierte Hardware nachgebildet wird und was nicht. Somit besitzen die hier beschriebenen Konzepte und Techniken einen großen Grad an Allgemeingültigkeit.



Teil V

Transaktionsverwaltung

13 Das Transaktionsparadigma

Dieses und die folgenden Kapitel befassen sich mit dem Transaktionskonzept und seiner Realisierung. Die Einhaltung dieses auch als ACID-Paradigma bezeichneten Konzepts ist Voraussetzung für die sichere und konsistente Ausführung von DB-Operationen, trotz gleichzeitiger DB-Zugriffe durch zahlreiche Benutzer und möglicher Fehlersituationen wie Rechner- oder Plattenausfällen. Die Grundlagen des Transaktionskonzepts sowie wesentliche Implementierungstechniken wurden bereits in den siebziger Jahren entwickelt, insbesondere in Verbindung mit der Implementierung der ersten relationalen DBS [GRAY78]. Die Unterstützung des Transaktionskonzepts ist seitdem längst eine obligatorische Funktion aller Datenbanksysteme, unabhängig vom zugrundeliegenden Datenmodell. Für wesentliche Aufgaben der Transaktionsverwaltung, insbesondere Synchronisation, Logging und Recovery, steht ein Fundus an leistungsfähigen und in der Praxis erprobten Verfahren zur Verfügung.

Die Bedeutung des Transaktionskonzepts geht jedoch weit über den Einsatz im Rahmen von DBS hinaus. Es stellt ein zentrales Paradigma der Informatik dar, das zur sicheren Verwendung unterschiedlichster Betriebsmittel eingesetzt werden kann. Insbesondere ist das Transaktionskonzept der Schlüssel zur zuverlässigen Nutzung verteilter Systeme [GRAY93]. Durch entsprechende Standardisierungen können dabei auch heterogene und autonome Teilsysteme in die Transaktionsverarbeitung eingebunden werden (siehe Abschnitt 1.4.5.1). Das Transaktionskonzept ist somit auch von zentraler Bedeutung für die sichere Abwicklung von Geschäftsvorgängen im Internet (Electronic Commerce), welche absehbar eine enorme wirtschaftliche Bedeutung erlangen werden [TYGA98].

Auf der anderen Seite zeigen sich für bestimmte Anwendungsbereiche mit komplexen Verarbeitungsvorgängen zunehmend auch Beschränkungen des „klassischen" Transaktionskonzepts. Dieses ist ausgelegt für sog. *flache Transaktionen*, welche aus einer linearen Folge von meist relativ wenigen DB-Operationen bestehen. Es wurden daher in der Literatur zahlreiche Erweiterungen vorgeschlagen [MOSS81, ELMA92, JAJO97], welche bestimmte Beschränkungen des herkömmlichen Transaktionskonzepts beheben sollen. Erste Implementierungen liegen insbesondere für *geschachtelte Transaktionsmodelle* vor, welche eine interne Untergliederung von Transaktionen in Teiltransaktionen unterstützen, für die besondere Funktionsmerkmale gelten.

Die Darstellung in diesem Buch konzentriert sich weitgehend auf das klassische ACID-Paradigma und seine Realisierung im Rahmen von (zentralisierten) Datenbanksystemen. Die dabei verwendeten Techniken sind von grundlegender Bedeutung und stellen auch die Basis für die Realisierung verteilter Transaktionen sowie die Implementierung erweiterter Transaktionsmodelle dar.

In diesem Kapitel erläutern wir einführend das ACID-Paradigma, die Benutzerschnittstelle zur Transaktionsverwaltung sowie Aspekte der Integritätskontrolle. Die folgenden Kapitel behandeln dann Anforderungen sowie die wichtigsten Realisierungsansätze für die grundlegenden Funktionen der Synchronisation (Kapitel 14) sowie von Logging und Recovery (Kapitel 15). Anschließend wird ein Überblick zu erweiterten Transaktionsmodellen gegeben, insbesondere geschachtelten Transaktionen (Kapitel 16).

13.1 Die ACID-Eigenschaften

Transaktionen stellen Abstraktionen von Verarbeitungsvorgängen der jeweiligen Anwendungs-Miniwelt dar (Geldauszahlung, Verkauf eines Artikels, Platzreservierung usw.). Wie in Abb. 13.1 illustriert, wird die Datenbank, welche die Miniwelt modellhaft repräsentiert, durch eine Transaktion von einem Zustand M in einen Zustand M’ überführt (M = M’ ist im Spezialfall von Lesetransaktionen möglich). Wesentlich für die Qualität und Korrektheit der Datenbankverarbeitung ist, daß auch nach Durchführung von Transaktionen eine hohe Übereinstimmung zwischen den resultierenden Datenbankzuständen und der Miniwelt (Realität) gewährleistet bleibt. Transaktionen müssen somit die Konsistenz der Datenbank bewahren (s. u.). Weiterhin wird zur Konsistenzwahrung verlangt, daß sämtliche DB-Zugriffe ausschließlich durch Transaktionen erfolgen.

Bezüglich der Ausführung von Transaktionen, welche aus einer oder mehreren Operationen der jeweiligen DB-Sprache (z. B. SQL) bestehen, garantiert das DBS die Einhaltung von vier grundlegenden Eigenschaften, nämlich Atomarität, Konsistenz, Isolation und Dauerhaf-

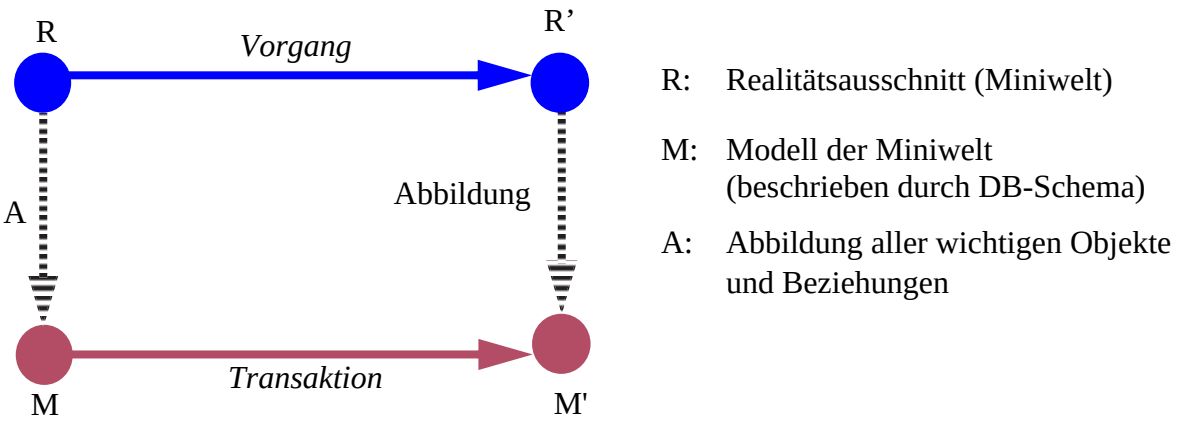


Abb. 13.1: Transaktionen als Modellbildung von Anwendungsvorgängen

tigkeit. Man spricht hierbei von den sog. ACID-Eigenschaften, abgeleitet von den Anfangsbuchstaben der englischen Begriffe ***A***tomicity, ***C***onsistency, ***I***solation und ***D***urability [HÄRD83b]. Diese im folgenden näher erläuterten Eigenschaften charakterisieren zugleich das Transaktionskonzept.

1. Atomarität (Atomicity, „Alles oder Nichts“)

Die Ausführung einer Transaktion soll aus Sicht des Benutzers ununterbrechbar verlaufen, so daß sie entweder vollständig oder gar nicht ausgeführt wird. Dies bezieht sich vor allem auf die im Rahmen der Transaktion auszuführenden Änderungen der Datenbank. Tritt während der Ausführung einer Transaktion ein Fehler auf (Programmfehler, Hardware-Fehler, Absturz des Betriebssystems usw.), der die ordnungsgemäße Fortführung verhindert, werden seitens des DBS sämtliche bereits erfolgten Änderungen der Transaktion zurückgesetzt. Durch eine sog. *Undo-Recovery* werden die „Spuren“ der unterbrochenen Transaktion vollständig aus der Datenbank entfernt. Um diese Fehlerbehandlung zu ermöglichen, führt das DBS ein *Logging* durch, d. h., zu den erfolgten Änderungen werden geeignete Informationen auf einer Protokoll- oder Log-Datei dauerhaft mitgeschrieben.
Das Zurücksetzen der Transaktion entspricht somit dem „Nichts-Fall“ der Atomarität. Der „Alles-Fall“ wird durch Eigenschaft 4 (Dauerhaftigkeit) gewährleistet.

2. Konsistenz (Consistency)

Die Transaktion ist die Einheit der Datenbankkonsistenz. Dies bedeutet, daß sie die Datenbank von einem konsistenten in einen wiederum konsistenten (nicht notwendigerweise unterschiedlichen) Zustand überführt. Von besonderer Bedeutung ist dabei die Einhaltung der *logischen Konsistenz*, so daß die Inhalte der Datenbank einem möglichst korrekten Abbild der modellierten Wirklichkeit entsprechen. Hierzu können beim Datenbankentwurf *semantische Integritätsbedingungen* (zulässige Wertebereiche, Schlüsseleigenschaften usw.) definiert werden, welche vom DBS automatisch zu überwachen sind. Das DBS garantiert somit, daß am Ende einer jeden Transaktion sämtliche Integritätsbedingungen erfüllt sind. Änderungen, welche zu einer Verletzung der Integritätsbedingungen führen, werden abgewiesen, d. h., sie führen zum Zurücksetzen der Transaktion. Voraussetzung für die logische ist die *physische Konsistenz* der Datenbank, d. h. die korrekte interne Repräsentation und Speicherung der Daten im Datenbanksystem.
Zu beachten ist, daß die Konsistenz i. allg. nur vor und nach Ausführung einer Transaktion gewährleistet wird. Während einer Transaktion dagegen können temporäre Konsistenzverletzungen eintreten bzw. notwendig werden. Als Beispiel diene eine Umbuchung zwischen Giro- und Sparkonto. Nach Abbuchen des Betrags vom Girokonto liegt ein inkonsistenter Datenbankzustand vor, wenn als Integritätsbedingung verlangt wird, daß bei Kontobewegungen innerhalb einer Bank die Summe der Kontostände unverändert bleiben soll. Die logische Konsistenz der Datenbank ist erst nach einer weiteren DB-Operation zum Gutschreiben des Betrags auf dem Sparkonto hergestellt. Solche Integritätsbedingungen, welche erst nach mehreren DB-Operationen erfüllbar sind, werden als *verzögerte Integritätsbedingungen (deferred integrity constraints)* bezeichnet; sie sind i. allg. am Transaktionsende zu überprüfen. Demge-

genüber lassen sich *unverzögerte Integritätsbedingungen (immediate integrity constraints)* unmittelbar bei einer DB-Änderung überwachen, z. B. einfache Bedingungen auf einem Attribut oder einem Satz (Einhaltung von Wertebereichsgrenzen, Eindeutigkeit von Attributwerten, usw.). Auf weitere Arten von Integritätsbedingungen gehen wir in Abschnitt 13.3.1 ein.

3. Isolation

Datenbanksysteme unterstützen typischerweise eine große Anzahl von Benutzern, die gleichzeitig auf die Datenbank zugreifen können. Trotz dieses *Mehrbenutzerbetriebs* wird garantiert, daß dadurch keine unerwünschten Nebenwirkungen eintreten, wie z. B. das gegenseitige Überschreiben desselben Datenbankobjektes. Vielmehr bietet das DBS jedem Benutzer und Anwendungsprogramm einen „logischen Einbenutzerbetrieb“, so daß parallele Datenbankzugriffe anderer Benutzer unsichtbar bleiben. Diese Isolation bzw. Ablaufintegrität der Transaktionen wird seitens des DBS durch geeignete *Synchronisationsmaßnahmen* erreicht, z. B. durch bestimmte Sperrverfahren. Ablaufintegrität ist Voraussetzung zur Einhaltung der Datenbankkonsistenz.

4. Dauerhaftigkeit (Durability)

Das DBS garantiert die Dauerhaftigkeit bzw. Persistenz erfolgreicher Transaktionen, deren Operationen vollständig ausgeführt wurden. Dies bedeutet, daß Änderungen dieser Transaktionen alle künftigen Fehler überleben, insbesondere auch Systemabstürze oder Externspeicherausfälle. Hierzu sind gegebenenfalls die Änderungen seitens des DBS im Rahmen einer *Redo-Recovery* zu wiederholen. Dafür sind wiederum geeignete Logging-Maßnahmen erforderlich, insbesondere sind vor Abschluß einer Transaktion die für die Recovery benötigten Informationen zu protokollieren.

Die automatische Gewährleistung der ACID-Eigenschaften durch das DBS bedeuten für den Benutzer bzw. Anwendungsprogrammierer eine erhebliche Erleichterung. Insbesondere kann bei der Entwicklung von Anwendungen aufgrund der Atomaritäts- und Dauerhaftigkeitszusicherungen von einer fehlerfreien Umgebung ausgegangen werden. So ist nach Bestätigung des erfolgreichen Transaktionsendes das weitere „Überleben“ der Änderungen gesichert. Tritt während der Transaktionsausführung ein Fehler auf, erfolgt ein automatisches Zurücksetzen der Transaktion, um wieder einen konsistenten DB-Zustand herzustellen. Es entfällt somit auch eine sehr aufwendige manuelle Ermittlung, welche Änderungen bis zu dem Fehlerzeitpunkt schon ausgeführt wurden, um diese zu eliminieren bzw. die restlichen Änderungen noch zur Ausführung zu bringen[1]. Durch die Undo-Recovery des DBS genügt zur Durchführung der unterbrochenen Änderungen ein erneuter Start der Transaktion. Die Isolationszusicherung gestattet die Datenbanknutzung wie im Einbenutzerbetrieb; potentielle Nebenwirkungen gleichzeitiger Datenbankzugriffe sind in der Anwendung nicht abzufangen. Schließlich entfällt aufgrund der automatischen Überwachung von Integritätsbedingungen die Durchführung entsprechender Überprüfungen in den Anwendungsprogrammen.

---

[1] Dies würde allein schon dadurch nahezu unmöglich werden, da eine Änderungsoperation im DBS i. allg. mehrere Teiländerungen umfaßt, die z. B. durch einen Hardware-Fehler an beliebiger Stelle unterbrochen werden können.

Die anwendungsseitige Behandlung von Fehlern und Mehrbenutzereffekten sowie Überprüfung von Integritätsbedingungen würde zudem eine i. allg. inakzeptable Abhängigkeit der Datenbankkonsistenz zur Korrektheit der Anwendungen bedeuten. Zudem verbietet sich eine solche Vorgehensweise schon deshalb, da somit die entsprechenden Kontrollaufgaben redundant in zahlreichen Anwendungen realisiert werden müßten. Allerdings bleibt eine Mitverantwortung des Programmierers insbesondere hinsichtlich der semantischen Integrität der Datenbank bestehen. Denn i. allg. wird schon aus Aufwandsgründen über die definierten Integritätsbedingungen die vollständige Übereinstimmung der Datenbank mit dem modellierten Realitätsausschnitt nicht garantiert werden können. Vor allem ist es Aufgabe des Anwendungsprogrammierers, die Zuordnung der DB-Operationen zu einer Transaktion so festzulegen, daß die damit verbundenen Änderungen tatsächlich einem konsistenzerhaltenden Vorgang der Realität entsprechen (vgl. obiges Beispiel der Umbuchung).

Das DBS kann über die Prüfung der Integritätsbedingungen hinaus keine Zusicherungen zur logischen Konsistenz treffen, sondern geht ansonsten von der Korrektheit der Anwendungen aus. Wird nachträglich ein logischer Fehler in der Anwendung festgestellt, kann daher das DBS auch keine Unterstützung zum Rückgängigmachen der jeweiligen Änderungen anbieten, da diese aufgrund der Dauerhaftigkeitszusicherung aus DBS-Sicht Bestand haben müssen. Die Korrektur solcher Änderungen kann somit nur auf Anwendungsebene durch kompensierende Transaktionen erfolgen.

13.2 Benutzerschnittstelle

Die weitgehende Kontrolle der Transaktionsausführung durch das DBS gestattet eine einfache Benutzerschnittstelle zur Transaktionsverwaltung. Benutzerseitig sind im wesentlichen nur die Transaktionsgrenzen bekanntzumachen, damit das DBS weiß, welche DB-Operationen als Ausführungseinheit zu behandeln sind. Hierzu stehen an der Anwendungsschnittstelle i. allg. drei Operationen zur Verfügung, nämlich

- BEGIN OF TRANSACTION (BOT)[2] zur Kennzeichnung des Beginns einer neuen Transaktion
- COMMIT zur erfolgreichen Beendigung der Transaktion sowie
- ROLLBACK, um die Transaktion abzubrechen, z. B. aufgrund von erkannten Eingabefehlern oder sonstigen in der Anwendung erkannten Ausnahmesituationen, welche der weiteren Ausführung der Transaktion entgegenstehen.

Damit ergeben sich aus Benutzersicht drei Fälle hinsichtlich der Beendigung einer Transaktion, die in Abb. 13.2 veranschaulicht sind. Der Normalfall stellt die erfolgreiche durch Ausführung der COMMIT-Operation abgeschlossene Transaktionsausführung dar. Im Fehlerfall

---

[2] In der DB-Sprache SQL92 [DATE97] erfolgt der Beginn einer Transaktion implizit bei Ausführung der ersten DB-Operation eines Benutzers. Zur Beendigung einer Transaktion stehen die Anweisungen COMMIT WORK und ROLLBACK WORK zur Verfügung.

