

Lernziele

Nach dem Durcharbeiten der vorliegenden Kurseinheit sollen Sie die aus dem Problemgegenstand der physischen Datenorganisation, d.h. aus dem Problem der Abbildung konzeptioneller Datenmodelle auf physische Datenstrukturen, resultierenden Gestaltungs- und Entwurfsaufgaben abgrenzen und beschreiben können. Zudem sollen Sie Ansätze zur Bewältigung dieser Gestaltungs- und Entwurfsaufgaben benennen und erläutern können. Dies setzt voraus, dass Sie die Kenntnis folgender Konzepte erworben haben:

- Konzepte für den Entwurf interner Sätze wie z.B. schlüsselfreie Sätze, variable Satzlängen und virtuelle Felder.
- Konzepte für die Organisation dateiinterner Zugriffspfade unter Rückgriff auf bekannte Verfahren der Primär- und der Sekundärorganisation von Dateien sowie auf spezielle Techniken wie z.B. Multilist-Strukturen.
- Konzepte für die Organisation dateiübergreifender Zugriffspfade unter Verwendung spezieller Techniken wie invertierter Dateien, Zeigersätze und Kettsätze.

Außerdem sollen Sie die drei Ebenen der Datenintegrität, nämlich die Ebenen Datenkonsistenz, Datensicherheit und Datenschutz, nicht nur gegeneinander abgrenzen können, sondern spezifische, auf diesen Ebenen auftretende Gestaltungsprobleme sowie Konzepte zur Lösung dieser Gestaltungsprobleme erläutern können. Im Einzelnen sollen Sie:

- die unterschiedlichen Arten von Konsistenzbedingungen charakterisieren, den Aufbau von Konsistenzbedingungen erklären und die Formulierung von Konsistenzbedingungen in der Sprache SQL beschreiben können,
- das Problem der Synchronisation paralleler Datenbankzugriffe sowie die grundsätzliche Funktionsweise von zwei Synchronisationsverfahren, und zwar des optimistischen Verfahrens und des Sperrverfahrens, erläutern können,
- die verschiedenen Ebenen des Datenschutzes und einige diesen Ebenen zuzuordnende Schutzmaßnahmen beschreiben können.

Schließlich sollen Sie noch ein verhaltensorientiertes Lernziel erreicht haben: Sie sollen in der Lage sein, Konsistenzbedingungen mit Hilfe der ASSERT-Anweisung der Sprache SQL zu formulieren.

5 Physische Datenorganisation

Ausgehend von dem konzeptionellen Datenmodell und den Benutzeranforderungen - insbesondere dem Informationsbedarf, den Zugriffshäufigkeiten zu Daten und den geforderten Systemantwortzeiten - ist es das Ziel der physischen Datenorganisation, konzeptionelle Daten auf physische Daten abzubilden und effiziente Zugriffspfade zu den physischen Daten festzulegen. Unter physischen Daten sind hierbei auf Speichern abgelegte Daten zu verstehen. Die physische Datenorganisation wird durch das in Abb. 5.1 schematisch dargestellte Speicherkonzept geprägt.

Abbildung des konzeptionellen Modells auf physische Strukturen

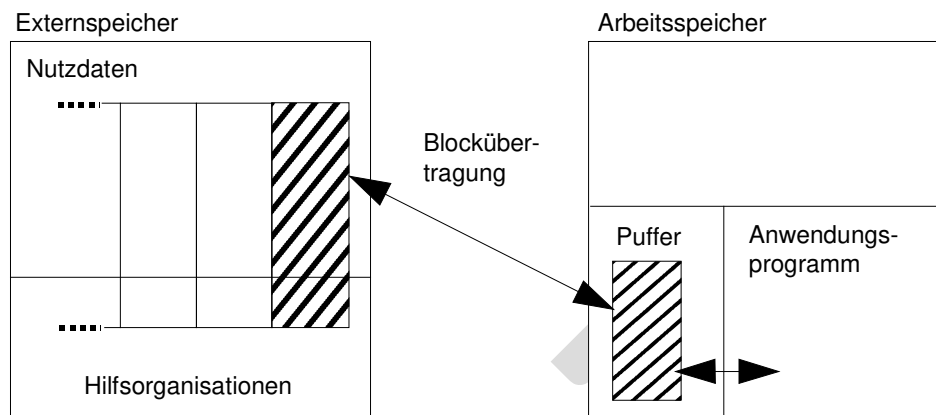


Abb. 5.1. Speicherkonzept bei großen Datenbeständen.

Da von großen Datenbeständen auszugehen ist, kommt eine ausschließlich interne Datenspeicherung nicht in Frage. Vielmehr erweist sich das in Abb. 5.1 skizzierte Speicherkonzept als unumgänglich. Es sieht die Speicherung der kompletten Datenbank einschließlich der Hilfsorganisationen für den Datenzugriff auf einem Externspeicher vor. Aus Effizienzgründen wird jeweils ein ganzer Datenblock zwischen dem Externspeicher und dem Arbeitsspeicher bzw. Puffer ausgetauscht. Ein im Arbeitsspeicher residierendes Anwendungsprogramm kann nun auf die in seinem Puffer befindlichen Daten zugreifen (vgl. hierzu auch Abb. 2.3 und die dazugehörigen Erläuterungen).

Das skizzierte Speicherkonzept lässt die Datenstrukturierung und die Vorgabe von Zugriffspfaden zu Daten noch völlig offen. Diese Gestaltungsspielräume zu nutzen ist das zentrale Anliegen der physischen Datenorganisation. Hierbei treten im Wesentlichen drei Gestaltungs- und Entwurfsaufgaben auf:

- Abbilden konzeptioneller Entitäten auf physische Datenstrukturen bzw. interne Sätze,
- Festlegen von Zugriffspfaden, welche Selektionen von internen Sätzen innerhalb eines Satztyps ermöglichen,
- Festlegen von Zugriffspfaden, welche Verbindungen zwischen internen Sätzen verschiedenen Satztyps herstellen und entsprechende Selektionen ermöglichen.

drei Entwurfsaufgaben

Auf die erste Entwurfsaufgabe, die man auch mit dem Begriff "Satzbildungsproblem" umschreiben kann, geht das Kap. 5.1 ein. Bei der zweiten Entwurfsaufgabe ist danach zu unterscheiden, ob zu Sätzen einer Datei aufgrund vorgegebener Primärschlüsselwerte oder Sekundärschlüsselwerte zugegriffen werden soll. Im ersten Fall geht es um die in Kap. 5.2 behandelte Primärorganisation von internen Sätzen und im zweiten Fall um die

Inhalt des Kapitels

in Kap. 5.3 behandelte Sekundärorganisation. Die dritte Entwurfsaufgabe schließlich, die Gestaltung von Verbindungen zwischen internen Sätzen unterschiedlichen Typs, wird in Kap. 5.4 erörtert.

Vor allem die Bereiche der Primär- und Sekundärorganisation von Daten bzw. Dateien wurden im Kurs Algorithmen und Datenstrukturen bereits ausführlich behandelt. Im vorliegenden Kapitel werden die genannten Entwurfsaufgaben daher nur in ihren Grundzügen beschrieben.

5.1 Entwurf interner Sätze

Bildung interner Sätze

Interne Sätze entstehen durch das Zusammenfassen von Attributen konzeptioneller Entitäten zu Feldern von Sätzen auf der Ebene des internen Modells. Dabei müssen die Entitäten der konzeptionellen Ebene nicht zwangsläufig auf analog zusammengesetzte Satztypen abgebildet werden. Meist bestehen jedoch die internen Satztypen aus den Attributen je eines Entitätstyps.

Mit dem Übergang von der konzeptionellen zur internen Ebene ändern sich die Begriffe zur Bezeichnung von Dateneinheiten. Die einander entsprechenden Begriffe gehen aus der Tabelle 5.1 hervor.

Tab. 5.1. Dateneinheiten auf der konzeptionellen und auf der internen Ebene.

Konzeptionelle Ebene	Interne Ebene
Attribut	Feld
Entität	Satz
Entitätstyp	Satztyp
Entitätsmenge	Datei

Das Problem des Entwurfs interner Sätze kann nun wie folgt formuliert werden:

Problem des Entwurfs interner Sätze

Bei einer gegebenen Gesamtmenge von Attributen, einer gegebenen Menge von Anwendungsprogrammen und gegebenen Benutzungshäufigkeiten der einzelnen Attribute durch die Anwendungsprogramme sind Teilmengen von Attributen - genannt interne Sätze - derart zu bilden, dass die Verarbeitungseffizienz über alle Anwendungsprogramme maximal ist.

schlüsselfreie Sätze

Eine derart verstandene Entwurfsaufgabe kann zu einem spezifischen Nachteil führen, dem Entstehen schlüsselfreier Sätze. Darunter sind Anordnungen von Attributen zu verstehen, die keinen Identifikationsschlüssel enthalten. Um Zugriffe zu schlüsselfreien Sätzen zu ermöglichen, bieten sich zwei Maßnahmen an: Zum einen die Verkettung zusammengehöriger schlüsselfreier Sätze und zum anderen die Verkettung schlüsselfreier Sätze mit inhaltlich in Beziehung stehenden Sätzen, welche Schlüssel enthalten. Der oben angesprochene Gewinn an Verarbeitungseffizienz muss also durch einen zusätzlichen Verwaltungsaufwand erkaufte werden.

Bei der direkten Abbildung eines Entitätstyps der konzeptionellen Ebene auf einen Satztyp der internen Ebene bietet es sich gegebenenfalls an, auf zwei spezielle Gestaltungskonzepte zurückzugreifen: variable Satzlengthen und virtuelle Felder.

Variable Satzlängen ergeben sich zwangsläufig im Falle nicht atomarer Attribute bzw. wiederholender Gruppen. In Kap. 3.4.1 wurde gezeigt, wie man wiederholende Gruppen auf der konzeptionellen Ebene durch den Vorgang der Normalisierung eliminieren und damit die von ihnen ausgehenden Nachteile vermeiden kann. Auf der physischen Ebene mag es angezeigt sein, wiederholende Gruppen explizit darzustellen. Zwei mögliche Darstellungsformen zeigt die Abb. 5.2. Zugrunde gelegt werde ein Beispiel, das die Sprachkenntnisse von Mitarbeitern einer Firma betrifft.

variable Satzlängen

Die Darstellungsform in Abb. 5.2 b) verwendet feste Satzlängen mit einer vorgegebenen maximalen Anzahl von Wiederholungen. Bei der Darstellungsform in Abb. 5.2 c) werden der Anfang und das Ende eines Spracheneintrags jeweils durch ein Begrenzungszeichen markiert. Dadurch entstehen Sätze variabler Länge. In beiden Fällen steht pro Eintrag ein Speicherplatz fester Länge zur Verfügung.

Mitarbeiter \ Sprachen	Steno-graphie	Englisch	Japanisch	Russisch	Spanisch
Abele	x	x			
Dreher		x			
Müller	x		x	x	
Muster	x	x		x	x
Zander			x		

a) Sprachkenntnisse von Mitarbeitern

1	Abele	Stenogr	Englisch		
2	Dreher	Englisch	Spanisch		
3	Müller	Stenogr	Japanisch	Russisch	
4	Muster	Stenogr	Englisch	Russisch	Spanisch
5	Zander	Japanisch			

Darstellung wiederholender Gruppen

b) Darstellung mit fester Satzlänge

1	Abele	*	Stenogr	Englisch	*	
2	Dreher	*	Englisch	Spanisch	*	
3	Müller	*	Stenogr	Japanisch	Russisch	*
4	Muster	*	Stenogr	Englisch	Russisch	Spanisch
5	Zander	*	Japanisch	*		

c) Darstellung mit variabler Satzlänge

Abb. 5.2. Zwei Darstellungsformen für wiederholende Gruppen.

Virtuelle Felder sind Felder, auf die der Benutzer zugreifen kann, obwohl ihre Werte nicht abgespeichert wurden. Dies ist deshalb möglich, weil bei einem Zugriff zu einem virtuellen Feld der Feldwert aus den Werten anderer Felder berechnet wird. Beispiels-

virtuelle Felder

weise könnte man ein Feld *Betrag*, dessen Werte sich als Produkte der Werte der Felder *Menge* und *Preis* darstellen, als virtuelles Feld definieren.

Variable Satzlängen und virtuelle Felder führen zu einer Einsparung an Speicherplatz. Diesem Vorteil steht der Nachteil des zusätzlichen Verwaltungs- und Berechnungsaufwandes gegenüber. Insgesamt ist bei dem Entwurf interner Sätze genau zu prüfen, ob die Vorteile, die sich aus der Verwendung von

- schlüsselfreien Sätzen,
- variablen Satzlängen und
- virtuellen Feldern

ergeben, den in Kauf zu nehmenden zusätzlichen Verwaltungs- und Berechnungsaufwand überwiegen. Diese Fragestellung berührt allerdings primär den Datenbankadministrator und weniger den Datenbankbenutzer.

Übungsaufgabe 5.1

Strebt man bei der Abbildung konzeptioneller Entitäten auf interne Sätze eine maximale Verarbeitungseffizienz an, so ist gegebenenfalls die Bildung schlüsselfreier Sätze in Kauf zu nehmen. Erläutern Sie diese Aussage an einem Beispiel aus dem betrieblichen Bereich.

5.2 Primärorganisation von Dateien

Mit den Verfahren der Primärorganisation von Dateien lassen sich Zugriffspfade erzeugen, welche die Selektion von internen Sätzen eines Satztyps für vorgegebene Primärschlüsselwerte unterstützen. Man unterscheidet zwischen Organisationsformen mit und ohne Hilfsorganisation.

a) Organisationsform ohne Hilfsorganisation

Die Möglichkeiten der Primärorganisation ohne Hilfsorganisation beschränken sich im Wesentlichen auf die sequentielle Organisationsform. Bei der sequentiellen Organisation werden die internen Sätze eines Satztyps - also die Sätze einer Datei - sortiert nach dem Primärschlüssel in aufeinanderfolgenden Blöcken abgespeichert. Die Blöcke, die man sich z.B. als Sektoren oder Spuren eines Plattenspeichers vorzustellen hat, müssen nicht zwingend physisch, aber logisch aufeinanderfolgen.

sequentielle Organisation

Für einige der wichtigsten Datenbankoperationen gilt für den Fall der sequentiellen Dateioorganisation:

- Ein sequentieller Zugriff zu den Sätzen einer Datei wird durch die Organisationsform unmittelbar unterstützt und ist daher sehr gut möglich.
- Die Selektion bzw. Suche eines Satzes für einen gegebenen Primärschlüssel ist nicht sehr aufwendig, da der Block, in dem sich der gesuchte Satz befindet, beispielsweise mittels binärer Suche recht schnell gefunden werden kann; nach dem Transfer des Blocks in den Arbeitsspeicher wird der gewünschte Satz mittels sequentieller Suche ermittelt.
- Das Löschen und Einfügen von Sätzen kann sehr aufwendig werden, falls Sätze über Blockgrenzen hinweg verschoben werden müssen, um Lücken zu vermeiden oder zu schaffen.

Operationen auf
sequentiell organisierten
Dateien

Wegen des im Vergleich zu anderen Organisationsformen doch erheblichen Selektionsaufwandes und des gegebenenfalls sehr hohen Änderungsaufwandes besitzt die sequentielle Organisation im Datenbankbereich nur eine geringe Bedeutung. Sie wird häufig zu Zwecken der Datensicherung und Datenrekonstruktion verwendet.

b) Organisationsformen mit Hilfsorganisation

Durch die Verwendung von Hilfsorganisationen lässt sich die Effizienz von Datenbankoperationen wesentlich steigern. In Frage kommen vor allem

- die indexsequentielle Organisation,
- die Organisation mit B- oder B*-Bäumen und
- die Hash-Organisation.

Bei der indexsequeentiellen Organisation wird die sequentielle Abspeicherung der Sätze einer Datei mit einem ein- oder mehrstufigen Primärindex kombiniert. Für einen gegebenen Primärschlüsselwert gestattet der Primärindex die sehr schnelle Ermittlung des Blocks, in dem sich der gesuchte Satz befindet. Der Block wird in den Arbeitsspeicher

indexsequentielle
Organisation

transferiert und dort kann der fragliche Satz z.B. mittels sequentieller Suche bestimmt werden.

Mutationen lassen sich ebenfalls auf effiziente Weise organisieren. Lediglich das Einfügen von Sätzen kann zu Problemen führen, wenn der angesprochene Block belegt ist. In diesem Fall behilft man sich mit dem Einfügen in einen separaten Überlaufblock oder mit dem Ersetzen des angesprochenen Blocks durch zwei neue, logisch aufeinander folgende Blöcke.

B-Bäume

B-Bäume und B*-Bäume sind spezielle Formen eines Primärindex. Sie besitzen eine Baumstruktur mit variierbarer Breite und Höhe und ermöglichen daher eine effizientere Gestaltung von Indexoperationen als übliche Indexformen. Vor allem bei sehr großen Datenbeständen kommt dieser Vorteil zum Tragen. Für Suchoperationen gelten in etwa die Aussagen zur indexsequentiellen Organisation, wobei sich allerdings das Einfügen und Ändern insofern einfacher gestaltet, als keine Überlaufprobleme auftreten.

Hash-Organisation

Die Hash-Organisation, auch gestreute Speicherung genannt, bedient sich einer Hash-Funktion, welche jedem in einer Datei auftretenden Primärschlüsselwert eine Satz- oder Blockadresse zuordnet. Mittels der Hash-Funktion können sämtliche Suchoperationen und Mutationen auf effiziente Weise abgewickelt werden. Dies gilt jedoch nicht, falls Kollisionen gehäuft auftreten. Die Behandlung von Kollisionen, sei es durch Abspeicherung von synonymen Sätzen in einem separaten Überlaufbereich oder in dem nächstfolgenden freien Speicherplatz oder mittels eines anderen Verfahrens, kann mit einem erheblichen Aufwand verbunden sein.

Eine Beurteilung der genannten Verfahren der Primärorganisation zeigt die Tabelle 5.2. Für Datenbanksysteme eignen sich vor allem die indexsequentielle Organisation und mehr noch die Primärorganisation mit B- und B*-Bäumen.

Tab. 5.2. Beurteilung von Verfahren der Primärorganisation von Dateien.

Organisationsform	Sequentielle Zugriffe	Suchoperationen	Mutationen
Sequentielle Organisation	sehr gut	schlecht	schlecht ²⁾
Indexsequentielle Organisation	gut	sehr gut	gut ³⁾
Primärorganisation mit B- und B*-Bäumen	gut	sehr gut	gut
Hash-Organisation	ungeeignet	sehr gut ¹⁾	gut ³⁾

1) Nicht aber bei gehäuften Kollisionen.

2) Jedoch bei der Mutation von Massendaten sinnvoll.

3) Nicht aber bei gehäuften Überläufen.

Beurteilung von Verfahren
der Primärorganisation

Übungsaufgabe 5.2

Bei den Verfahren der Primärorganisation von Dateien, welche eine Hilfsorganisation verwenden, erhöhen gehäufte Überläufe den Aufwand von Mutationen beträchtlich. Im Datenbankbetrieb können jedoch auch Unterläufe auftreten. Wie wirken sich Unterläufe auf den Mutationsaufwand aus?

5.3 Sekundärorganisation von Dateien

Verfahren der Sekundärorganisation unterstützen die Selektion von Datensätzen eines Satztyps für Selektionskriterien, welche keine Primärschlüssel darstellen. Bei diesen Selektionskriterien kann es sich um beliebige Nichtschlüsselattribute handeln. Auf den auch als Sekundärschlüssel bezeichneten Selektionskriterien lassen sich unterschiedliche Hilfsorganisationen für Zugriffszwecke realisieren. Nachfolgend werden exemplarisch zwei Sekundärorganisationen behandelt, Multilist-Strukturen und invertierte Dateien. Beide Organisationsformen überlagern die jeweilige Primärorganisation einer Datei.

Sekundärschlüssel

a) Multilist-Strukturen

In einer Multilist-Struktur werden die Sätze eines Satztyps miteinander verkettet, welche für einen gegebenen Sekundärschlüssel den gleichen Schlüsselwert aufweisen. Da für jeden Wert eines Sekundärschlüssels eine solche verkettete Liste anzulegen ist, ergibt sich eine Struktur mit mehreren Listen - hier als Multilist-Struktur bezeichnet. Auf das erste Element einer jeden zu einem Sekundärschlüssel gehörigen Liste verweist ein Anker. Sämtliche Anker einer Multilist-Struktur fasst man in einem speziellen Sekundärindex, der Ankertabelle, zusammen.

Multilist-Struktur

Ankertabelle		Primärorganisation (Auszug)					Zeiger	
Ort	VZOrt	KundenNr	Name	Gruppe	Ort	...	VZOrt	RZOrt
Aachen	0047	0019	Abel	A	Berlin		0103	1982
Berlin	0019	0047	Behrend	C	Aachen		1341	1341
Bremen	0053	0053	Bult	A	Bremen		0157	0161
		0103	Diemel	B	Berlin		1406	<u>0019</u>
		0157	Dunger	B	Bremen		0161	<u>0053</u>
		0161	Emster	C	Bremen		<u>0053</u>	0157
		1341	Gallus	A	Aachen		<u>0047</u>	<u>0047</u>
		1406	Gimpel	C	Berlin		1982	0103
		1982	Hilpert	B	Berlin		<u>0019</u>	1406

Legende: VZOrt - Vorwärtszeiger auf Ort

RZOrt - Rückwärtszeiger auf Ort

Abb. 5.3. Einfaches Beispiel für eine Multilist-Struktur.