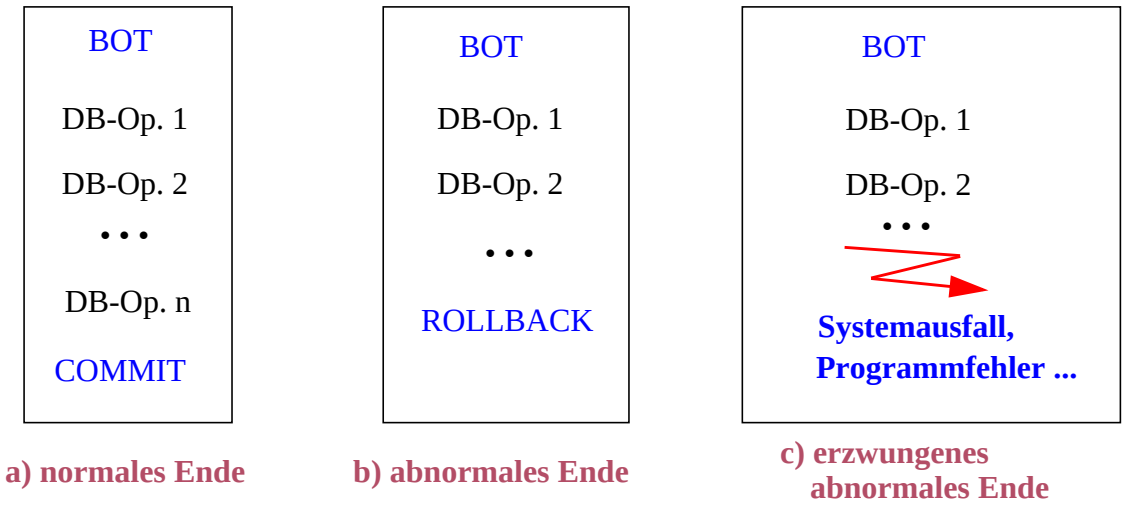


Abb. 13.2: Mögliche Ausgänge der Transaktionsbearbeitung

ist zu unterscheiden, zwischen der abnormalen Beendigung aufgrund einer expliziten ROLLBACK-Anweisung der Anwendung und der aufgrund eines Fehlers erzwungenen Rücksetzung der Transaktion. Eine Erweiterung der Benutzerschnittstelle – allerdings unter Abkehr vom Alles-oder-Nichts-Paradigma – ergibt sich bei Einführung transaktionsinterner Rücksetzpunkte (s. Abschnitt 16.2).

Abb. 13.3 zeigt den Kontrollfluß zwischen Anwendungsprogramm und DBS im Normalfall. Während der Ausführung der Transaktion sind Logging-Maßnahmen erforderlich, um zu Änderungsoperationen geeignete Informationen zu protokollieren, insbesondere um die Rücksetzbarkeit (Undo-Recovery) der Transaktion sicherzustellen. Weiterhin können bestimmte Integritätsbedingungen wie etwa Wertebereichsbeschränkungen von Attributen unmittelbar bei der Ausführung von Änderungsoperationen vom DBS geprüft werden. Besondere Bedeutung zur Transaktionsverwaltung kommt der Commit-Verarbeitung zu, die in zwei Phasen abläuft (*Zwei-Phasen-Commit*). Hierbei wird in Phase 1 zunächst geprüft, ob alle noch nicht direkt geprüften und von den vorgenommenen Änderungen betroffenen Integritätsbedingungen erfüllt sind. Ist dies der Fall, werden die für die Wiederholbarkeit der Änderungen (Redo-Recovery) erforderlichen Log-Daten gesichert. Danach ist das erfolgreiche Ende der Transaktion sichergestellt. In Phase 2 werden dann die vorgenommenen Änderungen in der Datenbank allgemein zugänglich gemacht (z. B. durch Freigabe von Sperren) und das erfolgreiche Transaktionsende der Anwendung bestätigt. Ein Zurücksetzen der Transaktion erfolgt, wenn die Transaktion durch einen Fehler vor oder während der Commit-Phase 1 unterbrochen wird oder die Verletzung einer Integritätsbedingung festgestellt wird.

Aufgabe der Transaktionsverwaltung des DBS ist es, die Abarbeitung der Transaktionen zu kontrollieren und die zur Wahrung der ACID-Eigenschaften benötigten Funktionen bereitzustellen. Dies erfordert Funktionen zur Synchronisation (Sicherstellung der Isolation), für Logging und Recovery (Atomarität und Dauerhaftigkeit) sowie zur Integritätskontrolle (Konsistenz). Zwischen diesen Funktionen bestehen zahlreiche Wechselwirkungen und Abhängig-

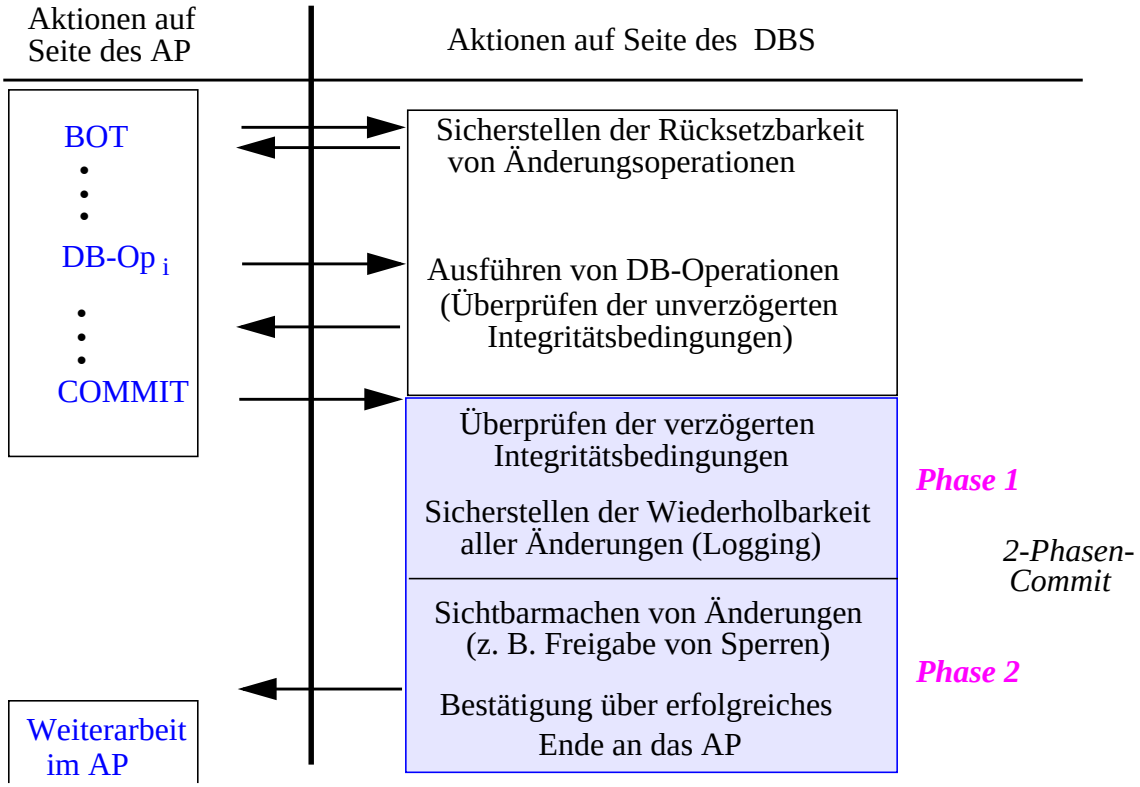


Abb. 13.3: Kontrollfluß zwischen Anwendungsprogramm (AP) und DBS

keiten, die für korrekte und leistungsfähige Implementierungen zu beachten sind. Ebenso bestehen enge Abhängigkeiten zu anderen DBS-Komponenten wie Pufferverwaltung, Seitenzuordnungs- und Speicherungsstrukturen. Die enge Beziehung zwischen Logging und Recovery ist dabei offensichtlich, da zum Durchführen der Recovery (Fehlerbehandlung) ausreichende Log-Daten im Normalbetrieb zu sammeln sind. Die Diskussion der Commit-Verarbeitung verdeutlichte Abhängigkeiten zwischen Integritätskontrolle, Logging und Synchronisation (Sichtbarmachen von Änderungen). Auf weitere Abhängigkeiten wird bei der Behandlung der einzelnen Funktionen näher eingegangen, u. a. in Abschnitt 15.3.

13.3 Integritätskontrolle

Aufgaben der Integritätskontrolle sind die Überwachung und Einhaltung der logischen DB-Konsistenz. Grundsätzlich lassen sich diese Aufgaben entweder auf Anwendungsebene oder zentral durch das DBS realisieren. Wie bei der Diskussion der ACID-Eigenschaften deutlich wurde, ist dabei eine möglichst weitgehende Integritätskontrolle durch das DBS anzustreben, insbesondere um die Abhängigkeiten zur Korrektheit der Anwendungsprogramme zu reduzieren. Die zentrale Definition und Überwachung semantischer Integritätsbedingungen ermöglicht zudem eine Vereinfachung der Anwendungsentwicklung, da die entsprechenden Prüfun-

gen nicht (redundant) in zahlreichen Programmen auszuprogrammieren sind, welche zudem bei Änderungen in den Integritätsbedingungen mit hohem Aufwand anzupassen wären (schlechte Wartbarkeit). Auch kann die DBS-interne Überprüfung von Integritätsbedingungen zu Leistungsvorteilen führen, insbesondere durch Einsparung von DBS-Aufrufen. Schließlich wird auch eine Integritätskontrolle für DB-Änderungen erreicht, welche nicht über den Aufruf von Transaktionsprogrammen, sondern direkt (ad hoc)[3] vorgenommen werden. In kommerziellen DBS stehen mittlerweile zunehmend Mechanismen zur Integritätskontrolle bereit, zumal im SQL92-Standard [DATE97] umfassende Möglichkeiten zur Spezifikation von Integritätsbedingungen festgelegt wurden.

Wir diskutieren im folgenden kurz unterschiedliche Typen von Integritätsbedingungen sowie ihre Unterstützung in SQL. Es folgt eine Diskussion von Triggern und ECA-Regeln, welche nach integritätsgefährdenden Änderungen eine aktive Reaktion ermöglichen, um die DB-Konsistenz aufrechtzuerhalten. Weiterhin wird auf Implementierungsaspekte der DBS-seitigen Integritätskontrolle eingegangen (Abschnitt 13.3.3).

13.3.1 Arten von Integritätsbedingungen

Semantische Integritätsbedingungen lassen sich hinsichtlich mehrerer Kategorien klassifizieren:

- Modellinhärente vs. sonstige (modellunabhängige) Integritätsbedingungen
  Modellinhärente Bedingungen folgen aus der Strukturbeschreibung des jeweiligen Datenmodells und sind somit für alle Anwendungen zu gewährleisten. Im Falle des relationalen Datenmodells sind dies die Relationalen Invarianten, also die Primärschlüsseleigenschaft sowie die referentielle Integrität für Fremdschlüssel. Außerdem sind die zulässigen Werte eines Attributs durch einen Definitionsbereich (Domain, Datentyp) zu beschränken.
- Reichweite der Bedingung
  Wichtige Fälle mit jeweils unterschiedlicher Datengranularität sind *Attributwert-Bedingungen* (z. B. Geburtsjahr > 1900), *Satzbedingungen* (z. B. Geburtsdatum < Einstellungsdatum), *Satztyp-Bedingungen* (z. B. Eindeutigkeit von Attributwerten) sowie *satztypübergreifende Bedingungen* (z. B. referentielle Integrität zwischen verschiedenen Tabellen)
- Statische vs. dynamische Bedingungen
  Statische Bedingungen (Zustandsbedingungen) beschränken zulässige Zustände der Datenbank (z. B. Gehalt < 500.000), während *dynamische Integritätsbedingungen* (Übergangsbedingungen) zulässige Zustandsübergänge festlegen (z. B. Gehalt darf nicht kleiner werden). Eine Variante dynamischer sind *temporale Integritätsbedingungen*, welche längerfristige Abläufe betreffen (z. B. Gehalt darf innerhalb von 3 Jahren nicht um mehr als 25 % steigen).
- Zeitpunkt der Überprüfbarkeit: *unverzögerte* vs. *verzögerte Integritätsbedingungen* (s. o.)

[3] Solche direkten Änderungen werden vom DBS implizit als Transaktionen abgewickelt.

SQL92 gestattet die deklarative Spezifikation all dieser Typen von Integritätsbedingungen mit Ausnahme dynamischer Integritätsbedingungen. Im einzelnen stehen hierzu folgende Sprachmittel zur Verfügung (selbsterklärende Beispiele dazu zeigt Abb. 13.4):

- Spezifikation relationaler Invarianten durch *Primary Key*- und *Foreign Key-Klauseln* bei der Definition von Tabellen. Für Fremdschlüssel können unterschiedliche Reaktionsmöglichkeiten für den Wegfall (Löschung, Änderung) eines referenzierten Satzes bzw. Primärschlüssels deklarativ festgelegt werden (CASCADE, SET NULL, SET DEFAULT, NO ACTION).
- Festlegung von Wertebereichen für Attribute durch Angabe eines Datentyps bzw. Domains. Dabei besteht die Möglichkeit, einen Default-Wert zu spezifizieren, Eindeutigkeit von Attributwerten zu verlangen (UNIQUE), Nullwerte auszuschließen (NOT NULL) sowie über eine CHECK-Klausel allgemeine Wertebereichsbeschränkungen festzulegen.
- Spezifikation allgemeiner, z. B. tabellenübergreifender Bedingungen durch die Anweisung *CREATE ASSERTION*.
- Für jede Integritätsbedingung kann eine direkte (IMMEDIATE) oder verzögerte (DEFERRED) Überwachung erreicht werden.

Zur Einhaltung der DB-Konsistenz (integrity enforcement) ist die Standard-Reaktion auf die erkannte Verletzung einer Integritätsbedingung der Abbruch der betreffenden Änderungstransaktion. In [ESWA75] wurde daneben für die Verletzung „schwacher Integritätsbedingungen“ (soft assertions) eine anwendungsseitige Reaktionsmöglichkeit vorgesehen, z. B. für interaktive Interventionsmöglichkeiten des Benutzers, Ausgabe von Fehlermeldungen, Aufruf von Korrekturprogrammen usw. Diese Vorgehensweise erlaubt zwar eine hohe Flexibilität, verlagert jedoch die Verantwortung zur Integritätskontrolle wieder auf die Anwendungsseite und widerspricht somit dem Hauptanliegen DBS-basierter Integritätskon-

```
CREATE TABLE  PERS
      ( PNR       INT  PRIMARY KEY,              // Personalnr.
        GEHALT INT  CHECK (VALUE < 500000),
        ANR       INT  NOT NULL                  // Abteilungsnr.
                       FOREIGN KEY  REFERENCES  ABT
                                ON DELETE CASCADE,
                       //   Löschen der Abteilung löscht Mitarbeiter
        ... );

CREATE ASSERTION A1
   CHECK       (NOT EXISTS (SELECT *
                            FROM ABT
                            WHERE ANR NOT IN
                              (SELECT ANR FROM PERS)))
   DEFERRED;
  // Abteilungen ohne Mitarbeiter werden nicht zugelassen
```

Abb. 13.4: Beispiele für Integritätsbedingungen in SQL92

trolle. Da zudem die Verletzung definierter Integritätsbedingungen mit dem Transaktionskonzept nicht vereinbar ist, wäre eine anwendungsseitige Reaktionsmöglichkeit nur dann akzeptabel, wenn danach vor dem endgültigen Commit die Einhaltung der Integritätsbedingungen erneut überprüft würde und bei Verletzung ein Transaktionsabbruch erfolgt.

Eine Alternative besteht darin, eine DBS-kontrollierte Reaktion auf die Verletzung einer Integritätsbedingung vorzusehen, welche zusammen mit der Integritätsbedingung zu spezifizieren ist. Solche erweiterten Integritätsbedingungen werden als *Integritätsregeln* bezeichnet und enthalten die Festlegung einer Reaktionsmöglichkeit zur Wahrung der DB-Konsistenz. Die in SQL92 vorgesehenen Spezifikationsmöglichkeiten zur Wartung der referentiellen Integrität (CASADE, SET NULL, SET DEFAULT, NO ACTION) sind Beispiele solcher Integritätsregeln (die Default-Reaktion, in SQL3 als RESTRICT-Option explizit zu spezifizieren, bewirkt die Abweisung einer Operation/Transaktion, welche zu einer Verletzung der referentiellen Integrität führt, und entspricht somit der Standard-Reaktion für verletzte Integritätsbedingungen).

Solche Regelmengen gestatten eine deklarative Beschreibung von Situationen/Ereignissen und den zugehörigen Reaktionen, ohne dabei die Programmabläufe, in denen sie auftreten können, vorausplanen und spezifizieren zu müssen. Die Erkennung solcher Situationen/Ereignisse und die prozedurale Umsetzung der spezifizierten Reaktion wird dabei dem DBS überlassen. Weiterhin gestatten sie eine leichte Erweiterbarkeit, was Hinzufügen, Löschen und Austauschen von Regeln sehr einfach gestaltet. Allerdings können Abhängigkeiten zwischen Regeln auftreten, wenn sie auf gemeinsame Daten Bezug nehmen und dabei Änderungen vollzogen werden. Das wird immer dann zu einem Problem bei der Regelausführung, wenn mehrere Regeln gleichzeitig ausgelöst und diese parallel bearbeitet werden sollen [REIN96].

13.3.2 Trigger-Konzept und ECA-Regeln

Die Überwachung dynamischer Integritätsbedingungen sowie die Spezifikation allgemeiner Integritätsregeln werden durch das *Trigger-Konzept* möglich, das von den meisten DBS bereits unterstützt wird; seine Standardisierung erfolgt jedoch erst in SQL3. Ein Trigger erlaubt für Änderungen auf Tabellen die Definition von automatisch ausgelösten Folgeaktivitäten. Eine derartige Folgeaktivität wird durch eine SQL-Anweisung spezifiziert; durch den Aufruf einer gespeicherten Prozedur (stored procedure) können beliebig umfangreiche Verarbeitungsvorgänge angestoßen werden. Die Ausführung der Folgeaktionen läßt sich von der Gültigkeit bestimmter Bedingungen abhängig machen, wobei zur Bedingungsauswertung die Werte vor und nach der Änderung berücksichtigt werden können. Wie in Abb. 13.5 a unter Verwendung der vorgesehenen SQL3-Syntax gezeigt, erlaubt dies u. a. die Realisierung dynamischer Integritätsbedingungen, indem bei einem unzulässigen Zustandsübergang (WHEN-Klausel) als Folgeaktion die betreffende Transaktion abgebrochen wird (ROLLBACK).

Trigger können auch zur Überwachung statischer Integritätsbedingungen sowie zur Wartung der referentiellen Integrität herangezogen werden. Für einige DBS, welche die SQL92-Konstrukte zur Festlegung von Integritätsbedingungen nur begrenzt unterstützen, ist dies teil-

```
CREATE TRIGGER GEHALTSTEST
    AFTER UPDATE OF GEHALT ON PERS
    REFERENCING OLD AS AltesGehalt,
            NEW AS NeuesGehalt
    WHEN (NeuesGehalt < AltesGehalt)
    ROLLBACK;
```

a) Realisierung einer dynamischen Integritätsbedingung
(Gehalt darf nicht kleiner werden)

```
CREATE TRIGGER MITARBEITERLÖSCHEN
    AFTER DELETE ON ABT
    REFERENCING OLD AS A
    DELETE FROM PERS P WHERE P.ANR = A.ANR;
```

b)Wartung der referentiellen Integrität

Abb. 13.5: Beispiele zum Trigger-Einsatz

weise auch die einzige Möglichkeit, die fehlende Funktionalität auszugleichen. In Abb. 13.5 b ist z. B. gezeigt, wie die Fremdschlüsselbedingung aus Abb. 13.4 über einen Trigger realisiert werden kann. Trigger gestatten darüber hinaus die Formulierung von allgemeinen Integritätsregeln, da als Folgeaktion beliebig komplexe Vorgänge zur Herstellung eines konsistenten DB-Zustandes möglich sind oder auch um vor dem Transaktionsabbruch noch Benachrichtigungen an den Benutzer zu liefern. Es handelt sich dabei um eine weitgehend prozedurale Spezifikation, bei welcher der Zeitpunkt der Trigger-Ausführung (i. allg. vor bzw. nach Ausführung einer Änderungsoperation), die Verwendung alter und neuer DB-Werte sowie die einzelnen Aktionen genau festzulegen sind.

Ein derartiger Einsatz von Triggern hat im Vergleich zur Nutzung der deklarativen SQL92-Konstrukte wesentliche Nachteile:

- Im Vergleich zur deklarativen Spezifikation von Integritätsbedingungen ist die prozedurale Definition von Triggern auf einer geringeren Abstraktionsstufe und damit umständlicher und fehleranfälliger. Insbesondere sind Trigger in derzeitigen Implementierungen jeweils an eine der drei Änderungsoperationen (Update, Insert, Delete) einer Tabelle gekoppelt. Dies erfordert oft die Festlegung von mehreren Triggern pro Integritätsbedingung. So müßten für eine einfache Wertebereichsbeschränkung (z. B. Gehalt < 500000) schon zwei Trigger (für Insert und Update) definiert werden.
- In derzeitigen Implementierungen sowie im SQL3-Standard wird die Trigger-Ausführung unmittelbar vor oder nach (BEFORE/AFTER) der betreffenden Änderungsoperation ausgeführt. Eine Ausführung am Transaktionsende, wie für verzögerte Integritätsbedingungen erforderlich, ist nicht möglich. Damit kann eine äußerst wichtige

Klasse von Integritätsbedingungen mit derzeitigen Trigger-Implementierungen nicht abgedeckt werden.

– Die von Triggern durchgeführten Änderungen können selbst wieder Trigger auslösen, wodurch die Auswirkungen auf die Datenbank oft nur schwer einschätzbar sind sowie sich die Gefahr zyklischer und möglicherweise nicht terminierender Aktivierungen ergibt. Weiterhin besteht das Problem der Konfluenz, inwieweit also der Effekt von Triggern unabhängig von der Abarbeitungsreihenfolge parallel aktivierter Trigger ist. Zur korrekten Erstellung von Transaktionsprogrammen muß der Anwendungsprogrammierer jedoch die Wirkung aller definierter Trigger genau kennen, z. B. um eine doppelte Durchführung bestimmter Änderungen zu vermeiden. Das Hinzufügen eines neuen Trigger kann so auch leicht zu unerwarteten Nebenwirkungen mit vorhandenen Transaktionsprogrammen führen.
  In [SIMO95] wird bemängelt, daß keine Tools existieren, um die Auswirkungen von Triggern zu erkennen und die Entwicklung sicherer Anwendungen bei Einsatz von Triggern zu unterstützen. Eine solche Unterstützung ist sicher nur schwer erreichbar; ihre Notwendigkeit steigt jedoch mit den Größe der Anwendung und der Anzahl benötigter Trigger.

Aus diesen Überlegungen folgt, daß benutzerseitig der Verwendung deklarativ definierter Integritätsbedingungen Vorrang einzuräumen ist. Der Trigger-Einsatz sollte auf Einsatzbereiche begrenzt werden sollte, die darüber nicht abgedeckt werden können, insbesondere die Realisierung dynamischer Integritätsbedingungen und spezifischer Integritätsregeln. Allerdings kann das Trigger-Konzept aufgrund seiner operationalen Semantik DBS-intern als Implementierungsmöglichkeit für nahezu alle Integritätsbedingungen genutzt werden (s. u.). Darüber hinaus gestattet die hohe Flexibilität des Trigger-Konzeptes, es neben der Integritätskontrolle zur Realisierung vielfältiger Kontrollaufgaben heranzuziehen, z. B. Aktualisierung replizierter Datenbestände, Protokollierung von DB-Zugriffen für Auditing-Zwecke, automatische Auslösung von Benachrichtigungen, usw. Ähnlich wie bei der Integritätskontrolle ist es äußerst wünschenswert, diese Aufgaben zentralisiert durch das DBS anstatt über Anwendungsprogramme zu realisieren (Umgehen redundanter Implementierungen in verschiedenen Anwendungsprogrammen, bessere Wartbarkeit und Modularisierung von Anwendungen, Leistungsvorteile, ...).

Eine Verallgemeinerung der Trigger-Funktionalität wird von *aktiven Datenbanksystemen* [JASP98, WIDO96a] verfolgt. Sie verwenden anstelle von Triggern sog. ECA-Regeln, welche durch drei Komponenten gekennzeichnet sind:

– *Event (E)*: regelauslösendes Ereignis. Neben Änderungsoperationen werden weitere Ereignisse zugelassen, z. B. lesende DB-Zugriffe, Beginn/Ende von Transaktionen, bestimmte absolute oder relative Zeitpunkte, usw. Durch Definition komplexer Ereignisse (z. B. Konjunktion oder Disjunktion von Teilereignissen) kann die Anzahl von Regeln deutlich reduziert werden.
– *Condition (C)*: optionale Bedingung (analog zur WHEN-Klausel von Triggern)
– *Action (A)*: auszuführende Aktionen.

Zwischen Event, Bedingungsauswertung sowie Ausführung der Aktionen wurden unterschiedliche „Kopplungsmodi" vorgeschlagen [HSU88]. Damit wird es z. B. möglich, Bedingungsauswertung und Ausführung von Aktionen vom auslösenden Ereignis zu entkoppeln, um sie verzögert am Transaktionsende oder gar innerhalb separater Transaktionen durchzuführen. Einige Nachteile des Trigger-Ansatzes können somit umgangen werden. Auf der anderen Seite verstärkt die höhere Ausdrucksmächtigkeit die Probleme bezüglich der sicheren Nutzung und erhöht die Notwendigkeit einer Tool-Unterstützung beim Entwurf von Anwendungen und ECA-Regeln. Überlegungen zum Entwurf von Triggern bzw. Regeln finden sich u. a. in [ZANI97].

13.3.3 Implementierungsaspekte

Die DBS-seitige Durchführung der Integritätskontrolle erfordert die Generierung und Ausführung zusätzlicher DB-Operationen für Änderungstransaktionen, um die Verletzung definierter Integritätsbedingungen zu erkennen. Hierzu ist vom DBS zu entscheiden,

- für welche Operationen und Transaktionen welche Überprüfungen vorzunehmen sind
- wann die Überprüfungen durchgeführt werden sollen (direkt bei der Änderungsoperation oder verzögert am Transaktionsende)
- wie die Überprüfungen realisiert werden (Erstellung eines Ausführungsplans für die durchzuführenden Prüfoperationen).

Die Unterstützung von Integritätsregeln bringt wesentlich weitergehende Anforderungen mit sich, insbesondere eine aufwendigere Überwachung von regelauslösenden Ereignissen, die effiziente Bedingungsauswertung und Ausführung der ausgelösten Folgeoperationen sowie die Überwachung der dabei rekursiv verursachten Regelaktivierungen.

Für Transaktionsprogramme können im Rahmen der Vorübersetzung durch einen *Prä-Compiler* Operationen zur Prüfung von Integritätsbedingungen sowie zur Reaktion auf Integritätsverletzungen eingebettet werden. Voraussetzung dafür ist – wie bei statischem eingebettetem SQL der Fall – daß neben den jeweiligen Änderungsoperationen auch die Namen der betroffenen Tabellen und Attribute erkannt werden. Die Prüfanweisungen können sich dabei auf einzelne Änderungsoperationen oder aber das gesamte Transaktionsprogramm beziehen, etwa um verzögerte Integritätsbedingungen zu überwachen.

Da mit dieser Technik jedoch keine dynamischen SQL-Anweisungen sowie direkte (Adhoc-) Änderungen unterstützt werden können, ist eine weitergehende Integritätskontrolle erforderlich. So sind bei der Übersetzung und Optimierung von Änderungsoperationen im Ausführungsplan direkt entsprechende Prüfoperationen zu erzeugen, oder es erfolgt der Aufruf dynamischer Prüfroutinen eines eigenen Integritäts-Subsystems (bzw. eines Regel-Subsystems). Aus Leistungsgründen ist beim Aufruf solcher Prüfroutinen möglichst viel Kontextinformation zur Überprüfung bereitzustellen, um die Anzahl zu prüfender Integritätsbedingungen auf ein Minimum zu beschränken. Insbesondere sollte es möglich sein, die Prüfungen auf die von einer Änderung betroffenen Daten zu beschränken, um nicht

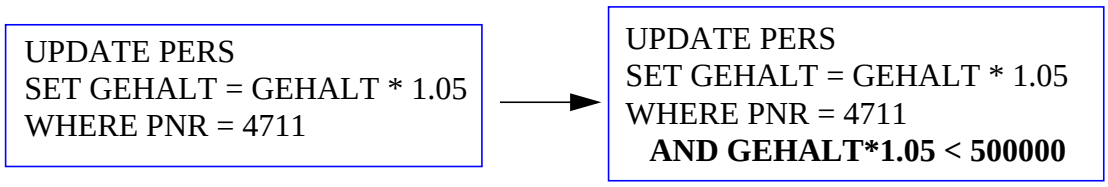


Abb. 13.6: Beispiel zur Anfragemodifikation
(Wahrung der Integritätsbedingung GEHALT < 500000)

große Teile der Datenbank auswerten zu müssen (z. B. bei Einfügen eines neuen Satzes sind die Integritätsbedingungen nur für diesen zu prüfen, nicht für die gesamte Tabelle).

Eine im Rahmen der Anfrageoptimierung nutzbare Technik zur Integritätskontrolle ist die für das DBS Ingres vorgeschlagene Anfragemodifikation (*Query Modification)* [STON75]. Dabei erfolgt eine Transformation von Änderungsoperationen durch Hinzunahme von Prädikaten einzuhaltender Integritätsbedingungen, so daß durch die resultierende Operation eine Verletzung der betreffenden Bedingungen umgangen wird. Diese in Abb. 13.6 beispielhaft illustrierte Methode ist jedoch nur für einfache statische und unverzögerte Integritätsbedingungen nutzbar. Sie führt zudem entgegen der üblichen Vorgehensweise nicht zum Abbruch der Transaktion, sondern verhindert lediglich die Ausführung einer integritätsgefährdenden Änderungsanweisung, was bei Transaktionen mit mehreren Operationen zu unerwarteten Effekten führen kann. Der offensichtliche Vorteil der Query-Modifikation liegt darin, daß die Integrität mit den vorhandenen Ausführungsmechanismen des DBS sichergestellt wird.

Eine allgemeine Realisierungsmöglichkeit zur Integritätskontrolle besteht darin, Trigger bzw. ECA-Regeln innerhalb des DBS heranzuziehen, selbst wenn eine deklarative Spezifikation der Integritätsbedingungen erfolgt. Denn wie erwähnt lassen sich darüber die meisten (einschließlich dynamischer) Integritätsbedingungen realisieren, wobei jedoch auch verzögerte Bedingungen zu unterstützen sind. Die Implementierung des ECA-Konzeptes ermöglicht darüber hinaus die Realisierung zahlreicher weiterer Kontrollaufgaben aktiver DBS. Die effektive Generierung von ECA-Regeln aus einer deklarativen Spezifikation von Integritätsbedingungen ist im allgemeinen Fall ein komplexes Problem und wurde u. a. in [GREF93, CERI94, GERT94] untersucht. Die Realisierung eines umfassenden Regelsystems steht für kommerzielle DBS noch aus, jedoch gibt es erste Implementierungen im Rahmen von Prototypen wie Postgres und Starburst.

In Postgres wurden zwei unterschiedliche Arten der Regelbehandlung realisiert [STON90, POTA96], wobei für jede Regel eine der beiden Realisierungsmöglichkeiten auszuwählen ist. Defaultgemäß wird ein zur Laufzeit wirksam werdendes und auf Tupelebene arbeitendes Regelsystem verwendet (tuple level rule system). Dabei werden bei der Spezifikation einer neuen Regel alle Tupel der Datenbank (bzw. ganze Relationen), welche die Qualifizierungsbedingung der Regel zu diesem Zeitpunkt erfüllen, mit einer permanenten Markierung für die Regel („rule lock“) versehen. Bei der Abarbeitung von Operationen wird für ein Tupel für alle Re-

```
CREATE RULE GehaltsCheck ON PERS
WHEN INSERTED, UPDATED (GEHALT) // zusammengesetztes Event (Disjunktion)
IF      EXISTS (SELECT *                    // Bedingung (Condition)
                FROM inserted UNION new-updated
                WHERE GEHALT >= 500000)
THEN ROLLBACK // Aktion
```

Abb. 13.7: Starburst-Regel zur Wahrung der Integritätsbedingung GEHALT < 500000

geln, zu denen eine Markierung vorgefunden wird, eine entsprechende Regelauswertung angestoßen. Als Alternative wird ein zur Übersetzungszeit von DB-Operationen wirkender Query-Rewrite-Ansatz zur Regelbehandlung unterstützt, der eine Erweiterung der diskutierten Query-Modifikation darstellt. Die Entwickler räumen ein, daß die beiden Ansätze unter Umständen zu unterschiedlichen Ergebnissen führen können. In der vorgenommenen Implementierung ist die Regelauswertung auch direkt an DB-Operationen gekoppelt; eine als machbar angesehene verzögerte Auswertung von Regeln am Transaktionsende wurde nicht realisiert.

Demgegenüber liegt der Schwerpunkt des Regelsystems für den DBS-Prototyp Starburst in der mengenorientierten Regelauswertung am Ende von Transaktionen [WIDO91, WIDO96b]. Pro Transaktion werden dabei für jede geänderte Tabelle vier temporäre Relationen (transition tables) mit den von Änderungen betroffenen Sätzen geführt. In den Tabellen *deleted* und *inserted* werden dabei die gelöschten und eingefügten Sätze aufgenommen, während die Tabellen *old-updated* und *new-updated* für geänderte Sätze die Werte vor und nach der Änderung enthalten. Diese Tabellen ermöglichen eine Begrenzung der Regelauswertung auf die minimale Menge relevanter Daten. Die Nutzung dieser Tabellen wird innerhalb der Regeln festgelegt, welche für deklarativ spezifizierte Integritätsbedingungen im allgemeinen automatisch vom Compiler erzeugt werden können (siehe Beispiel in Abb. 13.7 unter Verwendung der Starburst-Syntax). In der Realisierung wurde der effizienten Verarbeitung rekursiv ausgelöster Regeln besondere Beachtung gewidmet, worauf hier jedoch nicht näher eingegangen werden kann.