Ein einfaches Beispiel einer Multilist-Struktur zeigt die Abb. 5.3. Die Primärorganisation ist nicht vollständig dargestellt. Die Sekundärorganisation besteht aus einer Ankertabelle sowie aus Vorwärts- und Rückwärtszeigern auf das Attribut *Ort*, welches hier Sekundärschlüssel ist. Die Vorwärts- und Rückwärtszeiger wurden in die Primärorganisation einbezogen. Die Zeiger verketteten die zu je einem Sekundärschlüsselwert gehörigen Sätze in Vorwärts- und Rückwärtsrichtung.

Grundsätzlich können Zeiger realisiert werden als

Realisierung von Zeigern

- physische Zeiger, d.h. als Speicheradressen von internen Sätzen,
- logische Zeiger, d.h. als relative Adressen von internen Sätzen,
- symbolische Zeiger, d.h. als Primärschlüsselwerte von Sätzen.

Die erste Realisationsform ist unflexibel und in Datenbanksystemen daher unüblich. Die dritte Realisationsform besitzt zwar den Vorteil der völligen Unabhängigkeit von der Speicherung der Sätze, muss aber bei dem Zugriff zu Sätzen auf der Primärorganisation aufsetzen. Sie weist aus diesem Grunde ein ungünstigeres Zeitverhalten auf als die zweite Realisationsform.

Bei der in Abb. 5.3 verwendeten Realisationsform umfasst der Selektionsprozess für einen vorgegebenen Sekundärschlüsselwert, beispielsweise den Wert Berlin, mehrere Schritte:

Selektionsprozess für einen
Sekundärschlüsselwert

- (1) Zugriff zur Ankertabelle und Suche des Eintrags mit dem Sekundärschlüsselwert Berlin, beispielsweise mit dem binären Suchverfahren. Entnahme des Ankers aus dem ermittelten Eintrag; hier ist das ein symbolischer Zeiger mit dem Wert 0019.
- (2) Benutzung der hier nicht vollständig dargestellten Primärorganisation für den Einstieg in die Liste, welche die Sätze mit dem Sekundärschlüsselwert Berlin verkettet. Hier ist zufällig der erste Satz der Nutzdatei zugleich auch das erste Element dieser Liste. Lesen des ersten Listenelements einschließlich des Vorwärtszeigers *VZOrt*, der hier den Wert 0103 aufweist.
- (3) Benutzung der Primärorganisation für den Zugriff zu dem zweiten Listenelement, auf das der symbolische Zeiger *VZOrt* mit dem Wert 0103 verweist. Lesen des zweiten Listenelements einschließlich des Zeigers *VZOrt*, der den Wert 1406 aufweist.
- (4) Wiederholung von Schritt (3), bis das Listenende erreicht ist. Das Listenende wird durch den Rückverweis vom letzten Listenelement auf den Listenanfang angezeigt, hier also durch den Wert 0019 des Vorwärtszeigers *VZOrt*.

Während bei einem Dateizugriff mittels der Primärorganisation für einen vorgegebenen Primärschlüsselwert entweder kein Satz oder genau ein zutreffender Satz ermittelt wird, können für einen vorgegebenen Sekundärschlüsselwert kein, ein oder mehrere zutreffende Sätze gefunden werden. Im Falle mehrerer zutreffender Sätze ergeben sich lange Listen und entsprechend hohe Selektionszeiten.

Übungsaufgabe 5.3

Betrachtet werde das in Abb. 5.3 dargestellte Beispiel einer Multilist-Struktur. Welchem Zweck die in die Primärorganisation einbezogenen Vorwärtszeiger *VZOrt* dienen, geht aus dem beschriebenen Selektionsprozess hervor. Erläutern Sie, wozu sich die Rückwärtszeiger *RZOrt* verwenden lassen.

b) Invertierte Dateien

Eine invertierte Datei (engl. inverted file) ist eine Hilfsorganisation für einen Sekundärschlüssel, welche für einen beliebigen Sekundärschlüsselwert die Primärschlüsselwerte der Sätze ausweist, in denen dieser Sekundärschlüsselwert auftritt. Somit besitzt eine invertierte Datei die Form einer Tabelle. Jeder Eintrag in der Tabelle besteht aus einem Sekundärschlüsselwert sowie aus keinem oder einem oder mehreren Primärschlüsselwerten. Zur weiteren Verdeutlichung diene das in Abb. 5.4 angegebene Beispiel.

Invertierte Datei

Ort	ZOrt
Aachen	0047 1341
Berlin	0019 0103 1406 1982
Bremen	0053 0157 0161

Legende:

ZOrt - Zeiger auf Ort

Nutzdatendatei

KundenNr	Name	Gruppe	Ort	...
0019	Abel	A	Berlin	
0047	Behrend	C	Aachen	
0053	Bult	A	Bremen	
0103	Diemel	B	Berlin	
0157	Dunger	B	Bremen	
0161	Emster	C	Bremen	
1341	Gallus	A	Aachen	
1406	Gimpel	C	Berlin	
1982	Hilpert	B	Berlin	

Beispiel einer invertierten Datei

Abb. 5.4. Einfaches Beispiel für eine invertierte Datei.

Wie die Abb. 5.4 zeigt, liegt die Sekundärorganisation völlig außerhalb der Nutzdatendatei. Die Nutzdaten und die Zugriffspfade für die Sekundärschlüssel können daher - anders als bei Multilist-Strukturen - völlig unabhängig voneinander verwaltet werden.

Die Selektion von Sätzen für einen vorgegebenen Sekundärschlüsselwert, beispielsweise den Wert Berlin, verläuft in folgenden Schritten:

- (1) Zugriff zur invertierten Datei und Ermitteln des Eintrags mit dem Sekundärschlüsselwert Berlin unter Verwendung eines effizienten Suchverfahrens wie z.B. der binären Suche. Entnahme der Primärschlüsselwerte aus dem Eintrag. Hier sind das die Werte 0019, 0103, 1406 und 1982. Diese Werte repräsentieren symbolische Zeiger, welche genau auf diejenigen Sätze verweisen, in denen der Sekundärschlüssel *Ort* den Wert Berlin aufweist.

Selektionsprozess für einen Sekundärschlüsselwert

- (2) Zugriff zu den Sätzen der Nutzdatendatei, auf die die im ersten Schritt ermittelten Zeiger verweisen. Bei den Dateizugriffen wird die hier nicht dargestellte primäre Hilfsorganisation verwendet.

schnellerer Dateizugriff
kostet Speicherplatz und
Verwaltungsaufwand

Ebenso wie Multilist-Strukturen beanspruchen invertierte Dateien nicht eben wenig Speicherplatz. Beiden Organisationsformen ist außerdem der relativ hohe Verwaltungsaufwand gemein, der bei Mutationen auf der Nutzdatendatei entsteht. Dabei schneiden invertierte Dateien etwas besser ab als Multilist-Strukturen, weil sich Änderungen in der Sekundärorganisation nicht auf die Nutzdatendatei erstrecken.

5.4 Verbindung von Dateien

dateiübergreifende
Zugriffspfade

Operationen auf Datenbanken, welche die Selektion von Daten aus mehreren Dateien beinhalten, sind für betriebliche Datenbank Anwendungen geradezu typisch. Durch geeignet angelegte, dateiübergreifende Zugriffspfade kann die Effizienz solcher Operationen wesentlich gesteigert werden. Hinweise darauf, welche Dateien durch Zugriffspfade miteinander zu verbinden sind, ergeben sich bei der Entwicklung datenbankgestützter Anwendungssysteme bereits aus der Anforderungsdefinition. So weist das logische Datenmodell die grundlegenden konzeptionellen Verbindungen zwischen den einzelnen Relationen aus. Und aus den definierten Transaktionen gehen die darüber hinaus herzustellenden Verbindungen zwischen Relationen explizit hervor. Für die physische Datenorganisation sind diese Hinweise relevant, da Relationen häufig auf interne Satztypen abgebildet werden.

Dateiverbindungen mit und
ohne Hilfsorganisation

Zur Realisierung von Dateiverbindungen eignen sich die bekannten Verfahren der Hilfsorganisation, insbesondere Verfahren der Verkettung und invertierte Dateien. Zudem lassen sich Verbindungen über die physisch zusammenhängende Speicherung von Daten herstellen. Bei reinen Relationensystemen werden keine Hilfsorganisationen zur Herstellung von Dateiverbindungen verwendet. Dateiverbindungen ergeben sich vielmehr implizit aus den in den Relationen auftretenden globalen Attributen. Nachfolgend werden zunächst Dateiverbindungen behandelt, die nicht auf Hilfsorganisationen beruhen, und danach mit Hilfsorganisationen realisierte Verbindungsformen.

a) Dateiverbindung ohne zusätzliche Hilfsorganisation

Wie bereits erwähnt wurde, lässt sich eine Dateiverbindung durch die physisch zusammenhängende Speicherung von Sätzen realisieren. Dieser Ansatz, der ohne zusätzliche Hilfsorganisation auskommt, ist allerdings nur für hierarchische Beziehungen zwischen Dateien - also 1-1-, 1-c-, 1-m- und 1-mc- Beziehungen - geeignet.

physisch zusammen-
hängende Speicherung

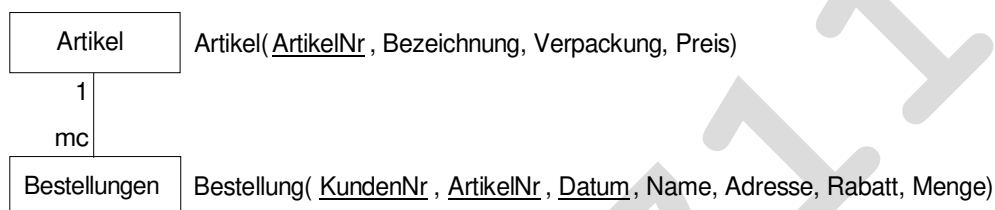
Bei der Behandlung des hierarchischen Datenmodells in Kap. 3.2 wurde gezeigt, dass hierarchische Beziehungen durch die sequentielle Speicherung von "Vätern" und "Söhnen" realisiert werden können. Man vergleiche hierzu vor allem die Abb. 3.19 und die Übungsaufgabe 3.10. In der Abb. 3.19 a) spielt die Datei *Kunden* die Rolle des Vaters und die Datei *Bestellungen* enthält die dazugehörigen Söhne. Wie die einzelnen Vatersätze mit den jeweiligen Sohnsätzen physisch zu organisieren sind, demonstriert das in Abb. 3.19 b) dargestellte Beispiel. Dieser Darstellung entnimmt man, dass die Verbindung

zweier Dateien durch die physisch zusammenhängende Speicherung der Sätze nur über eine Verzahnung der beiden Primärorganisationen der Dateien herstellbar ist.

In einem Relationensystem werden logische Beziehungen zwischen Relationen keinesfalls physisch realisiert. Insbesondere auch nicht durch die physisch zusammenhängende Speicherung von Tupeln. Die lediglich auf der logischen Ebene durch globale Attribute ausgedrückten Beziehungen zwischen Relationen müssen daher bei der Auswertung von Relationen hergestellt werden. An dem in der Übungsaufgabe 3.10 behandelten, einfachen Beispiel sei die prinzipielle Vorgehensweise demonstriert.

Beispiel 5.1

Gegeben sei folgender Ausschnitt aus einem Datenmodell:



Dateiverbindungen in
einem Relationensystem

Angenommen sei, dass die Tupel der Relationen *Artikel* und *Bestellung* je als physische Sätze zweier separater Dateien abgespeichert wurden. Gesucht sei der aus den Bestellungen resultierende Umsatz pro Artikel.

Die Ermittlung der genannten Umsatzgröße kann nur durch einen Auswertungsprozess erfolgen, in dem eine Verbindung zwischen den beiden Relationen bzw. Dateien hergestellt wird. Dies verdeutlicht das nachfolgend angegebene Algorithmusfragment zur Ermittlung des Artikelumsatzes:

```

...;
INPUT(Artikelsatz);
WHILE (Artikelsatz vorhanden) DO
    Artikelumsatz := 0;
    INPUT(Bestellsatz);
    WHILE(Bestellsatz vorhanden) DO
        IF(Artikel.ArtikelNr = Bestellung.ArtikelNr) THEN
            Artikelumsatz := Artikelumsatz + (Artikel.Preis *
                Bestellung.Menge) * (1-Bestellung.Rabatt))
        ENDIF;
        INPUT(Bestellsatz);
    ENDWHILE;
    OUTPUT(Artikel.ArtikelNr, Artikelumsatz);
    INPUT(Artikelsatz);
ENDWHILE;
...;
  
```

Algorithmusfragment zur
Ermittlung des
Artikelumsatzes

Die Dateiverbindung wird über zwei ineinander geschachtelte WHILE-Schleifen hergestellt. Während die äußere WHILE-Schleife nacheinander sämtliche Artikelsätze abarbeitet, erstreckt sich die innere WHILE-Schleife über die Menge der Bestellsätze. Man

beachte, dass die äußere Schleife pro Artikelsatz nur einmal und die innere Schleife i.a. mehrmals durchlaufen wird.

Zu einem in der äußeren Schleife gelesenen Artikelsatz werden - im Zuge des Durchlaufens der inneren Schleife - sämtliche Bestellungen zu der aktuellen Artikelnummer herausgefiltert und bei der Fortschreibung des Artikelumsatzes berücksichtigt. Nach dem Durchlaufen der inneren Schleife wird der Artikelumsatz für die aktuelle Artikelnummer ausgegeben und es wird zu dem nächsten Artikel übergegangen.

Angemerkt sei noch, dass die im Kurs "Algorithmen und Datenstrukturen" eingeführte Notation für Algorithmen verwendet wird.

b) Dateiverbindung mit zusätzlichen Hilfsorganisationen

Zur Realisierung von dateiübergreifenden Zugriffspfaden eignen sich u.a. folgende Verfahren der Hilfsorganisation:

Verfahren der
Hilfsorganisation

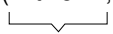
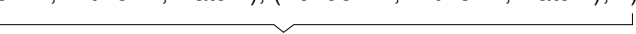
- invertierte Dateien,
- Verkettung,
- Zeigersätze und
- Kettsätze.

Drei dieser Verfahren werden nachfolgend nur kurz erläutert, das Verfahren der Kettsätze dagegen ausführlicher.

invertierte Datei

Zwischen zwei Dateien lässt sich durch das Anlegen einer invertierten Datei eine Verbindung herstellen. Geht man z.B. von den in Beispiel 5.1 angegebenen Relationen bzw. Dateien aus, so ist die invertierte Datei wie folgt aufzubauen:

Für jeden Wert des globalen Attributs *ArtikelNr* ist ein Eintrag in der invertierten Datei vorzusehen, der neben der aktuellen Artikelnummer die Primärschlüsselwerte sämtlicher Bestellsätze enthält, in denen die aktuelle Artikelnummer als Fremdschlüsselwert auftritt. Ein Eintrag in der invertierten Datei besitzt hier also folgendes Aussehen:

(ArtikelNr, (KundenNr, ArtikelNr, Datum), (KundenNr, ArtikelNr, Datum),...)	
 aktuelle Artikel- nummer in der Artikeldatei	 Primärschlüsselwerte der Sätze der Bestelldatei, in denen die aktuelle Artikelnummer als Fremd- schlüssel auftritt

Sollen nun für eine gegebene Artikelnummer sämtliche zugehörigen Bestellsätze selektiert werden, so lassen sich dem zutreffenden Eintrag der invertierten Datei die Primärschlüsselwerte dieser Bestellsätze entnehmen. Unter Verwendung der Primärorganisation der Bestelldatei kann jetzt auf die Bestellsätze zugegriffen werden.

Verkettung

Eine klassische Technik zur Realisierung dateiübergreifender Zugriffspfade ist die Verkettung. Hierbei werden die zu verbindenden Sätze in eine lineare Liste oder in eine nichtlineare dynamische Datenstruktur, wie z.B. Ringstruktur, Doppelkette usw., einbe-

zogen. Jede dieser Verkettungsformen erfordert die Erweiterung der zu verbindenden Sätze um Zeiger. Die Realisierung der in Beispiel 5.1 ausgewiesenen Verbindung zweier Dateien mittels linearer Listen veranschaulicht die Abb. 5.5.

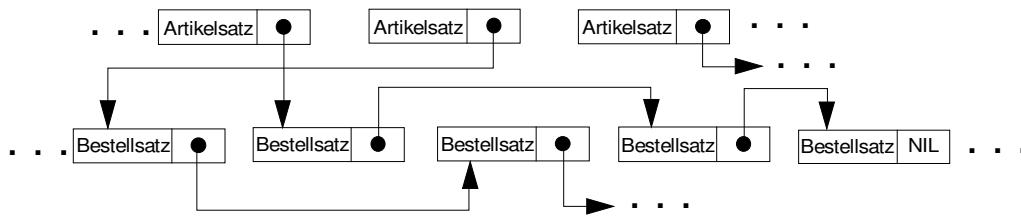


Abb. 5.5. Verbindung zweier Dateien mittels linearer Listen.

Beispielsweise werden mit dem ersten der in Abb. 5.5 dargestellten Artikelsätze genau drei Bestellsätze linear verkettet. Der Zeiger des letzten Bestellsatzes besitzt den Wert NIL und markiert so das Ende der Liste der Bestellsätze. Grundsätzlich kann ein Bestellsatz mehreren Listen angehören. Nämlich dann, wenn die Sätze der Bestelldatei mit den Sätzen weiterer Dateien verkettet werden. Insgesamt können auf diese Weise recht komplexe Strukturen entstehen, deren Pflege einen erheblichen Aufwand verursacht.

Bei der Verbindung von zwei Dateien mit Hilfe von Zeigersätzen werden die Datensätze eines Satztyps je um einen Zeigersatz erweitert. Die zu einem Zeigersatz zusammengefassten Zeiger verweisen auf Sätze eines zweiten Satztyps und stellen so eine dateiübergreifende Verbindung her. Zur Verdeutlichung diene das in Abb. 5.6 gezeigte Beispiel.

Zeigersätze

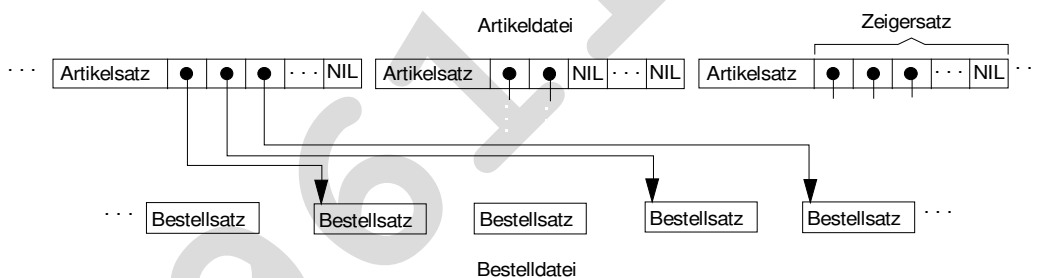


Abb. 5.6. Verbindung zweier Dateien mit Hilfe von Zeigersätzen.

Aus Gründen der Übersichtlichkeit wurden in Abb. 5.6 nur die Zeiger eingezeichnet, welche von dem ersten der dargestellten Artikelsätze ausgehen. Die Zeiger verweisen auf drei Bestellsätze, welche mit dem Artikelsatz in Beziehung stehen. Auch hier können aufgrund von weiteren Beziehungen komplexe Verbindungsstrukturen entstehen, deren Pflege recht aufwendig ist.

Kettsätze, die vierte der genannten Verbindungstechniken, gestatten die Realisierung netzwerkartiger Beziehungen zwischen Dateien. Mit einem Kettsatz lassen sich die gegenseitigen Beziehungen zwischen zwei Sätzen unterschiedlichen Satztyps darstellen. Zusammengehörige Kettsätze - also Kettsätze, welche die gegenseitigen Beziehungen zwischen sämtlichen Sätzen zweier Dateien beschreiben - fasst man zu einer Datei zusammen. Demonstriert werde diese Verbindungstechnik an einem einfachen Beispiel.

Kettsätze

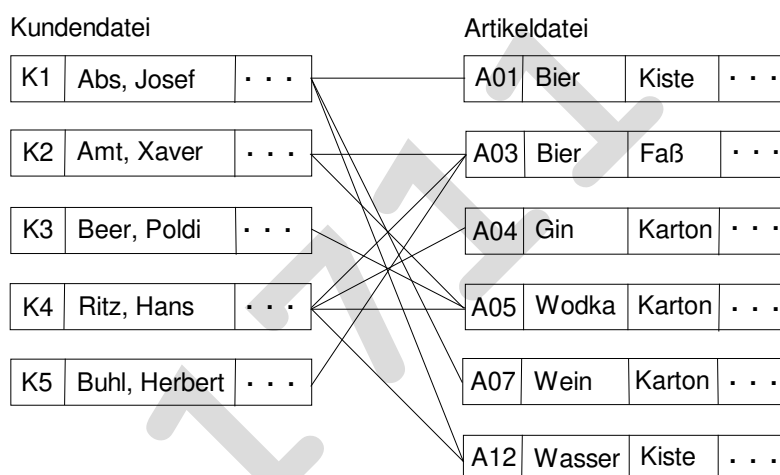
Beispiel für Kettsätze

Beispiel 5.2

Gegeben sei folgende, bereits bei der Behandlung des Netzwerkmodells in Kap. 3.3 betrachtete mc-mc-Beziehung (vgl. Abb. 3.22):



Auf der physischen Ebene entsprechen dieser Beziehung beispielsweise folgende, durch Verbindungslinien repräsentierte Einzelbeziehungen zwischen Kunden- und Artikelsätzen:



Jede der Einzelbeziehungen kann mit einem Kettsatz dargestellt werden, der folgenden Aufbau besitzt:

Aufbau eines Kettsatzes

KettsatzNr	ZArtikelkette	ZArtikel	ZKundenkette	ZKunde
------------	---------------	----------	--------------	--------

Hierbei bezeichnen:

- KettsatzNr - die Nummer des Kettsatzes,
- ZArtikelkette - einen Zeiger, der auf den nächsten Kettsatz in einer Artikelkette verweist,
- ZArtikel - einen Zeiger, der auf den Artikelsatz in der Artikeldatei verweist, den die durch den Kettsatz beschriebene Einzelbeziehung einschließt,
- ZKundenkette - einen Zeiger, der auf den nächsten Kettsatz in einer Kundenkette verweist,
- ZKunde - einen Zeiger, der auf den Kundensatz in der Kundendatei verweist, den die durch den Kettsatz beschriebene Einzelbeziehung einschließt.