Die zur Integritätskontrolle erforderlichen DB-Operationen führen natürlich generell zu einer Erhöhung der DBS-Last, welche auch beim physischen DB-Entwurf zu berücksichtigen ist. Anderenfalls kann die Überprüfung von Integritätsbedingungen zu intolerabel häufigen Relationen-Scans mit zahlreichen E/A-Vorgängen und hohem Sperrpotential führen. In [HÄRD92b, HÄRD96] wird die Zugriffspfadunterstützung zur effizienten Gewährleistung der Relationalen Invarianten untersucht. Es stellt sich u. a. heraus, daß aus Leistungsgründen dabei für jeden Primär- und Fremdschlüssel eine Indexunterstützung erforderlich ist. In [SIMO95] wurden für Trigger kommerzieller DBS zum Teil erhebliche Leistungsprobleme festgestellt, mitbedingt durch eine hohe Ausführungsfrequenz sowie verschärfte Sperrengpässe. In [BRÜC97] konnte für eine konkrete Anwendung der Einsatz von Triggern zur Integri-

tätskontrolle im Einbenutzerbetrieb etwas bessere Leistungsmerkmale als eine anwendungsseitige Realisierung bewirken, insbesondere aufgrund der Einsparung an DBS-Aufrufen; noch bessere Antwortzeiten wurden jedoch bei Einsatz deklarativer Integritätsbedingungen gemessen.

Literatur

ABBO89 Abbott, R., Garcia-Molina, H.: Scheduling Real-Time Transactions with Disk Resident Data. Proc. 15th Int. Conf. VLDB. Amsterdam. 1989. 385–396

AGHI82 Aghili, H., Severance, D.G.: A Practical Guide to the Design of Differential Files for Recovery of On-Line Databases. ACM Trans. Database Syst. 7:4. 1982. 540–565

AGRA83 Agrawal, R., Carey, M.J., DeWitt, D.J.: Deadlock Detection is Cheap. ACM SIGMOD Record 13:2. 1983. 19–34

AGRA87a Agrawal, R., Carey, M.J., Livny, M.: Concurrency Control Performance Modeling: Alternatives and Implications. ACM Trans. Database Syst. 12:4. 1987. 609–654

AGRA87b Agrawal, R., Carey, M.J., McVoy, L.W.: The Performance of Alternative Strategies for Dealing with Deadlocks in Database Management Systems. IEEE Trans. Software Eng. 13:12. 1987. 1348–1363

AGRA89 Agrawal, D., Sengupta, S.: Modular Synchronization in Multiversion Databases: Version Control and Concurrency Control. Proc. ACM SIGMOD Conf. Portland. 1989. 408–417

AGRA93 Agrawal, R., Imielinski, T., Swami, A.: Mining Association Rules between Sets of Items in Large Databases. Proc. ACM SIGMOD Conf. Washington. D.C. 1993. 207–216

ANDE94 Anderson, J.T., Stonebraker, M.: Sequoia 2000 Metadata Schema for Satellite Images. SIGMOD Record 23:4. 1994. 42–48

ALLE82 Allen, F.W., Loomis, E.S., Mannino, M.V.: The Integrated Dictionary/Directory System. ACM Computing Surv. 14:2. 1982. 245–286

ALON97 Alonso, G. et al.: Advanced Transaction Models in Workflow Contexts. Proc. 13th Int. Conf. Data Engineering. Los Angeles. 1987. 574–581

APER97 Apers, P.M.G., Blanken, H.M., Houtsma, M.A.W. (eds.): Multimedia Databases in Perspective. Springer. 1997

ARUN98 Arun, G., Joshi, A.: KODA – The Architecture and Interface of a Data Model Independent Kernel. Proc. 24th Int. Conf. VLDB. New York. 1998. 671–674

ASAI86 Asai, S.: Semiconductor Memory Trends. Proc. IEEE 74:12. 1986. 1623–1635

ASTR76 Astrahan, M.M., et al.: System R: Relational Approach to Database Management. ACM Trans. Database Syst. 1:1. 1976. 97–137

BABA77 Babad, J.M.: A Record and File Partitioning Model. Comm. ACM 20:1. 1977. 22–31

BACH74 Bachman, C.W.: Implementation Techniques for Data Structure Sets. Data Base Management Systems. Jardine, D.A. (ed.). North Holland. 1974. 147–157

BANC85 Bancilhon, F., Kim, W., Korth, H.F.: A Model of CAD Transactions. Proc. 11th Int. Conf. VLDB. 1985. 25–33

BARG91 Barghouti, N.S., Kaiser, G.E.: Concurrency Control in Advanced Database Applications. ACM Computing Surv. 23:3. 1991. 269–317

BARG95 Barga, R., Pu, C.: A Practical and Modular Method to Implement Extended Transaction Models. Proc. 21st Int. Conf. VLDB. Zurich. 1995. 206–217

BATO79 Batory, D.S.: On Searching Transposed Files. ACM Trans. Database Syst. 4:4. 1979. 531–544

BATO85 Batory, D.S.: Modeling the Storage Architectures of Commercial Database Systems. ACM Trans. Database Syst. 10:3. 1985. 463–528

BAYE72 Bayer, R., McCreight, E.M.: Organization and Maintenance of Large Ordered Indexes. Acta Informatica 1:3. 1972. 173–189

BAYE76 Bayer, R., Metzger, J.K.: On the Encipherment of Search Trees and Random Access Files. ACM Trans. Database Syst. 1:1. 1976. 37–52

BAYE77a Bayer, R., Schkolnick, M.: Concurrency of Operations on B-Trees. Acta Informatica 9:1. 1977. 1–21

BAYE77b Bayer, R., Unterauer, K.: Prefix-B-Trees. ACM Trans. Database Syst. 2:1. 1977. 11–26

BAYE80 Bayer, R., Heller, H., Reiser, A.: Parallelism and Recovery in Database Systems. ACM Trans. Database Syst. 5:2. 1980. 139–156

BAYE82 Bayer, R. et al.: Dynamic Timestamp Allocation for Transactions in Database Systems. Distributed Data Bases. North-Holland. 1982. 9–20

BAYE96 Bayer, R.: The Universal B-Tree for Multidimensional Indexing. Interner Bericht TUM-I9637. Technische Universität München. Nov. 1996

BECK90 Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. Proc. ACM SIGMOD Conf. Atlantic City. NJ. 1990. 322–331

BECK95 Becker, W.: Das HiCon-Modell: Dynamische Lastverteilung für datenintensive Anwendungen auf Rechnernetzen, Informatik – Forschung und Entwicklung 10:1. 1995. 14–25

BELA66 Belady, L.A.: A Study of Replacement Algorithms for Virtual Storage Computers. IBM Sys. J. 5:2. 1966. 78–101

BENT75 Bentley, J.L.: Multi-Dimensional Search Trees used for Associative Searching. Comm. ACM 18:9. 1975. 509–517

BENZ89 Benzaken, V., Delobel, C.: Dynamic Clustering Strategies in the O2 Object-Oriented Database System. Altair. BP 105. 78153 Le Chesnay Cedex. France. 1989. 1–27

BERC96 Berchtold, S., Keim, D.A., Kriegel, H.-P.: The X-Tree: An Index Structure for High-Dimensional Data. Proc. 22nd Int. Conf. VLDB. Bombay. 1996. 28–39

BERC98 Berchtold, S., Bohm, Ch., Kriegel, H.-P.: The Pyramid Technique: Towards Breaking the Course of Dimensionality. Proc. ACM SIGMOD Conf. Seattle. 1998. 142–153

BERE95 Berenson, H. et al.: A Critique of ANSI SQL Isolation Levels. Proc. ACM SIGMOD Conf. San Jose. 1995. 1–10

BERN87 Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison Wesley. 1987

BERN90 Bernstein, P.A., Hsu, M., Mann, B.: Implementing Recoverable Requests Using Queues. Proc. ACM SIGMOD Conf. Atlantic City. NJ. 1990. 112–122

BERN97 Bernstein, P.A., Newcomer : Transaction Processing. Morgan Kaufmann. 1997

BERR89 Berra, P.B., et al.: The Impact of Optics on Data and Knowledge Base Systems. IEEE Trans. Knowledge and Data Eng. 1:1. 1989. 111–132

BERT89 Bertino, E., Kim, W.: Indexing Techniques for Queries on Nested Objects. IEEE Trans. Knowledge and Data Eng. 1:2. 1989. 196–214

BEST97 Bestavros, A., Fay-Wolfe, V.: Real-Time Database and Information Systems: Research Advances. Kluwer. 1997

BHID88 Bhide, A.: An Analysis of Three Transaction Processing Architectures. Proc. 14th Int. Conf. VLDB. Los Angeles. 1988. 339–350

BILI92 Biliris, A.: The Performance of Three Database Storage Structures for Managing Large Objects. Proc. ACM SIGMOD Conf. San Diego. CA. 1992. 276–285

BILI94 Biliris, A. et al.: ASSET: A System Supporting Extended Transactions. Proc. ACM SIGMOD Conf. Minneapolis. 1994. 44–54

BLAK95 Blakeley, B., Harris, H., Lewis, R.: Messaging and Queuing Using the MQI. McGraw-Hill. 1995

BLAK96 Blakeley, J. A.: Data Access for the Masses through OLE DB. Proc. ACM SIGMOD Conf. Montreal. 1996. 161–172

BLAK97 Blakeley, J. A.: Universal Data Access with OLE DB. Proc. IEEE Spring CompCon Conf. san Jose. 1997. 2–7

BLAS77a Blasgen, M.W., Eswaran, K.P.: Storage and Access in Relational Data Bases. IBM Sys. J. 16:4. 1977. 363–377

BLAS77b Blasgen, M.W., Casey, R.G., Eswaran, K.P.: An Encoding Method for Multifield Sorting and Indexing. Comm. ACM 20:11. 1977. 874–876

BLAS81 Blasgen, M.W. et al.: System R: An Architectural Overview. IBM Sys. J. 20:1. 1981. 41–62

BLOO70 Bloom, B.H.: Space/Time Trade-Offs in Hash Coding with Allowable Errors. Comm. ACM 13:7. 1970. 422–426

BOBE92 Bober, P. M., Carey, M. J.: On Mixing Queries and Transactions via Multiversion Locking. Proc. 8th Int. Conf. on Data Engineering. Tempe. 1992. 535–545

BOHN89 Bohn, V., Wagner, T.: Transaktionsketten – Konzept und Implementierung. Proc. 4. GI-Tagung „Fehlertolerierende Rechensysteme“. Informatik-Fachberichte. Springer. 1989

BOHN91 Bohn, V., Härder, T., Rahm, E.: Extended Memory Support for High Performance Transaction Processing. Proc. 6. GI/ITG-Fachtagung „Messung, Modellierung und Bewertung von Rechensystemen“. München. Informatik-Fachberichte 286. Springer. 1991. 92–108

BORA83 Boral, H., DeWitt, D.J.: Database Machines: An Idea whose Time has Passed? A Critique of the Future of Database Machines. Proc. 3rd Int. Workshop on Database Machines. Munich. 1983. 166–187

BREI92 Breitbart, Y., Garcia-Molina, H., Silberschatz, A.: Overview of Multidatabase Transaction Management. VLDB Journal 1:2. 1992. 181–240

BRID97 Bridge, W., Joshi, A., Keihl, T., Lahiri, J., Loaiza, J., Macnaughton, N.: The Oracle Universal Server Buffer Manager. Proc. 23rd Int. Conf. VLDB. Athens. 1997. 590–594

BRIN93 Brinkhoff, T., Horn, H., Kriegel, H.-P., Schneider, R.: Eine Speicher- und Zugriffsarchitektur zur effizienten Anfragebearbeitung in Geo-Datenbanksystemen. Proc. BTW’93. Braunschweig. Informatik aktuell. Springer. 1993. 356–374

BROD98 Brodie, M.L.: The Cooperative Computing Initiative: A Contribution to the Middleware and Software Technologies. http://www.ccic.gov/ac/#whitepapers

BROW93 Brown, K.P., Carey, M.J., Livny, M.: Managing Memory to Meet Multiclass Workload Response Time Goals. Proc. 19th Int. Conf. VLDB. Dublin. 1993. 328–341

BROW94 Brown, K.P., Metha, M., Carey, M.J., Livny, M.: Towards Automated Performance Tuning for Complex Workloads. Proc. 20th Int. Conf. VLDB. Santiago. 1994. 72–84

BROW96 Brown, K.P., Carey, M.J., Livny, M.: Goal-Oriented Buffer Management Revisited. Proc. ACM SIGMOD Conf. Montreal. 1996. 353–364

BRÜC97 Brüchert, L., Zimmermann, J., Buchmann, A.: Einsatzmöglichkeiten von Triggermechanismen und deren Performanz in einem Wertpapier-Archivierungssystem. Proc. BTW‘97. Ulm. Informatik aktuell. Springer. 1997. 342–351

BUCH89 Buchmann, A. et al. (eds.): Proc. Symp. on the Design and Implementation of Large Spatial Databases. Santa Barbara. LNCS 409. Springer. 1989.

BUTT91 Butterworth P., Otis, A., Stein, J.: The GemStone Object Database Management System. Comm. ACM 34:10. 1991. 64–77

CARE83 Carey, M.: Granularity Hierarchies in Concurrency Control. Proc. ACM Symp. on Principles of Database Systems. Atlanta. 1983. 156–165

CARE86a Carey, M.J., Muhanna, W.A.: The Performance of Multiversion Concurrency Control Algorithms. ACM Trans. Computer Syst. 4:4. 1986. 338–378

CARE86b Carey, M.J., DeWitt, D.J., Richardson, J.E., Shekita, E.J.: Object and File Management in the EXODUS Extensible Database System. Proc.12th Int. Conf. VLDB. Kyoto. 1986. 91–100

CARE89 Carey, M.J., Jauhari, R., Livny, M.: Priority in DBMS Resource Scheduling. Proc.15th Int. Conf. VLDB. Amsterdam. 1989. 397–408

CARE90 Carey, M. J., Krishnamurthy, S., Livny, M.: Load Control for Locking: the „Half and Half" Approach. Proc. 9th ACM Symp. Principles of Database Systems. 1990. 72–84

CARE93 Carey, M.J., Haas, L.M., Livny, M.: Tapes Hold Data, Too: Challenges of Tuples on Tertiary Store. Proc. ACM SIGMOD Conf. Washington. D.C. 1993. 413–417

CARE98 Carey, M.J., Haas, L.M., Kleewein, J., Reinwald, B.: Data Access Interoperability in the IBM Database Family. IEEE Data Engineering 21:3. 1998. 4–11

CATT96 Cattell, R. (ed.): The Object Database Standard: ODMG-93 (Release 1.2). Morgan Kaufmann. 1996

CERI94 Ceri, S. et al.: Automatic Generation of Production Rules for Integrity Maintenance. ACM Trans. Database Syst. 19:3. 1994. 367–422. Addendum: ACM Trans. Database Syst. 20:3. 1995. 364

CHAM80 Chamberlin, D.D.: A Summary of User Experience with the SQL Data Sublanguage. Proc. Int. Conf. on Data Bases. Deen, S.M., Hammersley, P. (eds.). Heydon. London. 1980. 181–203

CHAM81a Chamberlin, D., Astrahan, M., Blasgen, M., Gray, J. et al.: A History and Evaluation of System R. Comm. ACM 24:10. 1981. 632–646

CHAM81b Chamberlin, D., Astrahan, King, W., Lorie, R. et al: Support for Repetitive Transactions and Ad Hoc Queries in System R. ACM Trans. Database Syst. 6:1. 1981. 70–94

CHAM96 Chamberlin, D.: Using the New DB2: IBM's Object-Relational Database System. Morgan Kaufmann. 1996

CHAN81 Chang, N.S., Fu, K.S.: Picture Query Languages for Pictorial Data-Base Systems. IEEE Computer 14:11. 1981

CHAN82 Chan, A. et al.: The Implementation of an Integrated Concurrency Control and Recovery Scheme. Proc. ACM SIGMOD Conf. Orlando. 1982. 184–191

CHAU98 Chaudhuri, S.: An Overview of Query Optimization in Relational Systems. Proc. ACM Symp. on Principles of Database Systems. Seattle. 1998. 34–43

CHEN84 Cheng, J.M., Loosley, C.R., Shibamiya, A., Worthington, P.S.: IBM Database 2 Performance: Design, Implementation, and Tuning. IBM Sys. J. 23:2. 1984. 189–210

CHES94 Cheswick, W., Bellovin, S.: Firewalls and Internet Security – Repelling the Wily Hacker. Addison-Wesley. 1994

CHOU85 Chou, H.-T., DeWitt, D.: An Evaluation of Buffer Management Strategies for Relational Database Systems. Proc.11th Int. Conf. VLDB. Stockholm. 1985. 127–141

CHOY93 Choy, D., Mohan, C.: Locking Protocols for Two-Tier Indexing of Partitioned Data. IBM Res. Rep. Almaden Research Center. 1993

CHRI88 Christmann, P., Härder, T., Meyer-Wegener, K., Sikeler, A.: Which Kinds of OS Mechanisms Should be Provided for Database Management? Proc. Experiences in Distributed Systems. Kaiserslautern. LNCS 309. Springer. 1988. 213–252

CHRY90 Chrysanthis, P.K., Ramamritham, K.: ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior. Proc. ACM SIGMOD Conf. Atlantic City. NJ. 1990. 194–203

CHRY94 Chrysanthis, P.K., Ramamritham, K.: Synthesis of Extended Transaction Models Using ACTA. ACM Trans. Database Syst. 19:3. 1994. 450–491

CHU76 Chu, W.W., Opderbeck, W.M.: Program Behavior and the Page Fault Frequency Replacement Algorithm. IEEE Computer 9:11. 1976. 29–38

CODA78 CODASYL Data Description Language Committee Report. Information Systems 3:4. 1978. 247–320

COFF71 Coffman, E.G., Elphick, M.J., Shoshani, A.: System Deadlocks. ACM Computing Surv. 3:2. 1971. 67–78

COHE89 Cohen, E.I., King, G.M., Brady, J.T.: Storage Hierarchies. IBM Sys. J. 28:1. 1989. 62–76

COME79 Comer, D.: The Ubiquitous B-Tree. ACM Computing Surv. 11:2. 1979. 121–137

CONR97 Conrad, S.: Föderierte Datenbanksysteme – Konzepte der Datenintegration. Springer. 1997

DADA86 Dadam, P., Küspert, K., Andersen, F., Blanken, H., Erbe, R., Günauer, J., Lum, V., Pistor, P., Walch, G.: A DBMS Prototype to Support Extended NF2 Relations: An Integrated View of Flat Tables and Hierarchies. Proc. ACM SIGMOD Conf. Washington. D.C. 1986. 183–195

DATE97 Date, C.J., Darwen, H.: A Guide to the SQL Standard. 4th Edition. Addison-Wesley. 1997

DB2 Verschiedene Aufsätze zu IBM Database 2. IBM Sys. J. 23:2. 1984

DEMO80 Demolombe, R.: Estimation of the Number of Tuples Satisfying a Query Expressed in Predicate Calculus Language. Proc. 6th Int. Conf. VLDB. Montreal. 1980. 55–63

DENN80 Denning, P.J.: Working Sets Past and Present. IEEE Trans. Software Eng. 6:1. 1980. 64–84

DENN97 Denning, P.J., Metcalfe, R.: Beyond Calculation – The Next Fifty Years of Computing. Copernicus Springer. 1997

DEPP86 Deppisch, U., Paul, H.-B., Schek, H.-J.: A Storage System for Complex Objects. Proc. Int. Workshop on Object-Oriented Database Systems. Dittrich, K., Dayal, U. (eds.). Pacific Grove. 1986. 183–195

DEUX90 Deux, O., et al.: The Story of O2. IEEE Trans. Knowledge and Data Eng. 2:1. 1990. 91–108

DEWI84 DeWitt, D., Katz, R., Olken, F., Shapiro, L., Stonebraker, M., Wood, D.: Implementation Techniques for Main Memory Database Systems. Proc. ACM SIGMOD Conf. Boston. 1984. 1–8

DEWI85 DeWitt, D., Gerber, R.: Multiprocessor Hash-Based Join Algorithms. Proc. 11th Int. Conf. VLDB. Stockholm. 1985. 151–164

DEWI90 DeWitt, D. J., Futtersack, P., Maier, D., Velez, F.: A Study of Three Alternative Workstation-Server Architectures. Proc.16th Int. Conf. VLDB. Brisbane. 1990. 107–121

DO98 Do, L., Drew, P., Jin, W., Jumani, W., Van Rossum, D.: Issues in Developing Very Large Data Warehouses. Proc. 24th Int. Conf. VLDB. New York. 1998. 633–636

EBER97 Eberle, H.: Architektur moderner RISC-Mikroprozessoren. Informatik-Spektrum 20. 1997. 259–267

EFFE80a Effelsberg, W., Härder, T., Reuter, A.: An Experiment in Learning DBTG Database Administration. Information Systems 5:2. 1980. 136–147

EFFE80b Effelsberg, W., Härder, T., Reuter, A.: Measurement and Evaluation of Techniques for Implementing COSETS: A Case Study. Proc. Int. Conf. on Data Bases. Deen, S.M. Hammersley, P. (eds.). Heydon. London. 1980. 135–159

EFFE84a Effelsberg, W., Härder, T.: Principles of Database Buffer Management. ACM Trans. Database Syst. 9:4. 1984. 560–595

EFFE84b Effelsberg, W., Loomis, M.E.S.: Logical, Internal and Physical Reference Behavior in CODASYL Database Systems. ACM Trans. Database Syst. 9:2. 1984. 187–213

EISE98 Eisenberg, A., Melton, J.: Standards in Practice. ACM SIGMOD Record 27:3. 1998. 53–58

ELHA84 Elhardt, K., Bayer, R.: A Database Cache for High Performance and Fast Restart in Database Systems. ACM Trans. Database Syst. 9:4. 1984. 503–525

ELMA92 Elmargarmid, A.K. (ed.): Database Transaction Models for Advanced Applications. Morgan Kaufmann. 1992

ENBO88 Enbody, R.J., Du, H.C.: Dynamic Hashing Schemes. ACM Computing Surv. 20:2. 1988. 85–113

ESWA75 Eswaran, K.P., Chamberlin, D.D.: Functional Specifications of a Subsystem for Data Base Integrity. Proc. 1st Int. Conf. VLDB. Framingham. Mass. 1975. 48–68

ESWA76 Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, I.L.: The Notions of Consistency and Predicate Locks in a Database System. Comm. ACM 19:11. 1976. 624–633

FAGI79b Fagin, R., et. al: Extendible Hashing – A Fast Access Method for Dynamic Files. ACM Trans. Database Syst. 4:3. 1979. 315–344

FALO88 Faloutsos, C.: Gray-Codes for Partial Match and Range Queries. IEEE Trans. Software Eng. 14. 1988. 1381–1393

FALO90 Faloutsos, C.: Signature-Based Text Retrieval Methods: A Survey. IEEE Data Engineering 13:1. 1990. 25–32

FERG93 Ferguson, D., Nikolau, C., Georgiadis, L., Davies, K.:.Goal Oriented, Adaptive Transaction Routing for High Performance Transaction Processing. Proc. 2nd Int. Conf. on Parallel and Distributed Information Systems. San Diego. 1993

FERR76 Ferrari, D.: The Improvement of Program Behavior. IEEE Computer 9:11. 1976. 39–47

FINK74 Finkel, R.A., Bentley, J.L.: Quad Trees: A Data Structure for Retrieval on Composite Keys. Acta Informatica 4:1. 1974. 1–9

FLOR98 Florescu, D., Levy, A., Mendelzon, A.: Database Techniques for the World-Wide Web: A Survey. ACM SIGMOD Record 27:3. 1998. 59–74

FORD95 Ford, D.A., Myllymaki, J.: A Log-Structured Organization for Tertiary Storage. IBM Res. Rep. RJ 9941. San Jose. CA. 1995

FRAN85 Franaszek, P.A., Robinson, J.T: Limitations of Concurrency in Transaction Processing. ACM Trans. Database Syst. 10:1. 1985. 1–28

FRAN92 Franaszek, P.A., Robinson, J.T., Thomasian, A.: Concurrency Control for High Contention Environments. ACM Trans. Database Syst. 17:2. 1992. 304–345

FRAN97 Franklin, M.J., Carey, M.J., Livny, M.: Transactional Client-Server Cache Consistency: Alternatives and Performance. ACM Trans. Database Syst. 22:3. 1997. 315–363

FRED61 Fredkin, E.: Trie Memory. Comm. ACM 3:9. 1961. 490–500

FREE87 Freeston, M.W.: The BANG file: A New Kind of Grid File. Proc. ACM SIGMOD Conf. San Francisco. 1987. 260–269

FREE89 Freeston, M.W. : A Well-Behaved File Structure for the Storage of Spatial Objects. Proc. Symp. on the Design and Implementation of Large Spatial Databases. Santa Barbara. LNCS 409. Springer. 1989. 287–300

GAED98 Gaede, V., Günther, O.: Multidimensional Access Methods. ACM Computing Surv. 30:2. 1998. 170–231

GANG94 Ganger, G.R., Worthington, B.L., Hou, R.Y., Patt, Y.N.: Disk-Arrays – High-Performance, High-Reliability Storage Subsystems. IEEE Computer 27: 3. 1994. 30–36

GARC87 Garcia-Molina, H., Salem, H.: Sagas. Proc. ACM SIGMOD Conf. San Francisco. 1987. 249–259

GARZ88 Garza, J.F., Kim, W.: Transaction Management in an Object-Oriented Database System. Proc. ACM SIGMOD Conf. Chicago. 1988. 37–45

GAWL85 Gawlick, D.: Processing “Hot Spots” in High Performance Systems. Proc. IEEE Spring CompCon Conf. 1985. 249–251

GAWL85b Gawlick, D., Kinkade, D.: Varieties of Concurrency Control in IMS/VS Fast Path. IEEE Database Engineering 8:2. 1985. 3–10

GELB89 Gelb, J.P.: System-Managed Storage. IBM Sys. J. 28:1. 1989. 77–103

GEOR94 Georgakopoulos, D. et al.: Specification and Management of Extended Transactions in a Programmable Transaction Environment. Proc. 10th Int. Conf. on Data Engineering. Houston. 1994. 462–473

GERT94 Gertz, M.: Specifying Reactive Integrity Control for Active Databases. Proc. 4th Int. Workshop on Research Issues in Data Engineering (RIDE-ADS 94). Houston. 1994. 62–70

GHOS75 Ghosh, S.P., Lum, V.Y.: Analysis of Collision when Hashing by Division. Information Systems 1:1. 1975. 15–22

GIBB97 Gibbons, P.B., Matias, Y., Poosala, V.: Fast Incremental Maintenance of Approximate Histograms. Proc. 23th Int. Conf. VLDB. Athens. 1997. 466–475

GOLD89 Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley. 1989

GRAE89 Graefe, G., Ward, K.: Dynamic Query Evaluation Plans. Proc. ACM SIGMOD Conf. Portland. 1989. 358–366

GRAE93 Graefe, G.: Query Evaluation Techniques for Large Databases. ACM Computing Surv. 25:2. 1993. 73–170

GRAY76 Gray, J.N. et al.: Granularity of Locks and Degrees of Consistency in a Shared Data Base. Proc. IFIP Working Conf. on Modelling in Data Base Management Systems. North-Holland. 1976. 365–394

GRAY78 Gray, J. N.: Notes on Data Base Operating Systems. Operating Systems: An Advanced Course. LNCS 60. Springer. 1979. 393–481

GRAY81a Gray, J.N., et al.: The Recovery Manager of the System R Database Manager. ACM Computing Surv. 13:2. 1981. 223–242. Ebenso in [KUMA98]

GRAY81b Gray, J.N.: A Straw Man Analysis of Probability of Waiting and Deadlock. IBM Research Report RJ3066. San Jose. 1981

GRAY81c Gray, J.N.: The Transaction Concept: Virtues and Limitations. Proc. 7th Int. Conf. VLDB. Cannes. 1981. 144–154

GRAY86 Gray, J.N.: An Approach to Decentralized Data Management Systems. IEEE Trans. Software Eng. 12:6. 1986. 684–692

GRAY87 Gray, J.N., Putzolu, G.R.: The Five Minute Rule for Trading Memory for Disk Accesses and the 10 Byte Rule for Trading Memory for CPU Time. Proc. ACM SIGMOD Conf. San Francisco. 1987. 395–398

GRAY93 Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann. 1993

GRAY96 Gray, J., Bosworth, A., Layman, A., Pirahesh, H.: Data Cube: A Relational Aggregation Operator Generalizing Group-By. Cross-Tab. and Sub-Total. Proc. 12th Int. Conf. on Data Engineering. New Orleans. 1996. 152–159

GRAY97 Gray, J.N., Graefe, G.: The Five Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb. ACM SIGMOD Record 26:4. 1997. 63–68

GREE89 Greene, D.: An Implementation and Performance Analysis of Spatial Data Access Methods. Proc. 5th Int. Conf. on Data Engineering. Los Angeles. 1989. 606–615

GREF93 Grefen, P.: Combining Theory and Practice in Integrity Control: A Declarative Approach to the Specification of a Transaction Modification Subsystem. Proc. 19th Int. Conf. VLDB. Dublin. 1993. 581–591

GUTT84 Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. Proc. ACM SIGMOD Conf. Boston. 1984. 47–57

HAAS90 Haas, L., Chang, W., Lohman, G., McPherson, J., et al.: Starburst Mid-Flight: As the Dust Clears. IEEE Trans. Knowledge and Data Eng. 2:1. 1990. 143–160

HÄRD77 Härder, T.: A Scan-Driven Sort Facility for a Relational Database System. 3rd Int. Conf. VLDB. Tokyo. 1977. 236–243

HÄRD78a Härder, T.: Implementierung von Datenbanksystemen. Hanser. 1978

HÄRD78b Härder, T.: Implementing a Generalized Access Path Structure for a Relational Database System. ACM Trans. Database Syst. 3:3. 1978. 285–298

HÄRD79 Härder, T., Reuter, A.: Optimization of Logging and Recovery in a Database System. Data Base Architecture. Bracchi, G., Nijssen, G.M. (eds.). North Holland. 1979. 151–168

HÄRD83a Härder, T., Reuter, A.: Concepts for Implementing a Centralized Database Management System. Proc. Int. Computing Symposium on Application Systems Development. Nürnberg. Teubner. 1983. 28–59

HÄRD83b Härder, T., Reuter, A.: Principles of Transaction-Oriented Database Recovery. ACM Computing Surv. 15 :4. 1983. 287–317. Ebenso in [KUMA98]

HÄRD84 Härder, T.: Observations on Optimistic Concurrency Control. Information Systems 9:2. 1984. 111–120

HÄRD86a Härder, T., Rahm, E.: Mehrrechner-Datenbanksysteme für Transaktionssysteme hoher Leistungsfähigkeit. Informationstechnik–Computer, Systeme, Anwendungen 28:4. 1986. 214–225

HÄRD86b Härder, T., Meyer-Wegener, K.: Transaktionssysteme und TP-Monitore – Eine Systematik ihrer Aufgabenstellung und Implementierung. Informatik – Forschung und Entwicklung 1:1. 1986. 3–25

HÄRD87a Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA – A DBMS Prototype Supporting Engineering Applications. Proc.13th Int. Conf. VLDB. Brighton. U.K. 1987. 433–442

HÄRD87b Härder, T.: On Selected Performance Issues of Database Systems. Proc. 4. GI/ITG-Fachtagung „Messung, Modellierung und Bewertung von Rechensystemen“. Informatik-Fachberichte 154. Erlangen. 1987. 294–312

HÄRD87c Härder, T., Petry, E.: Evaluation of Multiple Version Scheme for Concurrency Control. Information Systems 12:1. 1987. 83–98

HÄRD87d Härder, T., Rothermel, K.: Concepts for Transaction Recovery in Nested Transactions. Proc. ACM SIGMOD Conf. San Francisco. 1987. 239–248

HÄRD88 Härder, T., Hübel, Ch., Meyer-Wegener, K., Mitschang, B.: Processing and Transaction Concepts for Cooperation of Engineering Workstations and a Database Server. Data and Knowledge Eng. 3. 1988. 87–107

HÄRD90a Härder, T., Meyer-Wegener, K.: Transaktionssysteme in Workstation/Server-Umgebungen. Informatik – Forschung und Entwicklung 5:3. 1990. 127–143

HÄRD90b Härder, T.: An Approach to Implement Dynamically Defined Complex Objects. Proc. PRISMA Workshop. Nordwijk. Holland. LNCS 503. Springer. 1990. 71–98

HÄRD92a Härder, T., Mitschang, B., Schöning, H.: Query Processing for Complex Objects. Data and Knowledge Eng. 7. 1992. 181–200

HÄRD92b Härder, T., Rahm, E.: Zugriffspfad-Unterstützung zur Sicherung der Relationalen Invarianten. ZRI-Bericht 2/92. Univ. Kaiserslautern. 1992

HÄRD93 Härder, T., Rothermel, K.: Concurrency Control Issues in Nested Transactions. VLDB-Journal 2:1. 1993. 39–74

HÄRD95 Härder, T., Mitschang, B., Nink, U., Ritter, N.: Workstation/Server-Architekturen für datenbankbasierte Ingenieuranwendungen. Informatik – Forschung und Entwicklung 10:2. 1995. 55–72

HÄRD96 Härder, T., Reinert, J.: Access Path Support for Referential Integrity in SQL2. VLDB Journal 5:3. 1996. 196–214

HÄRD99 Härder, T., Sauter, G., Thomas, J.: The Intrinsic Problems of Structural Heterogeneity and an Approach to their Solution. VLDB Journal 8:1. 1999

HAMI96 Hamilton, G., Cattell, R.: JDBC: A Java SQL API, Version 1.10. SUN Microsystems Computer Comp. Oct. 1996. ftp://splash.javasoft.com/pub/jdbc-spec-0110.ps

HALA88 Halasz, F.G.: Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems. Comm. ACM 31:7. 1988. 836–852

HELD78 Held, G.D., Stonebraker, M.: B-Trees Re-Examined. Comm. ACM 21:2. 1978. 139–143

HELL89 Helland, P. et al.: Group Commit Timers and High Volume Transaction Systems. Proc. 2nd Int. Workshop on High Performance Transaction Systems. Asilomar. CA. LNCS 359. Springer. 1989

HELL95 Hellerstein, J. M., Naughton, J. F., Pfeffer, A.: Generalized Search Trees for Database Systems. Proc. 21st Int. Conf. VLDB. Zurich. 1995. 562–573

HENN90 Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach. Morgan Kaufmann. 1990

HENR89 Henrich, A., H.-W. Six, P. Widmayer: The LSD-Tree: Spatial Access to Multidimensional Point- and Non-Point Objects. Proc. 15th Int. Conf. VLDB. Amsterdam. 1989. 45–53

HERB97 Herbst, A.: Anwendungsorientiertes DB-Archivieren – Neue Konzepte zur Archivierung in Datenbanksystemen. Springer. 1997

HINR85 Hinrichs, K.: Implementation of the Grid File: Design Concepts and Experience. BIT 25. 1985. 569–592

HOAG85 Hoagland, A.S.: Information Storage Technology: A Look at the Future. IEEE Computer 18:7. 1985. 60–67

HOWA78 Howard, P.H., Borgendale, K.W.: System/38 Machine Indexing Support. IBM System/38 Technical Developments. 1978. 67–69

HSIA77 Hsiao, D.K., Madnick, S.E.: Database Machine Architecture in the Context of Information Technology Evolution. Proc. 3rd Int. Conf. VLDB. 1977. 63–84