In einer Artikelkette werden sämtliche Kettsätze verkettet, welche auf die Artikel verweisen, die ein bestimmter Kunde bestellt hat. Entsprechend werden in einer Kundenkette sämtliche Kettsätze verkettet, welche auf die Kunden verweisen, die einen bestimmten Artikel bestellt haben. Ein Kettsatz ist folglich gleichzeitig Mitglied in einer Kunden- und in einer Artikelkette.

Den Einstieg in eine Artikelkette ausgehend von einem Kunden gestattet ein Zeiger *FirstArtikel*, um den der Kundensatz zu erweitern ist. Entsprechend ist jeder Artikelsatz um einen Zeiger *FirstKunde* zu erweitern, der auf den ersten Kunden in einer Kundenkette verweist.

Damit ergibt sich folgende tabellarische Darstellung der Kunden-, der Artikel- und der Kettsatzdatei:

Kundendatei

KundenNr	Name	First-Artikel
K1	Abs, Josef	111
K2	Amt, Xaver	114
K3	Beer, Poldi	116
K4	Bitz, Hans	117
K5	Buhl, Herbert	121

Artikeldatei

ArtikelNr	Bezeichnung	Verpackung	First-Kunde
A01	Bier	Kiste	111
A03	Bier	Faß	114
A04	Gin	Karton	118
A05	Wodka	Karton	115
A07	Wein	Karton	112
A12	Wasser	Kiste	113

Kettsatzdatei

KettsatzNr	ZArtikelkette	ZArtikel	ZKundenkette	ZKunde
<u>111</u>	112	A01	NIL	K1
112	113	A07	NIL	K1
113	NIL	A12	120	K1
<u>114</u>	115	A03	117	K2
115	NIL	A05	116	K2
<u>116</u>	NIL	A05	118	K3
<u>117</u>	118	A03	121	K4
118	119	A04	NIL	K4
119	120	A05	NIL	K4
120	NIL	A12	NIL	K4
<u>121</u>	NIL	A03	NIL	K5

zwei Dateien verbindende
Kettsatzdatei

Exemplarisch werde die Artikelkette betrachtet, die von dem Kunden mit der Nummer K4 ausgeht. Der Zeiger *FirstArtikel* des entsprechenden Kundensatzes verweist auf den Kettsatz mit der Nummer 117. Der Zeiger *ZArtikel* dieses Kettsatzes verweist auf den Artikel A03 als ersten geordneten Artikel und der Zeiger *ZArtikelkette* auf den Kettsatz 118. Aus diesem Kettsatz geht A04 als nächster bestellter Artikel und der Kettsatz 119 als nächster Kettsatz hervor. Der Kettsatz 119 enthält Verweise auf den Artikel A05 und den nächsten Kettsatz 120. Der Kettsatz 120 schließlich enthält einen Verweis auf den Artikel A12 und das Endezeichen NIL für die aktuelle Artikelkette. Bei dem Durchlaufen der Artikelkette konnte also ermittelt werden, dass der Kunde K4 die Artikel A03, A04, A05 und A12 geordert hat.

Auch bei der Verwendung von Kettsätzen zur Darstellung von Dateiverbindungen ist ein erheblicher Pflegeaufwand in Kauf zu nehmen. Bei der Änderung von Beziehungen sind hier sogar Zugriffe zu drei Dateien erforderlich.

Übungsaufgabe 5.4

Betrachtet werde das Beispiel 5.2. Die in diesem Beispiel durch Kettsätze realisierten Beziehungen zwischen Kunden- und Artikelsätzen wurden bereits in Kap. 3.3 behandelt. So zeigt die Abb. 3.27 beispielhaft, wie die Einzelbeziehungen zwischen einem Kunden und den von ihm bestellten Artikeln sowie zwischen einem Artikel und den diesen Artikel ordernden Kunden mit Hilfe von Zeigern auf direkte Weise dargestellt werden können. Welcher Zusammenhang besteht zwischen der in Abb. 3.27 dargestellten Realisierungsform von Beziehungen zwischen Dateien und der in Beispiel 5.2 demonstrierten Kettsatztechnik?

6 Datenintegrität

Bei der Konzipierung, der Realisierung und dem Betrieb eines Datenbanksystems können sich Fehler einschleichen oder Ereignisse eintreten, welche negative Auswirkungen auf die Datenbank selbst, den Betreiber der Datenbank, die Datenbankbenutzer oder sonstige Personen zur Folge haben. So kann z.B. ein Fehler in einer Updateroutine zu inkorrekten Daten oder ein Brand zum Verlust von Daten führen. Während in diesen Fällen die Datenbank sowie die Interessen von Datenbankbetreiber und Benutzern Schaden nehmen, tangiert z.B. die Weitergabe geschützter personenbezogener Daten die Interessen der betroffenen Personen. Sämtliche der genannten Beispiele berühren den Bereich der Datenintegrität und damit einen Wissensbereich, der ebenso vielschichtig ist, wie die sich hinter den Beispielen verbergenden grundsätzlichen Probleme. Diese grundsätzlichen Probleme liegen auf unterschiedlichen Ebenen der Datenintegrität, nämlich den Ebenen

- Datenkonsistenz,
- Datensicherheit und
- Datenschutz.

Ebenen der
Datenintegrität

Nachfolgend wird der Inhalt dieser Begriffe erläutert.

Unter "Datenkonsistenz" (engl. data consistence) versteht man die logische Richtigkeit und Widerspruchsfreiheit der in einer Datenbank abgelegten Daten. Als Maßstab zur Beurteilung der Datenkonsistenz dient die logische Datenbeschreibung. Eine Datenbank heißt daher konsistent, falls sämtliche Daten den Regeln des konzeptionellen Schemas und insbesondere den sogenannten Konsistenzregeln genügen.

Datenkonsistenz

Bei der "Datensicherheit" (engl. data security) geht es um die physische Sicherung der Daten beim Betrieb der Datenbank. Unter den Begriff der Datensicherheit fallen daher sämtliche Überlegungen und Maßnahmen, die geeignet sind, die in einer Datenbank abgelegten Daten vor Verlust, Beschädigung, Verfälschung und unerlaubtem Zugriff zu schützen. Hierbei spielt es keine Rolle, ob die Gefahr für die Daten auf höhere Gewalt (z.B. Blitzschlag, Kurzschluss), unbeabsichtigtes Handeln (z.B. Bedienungsfehler, Programmierfehler) oder beabsichtigtes Handeln (z.B. Brandstiftung, Diebstahl) zurückgeht.

Datensicherheit

Der Begriff "Datenschutz" (engl. data protection) zielt auf den Schutz von Personen vor der missbräuchlichen Verwendung personenbezogener Daten ab. Um die Schutzinteressen betroffener Personen zu gewährleisten, wurden Datenschutzgesetze auf Bundes- und Länderebene erlassen. Die Gesetze regeln u.a. die Zulässigkeit der Verarbeitung personenbezogener Daten, die Rechte der Betroffenen und die Pflichten der datenverarbeitenden Stellen. Sie sehen darüber hinaus vor, dass bestimmte Kontrollinstanzen die Einhaltung der Datenschutzgesetze überprüfen.

Datenschutz

Datenkonsistenz-, Datensicherheits- und Datenschutzprobleme treten natürlich nicht nur bei dem Betrieb von Datenbanken auf, sondern bei sämtlichen Formen der (automatisierten) Datenverarbeitung. Die spezifischen Ansatzpunkte der drei Bereiche der Datenintegrität bei der datenbankgestützten Datenverarbeitung veranschaulicht die Abb. 6.1.

Die Lösung von Datenschutz-, Datensicherheits- und Datenschutzproblemen wird durch Interessenkonflikte der involvierten Personen und durch Trade-offs erschwert. Einige Beispiele mögen dies verdeutlichen:

- Dem geschäftsmäßigen Interesse eines Datenbankbetreibers bei der Verarbeitung personenbezogener Daten, z.B. im Rahmen eines Personalinformationssystems, stehen die Datenschutzinteressen der Betroffenen gegenüber.
- Spezielle Datenschutzmaßnahmen, wie z.B. die Aufzeichnung von Zugriffen zu geschützten Daten, vergrößern den zu schützenden Datenbestand.
- Benutzerseitiger Komfort, z.B. in Form interaktiver Mehrbenutzersysteme, erhöht den für die Datensicherung zu betreibenden Aufwand, da z.B. die Datenrekonstruktion in Fehlersituationen erschwert wird.
- Archive von Datenkopien, die aus Gründen der Datensicherung angelegt werden, schaffen zusätzliche Angriffspunkte für die missbräuchliche Datenverwendung und vergrößern so die Datenschutzprobleme.

erschwerende
Interessenskonflikte und
Trade-offs

Nachdem nun die Bereiche der Datenintegrität abgegrenzt und hinsichtlich ihres Problemcharakters und ihrer Interdependenzen in groben Zügen erläutert wurden, gehen die drei folgenden Kapitel detaillierter auf bereichsspezifische Probleme und Problemlösungsansätze ein. Kap. 6.1 behandelt den Bereich Datenkonsistenz, Kap. 6.2 den Bereich Datensicherheit und Kap. 6.3 den Bereich Datenschutz.

Inhalt des Kapitels

6.1 Datenkonsistenz

Im Zentrum der Überlegungen zur Konsistenz von Datenbanken stehen Konsistenzbedingungen, auch Konsistenzregeln oder Integritätsregeln (engl. integrity rules) genannt. Darunter sind Angaben in der Beschreibung von Daten zu verstehen, die festlegen, welche Datenwerte in welchen Zeitabschnitten unter welchen Bedingungen zulässig sein sollen. Falls die in einer Datenbank gespeicherten Daten sämtlichen formulierten Konsistenzbedingungen genügen, spricht man von einer konsistenten Datenbank.

Konsistenzregeln

Eine konsistente Datenbank wird dann inkonsistent, wenn auf ihr Operationen ausgeführt werden, welche zur Verletzung von Konsistenzbedingungen führen. Von besonderem Interesse sind daher die konsistenzerhaltenden Datenbankoperationen. Sie werden als Transaktionen bezeichnet. Festgehalten sei folgende vorläufige Begriffsbestimmung:

Eine Transaktion ist eine konsistenzerhaltende Datenbankoperation; eine konsistente Datenbank befindet sich nach der Ausführung einer Transaktion in einem konsistenten Zustand.

vorläufiger Transaktions-
begriff

Transaktionen stellen, wie die Abb. 6.2 zeigt, eine Teilmenge der Menge der Datenbankoperationen dar. In Abb. 6.2 werden die Transaktionen weiter unterteilt in Abfragen/Datenbankzugriffe und konsistenzerhaltende Mutationen. Versteht man unter Abfragen und Datenbankzugriffen solche Operationen, welche lediglich Daten lesen und extrahieren aber nicht schreiben bzw. verändern, so sind Abfragen/Datenbankzugriffe per

Abfragen sind
konsistenzerhaltend

se konsistenzhaltend. Und zwar ohne dass man diese Operationen an irgendwelche konsistenzhaltenden Bedingungen knüpft.

Einteilung der
Datenbankoperationen

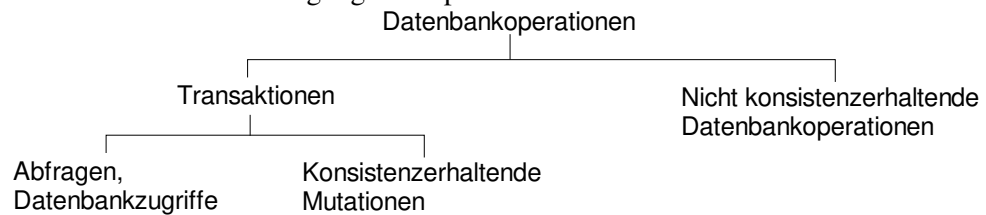


Abb. 6.2. Datenbankoperationen.

Andere Verhältnisse liegen bei Mutationen vor. Einfüge-, Änderungs- und Löschoperationen sowie komplexere, aus diesen Operationen zusammengesetzte Datenbankoperationen können zur Verletzung von Konsistenzbedingungen und damit zu einer inkonsistenten Datenbank führen. Zur Menge der Transaktionen gehören daher nur die Mutationen, die keine Konsistenzbedingungen verletzen. Wie nun Konsistenzbedingungen beschaffen sind und wie sie zu formulieren sind, wird im vorliegenden Kapitel behandelt. Im Einzelnen werden dargestellt:

Inhalt des Kapitels

- die Arten von Konsistenzbedingungen,
- der Aufbau von Konsistenzbedingungen und
- die Formulierung von Konsistenzbedingungen.

Der Zusammenhang zwischen Konsistenzbedingungen und Transaktionen wird bei der Betrachtung der Arten von Konsistenzbedingungen aufgegriffen.

a) Arten von Konsistenzbedingungen

Folgende Arten von Konsistenzbedingungen lassen sich unterscheiden:

Arten von Konsistenz-
bedingungen

- Starke und schwache Konsistenzbedingungen,
- primäre und sekundäre Konsistenzbedingungen,
- Zustands- und Übergangsbedingungen.

Außerdem kann der Objektbereich von Konsistenzbedingungen, d.h. die Menge der betroffenen Datenobjekte, variieren.

Starke und schwache Konsistenzbedingungen

Als Unterscheidungskriterium dient hier die Notwendigkeit des Einhaltens von Konsistenzbedingungen.

starke und schwache
Konsistenzbedingungen

Starke Konsistenzbedingungen (engl. strong assertions) müssen von allen Transaktionen uneingeschränkt eingehalten werden; *schwache* Konsistenzbedingungen (engl. soft assertions) sind in der Regel einzuhalten, jedoch nicht in unumgänglichen Ausnahmefällen.

Ausnahmefälle liegen dann vor, wenn bestimmte Daten aus unabänderlichen Gründen nicht zur Verfügung stehen oder fehlerhafte Werte aufweisen und dadurch Inkonsisten-

zen verursachen. Verdeutlicht sei dies an zwei Beispielen:

- Im Lagerwesen ist das Phänomen des Schwunds niemals vollständig eliminierbar. Abweichungen zwischen Buchbeständen und Zählbeständen müssen daher zugelassen bzw. Verletzungen der Konsistenzbedingung *Bestandsgleichheit* hingenommen werden.
- Bei manchen Kindern sind nicht beide Elternteile namentlich bekannt. In einem Familienregister können daher Verletzungen der Konsistenzbedingung *Eltern* unumgänglich sein.

Beispiele für schwache
Konsistenzbedingungen

Stets sollte die Verletzung einer schwachen Konsistenzbedingung eine Reaktion zur Folge haben. Das Problem der Reaktion wird bei der Behandlung des Aufbaus von Konsistenzbedingungen angesprochen.

Übungsaufgabe 6.1

Nennen Sie ein weiteres Beispiel für eine schwache Konsistenzbedingung. Wählen Sie ein Beispiel aus dem betrieblichen Bereich.

Primäre und sekundäre Konsistenzbedingungen

In bestimmten Fällen kann eine einzelne Mutation, z.B. eine Einfügeoperation, unweigerlich einen inkonsistenten Datenbankzustand auslösen. Durch nachgeschaltete weitere Operationen lässt sich eine solche Inkonsistenz wieder beheben. Die ursprünglich ausgelöste Inkonsistenz besteht somit nur temporär. Die (unvermeidliche) Inkaufnahme temporärer Verletzungen von Konsistenzbedingungen dient als Kriterium zur Unterscheidung zwischen primären und sekundären Konsistenzbedingungen. Außerdem ergeben sich Konsequenzen für den Transaktionsbegriff.

Eine Konsistenzbedingung heißt *primär*, wenn sie sich auf genau eine Relation bezieht und wenn sie durch die Ausführung einer elementaren Datenbankoperation, d.h. einer einzelnen Einfüge-, Änderungs- oder Löschoperation, auf dieser Relation nicht verletzt wird.

primäre
Konsistenzbedingung

Unter elementaren Datenbankoperationen sind einfache Mutationen wie Einfügen eines Tupels in eine Relation, Ändern des Werts eines Attributs oder der Werte mehrerer Attribute eines Tupels usw. zu verstehen. Aus dem Bezug auf eine Relation oder gar nur auf ein Tupel einer Relation folgt, dass primäre Konsistenzbedingungen grundlegende Eigenschaften einer Relation ansprechen. Beispielsweise die Identifizierung der Tupel einer Relation oder den Wertebereich eines Attributs.

Aus der letzten Bemerkung ergeben sich unmittelbar folgende Beispiele für primäre Konsistenzbedingungen:

Beispiele für primäre Konsistenzbedingungen

- Der Primärschlüssel einer Relation muss eindeutig sein. In einer Relation *Kunde* ist z.B. eine fortlaufend vergebene Kundennummer ein eindeutiger Primärschlüssel.
- Der Wertebereich eines Attributs, welches in einer Relation nicht Fremdschlüssel ist, muss statisch sein. Für das Schlüsselattribut *KundenNr* einer Relation *Kunde* könnte der statische Wertebereich z.B. auf [100000..999999] festgelegt sein.

Mit nur temporären Inkonsistenzen einer Datenbank steht der Begriff der sekundären Konsistenzbedingung in Verbindung.

sekundäre Konsistenzbedingung

Eine Konsistenzbedingung heißt *sekundär*, wenn mindestens eine elementare Datenbankoperation existiert, welche zu einer Verletzung der Konsistenzbedingung führt, und wenn jede Inkonsistenz, die von einer elementaren Datenbankoperation ausgeht, durch eine oder mehrere nachgeschaltete elementare Datenbankoperationen zu beheben ist.

Beispiele für sekundäre Konsistenzbedingungen

Die befristete Inkaufnahme inkonsistenter Datenbankzustände ist oft unvermeidbar. An zwei Beispielen für sekundäre Konsistenzbedingungen sei dies verdeutlicht:

Bilanz oder Gleichgewichtsbedingung

- Eine typische sekundäre Konsistenzbedingung ist eine Bilanz oder eine Gleichgewichtsbedingung. So muss z.B. bei dem Austausch von Hausteilen zwischen zwei Automobilwerken die Bilanz der Hausteilzu- und -abgänge ausgeglichen sein. Da ein Ausgleich stets zwei Buchungen erfordert, ist eine temporäre Inkonsistenz unvermeidlich. Die Abb. 6.3 veranschaulicht dies.

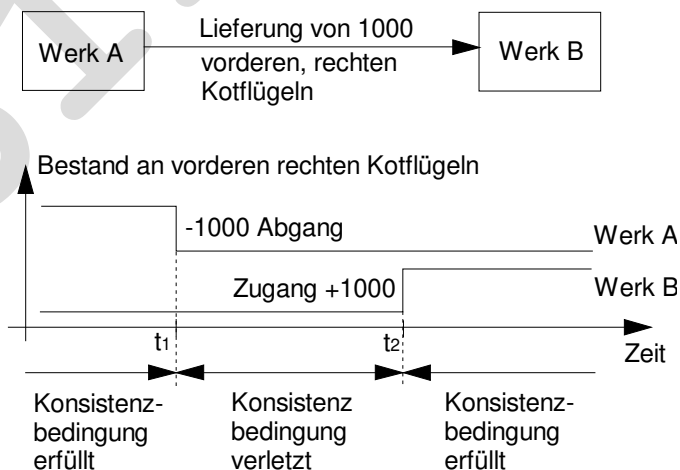


Abb. 6.3. Temporäre Verletzung einer sekundären Konsistenzbedingung.

In Abb. 6.3 bezeichnet t_1 den Zeitpunkt der Verbuchung des Lagerabgangs im Werk A und t_2 den Zeitpunkt der Gegenbuchung. Im Zeitraum zwischen t_1 und t_2 ist die Konsistenzbedingung verletzt.

referentielle Integrität

- Eine weitere typische sekundäre Konsistenzbedingung ist die Bedingung der referentiellen Integrität, die in Kap. 3.4.2 vorgestellt wurde. Die in Abb. 6.4 gezeigte 1-mc-Beziehung zwischen zwei Relationen *Artikel* und *Bestellung* z.B. impliziert eine Bedingung der referentiellen Integrität.

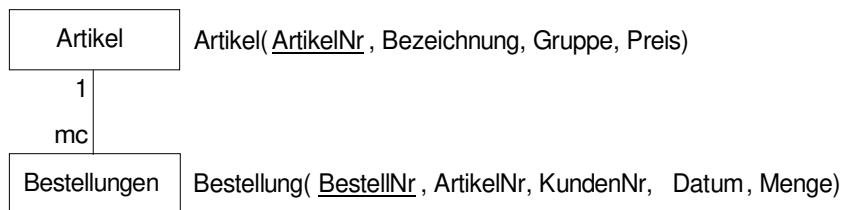


Abb. 6.4. Beispiel für eine 1-mc-Beziehung.

Bezogen auf die Abb. 6.4 besagt die Bedingung der referentiellen Integrität, dass der Fremdschlüssel *ArtikelNr* in der Relation *Bestellung* nur tatsächlich vorhandene Tupel der Relation *Artikel* referenzieren darf. Wird nun beispielsweise ein Tupel aus der Relation *Artikel* entfernt, weil der betreffende Artikel ausgelaufen ist, so müssen im Nachgang sämtliche diesen Artikel ordernden Tupel aus der Relation *Bestellung* entfernt werden. Falls derartige Tupel in der Relation *Bestellung* existieren, tritt eine temporäre Inkonsistenz auf.

Aus den Ausführungen zu den sekundären Konsistenzbedingungen ergibt sich die Notwendigkeit, den Transaktionsbegriff wie folgt zu erweitern und zu präzisieren:

Eine Transaktion ist eine elementare Datenbankoperation oder eine Folge von elementaren Datenbankoperationen, welche nach ihrer Anwendung auf eine konsistente Datenbank die Datenbank in einem konsistenten Zustand hinterlässt. Charakteristisch für eine Transaktion ist ihre Unteilbarkeit; sie darf entweder nur vollständig oder überhaupt nicht ausgeführt werden.

Begriff der Transaktion

Transaktionen, die aus mehreren elementaren Datenbankoperationen bestehen, dürfen unter keinen Umständen nur teilweise ausgeführt werden. Kann eine solche Transaktion aus irgendwelchen Gründen nicht korrekt beendet werden, so muss sie zurückgesetzt und damit der ursprüngliche Zustand der Datenbank wieder hergestellt werden. Auf die Problematik des Zurücksetzens geht das Kap. 6.2 näher ein.

Zustands- und Übergangsbedingungen

Die Einteilung von Konsistenzbedingungen in Zustands- und Übergangsbedingungen basiert auf dem Kriterium der Kopplung an Datenbankoperationen.

Zustandsbedingungen sind nicht an Datenbankoperationen gebunden. Bei Übergangsbedingungen liegt dagegen eine Bindung an bestimmte Datenbankoperationen vor.

Zustands- und Übergangsbedingungen

Beispiele für Zustandsbedingungen sind die oben angegebenen primären Konsistenzbedingungen, welche

- die Eindeutigkeit des Primärschlüssels einer Relation und
- statische Wertebereiche für die Attribute einer Relation, die nicht als Fremdschlüssel fungieren,

Beispiele für Zustandsbedingungen

fordern. Konsistenzbedingungen dieser Art gelten unabhängig davon, ob irgendwelche Datenbankoperationen ausgeführt werden oder nicht.

Die Bindung von Übergangsbedingungen an bestimmte Datenbankoperationen sei an zwei Beispielen erläutert:

Beispiele für
Übergangsbedingungen

- Firmenmitarbeiter, die einen Anspruch auf eine Firmenrente haben, dürfen nach dem Erreichen der Altersgrenze im Personalstamm nicht gelöscht werden.
- Der Familienstand eines Mitarbeiters einer Firma darf nicht von "verheiratet" auf "ledig" geändert werden.

Wie diese Beispiele zeigen, schließen Übergangsbedingungen gegebenenfalls bestimmte Übergänge zwischen konsistenten Datenbankzuständen aus. Festzuhalten ist daher:

verbotene Übergänge
zwischen konsistenten
Datenbankzuständen

Nicht jeder durch eine Datenbankoperation vollziehbare Übergang von einem konsistenten Datenbankzustand zu einem anderen konsistenten Datenbankzustand ist zulässig.

Übungsaufgabe 6.2

Betrachtet werde der Zusammenhang zwischen den unterschiedenen Arten von Konsistenzbedingungen. Kann eine Übergangsbedingung zugleich auch

- eine primäre,
- eine sekundäre,
- eine starke,
- eine schwache

Konsistenzbedingung darstellen? Begründen Sie Ihre Antwort gegebenenfalls mittels Beispielen aus dem betrieblichen Bereich.

Objektbereich von Konsistenzbedingungen

Umfang der betroffenen
Daten

Der Begriff "Objektbereich" bezeichnet den Umfang der Daten, auf den sich eine Konsistenzbedingung erstreckt. Der Objektbereich von Konsistenzbedingungen kann stark variieren. So existieren Konsistenzbedingungen, die lediglich ein Attribut einer Relation betreffen, und übergreifende Konsistenzbedingungen, die Daten aus mehreren Relationen einschließen. Beispiele für Konsistenzbedingungen unterschiedlichen Objektbereichs zeigt die Abb. 6.5.

Objektbereich	Konsistenzbedingung	Beispiel								
Einzelnes Attribut	Statischer Wertebereich eines Attributs.	Wird der Wertebereich [0000..9999] des Attributs <i>Umsatz</i> eingehalten?								
Mehrere Attribute eines Tupels	Zusammenhang zwischen den Werten mehrerer Attribute eines Tupels.	<table><tr><th>Umsatz</th><th>Kundengruppe</th></tr><tr><td>0000 - 0999</td><td>A</td></tr><tr><td>1000 - 4999</td><td>B</td></tr><tr><td>5000 - 9999</td><td>C</td></tr></table> zwischen <i>Umsatz</i> und <i>Kundengruppe</i> eingehalten?	Umsatz	Kundengruppe	0000 - 0999	A	1000 - 4999	B	5000 - 9999	C
Umsatz	Kundengruppe									
0000 - 0999	A									
1000 - 4999	B									
5000 - 9999	C									
Einzelner Tupel	Eindeutige Identifikation eines Tupels durch einen Identifikationsschlüsselwert.	Existiert in der Relation <i>Kunde</i> bereits ein Tupel mit dem Schlüsselwert <i>KundenNr</i> = 4712?								
Mehrere Tupel einer Relation	Summenrestriktion über die Werte eines Attributs in mehreren Tupeln einer Relation.	Überschreitet die Summe der an einem Tag vorgenommenen Abhebungen von einem Konto eine gegebene Obergrenze?								
Mehrere Tupel aus mehreren Relationen	Bedingung der referentiellen Integrität.	Referenziert der Fremdschlüssel <i>KundenNr</i> in der Relation <i>Bestellung</i> nur tatsächlich vorhandene Tupel der Relation <i>Kunde</i> ?								
Einzelne Relation	Summenrestriktion über die Werte eines Attributs in allen Tupeln einer Relation.	Überschreitet die Summe der Monatsgehälter ein gegebenes Gehaltsbudget?								
Mehrere Relationen	Bestandsrestriktion über die Tupel mehrerer Relationen.	Ist die Anzahl der in der Relation <i>Buchbestand</i> als entliehen vermerkten Bücher gleich der Anzahl der in der Relation <i>Leser</i> als ausgehändigt vermerkten Bücher?								

Objektbereich von
Konsistenzbedingungen

Abb. 6.5. Objektbereich von Konsistenzbedingungen.

In der Spalte "Konsistenzbedingung" wird jeweils ein allgemein gehaltenes Beispiel für eine Konsistenzbedingung mit dem fraglichen Objektbereich angegeben. Je ein zugehöriges konkretes Beispiel findet sich in der Spalte "Beispiel".

b) Aufbau von Konsistenzbedingungen

Zwischen Zustands- und Übergangsbedingungen bestehen lediglich geringe strukturelle Unterschiede. Zunächst sei der Aufbau von Zustandsbedingungen betrachtet. Eine Zustandsbedingung umfasst folgende Bestandteile (vgl. SCHLAGETER und STUCKY 1977):

- (1) Eine Menge von "Datenobjekten", auf die sich die Konsistenzbedingung bezieht.
- (2) Eine Bedingung, auch "Prädikat" genannt, die für die Objektmenge einzuhalten ist.
- (3) Eine "Auslöseregeln", die festlegt, wann die Einhaltung der Bedingung zu prüfen ist.

Bestandteile einer
Zustandsbedingung

- (4) Eine "Reaktionsregel", die vorgibt, welche Fehlerreaktion im Falle der Verletzung der Bedingung einzuleiten ist.

Objektbereich	Die einer Konsistenzbedingung zugrunde liegende Menge von Datenobjekten kann stark variieren. Im Minimalfall besteht der Objektbereich aus nur einem Attribut und im Maximalfall aus mehreren Relationen. Über die Variationsmöglichkeiten des Objektbereichs zwischen diesen beiden Grenzfällen gibt die Abb. 6.5 Auskunft.
Prädikat	Das einzuhaltende Prädikat beinhaltet die eigentliche Konsistenzbedingung. Bei dieser kann es sich folglich um eine starke oder schwache bzw. um eine primäre oder sekundäre Konsistenzbedingung handeln.
Auslöseregel	<p>Die Auslöseregel kann eine operationsabhängige oder eine zeitabhängige Form der Prüfung implizieren. Außerdem ist auch eine externe Auslösung durch die Datenbankbenutzer oder den Datenbankadministrator denkbar. Bei der operationsabhängigen Prüfung sind zwei Fälle zu unterscheiden:</p> <ul style="list-style-type: none"> - Das Prädikat wird nach jeder Elementaroperation geprüft; dies empfiehlt sich speziell bei primären Konsistenzbedingungen. - Das Prädikat wird nach einer Folge von zusammengehörigen Elementaroperationen, die beispielsweise eine Transaktion bilden, geprüft; diese Vorgehensweise ist bei sekundären Konsistenzbedingungen angezeigt. <p>Bei der zeitabhängigen Prüfung, auch periodische Prüfung genannt, erfolgt der Prüfungsvorgang jeweils nach einer vorgegebenen Zeitspanne. Beispielsweise nach einigen Stunden oder, im Falle einer geringen Bewegungsrate der Daten, nach einigen Tagen.</p>
Reaktionsregel	Da starke Konsistenzbedingungen von allen Transaktionen uneingeschränkt einzuhalten sind, kommt im Falle der Verletzung einer starken Konsistenzbedingung nur eine Reaktion in Frage: Abbruch bzw. Zurücksetzen der Transaktion und Ausgabe einer entsprechenden Meldung. Schwache Konsistenzbedingungen dürfen dagegen in bestimmten Ausnahmefällen verletzt werden. Für solche Situationen können z.B. folgende Fehlerreaktionen vorgesehen sein:
Fehlerreaktionen	<ul style="list-style-type: none"> - Ausgabe der inkonsistenten Daten sowie eines Hinweises auf die Verletzung einer Konsistenzbedingung. - Automatisches Verwenden von Ersatzwerten (engl. default values) für fehlende oder unbekannte Daten und Ausgabe eines entsprechenden Hinweises.

Betont sei, dass Fehlerreaktionen dieser Art nur dann ausgelöst werden, wenn eine in Ausführung befindliche Transaktion eine schwache Konsistenzbedingung nicht einhalten kann. Die Fehlerreaktion muss sich auf die beispielhaft angegebenen Maßnahmen beschränken. Keinesfalls darf eine Fehlerreaktion selbständig eine neue Transaktion starten, da dies eine Verletzung der Konsistenz der Datenbank bewirken könnte.

Grundsätzlich besitzen Konsistenzbedingungen für Zustandsübergänge den gleichen, vier Bestandteile umfassenden Aufbau wie Zustandsbedingungen. Einzelne Bestandteile weisen jedoch eine spezielle Ausprägung auf. Im Einzelnen gilt für eine Zustandsbedingung:

- (1) Die "Objektmenge" besteht aus den Datenobjekten, die durch die Datenbankoperationen manipuliert werden, welche einer Übergangsbedingung zugrunde liegen.

- (2) Das "Prädikat" stellt keine Bedingung im eigentlichen Sinne dar; vielmehr ist anzugeben, welche Änderungen auf der Objektmenge erlaubt bzw. nicht erlaubt sind.
- (3) Die "Auslöseregel" ist operationsabhängig zu konzipieren, da eine Konsistenzprüfung stets dann vorzunehmen ist, wenn ein Zustandsübergang stattfindet.
- (4) Die "Reaktionsregel" kann in analoger Weise wie bei einer Zustandsbedingung gestaltet werden.

Bestandteile einer
Übergangsbedingung

Beispiele für Zustands- und Übergangsbedingungen werden im Abschnitt c) - also in Verbindung mit der formalen Darstellung von Konsistenzbedingungen - angegeben.

Übungsaufgabe 6.3

Begründen Sie, warum operationsabhängige Auslöseregeln im Falle von sekundären Konsistenzbedingungen eine Prüfung jeweils nach einer Folge von zusammengehörigen Elementaroperationen vorsehen sollten.

c) Formulierung von Konsistenzbedingungen

Auf die Formulierung von Konsistenzbedingungen wirkt sich die Art des verwendeten DBVS unmittelbar aus. Abhängig davon, ob ein DBVS Konsistenzbedingungen unterstützt oder nicht, unterscheidet man zwischen modellinhärenten und modellexternen Konsistenzbedingungen (vgl. hierzu ZEHNDER 1985, S. 166 f.):

Modellinhärente Konsistenzbedingungen lassen sich mit der Sprachkomponente eines DBVS direkt formulieren; ihre Einhaltung wird durch das DBVS automatisch gewährleistet.

modellinhärente Konsistenzbedingungen

Modellexterne Konsistenzbedingungen lassen sich nicht mit der Sprachkomponente eines DBVS formulieren; sie müssen daher außerhalb des DBVS explizit programmiert werden.

modellexterne Konsistenzbedingungen

Aus der Unterscheidung in modellinhärente und in modellexterne Konsistenzbedingungen resultieren zwei Grundformen der Gewährleistung von Konsistenzbedingungen: Die zentrale Konsistenzsicherung durch den Datenbankadministrator mit den sprachlichen Mitteln des DBVS und die dezentrale Konsistenzsicherung durch die Datenbankbenutzer mit den sprachlichen Mitteln der Anwendungsprogrammierung. Modellinhärente Konsistenzbedingungen wird man in zentraler Form organisieren und modellexterne in dezentraler Form. Eine Gegenüberstellung der zentralen und der dezentralen Form der Konsistenzsicherung zeigt die Abb. 6.6.

zentrale und dezentrale
Konsistenzsicherung

Form der Konsistenzsicherung Vergleichskriterien	Zentrale Form der Konsistenzsicherung	Dezentrale Form der Konsistenzsicherung
Art der Konsistenzbedingungen	Modellinhärente Konsistenzbedingungen	Modellexterne Konsistenzbedingungen
Sprachliches Mittel	Sprachliches Mittel des DBVS, z. B. Datenbeschreibungssprache	Sprachliches Mittel der Anwendungsprogrammierung, z. B. COBOL
Organisatorische Zuständigkeit	Datenbankadministrator	Datenbankbenutzer

Abb. 6.6. Zentrale und dezentrale Form der Konsistenzsicherung.

Zweifellos ist der zentralen Form der Konsistenzsicherung der Vorzug zu geben, denn sie gewährleistet, dass die formulierten Konsistenzbedingungen für sämtliche Anwendungsprogramme in gleicher Weise gelten. Begrenzte Wirksamkeit besitzen dagegen dezentral in den einzelnen Anwendungsprogrammen formulierte Konsistenzbedingungen. Insbesondere dann, wenn mehrere Anwendungsprogramme auf gleiche Datenbankbereiche zugreifen und die in den Anwendungsprogrammen verwendeten Konsistenzbedingungen voneinander abweichen.

Ob man auf die Formulierung modellexterner Konsistenzbedingungen ganz verzichten kann, hängt von den sprachlichen Möglichkeiten ab, die das verwendete DBVS bietet. Nachfolgend wird lediglich auf die Formulierung von modellinhärenten Konsistenzbedingungen eingegangen. Zugrunde gelegt wird hierbei das System R.

ASSERT-Anweisung

Im System R dient die ASSERT-Anweisung zur Formulierung von Konsistenzbedingungen. Die ASSERT-Anweisung ist Bestandteil der Datendefinition. In einer ASSERT-Anweisung dürfen sämtliche SQL-Konstrukte verwendet werden. Eine Zustandsbedingung besitzt folgenden Aufbau:

Zustandsbedingung

ASSERT zustandsbedingungsname ON basistabellenname: zustandsbedingung
--

Auf das Schlüsselwort ASSERT folgen also der Name der Zustandsbedingung, das Schlüsselwort ON und der Name der Basistabelle, auf die sich die Zustandsbedingung bezieht. Die Zustandsbedingung selbst wird am Ende der Anweisung spezifiziert; ihr geht ein Doppelpunkt voran. Vor dem Doppelpunkt kann der Teil ON *basistabellenname* fehlen, falls mehrere Basistabellen angesprochen werden und die Basistabellen in der Zustandsbedingung selbst spezifiziert werden.

In einer Übergangsbedingung sind die Basistabellen zu spezifizieren, zwischen denen der Zustandsübergang stattfindet:

```
ASSERT übergangsbedingungsname ON basistabellenname1 basistabellenname2:
    übergangsbedingung
```

Übergangsbedingung

Nach dem Schlüsselwort ON sind die Namen der Basistabellen, die von dem Zustandsübergang betroffen sind, anzugeben. Es folgen ein Doppelpunkt und die Übergangsbedingung selbst.

Nachdem nun der grundsätzliche Aufbau von Konsistenzbedingungen im System R dargelegt wurde, seien einige konkrete Konsistenzbedingungen angegeben. Sie sind in Beispiel 6.1 zusammengestellt.

Beispiel 6.1

Sämtliche hier formulierten Konsistenzbedingungen beziehen sich auf die in Kap. 4.4.3, Beispiel 4.4 angegebene Datenbasis. Zuerst seien einige Zustandsbedingungen und danach einige Übergangsbedingungen betrachtet.

Beispiele für konkrete Konsistenzbedingungen

a) Zustandsbedingungen

Folgende Konsistenzbedingung gewährleistet die vorgegebenen Mindestbestände für die einzelnen Artikel:

```
ASSERT mindestbestand ON Artikel:
    Bestand >= MindBestand
```

Das Absinken des Bestandes unter den Mindestbestand wird für keinen der in der Relation *Artikel* vorhandenen Artikel akzeptiert.

Zustandsbedingungen

Dass nur die Artikel bewegt werden, die in der Relation *Artikel* auch tatsächlich vorhanden sind, sichert die folgende Konsistenzbedingung:

```
ASSERT artikelexistenz:
    (SELECT ArtikelNr FROM ArtikelBew) IS IN
    (SELECT ArtikelNr FROM Artikel)
```

Für jede der in den Tupeln der Relation *ArtikelBew* verzeichneten Artikelnummern wird geprüft, ob die Artikelnummer auch in einem Tupel der Relation *Artikel* vorhanden ist.

b) Übergangsbedingungen

Versteht man die in der Relation *Artikel* verzeichneten Preise der Fertigprodukte als Grundpreise, die bei der Annahme von Kundenaufträgen in jedem Fall zu überschreiten sind, so ist folgende Übergangsbedingung angezeigt:

Übergangsbedingungen

```

ASSERT angebotspreis ON AuftragPos Artikel:
    AuftragPos.Preis >= (SELECT Artikel.Preis FROM Artikel
                        WHERE Artikel.ArtikelNr = AuftragPos.ArtikelNr)

```

Nach der Überarbeitung einer Stückliste ist das Stücklistendatum zu aktualisieren. Dabei muss das neue Datum aktueller als das alte Datum sein. Im System R könnte eine solche Übergangsbedingung wie folgt formuliert werden:

```

ASSERT StklisteDatum ON Stkliste:
    NEW StklisteDatum >= OLD StklisteDatum

```

Übungsaufgabe 6.4

Gegeben sei eine Relation *Kunde* mit folgenden Tupeln:

Kunde					
<u>KundenNr</u>	Name	Adresse	Jahresumsatz	Gruppe	Rabatt
103097412	Abs, Josef	Ahornweg 12, 2800 Bremen	43968	A	2
103097419	Axt, Xaver	Tulpenweg 10, 5100 Aachen	114037	C	7
103097501	Beer, Poldi	Rosenweg 36, 5800 Hagen	73884	B	4
103097507	Bald, Hans	Nelkenweg 18, 1000 Berlin	61311	B	3
103097518	Buhl, Horst	Amselweg 12, 2800 Bremen	180207	C	10
⋮					

Abgängig vom Jahresumsatz werden die Kunden in drei Gruppen eingestuft. Die Gruppe A umfasst Kunden mit einem Jahresumsatz bis zu 50000 DM, die Gruppe B mit einem Jahresumsatz von mehr als 50000 DM und bis zu 100000 DM und die Gruppe C mit einem Jahresumsatz oberhalb 100000 DM. Nach dem Jahresumsatz richtet sich auch der den Kunden eingeräumte Rabatt, wobei hier allerdings die beschriebene Gruppeneinteilung ohne Einfluss ist. Keinesfalls wird ein Rabatt von mehr als 10 % gewährt.

Formulieren Sie zwei Konsistenzbedingungen, die sicherstellen, dass

- der Höchstbetrag von 10 % nicht überschritten und
- die Einordnung der Kunden in Gruppen korrekt vorgenommen wird.

6.2 Datensicherheit

Bekanntlich beinhaltet der Begriff der Datensicherheit die Sicherung der Daten in einer Datenbank gegen Verlust, Beschädigung, Verfälschung usw. Zur Gewährleistung der Datensicherheit im laufenden Datenbankbetrieb ergreift man unterschiedliche Sicherungsmaßnahmen. In Frage kommen u.a.

- bauliche Maßnahmen wie z.B. feuerhemmende Türen,
- technische Maßnahmen wie z.B. Notstromaggregate oder automatische Feuerlöschsysteme,
- organisatorische Maßnahmen wie z.B. Zugangskontrollen und
- programmtechnische Maßnahmen wie z.B. Verfahren zur Synchronisation paralleler Datenbankzugriffe.

Datensicherungs-
maßnahmen

Im vorliegenden Kapitel wird die Datensicherheit ausschließlich unter dem Blickwinkel programmtechnischer Maßnahmen betrachtet. Von herausragender Bedeutung in dieser Gruppe von Maßnahmen sind die Verfahren zur Synchronisation paralleler Datenbankzugriffe und die Verfahren zur Datenrekonstruktion. In den beiden folgenden Kapiteln wird auf diese beiden Problemkreise eingegangen.