HSU88 Hsu, M., Ladin, R., McCarthy, D.: An Execution Model for Active Database Management Systems. Proc. 3rd Int. Conf. on Data and Knowledge Bases. 1988. 171–179

HSU93 Hsu, M., Zhang, B.: Performance Evaluation of Cautious Waiting. ACM Trans. Database Syst. 17:3. 1993. 477–512

HÜBE92 Hübel, Ch., Sutter, B.: Supporting Engineering Applications by New Data Base Processing Concepts – An Experience Report. Engineering with Computers 8. 1992. 31–49

HUTF88 Hutflesz, A., Six, H.-W., Widmayer, P.: Twin Grid Files: Space Optimizing Access Schemes. Proc. ACM SIGMOD Conf. Chicago. 1988. 183–190

HUTF90 Hutflesz, A., Six, H.-W., Widmayer, P.: The R-File: An Efficient Access Structure for Proximity Queries. Proc. 6th Int. Conf. Data Engineering. Los Angeles. 1990. 372–379

IOAN87 Ioannidis, Y., Wong, E.: Ouery Optimization by Simulated Annealing. Proc. ACM SIGMOD Conf. San Francisco. 1987. 9–22

IOAN90 Ioannidis, Y., Kang, Y.: Randomized Algorithms for Optimizing Large Join Queries. Proc. ACM SIGMOD Conf. Atlantic City. NJ. 1990. 312–321

IOAN93 Ioannidis, Y.E.: Universality of Serial Histograms. Proc. 19th Int. Conf. VLDB. Dublin. 1993. 256–267

JABL97 Jablonski, S., Böhm, M., Schulze, W. (Hrsg.): Workflow-Management – Entwicklung von Anwendungen und Systemen. Facetten einer neuen Technologie. dpunkt.Verlag. 1997

JAGA90 Jagadish, H.V.: Linear Clustering of Objects with Multiple Attributes. Proc. ACM SIGMOD Conf. Atlantic City. NJ. 1990. 332–342

JIAN88 Jiang, B.: Deadlock Detection is Really Cheap. ACM SIGMOD Record 17:2. 1988. 2–13

JAJO97 Jajodia, S., Kerschberg, L. (eds.): Advanced Transaction Models and Architectures. Kluwer. 1997

JAKO78a Jakobssen, M.: Huffman Coding in Bit-Vector Compression. Information Processing Letters 7:6. 1978. 304–553

JAKO78b Jacobssen, M., Nevalainen, O.: On the Compression of Inverted Files. Angewandte Informatik 20:12. 1978. 552–553

JAKO79 Jacobssen, M.: Implementation of Compressed Bit-Vector Indexes. Proc. EURO IFIP79. Samet. P.A. (ed). North Holland. 1979. 561–566

JARK84 Jarke, M., Koch, J.: Query Optimization in Database Systems. ACM Computing Surv. 16:2. 1984. 111–152

JASP98 Jasper, H.: Aktive Informationssysteme – Systeme, Methoden, Anwendungen. Shaker-Verlag. 1998

JAUH90 Jauhari, R., Carey, M.J., Livny, M.: Priority-Hints: An Algorithm for Priority-Based Buffer Management. Proc.16th Int. Conf. VLDB. Brisbane. 1990. 708–721

JIAN94 Jiang, B.: Behandlung nichtrelationaler Datenbankanfragen – Eine Gegenüberstellung verschiedener Vorschläge. Informatik-Spektrum 17:6. 1994. 373–383

JORD81 Jordan, J.R., Banerjee, J., Batman, R.B.: Precision Locks, Proc. ACM SIGMOD Conf. Ann Arbor. 1981. 143–147

JOSH98 Joshi, A., Bridge, W., Loazia, J., Lahiri, T.: Checkpointing in Oracle. Proc. 24th Int. Conf. VLDB. New York. 1998. 665–668

KATA97 Katayama, N., Satoh, S. The SR-Tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. Proc. ACM SIGMOD Conf. Tuscon. 1997. 369–380

KEET98 Keeton, K., Patterson, D.A., Hellerstein, J.M.: A Case for Intelligent Disks (IDISKs). ACM SIGMOD Record 27:3. 1998. 42–52

KELT87 Kelter, U.: Concurrency Control for Design Objects with Versions in CAD Databases. Information Systems 12:2. 1987. 137–143

KEMP90 Kemper, A., Moerkotte, G.: Access Support in Object Bases. Proc. ACM SIGMOD Conf. Atlantic City. NJ. 1990. 364–374

KEMP95 Kemper, A., Kossmann, D.: Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis. VLDB Journal 4:3. 1995. 519–566

KENN97 Kennedy, A. R. et al.: A G3 PowerPC^{TM} Superscalar Low-Power Microprocessor. Proc. Spring Compcon'97 Conf. San Jose. 1997. 315–324

KEßL93 Keßler, U., Dadam, P.: Benutzergesteuerte, flexible Speicherungsstrukturen für komplexe Objekte. Proc. BTW'93. Braunschweig. Informatik aktuell. Springer. 1993. 206–225

KIM80 Kim, W.: A New Way to Compute the Product and Join of Relations. Proc. ACM SIGMOD Conf. New York. 1980. 179–188

KIM84 Kim, W. et al.: A Transaction Mechanism for Engineering Design Databases. Proc. 10th Int. Conf. VLDB. Singapore. 1984. 355–362

KIM89 Kim, W., et al.: Features of the ORION Object-Oriented Database System. Object-Oriented Concepts, Databases, and Applications. Kim, W., Lochovsky, F. (eds.). Addison-Wesley. 1989. 251–282

KITS83 Kitsuregawa, M., Tanaka, H., Moto-Oka, T.: Application of Hash to Database Machine and its Architecture. New Generation Computing 1:1. 1983. 63–74

KLAH85 Klahold, P. et al.: A Transaction Model Supporting Complex Applications in Integrated Information Systems. Proc. ACM SIGMOD Conf. Austin. 1985. 388–401

KRIE88 Kriegel, H.-P., Seeger, B.: PLOP-Hashing: A Grid File without Directory. Proc. 4th Int. Conf. Data Engineering. Los Angeles. 1988. 369–376

KRIS85 Krishnamurthy, R., Whang, K.-Y.: Multilevel Grid Files. IBM Res. Rep. Yorktown Heights. NY. 1985

KRAT90 Kratzer, K., Wedekind, H., Zörntlein, G.: Prefetching – A Performance Analysis. Information Systems. 15:4. 1990. 445–452

KROP79 Kropp, D., Schek, H.-J., Walch, G.: Text Field Indexing. Datenbanktechnologie. Niedereichholz, J. (Hrsg.). Teubner. 1979. 101–115

KÜSP83 Küspert, K.: Storage Utilization in B*-Trees with a Generalized Overflow Technique. Acta Informatica 19:1. 1983. 35–55

KUMA89 Kumar, A., Stonebraker, M.: Performance Considerations for an Operating System Transaction Manager. IEEE Trans. Software Eng. 15:6. 1989. 705–714

KUMA96 Kumar, V. (ed.): Performance of Concurrency Control Mechanisms in Centralized Database Systems. Prentice Hall. 1996

KUMA98 Kumar, V., Hsu, M. (eds.): Recovery Mechanism in Database Systems. Prentice Hall. 1998

KUNG80 Kung, H.T., Lehman, P.L.: Concurrent Manipulation of Binary Search Trees. ACM Trans. Database Syst. 5:3. 1980. 339–353

KUNG81 Kung, H.T., Robinson, J.T.: On Optimistic Methods for Concurrency Control. ACM Trans. Database Syst. 6:2. 1981. 213–226

LAME94 Lamersdorf, W.: Datenbanken in verteilten Systemen – Konzepte, Lösungen, Standards. Reihe Datenbanksysteme. Vieweg. 1994

LAMP79 Lampson, B.W., Sturgis, H.E.: Crash Recovery in a Distributed Data Storage System. XEROX Research Report. Palo Alto. 1979

LARS78 Larson, P.: Dynamic Hashing. BIT 18. 1978. 184–201

LARS80 Larson, P.-A.: Linear Hashing with Partial Expansions. Proc. 6th Int. Conf. VLDB. Montreal. 1980. 224–232

LARS83 Larson, P.-A.: Dynamische Hashverfahren. Informatik-Spektrum 6:1. 1983. 7–19

LARS84 Larson, P.-A., Kajla A.: File Organization: Implementation of a Method Guaranteeing Retrieval in One Access. Comm. ACM 27:7. 1984. 670–677

LARS85 Larson, P.-A.: Linear Hashing with Overflow-Handling by Linear Probing. ACM Trans. Database Syst. 10:1. 1985. 75–89

LARS88 Larson, P.-A.: Linear Hashing with Separators – A Dynamic Hashing Scheme Achieving One-Access Retrieval. ACM Trans. Database Syst. 13:3. 1988. 366–388

LARS98 Larson, P.-A., Graefe, G.: Memory Management during Run Generation in External Sorting. Proc. ACM SIGMOD Conf. Seattle. 1998. 472–483

LEBE94 Lebeck, A.R., Wood, D.A.: Cache Profiling and the SPEC Benchmarks: A Case Study. IEEE Computer. 27:10. 1994. 15–26

LEHM89 Lehman, T.J., Lindsay, B.G.: The Starburst Long Field Manager. Proc. 15th Int. Conf. VLDB. Amsterdam. 1989. 375–383

LEUN98 Leung, T.Y.C., Pirahesh, H., Seshadri, P., Hellerstein, J.: Query Rewrite Optimization Rules in IBM DB2 Universal Database. Readings in Database Systems (3rd ed.). Stonebraker, M., Hellerstein, J. M. (eds.). Morgan Kaufmann. 1998. 1–27

LEYM97 Leymann, F.: Transaktionsunterstützung für Workflows. Informatik – Forschung und Entwicklung 12. 1997. 82–90

LINN94 Linnemann, V., Pampel, H.: Sprachliche Formulierung rekursiver und iterativer Anfragen in Datenbanksystemen. Informatik-Spektrum 17:3. 1990. 151–163

LIPT90 Lipton, R., Naughton, J., Schneider, D.: Practical Selectivity Estimation through Adaptive Sampling. Proc. ACM SIGMOD Conf. Atlantic City. NJ. 1990. 1–11

LITW78 Litwin, W.: Virtual Hashing: A Dynamically Changing Hashing. Proc. 4th Int. Conf. VLDB. Berlin. 1978. 517–523

LITW80 Litwin, W.: Linear Hashing: A New Tool for Files and Tables Implementation. Proc. 6th Int. Conf. VLDB. Montreal. 1980. 212–223

LOCK87 Lockemann, P.C., Schmidt, J.W. (Hrsg.): Datenbank-Handbuch. Springer. 1987

LOES98 Loeser, H.: Techniken für Web-basierte Datenbankanwendungen – Anforderungen, Ansätze, Architekturen. Informatik – Forschung und Entwicklung 13:4. 1998. 196–216

LOHM91 Lohman, G.M., Lindsay, B., Pirahesh, H., Schiefer, K.B.: Extensions to Starburst: Objects, Types, Functions and Rules. Comm. ACM 34:10. 1991. 94–109

LOME90 Lomet, D.B., Salzberg, B.: The hB-Tree: a Multiattribute Indexing Method with Good Guaranteed Performance. ACM Trans. Database Syst. 15:4. 1990. 625–658

LOME97 Lomet, D.B., Salzberg, B.: Concurrency and Recovery for Index Trees. VLDB Journal 6:3, 1997. 224–240

LOME98 Lomet, D.B., Weikum, G.: Efficient Transparent Application Recovery in Client/Server Information Systems. Proc. ACM SIGMOD Conf. Seattle. 1998. 460–471

LORI77 Lorie, R.A.: Physical Integrity in a Large Segmented Database. ACM Trans. Database Syst. 2:1. 1977. 91–104

LORI79a Lorie, R.A., Wade, B.W.: The Compilation of a High Level Data Language. IBM Res. Rep. RJ 2589. San Jose. Calif. 1979

LORI79b Lorie, R.A., Nilsson, J.F.: An Access Specification Language for a Relational Database System. IBM J. Res. and Dev. 23:3. 1979. 286–298

LORI83 Lorie, R., Plouffe, W. Complex Objects and their Use in Design Transactions. Proc. IEEE Annual Meeting of Database Week. 1983. 115–121

LUM70 Lum, V.Y.: Multi-Attribute Retrieval with Combined Indices. Comm. ACM 13:11 1970. 660–665

LUM71 Lum, V.Y., Yuen, P.S.T., Dodd, M.: Key-to-Address Transform Techniques: a Fundamental Performance Study of Large Existing Formatted Files. Comm. ACM 14:4. 1971. 228–239

LYNC88 Lynch, C.: Selectivity Estimation and Query Optimization in Large Databases with Highly Skewed Distributions of Column Values. Proc. 14th Int. Conf. VLDB. Los Angeles. 1988. 240–251

MAIE86 Maier, D., Stein, J.: Indexing in an Object-Oriented DBMS. Proc. IEEE Int. Workshop on Object-Oriented Database Systems. Dittrich, K., Dayal, U. (eds.). Asilomar. CA. 1986. 171–182

MANN88 Mannino, M., Chu, P., Sager, T.: Statistical Profile Estimation in Database Systems. ACM Computing Surv. 20:3. 1988. 191–221

MARC83 March, S.T.: Techniques for Structuring Database Records. ACM Computing Surv. 15:1. 1983. 45–79

MARE94 Marek, R., Rahm, E.: TID Hash Joins. Proc. 3rd Int. Conf. on Information and Knowledge Management (CIKM'94). Gaithersburg. MD. 1994. 42–49

MARE95 Marek, R.: Ein Kostenmodell der parallelen Anfragebearbeitung in Shared-Nothing-Datenbanksystemen. Proc. BTW'95. Dresden. Informatik aktuell. Springer. 1995. 232–251

MARU77 Maruyama, K., Smith, S.E.: Analysis of Design Alternatives for Virtual Memory Indexes. Comm. ACM. 20:4. 1977. 245–254

MAUR75 Maurer, W.D., Lewis, T.G.: Hash Table Methods. ACM Computing Surv. 7:1. 1975. 5–19

MCCR77 McCreight, E.: Pagination of B*-Trees with Variable Length Records. Comm. ACM 20:9. 1977. 84–168

METH93 Metha, M., DeWitt, D.: Dynamic Memory Allocation for Multiple-Query Workloads. Proc.19th Int. Conf. VLDB. Dublin. 1993. 354–367

MEYE87 Meyer-Wegener, K.: Transaktionssysteme – verteilte Verarbeitung und verteilte Datenhaltung. Informationstechnik–Computer, Systeme, Anwendungen 29:3. 1987. 120–126

MEYE88 Meyer-Wegener, K.: Transaktionssysteme. Teubner. Stuttgart. 1988

MISH92 Mishra, P., Eich, M.H.: Join Processing in Relational Databases. ACM Computing Surv. 24:1. 1992. 63–113

MITS88 Mitschang, B.: Ein Molekül-Atom-Datenmodell für Non-Standard-Anwendungen: Anwendungsanalyse, Datenmodellentwurf und Implementierungskonzepte. Informatik-Fachberichte 185. Springer. 1988

MITS95 Mitschang, B.: Anfrageverarbeitung in Datenbanksystemen: Entwurfs- und Implementierungskonzepte. Reihe Datenbanksysteme. Vieweg. 1995

MOHA90 Mohan, C.: Commit_LSN: A Novel and Simple Method for Reducing Locking and Latching in Transaction Processing Systems. Proc. 16th Int. Conf. VLDB. Brisbane. 1990. 406–418

MOHA90b Mohan, C.: ARIES KVL: A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operating on B-Tree Indexes. Proc. 16th Int. Conf. VLDB. Brisbane. 1990. 392–405

MOHA92 Mohan, C. et al.: ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks using Write-Ahead Logging. ACM Trans. Database Syst. 17:1. 1992. 94–162. Ebenso in [KUMA98]

MOHA92b Mohan, C.: Less Optimism about Optimistic Concurrency Control. Proc. 2nd Workshop on Research Issues in Data Engineering (RIDE-2). Tempe. AZ. IEEE Computer Society Press. 1992. 199–204

MOHA92c Mohan, C., Pirahesh, H., Lorie, R.: Efficient and Flexible Methods for Transient Versioning of Records to Avoid Locking by Read-Only Transactions. Proc. ACM SIGMOD Conf. San Diego. 1992. 124–133

MOHA92d Mohan, C., Levine, F.: ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging. Proc. ACM SIGMOD Conf. San Diego. 1992. 371–380

MOHA93a Mohan, C.: A Survey of DBMS Research Issues in Supporting Very Large Tables. Proc. 4th Int. Conf. on Foundations of Data Organization and Algorithms. Evanston. 1993

MOHA93b Mohan, C.: IBM's Relational DBMS Products: Features and Technologies. Proc. ACM SIGMOD Conf. Washington. D.C. 1993. 445–448

MOHA93c Mohan, C., Narang, I.: An Efficient and Flexible Method for Archiving a Data Base. Proc. ACM SIGMOD Conf. Washington, D.C. 1993. 139–146

MOHA94 Mohan, C.: Disk Read-Write Optimizations and Data Integrity in Transaction Systems using Write-Ahead Logging. IBM Res. Rep. RJ 9741. Almaden Research Center. 1994

MÖNK92 Mönkeberg, A., Weikum, G.: Performance Evaluation of an Adaptive and Robust Load Control Method for the Avoidance of Data Contention Thrashing. Proc. 18th Int. Conf. VLDB. Vancouver. 1992. 432–443