6.2.1 Synchronisation paralleler Datenbankzugriffe

Operieren gleichzeitig mehrere Transaktionen auf einer Datenbank, so können Konsistenzprobleme auftreten. Ausschlaggebend sind die Art der Transaktionen und die angesprochenen Datenbereiche. Bezieht man auch die triviale Situation rein serieller Transaktionen mit ein, so lassen sich vier Fälle bei der Abwicklung von Transaktionen unterscheiden, die sich unterschiedlich auf die Datenkonsistenz auswirken:

(1) Serielle Transaktionen

Seriell ausgeführte Transaktionen greifen niemals gleichzeitig auf gleiche Datenbereiche zu. Konsistenzprobleme entstehen daher nicht. Diese Situation ergibt sich am ehesten noch bei sehr kurzen Transaktionen oder bei relativ selten auftretenden Transaktionen. Als Beispiel sei die automatische Verkehrszählung in Verbindung mit einer Verkehrsdatenbank genannt.

(2) Parallele Transaktionen auf unterschiedlichen Datenbankbereichen

Transaktionen, die gleichzeitig auf unterschiedlichen Datenbankbereichen operieren, beeinflussen sich gegenseitig nicht. Es treten daher keine Konsistenzprobleme auf. Diese Situation liegt beispielsweise bei der gleichzeitigen Erfassung von Brieftexten mit einem datenbankgestützten Mehrplatz-Erfassungssystem vor.

vier Fälle bei der
Abwicklung von
Transaktionen

(3) Parallele lesende Transaktionen

Angesprochen sind hier Transaktionen, die ausschließlich Leseoperationen ausführen. Da Leseoperationen eine konsistente Datenbank stets in einem unverändert konsistenten Zustand hinterlassen, können ausschließlich lesende Transaktionen beliebig parallel ausgeführt werden. Dieser Fall ist beispielsweise dann gegeben, wenn ein Datenbanksystem in bestimmten Zeitabschnitten als reines Auskunftssystem betrieben wird.

(4) Parallele Transaktionen auf gleichen Datenbankbereichen

Sofern mehrere Transaktionen gleichzeitig lesend und schreibend auf dem gleichen Datenbankbereich operieren, entstehen Konflikte beim Zugriff auf die Datenbank und damit auch Konsistenzprobleme. Um diese Probleme zu vermeiden, müssen die Transak-

tionen synchronisiert werden. Zur weiteren Verdeutlichung sei die grafische Veranschaulichung einer Konfliktsituation betrachtet (vgl. Abb. 6.7).

Konfliktsituation bei zwei parallelen Transaktionen

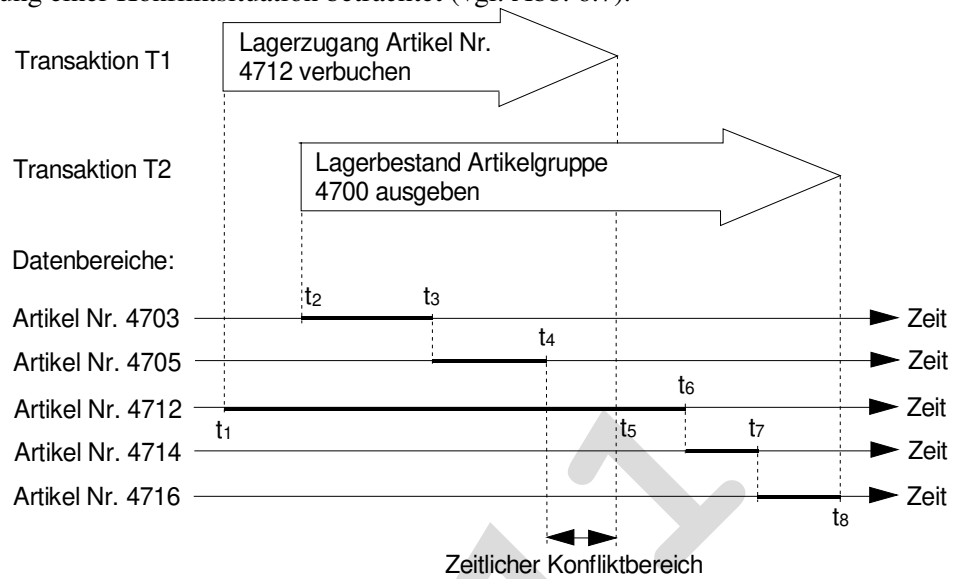


Abb. 6.7. Konflikt beim gleichzeitigen Zugriff paralleler Transaktionen auf den gleichen Datenbereich.

Erläuterung der Abbildung

Die Abb. 6.7 zeigt zwei Transaktionen und die Inanspruchnahme von fünf Datenbereichen im zeitlichen Ablauf. Die Transaktion $T1$ verbucht jeweils einen Lagerzugang, hier für den Artikel mit der Nummer 4712. Zu diesem Zweck greift $T1$ im Zeitraum von t_1 bis t_5 lesend und schreibend auf den Artikelsatz mit dem Schlüssel 4712 zu. Die Transaktion $T2$ gibt den Lagerbestand für jeweils eine Artikelgruppe aus, hier für die Gruppe 4700 mit fünf durch die Nummern 4703, 4705, 4712, 4714 und 4716 identifizierten Artikeln. $T2$ greift nacheinander lesend auf die fünf zu diesen Nummern gehörigen Artikelsätze zu. Im Zeitraum von t_2 bis t_3 auf den ersten Satz, danach bis zum Zeitpunkt t_4 auf den zweiten Satz usw. Ein Konflikt ergibt sich in der Zeitspanne von t_4 bis t_5 , da während dieser Spanne beide Transaktionen auf den Satz mit der Artikelnummer 4712 zugreifen möchten. Die von der Transaktion $T2$ gelesenen und ausgegebenen Daten hängen davon ab, ob die Transaktion $T1$ den Lagerzugang für den Artikel mit der Nummer 4712 bereits verbucht hat oder nicht. Solche auf Zufälligkeiten beruhenden, undefinierten Situationen sind mit einem geregelten Datenbankbetrieb nicht verträglich. Es bedarf daher eines Mechanismus, um gleichzeitige Zugriffsversuche auf gleiche Datenbereiche zu entflechten oder - wie es in der Fachsprache heißt - zu synchronisieren. Mechanismen, welche dies leisten, werden als Synchronisationsverfahren bezeichnet.

Synchronisationsverfahren

Man unterscheidet zwischen zwei Grundverfahren zur Synchronisation paralleler Datenbankzugriffe, dem Sperrverfahren und dem optimistischen Verfahren:

Sperrverfahren

Bei einem Sperrverfahren wird der Datenbereich, auf dem eine in Ausführung befindliche Transaktion operiert, für alle anderen Transaktionen gesperrt. Erst wenn die aktive Transaktion den gesperrten Datenbereich freigibt, können andere Transaktionen auf diesen Datenbereich zugreifen.

Bei einem optimistischen Verfahren werden Transaktionen beliebig gestartet und zunächst - quasi probeweise - auf einem Hilfsspeicher ausgeführt. Tritt kein Konflikt auf, so werden die auf den Hilfsspeicher geschriebenen Daten in die Datenbank übernommen. Andernfalls werden die an dem Konflikt beteiligten Transaktionen zurückgesetzt und erneut probeweise gestartet.

optimistisches Verfahren

Im Folgenden wird auf diese beiden grundlegenden Synchronisationsverfahren näher eingegangen. Zu berücksichtigen ist hierbei einerseits, dass eine konkrete Ausgestaltung des Sperrverfahrens - das sogenannte Zweiphasen-Sperrprotokoll - eine herausragende Bedeutung erlangt hat, und andererseits, dass trotz der Anwendung eines Sperrverfahrens die Konfliktsituation der Verklemmung und damit ein Stillstand eintreten kann. Im Folgenden werden daher vier Themenkreise behandelt:

- Sperrverfahren,
- Zweiphasen-Sperrprotokoll,
- Verklemmung,
- optimistisches Verfahren.

Inhalt des Kapitels

Übungsaufgabe 6.5

Geben Sie an, in welcher der beiden Situationen

- (1) Datenbankbetrieb mit hoher Transaktionsdichte und
- (2) Datenbankbetrieb mit geringer Transaktionsdichte

das Sperrverfahren bzw. das optimistische Verfahren geeigneter ist. Begründen Sie Ihre Antwort.

a) Sperrverfahren

Merkmale eines
Sperrverfahrens

Ein Sperrverfahren wird durch folgende Merkmale charakterisiert:

- die Sperrobjekte,
- die Sperrmodi und
- das Sperrprotokoll.

Als Sperrobjekte bezeichnet man die durch Sperren betroffenen Datenobjekte. Die Sperrobjekte können durchweg genau einer Objektebene angehören, beispielsweise der Ebene von Sätzen bzw. Tupeln, oder aber eine mehrere Objektebenen umfassende Sperrhierarchie bilden. Im letzten Fall ist folgende hierarchische Abstufung der Sperrobjekte denkbar:



Trade-off zwischen dem
Grad der Parallelität und
dem Synchronisations-
aufwand

Je feiner die Sperrobjekte gewählt werden, desto größer ist die durch die Synchronisation erzielbare Parallelität von Transaktionen. Anders formuliert: In desto größerem Umfang können Transaktionen gleichzeitig ausgeführt werden. Wird andererseits die gesamte Datenbank als Sperrobject gewählt, so ist nur eine serielle Ausführung von Transaktionen möglich. Da mit zunehmender Feinheit der Sperrobjekte der Synchronisationsaufwand steigt, liegt die Situation eines Trade-off zwischen dem Grad der Parallelität von Transaktionen und dem Synchronisationsaufwand vor. Um eine hinreichende Parallelität von Transaktionen zu ermöglichen, sollte das Setzen von Sperren zumindest auf der Ebene von Relationen, aber besser noch auf der Ebene von Tupeln, möglich sein.

Welche Wirksamkeit eine Sperre für die verschiedenen elementaren Datenbankoperationen besitzt, geht aus dem Sperrmodus hervor. Entsprechend der groben Einteilung elementarer Datenbankoperationen in die Gruppen lesende und schreibende Operationen lassen sich zwei Sperrmodi unterscheiden:

Sperrmodi

- Lesesperren und
- Schreibesperren.

Eine Transaktion, die eine Lese- oder eine Schreibsperre setzt, kann auf das Sperrobject uneingeschränkt lesend und schreibend zugreifen. Für andere Transaktionen sind die Zugriffsmöglichkeiten dagegen eingeschränkt und zwar wie nachfolgend beschrieben.

Begriff der Lesesperre

Eine Lesesperre, auch Teilsperre (engl. shared lock) genannt, gestattet anderen Transaktionen einen lesenden Zugriff auf das Sperrobject sowie das Setzen weiterer Lesesperren; für andere Transaktionen verboten sind dagegen ein schreibender Zugriff auf das Sperrobject sowie das Setzen von Schreibesperren auf das Sperrobject.

Eine Schreibsperre, auch Vollsperre (engl. exclusive lock) genannt, gestattet anderen Transaktionen weder einen lesenden Zugriff noch einen schreibenden Zugriff auf das Sperrobjekt; außerdem dürfen andere Transaktionen weder Lese- noch Schreibsperren auf das Sperrobjekt setzen.

Begriff der Schreibsperre

Aus den Begriffsbestimmungen der beiden Sperrmodi folgt, dass Lese- und Schreibsperren einem Sperrobjekt nicht beliebig auferlegt werden dürfen. Die Abb. 6.8 zeigt, in welchen Ausgangssituationen weitere Sperren gesetzt werden dürfen oder aber verboten sind.

Ausgangs- situation \ neue Sperre	Lesesperre durch andere Transaktion ?	Schreibsperre durch andere Transaktion ?
gesetzte Lesesperre	erlaubt	verboten
gesetzte Schreib- sperre	verboten	verboten

Abb. 6.8. Erlaubte und verbotene Folgen von Sperrmodi.

Wurde auf ein Sperrobjekt eine Lesesperre gesetzt, so dürfen laut Abb. 6.8 durch andere Transaktionen zwar weitere Lesesperren, aber keine Schreibsperren gesetzt werden. Eine noch restriktivere Situation liegt vor, falls eine Transaktion eine Schreibsperre gesetzt hat. Andere Transaktionen dürfen dem Sperrobjekt dann keinerlei weitere Sperren auferlegen.

Eine Transaktion wird dann eine Lesesperre anfordern, wenn lediglich ein lesender Zugriff auf ein bestimmtes Datenobjekt erfolgen soll. Die Lesesperre verhindert, dass das Sperrobjekt unterdessen durch eine andere Transaktion verändert wird. Andererseits ermöglicht die Lesesperre anderen Transaktionen das Setzen von Lesesperren und einen lesenden Zugriff auf das Datenobjekt.

Anforderung einer
Lesesperre

Beinhaltet eine Transaktion eine Schreiboperation, so ist das betreffende Datenobjekt mit einer Schreibsperre zu belegen. Dadurch wird erreicht, dass andere Transaktionen erst dann wieder lesend oder schreibend auf das Sperrobjekt zugreifen können, wenn die Schreiboperation abgeschlossen ist.

Anforderung einer
Schreibsperre

Übungsaufgabe 6.6

Aus welchen Gründen ist es sinnvoll, zu fordern, dass andere Transaktionen keinerlei weitere Sperren auf ein Sperrobjekt setzen dürfen, falls eine Transaktion eine Schreibsperre auf dieses Objekt gesetzt hat.

Sperrprotokoll

Zu erläutern ist schließlich noch der Begriff des Sperrprotokolls. Ein Sperrprotokoll besteht aus Regeln für das Setzen von Sperren und das Aufheben von Sperren durch Transaktionen. Als Beispiel sei das später noch eingehend behandelte Zweiphasen-Sperrprotokoll genannt.

Die Funktionsweise des Sperrverfahrens soll nun an einem Beispiel demonstriert werden. In diesem Beispiel werden u.a. die nachfolgend angegebenen Vereinbarungen und Anweisungen verwendet:

Notation zur Vereinbarung
von Sperren und
Transaktionen

```

TRANSACTION transaktionsname;
    {Eine Transaktion mit dem Namen transaktionsname wird vereinbart.}

ENDTRANSACTION;
    {Das Ende einer Transaktion wird markiert.}

EXCLUSIVELOCK (sperrobjekt);
    {Es wird eine Schreibsperre auf das Sperrobjekt mit dem Namen sperrobjekt
    gesetzt. Sperrobjekte können Relationen oder Attribute von Relationen sein;
    letztere werden mittels Punktqualifizierung angesprochen.}

SHAREDLOCK (sperrobjekt);
    {Es wird eine Lesesperre auf das Sperrobjekt namens sperrobjekt gesetzt. Sper-
    robjekte können Relationen oder Attribute von Relationen sein.}

UNLOCK (sperrobjekt);
    {Die erste dem Sperrobjekt namens sperrobjekt auferlegte Lesesperre oder die
    dem Sperrobjekt auferlegte (einzige) Schreibsperre wird aufgehoben.}

```

Bei der UNLOCK-Anweisung ist zu beachten, dass auf ein Sperrobjekt entweder beliebig viele Lesesperren gesetzt werden dürfen oder aber genau eine Schreibsperre. Im Falle mehrerer gesetzter Lesesperren bezieht sich die UNLOCK-Anweisung auf die noch nicht aufgehobene Lesesperre, die zuerst gesetzt wurde.

Beispiel zur
Funktionsweise des
Sperrverfahrens

Beispiel 6.1

Betrachtet werden die in Kap. 4, Beispiel 4.4 eingeführten Relationen *Artikel* und *ArtikelBew*. Diese Relationen lauten bekanntlich:

Artikel(ArtikelNr, Bezeichnung, TechnDaten, Preis, Bestand, MindBestand,
DispoBestand, AuftrBestand)

ArtikelBew(ArtikelNr, ProjektNr, BewDatum, Menge, DispoMenge)

Eine Transaktion zum Verbuchen von Abgängen in der Relation *ArtikelBew* könnte beispielsweise wie folgt aussehen:

```
TRANSACTION abgangverbuchen;
BEGIN
    EXCLUSIVELOCK(Artikel);
    EXCLUSIVELOCK(ArtikelBew);
    IF (Artikel(ArtikelNr).Bestand - Artikel(ArtikelNr).MindBestand) >= Menge
    THEN
        UPDATE Artikel(ArtikelNr) SET Bestand := Bestand - Menge
        INSERT INTO ArtikelBew
            < ArtikelNr, ProjektNr, BewDatum, Menge,
            Artikel(ArtikelNr).DispoBestand >;
    ELSE
        OUTPUT('UNTERSCHREITUNG MINDESTBESTAND');
    ENDIF;
    UNLOCK(Artikel);
    UNLOCK(ArtikelBew);
ENDTRANSACTION;
```

Ein Artikelabgang darf in der Relation *ArtikelBew* nur dann verbucht werden, wenn der Mindestbestand nicht unterschritten wird. Es ist also zu prüfen, ob für den fraglichen Artikel der Bestand den Mindestbestand um die Abgangsmenge überschreitet. Diese Prüfung betrifft die Relation *Artikel*.

Zu Beginn der Transaktion werden die Relationen *Artikel* und *ArtikelBew* für lesende und schreibende Zugriffe durch andere Transaktionen gesperrt. Dann wird geprüft, ob die Differenz zwischen Bestand und Mindestbestand für den fraglichen Artikel die Abgangsmenge erreicht oder überschreitet. In der entsprechenden Bedingung bezeichnen *ArtikelNr* die Nummer des fraglichen Artikels und *Menge* die Abgangsmenge. Falls das Prüferergebnis "wahr" ist, wird zuerst der Artikelbestand in der Relation *Artikel* korrigiert und dann die Artikelbewegung in die Relation *ArtikelBew* eingefügt. Letzteres geschieht mit einer INSERT-Anweisung, in der die einzufügenden Attributwerte den aktuellen Werten der zwischen den Dreieckklammern angegebenen Variablen bzw. Attribute entsprechen.

Lautet das Prüfergebnis jedoch "falsch", so ist eine Verbuchung nicht zulässig und es wird eine entsprechende Fehlermeldung ausgegeben.

Die Transaktion endet mit dem Aufheben der auf den Relationen *Artikel* und *ArtikelBew* liegenden Schreibsperrern.

Für das dem Beispiel 6.1 zugrundeliegende Einfügeproblem stellt die eben angegebene Transaktion *abgangverbuchen* eine Lösungsmöglichkeit dar. Eine alternative Transaktion, die sich durch ein anderes Setzen von Sperren auszeichnet, wird in Übungsaufgabe 6.7 vorgestellt.

Übungsaufgabe 6.7

Betrachtet werde die in Beispiel 6.1 angegebene Transaktion *abgangverbuchen*. Den gleichen Zweck wie *abgangverbuchen* erfüllt die folgende Transaktion:

```
TRANSACTION abgverb;  
BEGIN  
    SHAREDLOCK(Artikel);  
    IF (Artikel(ArtikelNr).Bestand - Artikel(ArtikelNr).MindBestand) >= Menge  
    THEN  
        EXCLUSIVELOCK(Artikel);  
        EXCLUSIVELOCK(ArtikelBew);  
        UPDATE Artikel(ArtikelNr) SET Bestand := Bestand - Menge  
        INSERT INTO ArtikelBew  
            < ArtikelNr, ProjektNr, BewDatum, Menge,  
              Artikel(ArtikelNr).DispoBestand >;  
        UNLOCK(Artikel);  
        UNLOCK(ArtikelBew);  
    ELSE  
        UNLOCK(Artikel);  
        OUTPUT('UNTERSCHREITUNG MINDESTBESTAND');  
    ENDIF;  
ENDTRANSACTION;
```

anders gesetzte Sperren

Offensichtlich unterscheidet sich die Transaktion *abgverb* von der Transaktion *abgangverbuchen* ausschließlich durch anders gesetzte Sperren. Erläutern Sie die Unterschiede und gehen Sie insbesondere auch auf Vor- und Nachteile der Transaktion *abgverb* im Vergleich zur Transaktion *abgangverbuchen* ein.

b) Zweiphasen-Sperrprotokoll

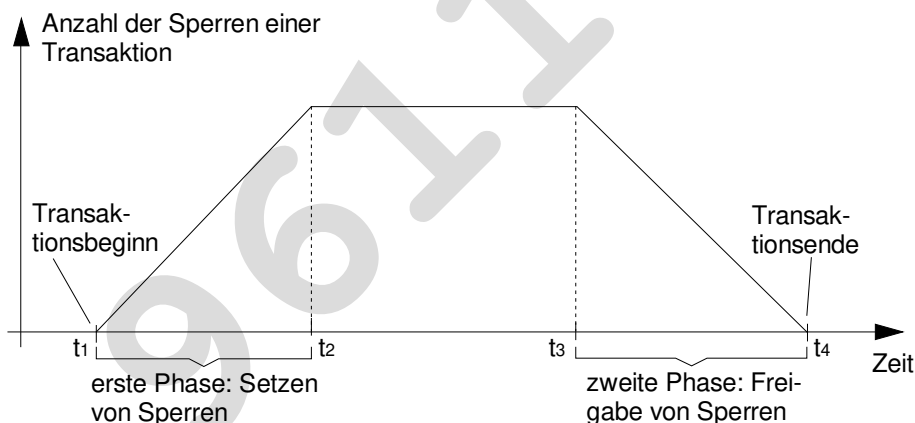
Der Grundansatz des Sperrverfahrens weist in Bezug auf das Setzen und Aufheben von Sperren noch Freiheitsgrade auf. Durch die mit einem Sperrprotokoll vorgegebenen Regeln für das Setzen und Aufheben von Sperren werden diese Freiheitsgrade ausgefüllt. Für das Zweiphasen-Sperrprotokoll gilt speziell:

Sobald in einer Transaktion die erste der von der Transaktion zuvor gesetzten Sperren aufgehoben wird, darf die Transaktion keine weitere Sperre mehr setzen. In einer Transaktion lassen sich damit zwei Phasen unterscheiden: Eine erste Phase, in der Sperren gesetzt, und eine zweite Phase, in der die gesetzten Sperren aufgehoben werden. Aus der Unterteilung der Sperraktivitäten in zwei Phasen leitet sich der Begriff "Zweiphasen-Sperrprotokoll" ab.

Zweiphasen-Sperrprotokoll

Das Anwachsen von Sperren und das anschließende Abbauen der Sperren beim Zweiphasen-Sperrprotokoll werden in Abb. 6.9 grafisch veranschaulicht.

In Abb. 6.9 wird vereinfachend unterstellt, dass die Anzahl der Sperren ab dem Transaktionsbeginn, also dem Zeitpunkt t_1 , bis zum Zeitpunkt t_2 linear ansteigt. Ebenso wird ein linearer Abbau der Sperren im Zeitraum von t_3 bis zu dem Transaktionsende t_4 unterstellt. Im Zeitraum zwischen t_2 und t_3 werden weder Sperren gesetzt, noch Sperren aufgehoben; im Grenzfall besitzt dieser Zeitraum die Dauer Null, d.h. die Zeitpunkte t_2 und t_3 fallen zusammen.



Veranschaulichung des Zweiphasen-Sperrprotokolls

Abb. 6.9. Setzen und Freigeben von Sperren beim Zweiphasen-Sperrprotokoll.

Bislang wurde lediglich dargelegt, welches Vorgehen ein Zweiphasen-Sperrprotokoll beinhaltet, aber nicht, welchen Sinn dieses Vorgehen besitzt. Offensichtlich wird bei dem Zweiphasen-Sperrprotokoll ein Datenbereich, auf dem eine Transaktion operiert, nur einmal gesperrt. Anders ausgedrückt: Solange noch Zugriffe einer Transaktion auf ein Sperrobjekt ausstehen, wird das Sperrobjekt nicht freigegeben. Andere Transaktionen können auf den entsprechenden Datenbereich daher erst dann zugreifen, wenn der Datenbereich von der aktuell betrachteten Transaktion nicht mehr benötigt wird. Auf diese Weise wird verhindert, dass während der Manipulation eines Datenobjekts durch eine Transaktion eben dieses Datenobjekt durch andere Transaktionen ebenfalls manipuliert wird. Würde man dies zulassen, so würde man einerseits nicht kontrollierbare Wertveränderungen des Datenobjekts und damit Inkonsistenzen in Kauf nehmen, und andererseits würde man das Risiko des Eintretens von Verklemmungen aufgrund konkurrierender Zugriffsversuche zu einem Datenbereich eingehen.

Motivation des Zweiphasen-Sperrprotokolls

Serialisierung paralleler Transaktionen

Das Zweiphasen-Sperrprotokoll verhindert Inkonsistenzen und Verklemmungen, indem es die Zugriffsversuche zu einem Datenbereich, die gleichzeitig von mehreren parallel abgewickelten Transaktionen ausgehen, serialisiert bzw. in eine serielle Ablauffolge bringt. Die serialisierende Wirkung des Zweiphasen-Sperrprotokolls lässt sich verdeutlichen, wenn man unterschiedliche Ablaufpläne für eine Menge von abzuwickelnden Transaktionen betrachtet.

Wie die in Beispiel 6.1 vorgestellte Transaktion *abgangverbuchen* zeigt, besteht eine Transaktion aus einer Folge von Anweisungen. Einige dieser Anweisungen, jedoch nicht notwendigerweise alle, stellen (elementare) Datenbankoperationen dar. Bei der Bildung und Untersuchung von Ablaufplänen begibt man sich auf die Ebene von Anweisungen bzw. elementaren Datenbankoperationen. Für den Begriff des Ablaufplans gilt im gegebenen Zusammenhang:

Begriff des Ablaufplans

Ein Ablaufplan einer Transaktionsmenge ist eine Folge von Anweisungen bzw. Datenbankoperationen, die zwei Bedingungen genügt:

- (1) Die Anweisungsfolge enthält die Anweisungen bzw. Datenbankoperationen sämtlicher Transaktionen der zugrundeliegenden Transaktionsmenge.
- (2) In der Anweisungsfolge treten die Anweisungen bzw. Datenbankoperationen einer Transaktion in der unveränderten, ursprünglichen Reihenfolge auf.

Die Bedingung (1) stellt sicher, dass jede Transaktion vollständig in einen Ablaufplan eingeht, und die Bedingung (2) gewährleistet für jede Transaktion die Beibehaltung der Ablauflogik.

In einem Ablaufplan können die Anweisungen der beteiligten Transaktionen in unterschiedlicher Weise verzahnt sein. Abhängig von der Art der Verzahnung liegt ein serieller oder ein nichtserieller Ablaufplan vor:

serieller und nichtserieller Ablaufplan

Bei einem seriellen Ablaufplan werden die Transaktionen einer Transaktionsmenge nacheinander und vollständig ausgeführt, bei einem nichtseriellen Ablaufplan werden die Anweisungen der einzelnen Transaktionen dagegen in verzahnter Form ausgeführt.

Ein serieller Ablaufplan weist folglich keinerlei Parallelität der beteiligten Transaktionen auf. Dagegen nimmt die Parallelität bei einem nichtseriellen Ablaufplan mit dem Grad der Verzahnung zu.

Da bei einem seriellen Ablaufplan keine Konfliktsituationen aufgrund konkurrierender Datenbankzugriffe eintreten können, ist ein serieller Ablaufplan stets konsistenzhaltend. Ein nichtserieller Ablaufplan muss dagegen nicht notwendigerweise konsistenzhaltend sein.

An einem einfachen Beispiel seien nun einige alternative Ablaufpläne demonstriert.

Beispiel 6.2

Aus Gründen der Vereinfachung seien hier lediglich zwei Datenobjekte, bezeichnet mit *Bestand* und *Menge*, einer nicht weiter erläuterten Datenbasis betrachtet. Zur Manipulation der beiden Datenobjekte dienen u.a. die beiden folgenden Transaktionen A und B:

```
TRANSACTION A;
BEGIN
  SHAREDLOCK (Menge);
  READ (Menge);
  EXCLUSIVELOCK (Bestand);
  READ (Bestand);
  Bestand := Bestand + Menge;
  WRITE (Bestand);
  UNLOCK (Menge);
  UNLOCK (Bestand);
ENDTRANSACTION A;
```

```
TRANSACTION B;
BEGIN
  SHAREDLOCK (Menge);
  READ (Menge);
  SHAREDLOCK (Bestand);
  READ (Bestand);
  OUTPUT (Menge, Bestand);
  UNLOCK (Menge);
  UNLOCK (Bestand);
ENDTRANSACTION B;
```

zwei einfache
Transaktionen

Die Transaktion A ermittelt die Werte der beiden Datenobjekte und schreibt ein Datenobjekt fort. Die Transaktion B ermittelt die Werte der beiden Datenobjekte und gibt die Werte aus.

Ein serieller und ein nichtserieller Ablaufplan der Transaktionsmenge {A, B} sind nachfolgend angegeben.

Ablaufplan 1

```
A TRANSACTION A;
A BEGIN
A  SHAREDLOCK (Menge);
A  READ (Menge);
A  EXCLUSIVELOCK (Bestand);
A  READ (Bestand);
A  Bestand := Bestand + Menge;
A  WRITE (Bestand);
A  UNLOCK (Menge);
A  UNLOCK (Bestand);
A ENDTRANSACTION A;
B TRANSACTION B;
B BEGIN
B  SHAREDLOCK (Menge);
B  READ (Menge);
B  SHAREDLOCK (Bestand);
B  READ (Bestand);
B  OUTPUT (Menge, Bestand);
B  UNLOCK (Menge);
B  UNLOCK (Bestand);
B ENDTRANSACTION B;
```

Ablaufplan 2

```
A TRANSACTION A;
A BEGIN
B TRANSACTION B;
B BEGIN
A  SHAREDLOCK (Menge);
B  SHAREDLOCK (Menge);
A  READ (Menge);
B  READ (Menge);
A  EXCLUSIVELOCK (Bestand);
A  READ (Bestand);
A  Bestand := Bestand + Menge;
A  WRITE (Bestand);
A  UNLOCK (Menge);
A  UNLOCK (Bestand);
A ENDTRANSACTION A;
B  SHAREDLOCK (Bestand);
B  READ (Bestand);
B  OUTPUT (Menge, Bestand);
B  UNLOCK (Menge);
B  UNLOCK (Bestand);
B ENDTRANSACTION B;
```

Beispiele für alternative
Ablaufpläne

Der Ablaufplan 1 ist seriell und damit auch konsistenz-erhaltend. Ebenfalls konsistenz-erhaltend ist der nichtserielle Ablaufplan 2. Man beachte, dass die mit der Anweisung *OUTPUT (Menge, Bestand)* ausgegebenen Werte für beide Ablaufpläne identisch sind.

Betrachtet seien nun zwei weitere Ablaufpläne. Sie sind nachfolgend dargestellt. Der Ablaufplan 3 ist seriell und konsistenz-erhaltend und der Ablaufplan 4 ist nichtseriell und außerdem nicht konsistenz-erhaltend.

Ablaufplan 3	Ablaufplan 4
B TRANSACTION B; B BEGIN B SHAREDLOCK (Menge); B READ (Menge); B SHAREDLOCK (Bestand); B READ (Bestand); B OUTPUT (Menge, Bestand); B UNLOCK (Menge); B UNLOCK (Bestand); B ENDTRANSACTION B; A TRANSACTION A; A BEGIN; A SHAREDLOCK (Menge); A READ (Menge); A EXCLUSIVELOCK (Bestand); A READ (Bestand); A Bestand := Bestand + Menge; A WRITE (Bestand); A UNLOCK (Menge); A UNLOCK (Bestand); A ENDTRANSACTION A;	A TRANSACTION A; A BEGIN B TRANSACTION B; B BEGIN A SHAREDLOCK (Menge); B SHAREDLOCK (Menge); A READ (Menge); B READ (Menge); B SHAREDLOCK (Bestand); A EXCLUSIVELOCK (Bestand); A READ (Bestand); A Bestand := Bestand + Menge; A WRITE (Bestand); B READ (Bestand); B OUTPUT (Menge, Bestand); A UNLOCK (Menge); A UNLOCK (Bestand); A ENDTRANSACTION A; B UNLOCK (Menge); B UNLOCK (Bestand); B ENDTRANSACTION B;

Ein Problem besteht bei Ablaufplan 4 bezüglich der beiden aufeinanderfolgenden Sperren:

B SHAREDLOCK (Bestand);
 A EXCLUSIVELOCK (Bestand);

Laut Abb. 6.8 darf auf die Lesesperre, welche die Transaktion *B* auf das Datenobjekt *Bestand* gesetzt hat, keine Schreibsperre einer anderen Transaktion auf das gleiche Objekt folgen. Der Ablaufplan 4 ist aus diesem Grunde unzulässig.

Vergleicht man die beiden nichtseriellen Ablaufpläne 2 und 4, so zeigt sich ein weiteres Problem: Der Ablaufplan 2 sieht zuerst die Bestandsfortschreibung mit der Anweisung

A Bestand := Bestand + Menge;

und später die Datenausgabe mit der Anweisung

B OUTPUT (Menge, Bestand);

vor. Im Ablaufplan 4 ist die Reihenfolge dieser Operationen genau umgekehrt. Der Ablaufplan 4 würde also, falls er zulässig wäre, ein anderes Ergebnis als der Ablaufplan 2 liefern.

unzulässiger Ablaufplan

Wie das Beispiel 6.2 zeigt, ist in einem Ablaufplan die Reihenfolge der Lese- und Schreiboperationen, die auf den gleichen Datenbereich zugreifen, signifikant. Das Ergebnis einer Leseoperation hängt nämlich davon ab, ob zuvor eine Schreiboperation durchgeführt wurde oder nicht. Aus diesem Grunde sind auch die beiden seriellen Ablaufpläne in Beispiel 6.2, Ablaufplan 1 und Ablaufplan 3, nicht äquivalent.

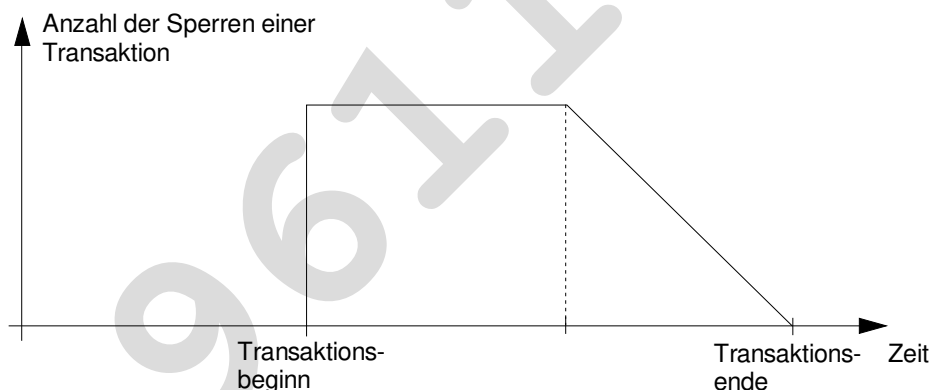
Die in Beispiel 6.2 vorgenommene Betrachtung gestattet es, folgende präzisere Formulierung des Zweiphasen-Sperrprotokolls leicht nachzuvollziehen (vgl. auch ZEHNDER 1985, S. 182):

Eine Transaktion umgeht die aus der parallelen Abwicklung von Transaktionen resultierenden Konsistenzprobleme, falls sie Sperren wie folgt handhabt:

- Jedes Datenobjekt, auf das die Transaktion lesend zugreift, wird mit einer Lesesperre belegt.
- Jedes Datenobjekt, auf das die Transaktion schreibend zugreift, wird mit einer Schreibsperre belegt.
- Sperren werden erst dann wieder freigegeben, wenn bereits alle Sperren gesetzt sind.

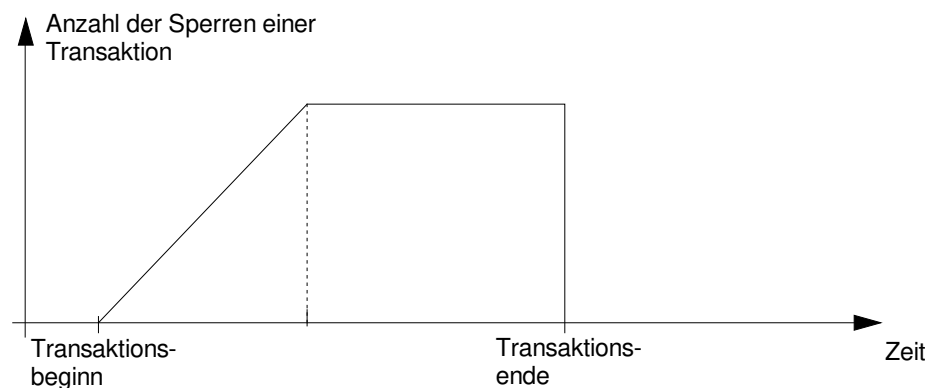
Präzisierung des
Zweiphasen-
Sperrprotokolls

Hinzuweisen ist noch auf zwei Grenzfälle des Zweiphasen-Sperrprotokolls, das Preclaiming und das Sperren bis zum Transaktionsende. Beide Grenzfälle werden in Abb. 6.10 grafisch veranschaulicht.



Grenzfälle des
Zweiphasen-
Sperrprotokolls

a) Preclaiming



b) Sperren bis zum Transaktionsende

Abb. 6.10. Preclaiming und Sperren bis zum Transaktionsende.

Preclaiming

Im Fall des Preclaiming werden sämtliche Sperren einer Transaktion zu Beginn der Transaktion gesetzt. Erst nach dem Setzen der Sperren wird mit der Verarbeitung begonnen. Gelangt eine Transaktion, die sich des Preclaiming bedient, erst einmal in die Phase der Verarbeitung, so kann ihre Ausführung nicht mehr durch Zugriffskonflikte in Frage gestellt werden. Da alle Sperren erfolgreich gesetzt wurden, sind Konflikte mit anderen Transaktionen ausgeschlossen. Das Preclaiming ist also geeignet, Verklemmungen zu verhindern.

Sperren bis zum Transaktionsende

Einen spezifischen Vorteil besitzt auch das Sperren bis zum Transaktionsende. Da die Datenobjekte, auf die ein schreibender Zugriff erfolgt, erst am Ende einer Transaktion freigegeben werden, kann die Transaktion jederzeit abgebrochen und zurückgesetzt werden. Andere Transaktionen werden durch das Zurücksetzen nicht betroffen. Die Notwendigkeit des Zurücksetzens einer Transaktion kann verschiedene Ursachen haben. Mögliche Ursachen sind beispielsweise ein Hardwarefehler oder eine Verklemmung.

Übungsaufgabe 6.8

Beide betrachteten Grenzfälle des Zweiphasen-Sperrprotokolls, das Preclaiming und das Sperren bis zum Transaktionsende, weisen im Gegensatz zum üblichen Zweiphasen-Sperrprotokoll einen gemeinsamen Nachteil auf. Um welchen Nachteil handelt es sich?

c) Verklemmung

Auf die unerwünschte Situation der Verklemmung wurde bereits mehrfach hingewiesen. Hier soll diese Situation zunächst erläutert werden und es sollen Möglichkeiten zur Vermeidung von Verklemmungen aufgezeigt werden.

Zunächst sei die Situation der Verklemmung verbal und danach anhand eines einfachen Beispiels erläutert.

Begriff der Verklemmung

Eine Verklemmung (engl. deadlock) liegt vor, falls parallel auszuführende Transaktionen wechselseitig auf das Aufheben gesetzter Sperren warten und falls sie von sich aus nicht in der Lage sind, den Wartezustand aufzulösen.

Wie eine Verklemmung eintreten kann, verdeutlicht das Beispiel 6.3.

Beispiel 6.3

Gegeben seien zwei nicht vollständig dargestellte Transaktionen *C* und *D*:

```
TRANSACTION C;
BEGIN
    EXCLUSIVELOCK (Menge);
    EXCLUSIVELOCK (Betrag);
    :
ENDTRANSACTION C;
```

```
TRANSACTION D;
BEGIN
    EXCLUSIVELOCK (Betrag);
    EXCLUSIVELOCK (Menge);
    :
ENDTRANSACTION D;
```

Beispiel für eine
Verklemmung

Werden beide Transaktionen parallel ausgeführt, so kann folgende Situation eintreten:

Die Transaktion *C* belegt das Datenobjekt *Menge* mit einer Schreibsperre und die Transaktion *D* belegt das Datenobjekt *Betrag* ebenfalls mit einer Schreibsperre. Nun kann weder die Transaktion *C*, noch die Transaktion *D* fortgesetzt werden. *C* kann die nächste Anweisung, das Setzen einer Schreibsperre auf *Betrag*, nicht ausführen, da *Betrag* bereits von *D* mit einer Schreibsperre belegt wurde. Und in analoger Weise kann *D* das bereits gesperrte Objekt *Menge* nicht mit einer weiteren Schreibsperre belegen.

Von sich aus können die beiden Transaktionen das gegenseitige Warten auf das Aufheben der Sperren nicht beenden - es liegt eine Verklemmung vor.

Ein Ansatz zur Vermeidung von Verklemmungen, das Preclaiming, wurde bereits angesprochen. Allerdings besitzt das Preclaiming auch den aus der Übungsaufgabe 6.8 bekannten Nachteil. Es ist daher sinnvoll, einen anderen Weg zu beschreiben und die Lösung von Synchronisationsproblemen einem sogenannten Lock-Manager zu übertragen.

Ein Lock-Manager ist eine Komponente eines DBVS, die für die Synchronisation der unter dem DBVS laufenden Transaktionen zuständig ist. Zu den Aufgaben eines Lock-Managers gehören:

Lock-Manager

- Die Umsetzung der Sperranweisungen der abgewickelten Transaktionen, d.h. das konkrete Setzen und Freigeben von Sperren auf der Datenbank.
- Die Analyse der Sperranforderungen der zur Ausführung anstehenden Transaktionen und das Entschärfen potentieller Konfliktsituationen durch die zeitliche Entzerrung konkurrierender Sperranforderungen.
- Das Aufdecken dennoch eingetretener Verklemmungen und das Auflösen von Verklemmungen durch das Zurücksetzen beteiligter Transaktionen.

Aufgaben eines Lock-
Managers

Die genannten Aufgaben machen deutlich, dass der in einem DBVS verwendete Lock-Manager die Verarbeitungseffizienz der unter dem DBVS abgewickelten Anwendungen unmittelbar beeinflusst.

d) Optimistisches Verfahren

optimistisches Verfahren	Zu Beginn dieses Kapitels wurde der Begriff des optimistischen Verfahrens bereits erläutert. Demnach verzichtete dieses Synchronisationsverfahren auf das Setzen von Sperren. Stattdessen werden die anstehenden Transaktionen - ohne Rücksicht auf möglicherweise eintretende Zugriffskonflikte - parallel auf einem Hilfsspeicher abgewickelt und es wird geprüft, ob ein Zugriffskonflikt aufgetreten ist. Falls dies zutrifft, werden bestimmte Transaktionen zurückgesetzt und später erneut gestartet. Die vom Zurücksetzen nicht betroffenen Inhalte des Hilfsspeichers werden in die Datenbank übernommen.
Phasen des optimistischen Verfahrens	Jede Transaktion wird somit in drei aufeinander folgenden Phasen, hier bezeichnet als Ausführungs-, Validierungs- und Übernahmephase, abgewickelt. Für diese Phasen gilt:
Ausführungsphase	(1) In der Ausführungsphase werden sämtliche Anweisungen einer Transaktion, einschließlich der Datenbankoperationen, durchgeführt. Schreiboperationen allerdings nicht auf der Datenbank, sondern auf einem Hilfsspeicher. Außerdem wird im Hilfsspeicher die anteilige Hilfsorganisation für die geschriebenen Daten aufgebaut.
Validierungsphase	(2) In der Validierungsphase wird geprüft, ob andere Transaktionen gleiche Datenbereiche benutzen. Trifft dies zu, so können Zugriffskonflikte auftreten. Aus der Menge der kollidierenden Transaktionen wird dann lediglich die Transaktion nicht zurückgesetzt, die zuerst in die Validierungsphase eingetreten ist. Die übrigen der kollidierenden Transaktionen werden nach dem Zurücksetzen erneut gestartet.
Übernahmephase	(3) In der Übernahmephase werden die Daten, welche die nicht abgebrochene Transaktion in den Hilfsspeicher geschrieben hat, in die Datenbank kopiert. Führt die Validierung allerdings zu keiner Konfliktsituation, so werden die zwischengespeicherten Daten sämtlicher nicht kollidierender Transaktionen in die Datenbank kopiert. Für die Transaktionen, die keine Schreiboperationen enthalten, entfällt die Übernahmephase.
Anwendung der Synchronisationsverfahren	Laut Übungsaufgabe 6.5 eignet sich das optimistische Verfahren eher für einen Datenbankbetrieb mit geringer Transaktionsdichte und das Sperrverfahren eher für einen Datenbankbetrieb mit hoher Transaktionsdichte. Leistungsfähige, in der Praxis eingesetzte DBVS verwenden in der Regel das Sperrverfahren in Verbindung mit dem Zweiphasen-Sperrprotokoll.