MORR68 Morrison, D.R.: PATRICIA – Practical Algorithm to Retrieve Information Coded in Alphanumeric. Journ. ACM 15:4. 1968. 514–534

MOSS81 Moss, J.E.B.: Nested Transactions: An Approach to Reliable Distributed Computing. MIT Report LCS TR 260. 1981. MIT Press. 1985

MOSS92 Moss, B., Eliot, J.: Working with Persistent Objects: To Swizzle or not to Swizzle. IEEE Trans. Software Eng. 18:8. 1992. 657–673

MÜLL99 Müller, R., Rahm, E.: Rule-Based Dynamic Modification of Workflows in a Medical Domain. Proc. BTW'99. Freiburg. Informatik aktuell. Springer. 1999

MURA88 Muralikrishna, M., DeWitt, D.: Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries. Proc. ACM SIGMOD Conf. Chicago. 1988. 28–36

MUTH93 Muth, P. et al.: Semantic Concurrency Control in Object-Oriented Database Systems. Proc. 9th Int. Conf. on Data Engineering. Vienna. 1993. 233–242

NAKA78 Nakamura, T., Mizzogushi, T.: An Analysis of Storage Utilization Factor in Block Split Data Structuring Scheme. Proc. 4th Int. Conf. VLDB. Berlin. 1978. 489–495

NARA97 Narang, I., Mohan, C., Brannon, K.: Coordinated Backup and Recovery between DBMSs and File Systems. submitted

NAVA85 Navathe, S.B., et al.: Vertical Partitioning Algorithms for Database Design. ACM Trans. Database Syst. 9:4. 1984. 680–710

NEUM92 Neumann, K.: Kopplungsarten von Programmiersprachen und Datenbanksprachen. Informatik-Spektrum 15:4. 1992. 185–194

NEVA79 Nevalainen, O., Muurinen, K., Rantala, S.: A Note on Character Compression. Angewandte Informatik 21:7. 1979. 313–318

NG91 Ng, R., Falutsos, C., Sellis, T.: Flexible Buffer Allocation Based on Marginal Gains. Proc. ACM SIGMOD Conf. Denver. 1991. 387–396

NIEV84 Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The Grid File: An Adaptable, Symmetric Multikey File Structure. ACM Trans. Database Syst. 9:1. 1984. 38–71

NIKO92 Nikolaou, C., Ferguson, D., Constantopoulos, P.: Towards Goal Oriented Resource Management. IBM Res. Rep. RC17919. Yorktown Heights. NY. 1992

NINK98 Nink, U.: Anwendungsprogrammierschnittstellen für strukturell objektorientierte Datenbanksysteme. Univ. Kaiserslautern. Dissertation.1998

NODI92 Nodine, M.H., Zdonik, S.B.: Cooperative Transaction Hierarchies: Transaction Support for Design Applications. VLDB Journal 1:1. 1992. 41–80

NOE87 Noe, J.D., Wagner, D.B.: Measured Performance of Time Interval Concurrency Control Techniques. Proc. 13th Int. Conf. VLDB. Brighton. U.K. 1987. 359–367

ONEI86 O'Neil, P.E.: The Escrow Transactional Method. ACM Trans. Database Syst 11:4. 1986. 405–430

ONEI93 O'Neil, E.J., O'Neil, P.E., Weikum, G.: The LRU-K Page Replacement Algorithm for Database Disk Buffering. Proc. ACM SIGMOD Conf. Washington. D.C. 1993. 297–306

ONEI95 O'Neil, P.E., Graefe, G.: Multi-Table Joins Through Bitmapped Join Indices. ACM SIGMOD Record 24:3. 1995. 8–11

ONEI97 O'Neil, P.E., Quass, D.: Improved Query Performance with Variant Indexes. Proc. ACM SIGMOD Conf. Tucson. 1997. 38–49

OOI89 Ooi, B.C., R. Sacks-Davis, K.J. Mc Donell: Extending a DBMS for Geographic Applications. Proc. 5th Int. Conf. Data Engineering. Los Angeles. 1989. 590–597

OREN84 Orenstein, J.A., Merrett, T.H.: A Class of Data Structures for Associative Searching. Proc. ACM Symp. on Principles of Database Systems. Waterloo. 1984. 181–190

ORFA96 Orfali, R., Harkey, D., Edwards, J.: The Essential Client/Server Survival Guide. 2nd edition. John Wiley & Sons. New York. 1996

OTTM96 Ottmann, Th., Widmayer, P.: Algorithmen und Datenstrukturen. 3. Auflage. BI-Wissenschaftsverlag. Mannheim. 1996

OUKS85 Ouksel, M.: The Interpolation-Based Grid File. Proc. ACM Symp. on Principles of Database Systems. Portland. 1985. 20–27

OUST98 Ousterhout, J.K.: Scripting: Higher-Level Programming for the 21st Century. IEEE Computer 31:3. 1998. 23–30

OZKA85 Ozkarahan, E.A., Ouksel: Dynamic and Order Preserving Data Partitioning for Database Machines. Proc. 11th Int. Conf. VLDB. Stockholm. 1985. 358–368

PAPA86 Papadimitriou, C.H.: The Theory of Database Concurrency Control. Computer Science Press, 1986

PARN72 Parnas, D.L.: On the Criteria to be Used in Decomposing Systems into Modules. Comm. ACM. 15:12. 1972. 1053–1058

PARN75 Parnas, D.L., Siewiorek, D.P.: Use of the Concept of Transparency in the Design of Hierarchically Structured Systems. Comm. ACM. 18:7. 1975. 401–408

PATT88 Patterson, D.A, Gibson, G., Katz, R.H.: A Case for Redundant Arrays of Inexpensive Disks (RAID). Proc. ACM SIGMOD Conf. Chicago. 1988. 109–116

PATT98 Patterson, D.A, Keeton, K.K.: Hardware Technology Trends and Database Opportunities. Invited talk at the ACM SIGMOD Conf. Seattle. 1998. http://cs.berkeley.edu/~patterson/talks

PAUL87 Paul, H.-B., Schek, H.-J., Scholl, M.H., Weikum, G., Deppisch, U.: Architecture and Implementation of the Darmstadt Database Kernel System. Proc. ACM SIGMOD Conf. San Francisco. 1987. 196–207

PEAR93 Pearson, C., Finkelstein, S.: Requirements for Automated Storage Management for OLTP Systems. Proc. Int. Workshop on High Performance Transaction Systems. Asilomar. CA. 1993

PEIN87 Peinl, P.: Synchronisation in zentralisierten Datenbanksystemen. Informatik-Fachberichte 161. Springer. 1987

PIAT84 Piatesky-Shapiro, G., Connell, C.: Accurate Estimation of the Number of Tuples Satisfying a Condition. Proc. ACM SIGMOD Conf. Boston. 1984. 256–276

PIRA92 Pirahesh, H., Hellerstein, J., Hasan W.: Extensible/Rule Based Query Rewrite Optimization in Starburst. Proc. ACM SIGMOD Conf. San Diego. 1992. 39–48

POOS97 Poosala, V., Ioannidis, Y. E.: Selectivity Estimation Without the Attribute Value Independence Assumption. Proc. 23th Int. Conf. VLDB. Athens. 1997. 486–495

POTA96 Potamianos, S., Stonebraker, M.: The POSTGRES Rule System. In [WIDO96a]. 43–61

PRÄD82 Prädel, U., Schlageter, G., Unland, R.: Einige Verbesserungen optimistischer Synchronisationsverfahren. Proc. 12. GI-Jahrestagung. Informatik-Fachberichte 57. Springer. 1982. 684–698

PU86 Pu, C.: On-the-Fly, Incremental, Consistent Reading of Entire Databases. Algorithmica. 1986. 271–287

RAHM88a Rahm, E.: Optimistische Synchronisationskonzepte in zentralisierten und verteilten Datenbanksystemen. Informationstechnik –Computer, Systeme, Anwendungen 30:1. 1988. 28–47

RAHM88b Rahm, E.: Synchronisation in Mehrrechner-Datenbanksystemen. Informatik-Fachberichte 186. Springer. 1988

RAHM89 Rahm, E.: Der Database-Sharing-Ansatz zur Realisierung von Hochleistungs-Transaktionssystemen. Informatik-Spektrum 12:2. 1989. 65–81

RAHM92 Rahm, E.: Performance Evaluation of Extended Storage Architectures for Transaction Processing, Proc. ACM SIGMOD Conf. San Diego. 1992. 308–317

RAHM93 Rahm, E.: Hochleistungs-Transaktionssysteme – Konzepte und Entwicklungen moderner Datenbankarchitekturen. Vieweg. 1993

RAHM94 Rahm, E.: Mehrrechner-Datenbanksysteme – Grundlagen der verteilten und parallelen Datenbankverarbeitung. Addison-Wesley. Bonn. 1994

RAHM95 Rahm, E., Marek, R.: Dynamic Multi-Resource Load Balancing for Parallel Database Systems. Proc. 21th Int. Conf. VLDB. Zurich. 1995. 395–406

RAHM96 Rahm, E.: Dynamic Load Balancing in Parallel Database Systems. Proc. EURO-PAR 96, LNCS 1123. Springer. 1996. 37–52

RAMA84 Ramamohanarao, K., Sacks-Davis, R.: Recursive Linear Hashing. ACM Trans. Database Syst. 9:3. 1984. 369–391

RAMA96 Ramamritham, K., Chrysanthis, P.K.: A Taxonomy of Correctness Criteria in Database Applications. VLDB Journal 5:1. 1996. 85–97

RAMA97 Ramamritham, K., Chrysanthis, P.K.: Advances in Concurrency Control and Transaction Processing. IEEE Computer Society Press. 1997

RAMA98 Ramakrishnan, R.: Database Management Systems. McGraw-Hill. Boston. 1998

REGN85 Regnier, M.: Analysis of Grid File Algorithms. BIT 25. 1985. 335–357

REIC98 Reichert, M., Dadam, P.: $ADEPT_{FLEX}$ - Supporting Dynamic Changes of Workflows Without Loosing Control. Journal of Intelligent Information Systems 10. 1998. 93–129

REIN96 Reinert, J.: Ein Regelsystem zur Integritätssicherung in aktiven relationalen Datenbanksystemen. Univ. Kaiserslautern. Dissertation DISDBIS 11. infix-Verlag. 1996

RELL98 Relly, L., Schuldt, H., Schek, H.-J.: Exporting Database Functionality – The CONCERT Way. IEEE Data Engineering 21:3. 1998. 43–51

REUT80a Reuter, A.: A Fast Transaction-Oriented Logging Scheme for UNDO-Recovery. IEEE Trans. Software Eng. 6:4. 1980. 348–356

REUT80b Reuter, A.: Schnelle Datenbankrecovery mit Hilfe eines hardwaregestützten Schattenspeicheralgorithmus. Hardware für Software. Hauer, K.-H., Seeger, C. (Hrsg.). Teubner. 1980. 258–272

REUT81 Reuter, A.: Fehlerbehandlung in Datenbanksystemen. Hanser. 1981

REUT82 Reuter, A.: Concurrency on High-Traffic Data Elements. Proc. ACM Symp. on Principles of Database Systems. Los Angeles. 1982. 83–92

REUT83 Reuter, A.: An Analytic Model of Transaction Interference. Interner Bericht 68/83. Univ. Kaiserslautern. 1983. Ebenso in [KUMA96]

REUT86 Reuter, A.: Load Control and Load Balancing in a Shared Database Management System. Proc. 2nd Int. Conf. on Data Engineering. Los Angeles. 1986. 188–197

REUT90 Reuter, A.: Performance and Reliability Issues in Future DBMSs. Proc. Int. Symp. Database Systems of the 90s. A. Blaser (ed.). LNCS 466. Springer. 1990. 294–315

REUT97 Reuter, A., Schneider, K., Schwenkreis, F.: ConTracts Revisited. In [JAJO97]. 1997. 127–151

REZE97 Rezende, F.F., Härder, T.: Exploiting Abstraction Relationships‘ Semantics for Transaction Synchronization in KBMSs. Data and Knowledge Eng. 22:3. 1997. 233–259

REZE98 Rezende, F.F., Hergula, K.: The Heterogeneity Problem and Middleware Technology: Experiences with and Performance of Database Gateways. Proc. 24th Int. Conf. VLDB. New York. 1998. 146–157

RITT94 Ritter, N., Mitschang, B., Härder, T., Nink, U., Schöning, H..: Capturing Design Dynamics – The CONCORD Approach. Proc. 10th Int. Conf. Data Engineering. Houston. 1994. 440–451

RITT97 Ritter, N.: DB-gestützte Kooperationsdienste für technische Entwurfsanwendungen. Univ. Kaiserslautern. Dissertation DISDBIS 33. infix-Verlag. 1997

ROBI81 Robinson, J.T.: The k-d-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes. Proc. ACM SIGMOD Conf. Ann Arbor. 1981. 10–18

RODR76 Rodriguez-Rosell, J.: Empirical Data Reference behavior in Data Base Systems. IEEE Computer. 9:11. 1976. 9–13

ROSE78 Rosenkrantz, D.J., Stearns, R., Lewis, P.: System Level Concurrency Control for Distributed Database Systems. ACM Trans. Database Syst. 3:2. 1978. 178–198

ROSE92 Rosenblum, M., Ousterhout, J.K.: The Design and Implementation of a Log-Structured File System. ACM Trans. Computer Syst. 10:1. 1992. 26–52

ROTH97 Roth, M. T, Schwarz, P.: Don’t Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. Proc. 23rd Int. Conf. VLDB. Athens. 1997. 266–275

ROUS85 Roussopoulos, N., Leifker, D.: Direct Spatial Search of Pictorial Databases using Packed R-Trees. Proc. ACM SIGMOD Conf. Austin. 1985. 17–31

RUEM94 Ruemmler, C., Wilkes, J.: An Introduction to Disk Drive Modeling. IEEE Computer 27: 3. 1994. 17–28

RUSI95 Rusinkiewicz, M. et al.: Towards a Cooperative Transaction Model – The Cooperative Activity Model. Proc. 21th Int. Conf. VLDB. Zurich. 1995. 194–205

RYSK80 Ryska, N., Herda, S.: Kryptographsiche Verfahren in der Datenverarbeitung. Informatik-Fachberichte 24. Springer. 1980

SACC82 Sacco, G.M., Schkolnick, M.: A Mechanism for Managing the Buffer Pool in a Relational Database System using the Hot Set Model. Proc. 8th Int. Conf. VLDB. Mexico City. 1982. 257–262

SACC86 Sacco, G.M., Schkolnick, M.: Buffer Management in Relational Database Systems. ACM Trans. Database Syst. 11:4. 1986. 473–498

SALT87 Salton, G., McGill, M.J.: Information Retrieval – Grundlegendes für Informationswissenschaftler. McGraw-Hill. Hamburg. 1987

SAME84 Samet, H.: The Quadtree and Related Hierarchical Data Structures. ACM Computing Surv. 16:2. 1984. 187–260

SAME88 Samet, H.: Hierarchical Representations of Collections of Small Rectangles. ACM Computing Surv. 20:4. 1988. 271–309

SAME90 Samet, H.: Applications of Spatial Data Structures. Addison-Wesley. Reading. 1990

SAUT98 Sauter, G.: Interoperabilität von Datenbanksystemen bei struktureller Heterogenität – Architektur, Beschreibungs- und Ausführungsmodell zur Unterstützung der Integration und Migration. Univ. Kaiserslautern. Dissertation DISDBIS. infix-Verlag. 1998

SCHA98 Schaarschmidt, R., Bühnert, K., Herbst, A., Küspert, K., Schindler, R.: Konzepte und Implementierungsaspekte anwendungsorientierten Archivierens in Datenbanksystemen. Informatik – Forschung und Entwicklung 13:2. 1998. 79–89

SCHE78 Schek, H.-J.: The Reference String Indexing Method. LNCS 65. Springer. 1978. 432–459

SCHE80 Schek, H.-J.: Optimal Index Intervals. Information Processing‘80. North-Holland. 1980. 493–498

SCHK78 Schkolnik, M.: A Survey of Physical Database Design Methodology and Techniques. Proc. 4th Int. Conf. VLDB. Berlin. 1978. 479–487

SCHK85 Schkolnik, M., Tiberio, P.: Estimating the Cost of Updates in a Relational Database. ACM Trans. Database Syst. 10:2. 1985. 163–179

SCHÖ89 Schöning, H., Sikeler, A.: Cluster Mechanisms Supporting the Dynamic Construction of Complex Objects. Proc. 3rd Int. Conf. on Foundations of Data Organization and Algorithms. Paris. LNCS 367. Springer. 1989. 31–46

SCHÖ90 Schöning, H.: Integrating Complex Objects and Recursion. Proc. Int. Conf. on Deductive and Object-Oriented Databases. North-Holland. 1990. 573–592

SCHÖ98 Schöning, H.: The ADABAS Buffer Pool Manager. Proc. 24th Int. Conf. VLDB. New York. 1998. 675–679

SCHO81 Scholl, M.: New File Organization Based on Dynamic Hashing. ACM Trans. Database Syst. 6:1. 1981. 194–211

SCHU89 Schulze, M., Gibson, G., Katz, R., Patterson, D.A.: How Reliable is a RAID. Proc. 34th IEEE Comcon. 1989. 118–123

SCHW93 Schwenkreis, F.: APRICOTS – A Prototype Implementation of a ConTract System. Proc. 12th Symp. Reliable Distributed Systems. 1993

SEEG88 Seeger, B., Kriegel, H.-P.: Techniques for Design and Implementation of Spatial Access Methods. Proc. 14th Int. Conf. VLDB. Los Angeles. 360–371

SEEG90 Seeger, B., Kriegel, H.-P.: The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems. Proc. 16th Int. Conf. VLDB. Brisbane. 1990. 590–601

SELI79 Selinger, P., Astrahan, M., Chamberlin, D., Lorie, R., Price, T.: Access Path Selection in a Relational Database Management System. Proc. ACM SIGMOD Conf. Boston. 1979. 23–34

SELL87 Sellis, T., Roussopoulos, N., Faloutsos, C.: The R+-Tree: A Dynamic Index for Multidimensional Objects. Proc. 13th Int. Conf. VLDB. Brighton. 1987. 507–518

SESH98 Seshadri, P.: Enhanced Abstract Data Types in Object-Relational Databases. VLDB Journal 7:3. 1998. 130–140

SESS98 Sessions, R.: COM and DCOM. Wiley Computer Publishing. 1998

SEVE76a Severance, D.G., Lohman, G.M.: Differential Files: Their Application to the Maintenance of Large Databases. ACM Trans. Database Syst. 1:3. 1976. 256–267

SEVE76b Severance, D.G., Duhne, R.: A Practicioner‘s Guide to Addressing Algorithms. Comm. ACM 19:6. 1976. 314–326

SEVE77 Severance, D.G., Carlis, J.V.: A Practical Approach to Selecting Record Access Paths. ACM Computing Surv. 9:4. 1977. 259–272

SHAP86 Shapiro, L.: Join Processing in Database Systems with Large Main Memories. ACM Trans. Database Syst. 11:3. 1986. 239–264

SHAS88 Shasha, D., Goodman, N.: Concurrent Search Structure Algorithms. ACM Trans. Database Syst. 13:1. 1988. 53–90

SHAW89 Shaw, G., Zdonik, S.: An Object-Oriented Query Algebra. IEEE Data Engineering 12:3. 1989. 29–36

SHEK90 Shekita, E., Carey, M.: A Performance Evaluation of Pointer-Based Joins. Proc. ACM SIGMOD Conf. Atlantic City. NJ. 1990. 300–311.