6.2.2 Datenrekonstruktion

Ereignisse wie der Abbruch einer Transaktion, der Zusammenbruch des Betriebssystems, das Auftreten eines Plattenfehlers usw. können zu einem undefinierten Datenbankzustand führen. Beispielsweise derart, dass Zeiger auf falsche Datenobjekte verweisen oder physische Sätze falsche Daten enthalten. Die Behebung solcher Inkonsistenzen ist das Ziel der Datenrekonstruktion. Darunter ist folgendes zu verstehen:

Begriff der Datenrekonstruktion

Als Datenrekonstruktion (engl. recovery) bezeichnet man das Wiederherstellen eines korrekten Datenbankzustandes nach dem Auftreten eines Fehlers. Je nach der Fehlerart unterscheidet man folgende Formen der Datenrekonstruktion:

- (1) Das Zurücksetzen (engl. rollback) zur Behebung der Folgen von Transaktionsfehlern.
- (2) Den Neustart (engl. restart), auch Wiederanlauf genannt, zur Behebung der Folgen von Betriebssystem- bzw. DBVS-Fehlern.

(3) Die Datenrekonstruktion im engeren Sinne zur Behebung von Speicherfehlern.

Nachfolgend werden die genannten Rekonstruktionsformen detaillierter behandelt.

a) Zurücksetzen

Bei der Ausführung von Transaktionen auftretende Fehler wie Verklemmungen, Programmfehler, Integritätsverletzungen usw. können durch das Zurücksetzen von Transaktionen behoben werden. Das Zurücksetzen betroffener Transaktionen erfolgt im laufenden Datenbankbetrieb parallel zur Abarbeitung anderer Transaktionen. Durch das Zurücksetzen von Transaktionen werden alle Änderungen, welche diese Transaktionen in der Datenbank bewirkt haben, wieder rückgängig gemacht. Ein Zurücksetzen ist allerdings nur dann möglich, wenn die auf den veränderten Datenobjekten liegenden Sperren noch nicht freigegeben wurden.

Für das Vorgehen bei dem Zurücksetzen bieten sich zwei grundlegende Möglichkeiten an:

- Rollback mit Hilfe von Update-Kopien,
- Rollback mit Hilfe einer Logdatei.

Vorgehensweisen des
Zurücksetzens

Die Update-Kopien bzw. die Logdatei (engl. logfile) enthalten die für das Zurücksetzen erforderlichen Informationen.

Rollback mit Update-Kopien

Das Zurücksetzen mit Hilfe von Update-Kopien setzt voraus, dass bei der Ausführung von Transaktionen die Änderungen von Objektwerten zunächst ausschließlich auf Kopien der betroffenen Datenbankobjekte ausgeführt werden. Nach der fehlerfreien Ausführung einer Transaktion werden die in der zugehörigen Update-Kopie verzeichneten Änderungen in die Datenbank eingetragen.

Rollback mit Update-
Kopien

Tritt bei der Ausführung einer Transaktion jedoch ein Fehler auf, so ist die Transaktion zurückzusetzen. Dies geschieht dadurch, dass die erzeugte Update-Kopie gelöscht wird. Das Eintragen der auf der Update-Kopie verzeichneten Änderungen von Objektwerten in die Datenbank unterbleibt somit.

Bei dieser Form des Zurücksetzens enthält die Datenbank stets die Objektwerte, die den Zustand nach dem Zurücksetzen kennzeichnen. Dies ist deshalb möglich, weil die durch eine Transaktion bewirkten Änderungen nur dann in die Datenbank eingetragen werden, wenn feststeht, dass die Transaktion nicht zurückgesetzt wird.

Rollback mit einem Logfile

Rollback mit einem
Logfile

Auch das Rollback mit einem Logfile basiert auf der Verwendung von kopierten Objektwerten. Anders als bei dem Anlegen von Update-Kopien werden jedoch vom DBVS Änderungen sowohl in die Datenbank, als auch in den Logfile eingetragen. Das Zurücksetzen einer Transaktion erfordert daher Wertkorrekturen in der Datenbank. Hierbei wird der Logfile als Informationsbasis verwendet.

Ebenso wie Update-Kopien wird der Logfile vom DBVS angelegt. Pro ausgeführter Transaktion enthält der Logfile etwa folgende Informationen:

Inhalt eines Logfile

- Kennzeichen für den Beginn eines Transaktionseintrags,
- Identifikation der Transaktion,
- Vorabzüge (engl. before images) der durch die Transaktion veränderten Objekte,
- Nachabzüge (engl. after images) der durch die Transaktion veränderten Objekte,
- Kennzeichen für das Ende eines Transaktionseintrags.

Vorabzug

Pro verändertem Objekt enthält ein Vorabzug folgende Informationen:

- Identifikation der Transaktion,
- Identifikation des Datenobjekts,
- Objektwert vor der Veränderung.

Nachabzug

Entsprechend geht aus einem Nachabzug pro verändertem Objekt folgendes hervor:

- Identifikation der Transaktion,
- Identifikation des Datenobjekts,
- Objektwert nach der Veränderung.

Tritt bei der Ausführung einer Transaktion kein Fehler auf, so sind keine weiteren Aktionen erforderlich, da Veränderungen von Objektwerten auch in der Datenbank vorgenommen wurden. Im Falle eines Transaktionsfehlers wird das erforderliche Zurücksetzen der Transaktion wie folgt realisiert:

Der Logfile wird rückwärts gelesen und für jedes von der Transaktion veränderte Datenobjekt wird der alte Objektwert in die Datenbank geschrieben. Die alten Objektwerte ergeben sich unmittelbar aus den entsprechenden Vorabzügen.

Übungsaufgabe 6.9

Bei dem Rollback mit Hilfe eines Logfile ist es nicht gleichgültig, ob Änderungen von Objektwerten zuerst in die Datenbank und dann in den Logfile eingetragen werden oder umgekehrt. Geben Sie an, in welcher Reihenfolge Änderungseinträge vorzunehmen sind und begründen Sie Ihre Antwort.

b) Neustart

Ein Neustart des Datenbankbetriebs bzw. des DBVS ist im Falle einer eingetretenen Funktionsunfähigkeit des DBVS erforderlich. Als auslösende Ursachen für eine solche Situation kommen z.B. ein Zusammenbruch des Betriebssystems oder Hardwarefehler - allerdings keine Defekte des Massenspeichers - in Frage. Voraussetzung für einen Neustart ist ein intakter Massenspeicher; der Arbeitsspeicherinhalt darf dagegen verlorengegangen sein.

Neustart

Unter der genannten Voraussetzung ist es das Ziel eines Neustarts, die Datenbank wieder in einen konsistenten Zustand zu versetzen. Denn es ist davon auszugehen, dass sich die Datenbank nach einem durch einen Systemfehler verursachten Abbruch des Datenbankbetriebs in einem inkonsistenten Zustand befindet. Die Inkonsistenzen resultieren aus dem Abbruch von Transaktionen. Ein konsistenter Zustand lässt sich somit durch das Zurücksetzen der Transaktionen herstellen, die bei dem Abbruch noch nicht beendet waren.

Als Informationsbasis für einen Neustart bzw. für ein Zurücksetzen nicht beendeter Transaktionen dient der Logfile. Zur leichten Ermittlung der zurückzusetzenden Transaktionen werden in regelmäßigen Zeitabständen, z.B. wenigen Minuten, Kontrollmarken (engl. checkpoints) auf den Logfile geschrieben. Für den Inhalt und den Zweck von Kontrollmarken gilt:

Kontrollmarken

- Ein Checkpoint enthält eine Liste der zu einem bestimmten Zeitpunkt aktiven, d.h. in Ausführung befindlichen Transaktionen.
- Auf das Schreiben eines Checkpoints folgt stets und unmittelbar das Aktualisieren der Datenbank. Es werden also die Arbeitsspeicherblöcke bzw. -seiten, in denen seit dem letzten Checkpoint Objektwerte geändert wurden, in die Datenbank übertragen.

Anmerkung:

Aus Vereinfachungsgründen wurde bei der Erläuterung des Zurücksetzens unter Punkt a) nicht erwähnt, dass Änderungen von Objektwerten nicht unmittelbar im Massenspeicher vorgenommen werden. Wie aus den Ausführungen zum Datenbankkonzept (vgl. Kap. 1.3) bekannt ist, werden Änderungen jeweils nur in den im Arbeitsspeicher befindlichen Ausschnitten der Datenbank, den sogenannten Pages oder Seiten, vorgenommen. Erst durch das Übertragen geänderter Seiten in den Externspeicher wird die Datenbank selbst nachgeführt.

Nachführen von
geänderten Seiten

Zur weiteren Verdeutlichung des Zwecks von Kontrollmarken diene die Abb. 6.11. Sie zeigt einen Zeitausschnitt aus einem Datenbankbetrieb, in dem fünf Transaktionen - bezeichnet mit *T1* bis *T5* - aktiv sind.

Schreiben eines
Checkpoints

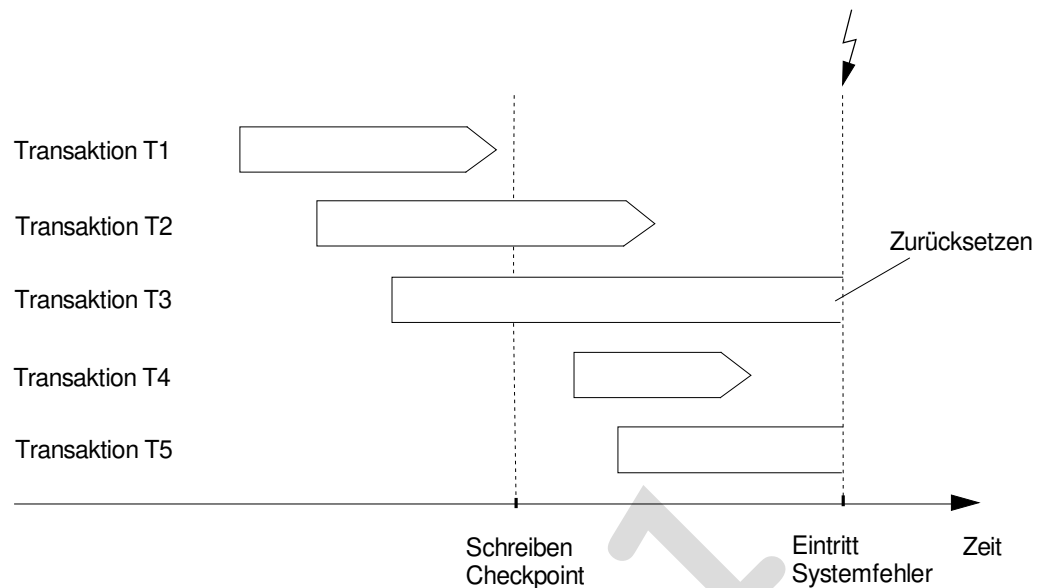


Abb. 6.11. Schematische Darstellung eines Datenbankbetriebs mit dem Schreiben eines Checkpoints und dem Eintritt eines Systemfehlers.

Die in Abb. 6.11 ausgewiesenen Transaktionen lassen sich in drei Gruppen unterteilen:

- (1) Transaktionen, die vor dem zuletzt gesetzten Checkpoint enden. Diese Gruppe besteht hier aus der Transaktion *T1*.
- (2) Transaktionen, die nach dem zuletzt gesetzten Checkpoint enden. Im gegebenen Fall sind dies die Transaktionen *T2* und *T4*.
- (3) Transaktionen, die nach dem zuletzt gesetzten Checkpoint noch aktiv waren, wegen des eingetretenen Systemfehlers aber nicht beendet werden konnten. Hier sind dies die Transaktionen *T3* und *T5*.

Jede dieser Gruppen ist bei einem Neustart unterschiedlich zu behandeln. Im Einzelnen gilt:

abgeschlossene
Transaktionen

- Die Transaktionen der Gruppe (1) sind abgeschlossen und die von Ihnen bewirkten Änderungen wurden bereits in die Datenbank eingetragen. Eine weitere Behandlung dieser Transaktionen entfällt daher.

nochmals
nachzuvollziehende
Transaktionen

- Die Transaktionen der Gruppe (2) sind auch abgeschlossen. Jedoch konnten die von Ihnen bewirkten Änderungen wegen des Betriebsabbruchs nicht mehr in die Datenbank eingetragen werden. Die von diesen Transaktionen bewirkten Änderungen müssen daher nochmals nachvollzogen und in die entsprechenden Arbeitsspeicherseiten eingetragen werden. Als Informationsbasis dienen hierbei die After Image-Einträge auf dem Logfile. Bei dem Schreiben des nächsten Checkpoints werden die Änderungen in die Datenbank übernommen.

zurückzusetzende
Transaktionen

- Wegen des Betriebsabbruchs konnten die Transaktionen der Gruppe (3) nicht beendet werden. Folglich liegen Inkonsistenzen vor, die durch das Zurücksetzen dieser Transaktionen zu beheben sind. Als Informationsbasis verwendet man hierbei bekanntlich die zutreffenden After Image-Einträge auf dem Logfile. Im Zuge der Wiederaufnahme des Datenbankbetriebs werden die zurückgesetzten Transaktionen wiederholt.

Genau genommen ist bei den Transaktionen der Gruppe (3) eine weitere Differenzierung erforderlich. Die folgende Übungsaufgabe geht darauf ein.

Übungsaufgabe 6.10

Nicht alle Transaktionen der Gruppe (3) müssen zwangsläufig Inkonsistenzen verursachen. Die obigen Ausführungen gelten daher nur für einen Teil der Transaktionen der Gruppe (3). Geben Sie an, wodurch dieser Teil der Transaktionen charakterisiert ist. Stellen Sie außerdem dar, welche Eigenschaft den anderen Teil der Transaktionen kennzeichnet und wie der andere Teil bei dem Neustart zu behandeln ist.

c) Datenrekonstruktion

Zweifelloos wiegt ein Systemfehler schwerer als ein Transaktionsfehler, da letzterer im laufenden Datenbankbetrieb behoben werden kann, während ein Systemfehler zu einem Betriebsabbruch führt und eine Wiederaufnahme des Betriebs erfordert. Noch schwerer als ein Systemfehler wiegt allerdings ein Speicherfehler, genauer ein Fehler im Massenspeicher wie z.B. ein Defekt des Lese-/Schreibkopfes (engl. disc head crash). Denn unweigerlich führt ein Speicherfehler zur Zerstörung oder Unleserlichkeit von Daten im Massenspeicher. Notwendig ist in einem solchen Fall die Rekonstruktion der zerstörten oder verfälschten Daten. Wie nachfolgend dargelegt wird, lässt sich ein korrekter Datenbankzustand mittels älterer Datenbankkopien, auch Dump (engl. dump) genannt, rekonstruieren.

Speicherfehler als Anlass
der Datenrekonstruktion

Ein Dump ist nichts anderes als ein Abzug der gesamten Datenbank. Da das Erzeugen eines Dumps einen erheblichen Aufwand verursacht, zieht man Dumps nur in größeren Zeitabständen, beispielsweise in Abständen von mehreren Stunden. In welcher Weise nun ausgehend von einem Dump ein korrekter Datenbankzustand rekonstruiert werden kann, sei anhand von Abb. 6.12 erläutert.

Dump

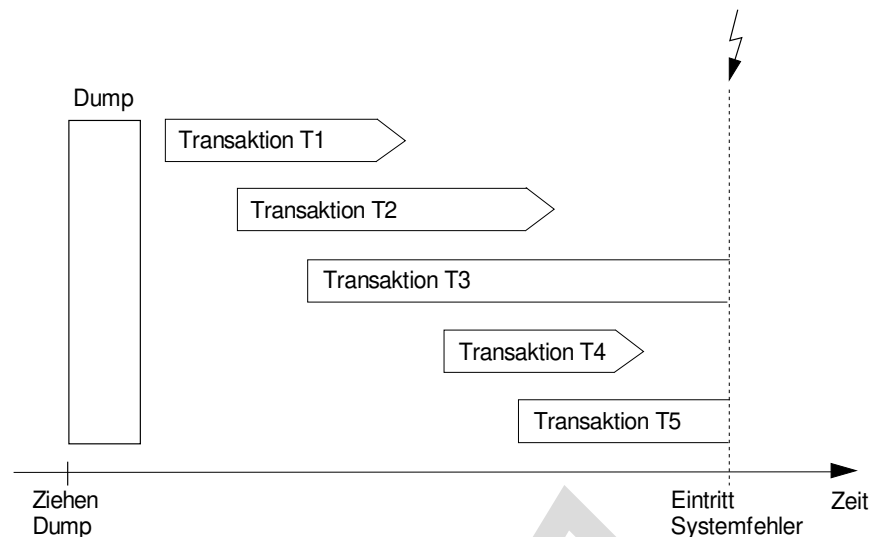


Abb. 6.12. Schematische Darstellung eines Datenbankbetriebs mit dem Ziehen eines Dumps und dem Eintritt eines Speicherfehlers.

Ziehen eines Dumps

Wie die Abb. 6.12 zeigt, dürfen während des Ziehens eines Dumps keine Transaktionen aktiv sein. Die nach dem Erzeugen des Dumps und vor dem Eintritt des Speicherfehlers gestarteten Transaktionen lassen sich in zwei Gruppen einteilen:

- (1) Transaktionen, welche vor dem Eintritt des Speicherfehlers beendet wurden. Hier sind dies die Transaktionen *T1*, *T2* und *T4*.
- (2) Transaktionen, welche durch den Eintritt des Speicherfehlers bzw. durch den dann sofort erfolgenden Abbruch des Datenbankbetriebs unterbrochen wurden. Zur Gruppe (2) gehören im gegebenen Fall die Transaktionen *T3* und *T5*.

zwei Gruppen von Transaktionen:

Ausgehend von dem Datenbankzustand, der durch den zuletzt erzeugten Dump definiert ist, kann nun mittels zweier Gruppen von Maßnahmen ein korrekter Datenbankzustand rekonstruiert werden:

zu wiederholende Transaktionen

- Sämtliche Transaktionen der Gruppe (1) werden aufsetzend auf dem Dump wiederholt. Als Informationsbasis dienen hierbei die After Image-Einträge im Logfile. Um diesen Rekonstruktionsschritt durchführen zu können, müssen die seit dem letzten Dump erzeugten After Image-Einträge vollständig aufbewahrt werden.

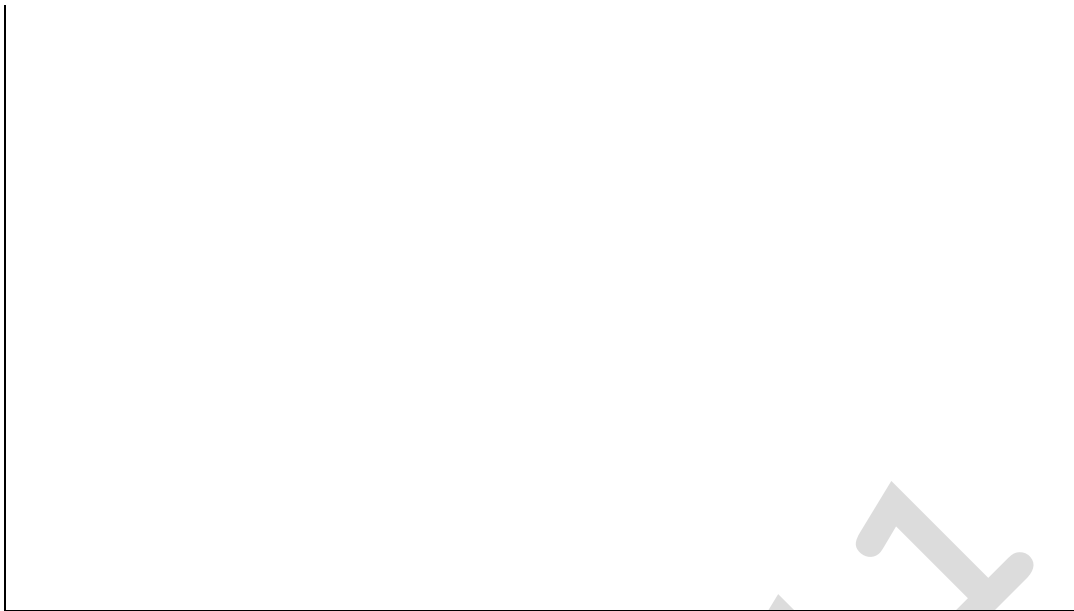
neu zu startende Transaktionen

- Sämtliche Transaktionen der Gruppe (2) werden neu gestartet. Dies geschieht allerdings erst, nachdem sämtliche Transaktionen der Gruppe (1) wiederholt wurden.