SHER76 Sherman, S.W., Brice, R.S.: Performance of a Database Manager in a Virtual Memory System. ACM Trans. Database Syst. 1:4. 1976. 317–343

SHET97 Sheth, A.: From Contemporary Workflow Process Automation to Adaptive and Dynamic Work Activity Coordination and Collaboration. Proc. 8th Int. Workshop on Database and Expert Systems Applications (DEXA‘97), Toulouse. 1997

SHNE77 Shneiderman, B.: Reduced Combined Indexes for Efficient Multiple Attribute Retrieval. Information Systems 2:4. 1977. 149–154

SHNE78 Shneiderman, B.: Jump Searching: A Fast Sequential Search Technique. Comm. ACM 21:10. 1978. 831–834

SIKE88 Sikeler, A.: VAR-PAGE-LRU – A Buffer Replacement Algorithm Supporting Different Page Sizes. Proc. 1st Int. Conf. on Extending Data Base Technology. Venice. LNCS 303. Springer. 1988. 336–351

SIMO95 Simon, E., Kotz-Dittrich, A.: Promises and Realities of Active Database Systems. Proc. 21th Int. Conf. VLDB. Zurich. 1995. 642–653

SING97 Singhal, V., Smith, A.J.: Analysis of Locking Behavior in Three Real Database Systems. VLDB Journal 6:1. 1997. 40–52

SIX88 Six, H.-W., Widmayer, P.: Spatial Searching in Geometric Databases. Proc. 4th Int. Conf. Data Engineering. Los Angeles. 1988. 496–503

SMIT78 Smith, J.A.: Sequentiality and Prefetching in Data Base Systems. ACM Trans. Database Syst. 3:3. 1978. 223–247

SQL3 ISO/IEC CD 9075 Committee Draft. Database Language SQL. Jim Melton (ed.). July 1996

SRIN93 Srinivasan, V., Carey, M.J.: Performance of B+-Tree Concurrency Control Algorithms. VLDB Journal 2:4, 1993. 361–406

STEI98 Steiert, H.-P., Zimmermann, J.: JPMQ - An Advanced Persistent Message Queuing Service. Advances in Databases. Proc. 16th British Nat. Conf. on Data Management (BNCOD‘16), LNCS 1405. Springer. 1998. 1–18

STEN90 Stenström, P.: A Survey of Cache Coherence Schemes for Multiprocessors. IEEE Computer 23:6. 1990. 12–24

STON75 Stonebraker, M.: Implementation of Integrity Constraints and Views by Query Modification. Proc. ACM SIGMOD Conf. San Jose. 1975. 65–78

STON81 Stonebraker, M.: Operating System Support for Database Management. Comm. ACM 24:7. 1981. 412–418

STON83 Stonebraker, M., et al.: Performance Enhancements to a Relational Database System. ACM Trans. Database Syst. 8:2. 1983. 167–185

STON84 Stonebraker, M.: Virtual Memory Transaction Management. ACM Operating Systems Review 18:2. 1984. 8–16

STON86a Stonebraker, M.: The Case for Shared Nothing. IEEE Database Engineering 9:1. 1986. 4–9

STON86b Stonebraker, M.: Inclusion of New Types in Relational Data Base Systems. Proc. 2nd Int. Conf. Data Engineering. Los Angeles. 1986. 262–269

STON87 Stonebraker, M. et al.: Extensibility in POSTGRES. IEEE Database Engineering 10:2. 1987. 16–23

STON90 Stonebraker, M. et al.: On Rules, Procedures, Caching and Views in Data Base Systems. Proc. ACM SIGMOD Conf. Atlantic City. NJ. 1990. 281–290

STON93 Stonebraker, M., Olson, M.: Large Object Support in POSTGRES. Proc. 9th Int. Conf. Data Engineering. Vienna. 1993. 355–362

STON96a Stonebraker, M.: Object-Relational DBMSs – The Next Great Wave. Morgan Kaufmann. 1996

STON96b Stonebraker, M., Aoki, P.M., Devine, R., Litwin, W., Olson, M.: Mariposa: A New Architecture for Distributed Data. VLDB Journal 5:1. 1996. 48–63

SWAM93 Swami, A., Iyer, B.: A Polynomial Time Algorithm for Optimizing Join Queries. Proc. 9th Int. Conf. Data Engineering. Vienna. 1993. 345–354

SWAM94 Swami, A., Schiefer, B.: On the Estimation of Join Result Sizes. Proc. 4th Int. Conf. on Extending Data Base Technology. Cambridge, UK. LNCS 779. Springer. 1994. 287–300

TAMM82 Tamminen, M.: The Extendible Cell Method for Closest Point Problems. BIT 22. 1982. 27–41

TANE94 Tanenbaum, A.S.: Moderne Betriebssysteme. Hanser. 1994

TAFV74 Tafvelin, S.: Sequential Files on Cycling Storage. Proc. Information Processing‘74. North-Holland. 1974. 983–987

TAY85 Tay, Y.C., Goodman, N., Suri, R.: Locking Performance in Centralized Databases. ACM Trans. Database Syst. 10:4. 1985. 415–462

TENG84 Teng, J.Z., Gumaer, R.A.: Managing IBM Database 2 Buffers to Maximize Performance. IBM Sys. J. 23:2. 1984. 211–218

TEUH78 Teuhola, J.: A Compression Method for Clustered Bit-Vectors. Information Processing Letters 7:6. 1978. 308–311

THOM90 Thomasian, A., Rahm, E.: A New Distributed Optimistic Concurrency Control Method and a Comparison of its Performance with Two-Phase Locking. Proc. 10th IEEE Int. Conf. on Distributed Computing Systems. 1990. 294–301

THOM91 Thomasian, A., Ryu, I.K.: Performance Analysis of Two-Phase Locking. IEEE Trans. Software Eng. 17:5. 1991. 386–402

THOM93 Thomasian, A.: Two-Phase Locking and its Thrashing Behavior. ACM Trans. Database Syst. 18:4. 1993. 579–625, Ebenso in [KUMA96]

THOM94 Thomasian, A.: On a More Realistic Lock Contention Model and its Analysis. Proc. 10th Int. Conf. Data Engineering. Houston. 1994. 2–9

THOM95 Thomas, J., Gerbes, T., Härder, T., Mitschang, B.: Dynamic Code Assembly for Client-Based Query Processing. Proc. DASFAA’95, Singapore. 1995. 264–272

THOM96 Thomas, J.: An Approach to Query Processing in Advanced Database Systems. Univ. Kaiserslautern. Dissertation DISDBIS 16. infix-Verlag. 1996

THOM97 Thomasian, A.: A Performance Comparison of Locking Methods with Limited Wait Depth. IEEE Trans. on Knowledge and Data Eng. 9:3. 1997. 421–434

THOM98 Thomasian, A.: Concurrency Control: Performance, and Analysis. ACM Computing Surv. 30:1. 1998. 70–119

TPCW98 TPC Launches New E-Commerce Transactional Web Benchmark Effort. Transaction Processing Performance Council. http://www.tpc.org. 1998

TSIC78 Tsichritzis, D. C., Klug, A.: The ANSI/X3/Sparc DBMS Framework Report of the Study Group on Database Management Systems. Information Systems 3:3. 1978. 173–191

TYGA98 Tygar, J.D.: Atomicity versus Anonymity: Distributed Transactions for Electronic Commerce. Proc. 24th Int. Conf. VLDB. New York. 1998. 1–12

UHRO73 Uhrowczik, P.P.: Data Dictionary/Directories. IBM Sys. J. 12:4. 1973. 332–350

ULLM88 Ullman, J.D.: Principles of Database and Knowledge-Base Systems. Vol. 1. Computer Science Press. 1988

UNTE90 Unterauer, K.: Synchronisation des Logpuffers in Mehrprozeß-Datenbanksystemen. Informationstechnik–Computer, Systeme, Anwendungen 32:4. 1990. 281–286

VALD87 Valduriez, P.: Join Indices. ACM Trans. Database Syst. 12:2. 1987. 218–246

VAND91 Vandenberg, S., DeWitt, D.: Algebraic Support for Complex Objects with Arrays, Identity, and Inheritance. Proc. ACM SIGMOD Conf. Denver. 1991. 158–167

VARV89 Varvel, D.A., Shapiro, L.: The Computational Completeness of Extended Database Query Languages. IEEE Trans. Software Eng. 15. 1989. 632–638

VEKL85 Veklerov, E.: Analysis of Dynamic Hashing with Deferred Splitting. ACM Trans. Database Syst. 10:1. 1985. 90–96

VOSS93 Vossen, G., Gross-Hardt, M.: Grundlagen der Transaktionsverarbeitung. Addison-Wesley, 1993

WÄCH90 Wächter, H., Reuter, A.: Grundkonzepte und Realisierungstrategien des ConTract-Modells. Informatik – Forschung und Entwicklung 5:4. 1990. 202–212

WÄCH92 Wächter, H., Reuter, A.: The ConTract Model. In [ELMA92]. Chapter 7. 1992

WAGN73 Wagner, R.E.: Indexing Design Considerations. IBM Sys. J. 12:4. 1973. 351–367

WATE75 Waters, S.J.: Analysis of Self-Indexing Disk Files. The Computer Journal. 18:3. 1975. 200–205

WEDE74 Wedekind, H.: On the Selection of Access Paths in a Data Base System. Data Base Management. Klimbie, J.W., Koffeman, K.L. (eds.). North-Holland. 1974. 385–397

WEDE76 Wedekind, H., Härder, T.: Datenbanksysteme II. Reihe Informatik/18. BI-Wissenschaftsverlag. 1976

WEDE86 Wedekind, H., Zörntlein, G.: Prefetching in Real-Time Database Applications. Proc. ACM SIGMOD Conf. Washington. D.C. 1986. 215–226

WEIK86 Weikum, G.: Pros and Cons of Operating System Transactions for Data Base Systems. Proc. ACM/IEEE Fall Joint Computer Conf., 1986

WEIK89 Weikum, G.: Set-Oriented Disk Access to Large Complex Objects. Proc. 5th Int. Conf. Data Engineering. Los Angeles. 1989. 426–433

WEIK87 Weikum, G., Neumann, B., Paul, H.-B.: Konzeption und Realisierung einer mengenorientierten Seitenschnittstelle zum effizienten Zugriff auf komplexe Objekte. Proc. BTW‘87. Darmstadt. Informatik-Fachberichte 136. Springer. 1987. 212–230

WEIK91 Weikum, G.: Principles and Realization Strategies of Multilevel Transaction Management. ACM Trans. Database Syst. 16:1. 1991. 132–180

WEIK92 Weikum, G., Schek, H.: Concepts and Applications of Multilevel Transactions and Open Nested Transactions. In [ELMA92]. Chapter 13. 1992

WEIK93a Weikum, G., Hasse, C.: Multi-Level Transaction Management for Complex Objects: Implementation, Performance, Parallelism. VLDB Journal 2:4. 1993. 407–453

WEIK93b Weikum, G., Zabback, P.: I/O-Parallelität und Fehlertoleranz in Disk-Arrays – Teil 1 und 2. Informatik-Spektrum 16:3. 1993. 133–142 und 16:4. 1993. 206–214

WEIK94 Weikum, G., Hasse, C., Mönkeberg, A., Zabback, P.: The COMFORT Automatic Tuning Project. Information Systems 19:5. 1994. 381–432

WHAN90 Whang, K.-Y., Vander-Zanden, B. T., Taylor, H. M.: A Time-Linear Probabilistic Counting Algorithm for Database Applications. ACM Trans. Database Syst. 15:2. 1990. 208–229

WHAN98 Whang, S., Hellerstein, J.M., Lipkind, I.: Near-Neighbor Query Performance in Search Trees. submitted

WHIT92 White, S.J.: A Performance Study of Alternative Object Faulting and Pointer Swizzling Strategies. Proc. 18th Int. Conf. VLDB. Vancouver. 1992. 419–431

WHIT95 White, S.J., DeWitt, D.J.: Quickstore: A High Performance Mapped Object Store. VLDB Journal 4:4. 1995. 629–674.

WHIT96 White, D.A., Jain, R.: Similarity Indexing with the SS-Tree. Proc. 12th Int. Conf. Data Engineering. New Orleans. 1996. 516–523

WIDM91 Widmayer, P.: Datenstrukturen für Geodatenbanken. Entwicklungstendenzen bei Datenbanksystemen. Vossen, G., Witt, K.-U. (Hrsg). Oldenbourg. München. 317–361

WIDO91 Widom, J., Cochrane, R.J., Lindsay, B.: Implementing Set-Oriented Production Rules as an Extension to Starburst. Proc. 17th Int. Conf. VLDB. Barcelona. 1991. 275–285

WIDO96a Widom, J., Ceri, S.: Active Database Systems: Triggers and Rules for Advanced Database Processing. Morgan Kaufmann. 1996

WIDO96b Widom, J.: The Starburst Rule System. In [WIDO96a]. 87–109.

WILL82 Williams, R., Daniels, D., Haas, L.M., Lapis, G., Lindsay, B.G., Ng, P., Obermarck, R., Selinger, P.G., Walker, A., Wilms, P.F., Yost, R.A.: R*: An Overview of the Architecture. Reprinted in Readings in Database Systems (3rd ed.). Stonebraker, M., Hellerstein, J. M. (eds.), Morgan Kaufmann. 1998. 1–27

WILS90 Wilson, P.R.: Pointer Swizzling at Page Fault Time: Efficiently Supporting Huge Address Spaces on Standard Hardware. Tech. Rep. UIC-EECS-90-6. Univ. of Illinois at Chicago. 1990

WORA97 Worah, D., Sheth, A.: Transactions in Transactional Workflows. In [JAJO97]. 1997

XML XML: Principles, Tools, and Techniques. World Wide Web Journal 2:4. 1997

XOPE93 X/Open „Distributed Transaction Processing“. Dokumente. The X/Open Company Ltd. Reading, U.K. 1993

YAMA97 Yamada, H.: DVD Overview. Proc. Spring Compcon’97 Conf. San Jose. 1997. 287–290

YAN91 Yan, W.: Auswertung rekursiver Anfragen in Deduktiven Datenbanksystemen – eine Untersuchung der Strategien, des Leistungsverhaltens und der Realisierungsmöglichkeiten. Dissertation. Univ. Kaiserslautern. 1991

YU85 Yu, C.T. et al.: Adaptive Record Clustering. ACM Trans. Database Syst. 10:2. 1985. 180–204

YU96 Yu, P.S.: Modeling and Analysis of Concurrency Control Schemes. In [KUMA96]. 1996. 106–147

ZABB90 Zabback, P.: Optische und magneto-optische Platten in File- und Datenbanksystemen. Informatik-Spektrum 13:5. 1990. 260–275

ZABB94 Zabback, P.: I/O-Parallelität in Datenbanksystemen – Entwurf, Implementierung und Evaluation eines Speichersystems für Disk-Arrays. Diss. ETH Nr. 10629. ETH Zürich. 1994

ZANI97 Zaniolo, C. et al.: Advanced Database Systems. Chapter 4: Design Principles for Active Rules. Morgan Kaufmann. 1997

ZELL90 Zeller, H., Gray, J.: An Adaptive Hash Join Algorithm for Multiuser Environments. Proc. 16th Int. Conf. VLDB. Brisbane. 1990. 186–197

ZHAN97 Zhang, N., Härder, T.: On the Modeling Power of Object-Relational Data Models inTechnical Applications. Proc. 1st East-European Symposium on Advances in Databases and Information Systems (ADBIS‘97). St. Petersburg. 1997. 318–325

ZHAN99 Zhang, N., Härder, T.: On a Buzzword “Extensibility“ – What we have Learned from the ORIENT Project? Interner Bericht. SFB 501. Fachbereich Informatik. Univ. Kaiserslautern. 1999