Auch im Verlaufe einer durch einen Speicherfehler notwendig gewordenen Datenrekonstruktion werden Checkpoints geschrieben, um mögliche Systemfehler beheben zu können. Aus Vereinfachungsgründen wurde dieser Aspekt bei den obigen Ausführungen vernachlässigt.

Übungsaufgabe 6.11

Betrachtet seien die oben skizzierten Maßnahmen der Datenrekonstruktion. Erläutern Sie, worin genau der Unterschied zwischen dem Wiederholen und dem erneuten Starten von Transaktionen besteht.



Datenschutzgesetze

6.3 Datenschutz

Unter dem Begriff "Datenschutz" versteht man den Schutz von Personen vor der Beeinträchtigung ihrer schutzwürdigen Belange durch die Verarbeitung der sie betreffenden Daten. Anders als in einigen anderen Staaten gelten in der Bundesrepublik Deutschland die auf Bundes- und Landesebene erlassenen Datenschutzgesetze ausschließlich für natürliche Personen. Daten betreffen eine Person dann, wenn sie Angaben über persönliche oder sachliche Verhältnisse dieser Person einschließen. Was schutzwürdige Belange sind, ist im Einzelfall festzulegen. Denn den schutzwürdigen Belangen von Personen stehen die Interessen der Stellen gegenüber, welche personenbezogene Daten (geschäftsmäßig) verarbeiten.

Zwei Teilprobleme des Datenschutzproblems seien noch näher charakterisiert, das Datenspeicherungsproblem und das Datenverwendungsproblem. Was die Datenspeicherung betrifft, gestatten die gesetzlichen Bestimmungen lediglich die Aufzeichnung solcher personenbezogenen Daten, die zur Abwicklung offizieller betrieblicher Aufgaben wie z.B. Lohn- und Gehaltsabrechnung unerlässlich sind. Hierbei sind allerdings spezielle Bestimmungen zu beachten. So beispielsweise

- das Recht des Einzelnen auf Bekanntgabe und gegebenenfalls Korrektur der über ihn gespeicherten Daten,
- definitive Löschfristen für Daten mit einem gewissen Alter und

Datenspeicherungsproblem

- die Ergänzung bewertender Daten um Kontexte, welche Nachteile aufgrund von Fehlinterpretationen ausschließen.

In analoger Weise unterliegt die Datenverwendung Regelungen, welche die Nutzung personenbezogener Daten auf den offiziellen betrieblichen Verarbeitungszweck beschränken. Hierbei sind u.a. folgende Bestimmungen einzuhalten:

Datenverwendungsproblem

- Geschützte personenbezogene Daten dürfen nur für autorisierte Personen bereitgestellt werden. Also solchen Personen, die offiziell mit der Erfüllung entsprechender Aufgaben beauftragt sind.
- Anderen Personen und insbesondere externen Institutionen dürfen personenbezogene Daten keinesfalls zur "Einsicht" zur Verfügung gestellt werden.
- Personenbezogene Daten dürfen nur in einer Form bereitgestellt werden, welche lediglich Auswertungen im Rahmen der offiziellen Aufgaben gestattet.

Gruppen von Datenschutzmaßnahmen:

Mit Hilfe von drei Gruppen von Maßnahmen versucht man, die Datenschutzproblematik in den Griff zu bekommen, nämlich mit

- legislativen Maßnahmen,
- organisatorischen Maßnahmen und
- technischen Maßnahmen.

legislative Maßnahmen

Zu den legislativen Maßnahmen zählen die oben erwähnten Datenschutzgesetze. Sie wurden bereits im Kurs "Einführung in die EDV" angesprochen (vgl. Kurs 00008, KE 2, S. 85 ff.) und werden daher hier nicht weiter behandelt.

organisatorische Maßnahmen

Organisatorische Maßnahmen sind von den Betreibern von Datenverarbeitungsgeräten zu ergreifen. Diese Maßnahmen liegen primär auf Rechenzentrumsebene und zielen darauf ab, Unbefugten den Zugang zu Informationen zu erschweren. Beispielsweise mittels baulicher Maßnahmen oder mittels Personenkontrollen.

technische Maßnahmen

Für technische Maßnahmen sind System- bzw. Datenbankspezialisten zuständig, denn diese Maßnahmen sind auf der Datenbankebene zu realisieren. Im Einzelnen lassen sich

- Identitätskontrollen,
- Zugriffskontrollen und
- kryptographische Maßnahmen

unterscheiden.

Insgesamt ergeben sich damit fünf Ebenen des Datenschutzes. Die einzelnen Ebenen zeichnen sich durch unterschiedliche Nähe zur Datenbank aus. Sie lassen sich daher als übereinander liegende und gleichsam über die Datenbank gestülpte Schutzschichten begreifen. Die Abb. 6.13 stellt die Ebenen des Datenschutzes in schematischer Form dar.

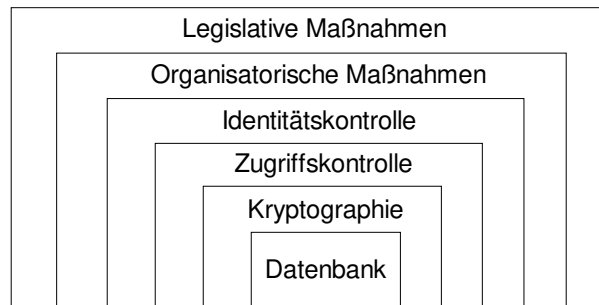


Abb. 6.13. Ebenen des Datenschutzes.

Das Wesen organisatorischer Datenschutzmaßnahmen ergibt sich aus den obigen Hinweisen. Hier erübrigen sich daher weitere Ausführungen. Ebenso genügt es hier, darauf hinzuweisen, dass es sich bei kryptographischen Maßnahmen um sehr spezielle Verfahren zur Verschlüsselung und damit zur Unkenntlichmachung von Daten handelt. Für die weitere Behandlung im vorliegenden Kapitel verbleiben folglich die Ebenen Identitätskontrolle und Zugriffskontrolle.

a) Identitätskontrolle

Formen der
Identitätskontrolle

Das Ziel der Identitätskontrolle besteht darin, einen Datenbankbenutzer vor seinem Zugang zur Datenbank als befugten Benutzer zu erkennen und ihm erst nach seiner Identifikation als autorisiertem Benutzer den Zugang bzw. Zugriff zur Datenbank zu ermöglichen. Folgende Formen der Identitätskontrolle sind üblich:

- Identifikationskarten,
- Prüfen physischer Benutzermerkmale,
- Passwortverfahren.

Identifikationskarten sind meist kleinformatige, mit einem Magnetstreifen versehene Kunststoffkarten. Der Magnetstreifen trägt Informationen, welche den Kartenbesitzer eindeutig identifizieren. Eine Identifikationskarte ermöglicht beispielsweise den Zugang zu bestimmten Geräteräumen und die Gerätebenutzung. Die Identitätsprüfung wird durch die Eingabe der Karte in eine Leseeinrichtung ausgelöst. Gelegentlich wird in die Identitätsprüfung das weiter unten erläuterte Passwortverfahren einbezogen. In diesem Fall ist nicht nur die Karte einzuführen, sondern - nach Aufforderung - zusätzlich ein Passwort einzugeben.

Identifikationskarten

Im Unterschied zu den identifizierenden Informationen auf einer Magnetstreifenkarte sind physische Benutzermerkmale unmittelbar an den einzelnen Benutzer gebunden und nicht losgelöst von ihm abprüfbar. Als Merkmale kommen vor allem Fingerabdrücke oder die menschliche Stimme in Frage. Zugang zu bestimmten Geräteräumen erhält ein Benutzer, nachdem der Vergleich seines Fingerabdrucks mit einem gespeicherten Ab-

physische
Benutzer-merkmale

druckmuster oder die Frequenzanalyse seiner Stimme seine Identität zweifelsfrei erwiesen hat.

Passwortverfahren

Meist kommt das Passwortverfahren dann zum Zuge, wenn sich ein Benutzer an einem Computer-Arbeitsplatz in den laufenden Betrieb einschalten möchte, um beispielsweise ein bestimmtes Softwarepaket zu nutzen oder bestimmte, im Netzbetrieb verfügbare Dienste in Anspruch zu nehmen. Durch die Eingabe eines Kennworts, des sogenannten Passworts, hat sich der Benutzer zu legitimieren. Ergibt die Prüfung ein gültiges Passwort, so erhält der Benutzer Rechenerlaubnis. Das Passwortverfahren wird in zwei Varianten praktiziert, zum einen mit festen und zum anderen mit variablen Passwörtern:

feste Passwörter

- Bei der Verwendung fester Passwörter ändern sich die den Benutzern zugeteilten Passwörter im Zeitablauf nicht. Nachteilig ist in diesem Fall das nicht zu unterschätzende Risiko des Entwendens von Passwörtern. Beispielsweise durch die Beobachtung der Passwordeingabe von Seiten unbefugter Dritter.

variable Passwörter

- Variable Passwörter ändern sich im Zeitablauf. Die Gefahr der unbefugten missbräuchlichen Verwendung ist daher geringer als bei festen Passwörtern. Eine Passwortänderung ist dann jederzeit möglich, wenn der Benutzer sein Passwort vor der Eingabe erst ermitteln muss. Die Passwörtermittlung lässt sich z.B. derart organisieren, dass dem Benutzer ein von Fall zu Fall variierender Text angezeigt wird, aus dem er das Passwort nach einem festen Verfahren entnehmen kann.

mehrstufiges Passwortverfahren

Natürlich ist es auch möglich, das Passwortverfahren mehrstufig einzusetzen. Denkbar ist beispielsweise eine Regelung, bei der ein Benutzer ein erstes Passwort für den Zugang zu einem Rechnersystem und ein zweites Passwort für den Zugang zu einer mit diesem Rechnersystem betriebenen Datenbank benötigt.

Übungsaufgabe 6.12

Betrachtet seien die eben angesprochenen Formen der Identitätskontrolle, nämlich die Benutzung von Identifikationskarten, das Prüfen physischer Benutzermerkmale und das Passwortverfahren. Nennen Sie je einen Vorteil und einen Nachteil dieser Formen der Identitätskontrolle.

b) Zugriffskontrolle

Datenbanken sind sehr komplexe Gebilde. Die Gewährleistung der Datenintegrität erfordert daher sorgfältige Kontrollen des Datenbankzugangs. Das Ziel solcher Zugriffskontrollen besteht in der Verhinderung von Zugriffen auf die Datenbasis durch Unbefugte. Insbesondere gehört dazu auch das Vereiteln von Versuchen, unter Umgehung des DBVS auf die Datenbasis zuzugreifen. Autorisierten Benutzern sollen dagegen lesende und schreibende Zugriffe auf die Datenbasis ermöglicht werden. Die Art und der Umfang der Benutzerzugriffe haben sich hierbei an den erteilten Zugriffsberechtigungen zu orientieren.

Was die Art von Zugriffsrechten betrifft, ist zu unterscheiden zwischen:

- Leseberechtigung,
- Schreibberechtigung,
- keine Zugriffsberechtigung.

Arten von Zugriffs-
berechtigungen

Wie später am Beispiel des Systems R noch gezeigt wird, kann das Schreibrecht weiter ausdifferenziert werden in das Recht zum Einfügen, zum Ändern, zum Löschen usw.

Abstufungen sind auch bei dem Umfang von Zugriffsrechten möglich. Sie lassen sich beispielsweise durch die Vorgabe von Zugriffsbedingungen realisieren. Man unterscheidet folgende Arten von Zugriffsbedingungen:

Arten von Zugriffs-
bedingungen

- wertunabhängige Zugriffsbedingungen,
- wertabhängige Zugriffsbedingungen,
- kontextabhängige Zugriffsbedingungen,
- funktionsbezogene Zugriffsbedingungen.

Wertunabhängige Zugriffsbedingungen sind nicht an die Werte von Datenobjekten gebunden. Sie können also wertunabhängig vergeben werden. Beispielsweise in Form einer Berechtigungsmatrix, wie sie in Abb. 6.14 dargestellt ist.

wertunabhängige
Zugriffsbedingungen

Objekte Benutzer	Personal- nummer	Name	Gehalt	Ausbildung	Beurteilung
Vorgesetzter	L	L	L	L	L, S
Personal- abteilung	L, S	L, S	L, S	L, S	L
Gehalts- abteilung	L	N	L	N	N

Legende: L - Leseberechtigung
S - Schreibberechtigung
N - keine Zugriffsberechtigung (engl. no access)

Abb. 6.14. Darstellung wertunabhängiger Zugriffsbedingungen in Form einer Berechtigungsmatrix.

Die in Abb. 6.14 formulierten Zugriffsbedingungen regeln den Zugang zu Personaldaten. Sie sehen beispielsweise vor, dass Daten über die Ausbildung eines Mitarbeiters von dem Vorgesetzten des Mitarbeiters lediglich eingesehen und von der Personalabteilung eingegeben und fortgeschrieben werden dürfen. Der Gehaltsabteilung ist dagegen jeglicher Zugang zu Ausbildungsdaten verwehrt.

Wertabhängige Zugriffsbedingungen machen den Datenzugang vom Wert des jeweiligen Datenobjekts abhängig. Für das Datenobjekt *Gehalt* in einer betrieblichen Datenbank kann die Zugriffsberechtigung der Gehaltsabteilung beispielsweise wie folgt geregelt sein:

wertabhängige
Zugriffsbedingungen

L, falls $\text{Gehalt}(\text{PersonalNr}) \leq 60000$ und
 N, falls $\text{Gehalt}(\text{PersonalNr}) > 60000$.

Hierbei bezeichnet $\text{Gehalt}(\text{PersonalNr})$ das Gehalt eines durch die Personalnummer identifizierten Mitarbeiters in DM.

Kontextabhängige Zugriffsbedingungen betreffen den gleichzeitigen Zugriff auf mehrere Datenobjekte bzw. Attribute. Sie sind von erheblicher Bedeutung, da häufig erst die Verbindung von bestimmten (Einzel-)Informationen eine Schutzwürdigkeit begründet. An folgendem einfachen Beispiel, das sich auf die Zugriffsberechtigung der Gehaltsabteilung eines Betriebes bezieht, sei dies demonstriert:

kontextabhängige
Zugriffsbedingungen

L für Zugriffe auf den Kontext (PersonalNr, Gehalt) und
 N für Zugriffe auf den Kontext (Name, Gehalt).

Gestattet ist also lediglich das Lesen von Gehaltsdaten in Verbindung mit Personalnummern, aber nicht in Verbindung mit Namen.

Funktionsbezogene Zugriffsbedingungen gestatten nur einen mittelbaren Zugriff zu bestimmten Datenobjekten. Der mittelbare Datenzugang wird über Auswertungsfunktionen, insbesondere über statistische Funktionen, hergestellt. An einem Beispiel sei dies demonstriert. Für die Zugriffsberechtigung der Planungsabteilung eines Betriebes zu Gehaltsdaten möge gelten:

funktionsbezogene
Zugriffsbedingungen

N für Zugriffe auf Gehalt,
 L für die Auswertungen SUM (Gehalt) und AVG (Gehalt).

Auf Gehaltsdaten direkt darf die Planungsabteilung demnach nicht zugreifen. Jedoch dürfen Gehaltssummen und durchschnittliche Gehälter gebildet werden.

Reaktionen auf
unberechtigte
Zugriffsversuche

Im Falle unberechtigter Datenbankzugriffe sind die Reaktionen auf die Art des Zugriffsversuchs bzw. den Schweregrad der Verletzung von Zugriffsbedingungen abzustimmen. Denkbar sind u.a. folgende Reaktionsmuster:

- (1) Zugriffsversuche durch Unbefugte werden registriert, d.h. es werden Daten wie Benutzer-ID, Gerät, Datum, Uhrzeit und angesprochenes Datenobjekt aufgezeichnet.
- (2) Bei einer erstmaligen (vermutlich unbeabsichtigten) Verletzung einer Zugriffsbedingung wird der betreffende Benutzer lediglich über sein Fehlverhalten informiert.
- (3) Bei wiederholter Verletzung einer Zugriffsbedingung besteht Missbrauchsverdacht. Dem Datenbankadministrator wird daher sofort eine Meldung übermittelt. Gegebenenfalls wird die Identifikationskarte des Verdächtigen einbehalten.

Zugriffskontrolle im
System R

Abschließend sei noch auf die Handhabung von Zugriffskontrollen im System R eingegangen. Bekanntlich gestattet das System R jedem Benutzer das Einrichten und Führen privater Relationen. Als Eigentümer privater Relationen kann ein Benutzer anderen Benutzern differenzierte Zugriffsrechte auf seine Relationen einräumen und diese Rechte auch wieder entziehen. Das Zuweisen von Rechten erfolgt mit der GRANT-Anweisung:

GRANT-Anweisung

```
GRANT {ALL RIGHTS | privilegnamen | ALL BUT privilegnamen}
      ON relationsname TO benutzername [WITH GRANT OPTION]
```

Auf das Schlüsselwort GRANT folgt eine geschweifte Klammer mit drei Fallunterscheidungen bezüglich der Vergabe von Rechten bzw. Privilegien (engl. privileges):

- (1) Der Fall ALL RIGHTS drückt die Vergabe sämtlicher möglicher Privilegien aus.
- (2) Sollen nur bestimmte Rechte vergeben werden, so sind diese im Anschluss an das Schlüsselwort GRANT einzeln zu benennen. Vergabe von Privilegien
- (3) Ausnahmen von der Vergabe sämtlicher Rechte sind im Anschluss an die Schlüsselworte GRANT ALL BUT zu benennen.

Nach der Spezifizierung der vergebenen Rechte ist nach dem Schlüsselwort ON die Relation anzugeben, auf die sich diese Rechte beziehen, und nach dem Schlüsselwort TO der privilegierte Benutzer. Rechte sind also für jede private Relation separat zu vergeben. Den Abschluss bildet eine Option, deren Zweck in der Möglichkeit zur Weitergabe von Rechten besteht.

Im System R ist die Gewährung folgender Rechte vorgesehen:

- | | |
|--|-------------------------|
| <p>READ - Recht zum Lesen einer Relation,</p> <p>INSERT - Recht zum Einfügen von Tupeln in eine Relation,</p> <p>DELETE - Recht zum Löschen von Tupeln einer Relation,</p> <p>UPDATE - Recht zum Ändern von Attributwerten von Tupeln einer Relation,
das gegebenenfalls auf bestimmte Attribute begrenzt ist,</p> <p>DROP - Recht zum Löschen einer Relation.</p> | Privilegien im System R |
|--|-------------------------|

An einem Beispiel sei der Gebrauch der GRANT-Anweisung erläutert:

GRANT ALL BUT DELETE, DROP ON Personal TO Hempel

Dem Benutzer *Hempel* werden in Bezug auf die Relation *Personal* die Rechte READ, INSERT und UPDATE eingeräumt.

Mit Hilfe der REVOKE-Anweisung können einem privilegierten Benutzer Rechte wieder entzogen werden:

REVOKE {ALL RIGHTS | privilegenames} ON relationsname
FROM benutzername

REVOKE-Anweisung

Auch hierzu ein Beispiel:

REVOKE ALL RIGHTS ON Personal FROM Hempel

Dem Benutzer *Hempel* werden in Bezug auf die Relation *Personal* sämtliche eingeräumten Privilegien entzogen.

Übungsaufgabe 6.13

Betrachtet werde das obige Beispiel für die Gewährung von Zugriffsrechten. Es lautet:

GRANT ALL BUT DELETE, DROP ON Personal TO Hempel

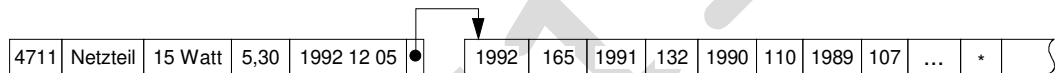
Geben Sie eine alternative Formulierung für diese Anweisung an.

Lösungen zu den Übungsaufgaben

Übungsaufgabe 5.1

Betrachtet werde eine Relation *Artikel*, die Bestandteil der Datenbasis eines Handelsbetriebs ist. Zu jedem Artikel werden auch Daten über den Umsatz der zurückliegenden Jahre festgehalten. Lediglich die Einkaufsabteilung benötigt diese Daten. Sämtliche anderen Benutzer sind an einem beschleunigten Zugriff auf die Artikelsätze interessiert. Folglich bietet es sich hier an, die Umsatzdaten als schlüsselfreie Sätze zu organisieren.

Die folgende Abbildung veranschaulicht die Verkettung eines Artikelsatzes mit den dazugehörigen Umsatzdaten:



Übungsaufgabe 5.2

Durch Unterläufe entstehen Lücken in der jeweiligen Datei. Das Schließen der Lücken ist mit einem beträchtlichen Aufwand verbunden, da Umspeicherungen zwischen Datenblöcken vorzunehmen sind und die durch einen Unterlauf verursachten Umspeicherungen gegebenenfalls mehr als zwei Datenblöcke betreffen können. Lässt man dagegen die Lücken bestehen, um auf diese Weise den Mutationsaufwand gering zu halten, so sind schwach belegte oder gar leere Datenblöcke in Kauf zu nehmen.

Übungsaufgabe 5.3

Die Rückwärtszeiger *RZOrt* ermöglichen das Verarbeiten der für einen Schlüsselwert gebildeten verketteten Liste in Rückwärtsrichtung. So kann man ganz einfach das Ende einer Liste ermitteln, indem man für das erste Element einer Liste den Wert von *RZOrt* liest: Für das erste Element der verketteten Liste für den Sekundärschlüsselwert "Berlin" ist *RZOrt* = 1982, dieser Zeigerwert verweist auf das letzte Element der Liste und ermöglicht so den Einstieg in die Liste von ihrem Ende her. Da nun der Zeiger *RZOrt* des letzten Elements der Liste auf das vorletzte Element verweist, der Zeiger *RZOrt* des vorletzten Elements auf das drittletzte Element verweist usw., kann die Liste Element für Element in Rückwärtsrichtung durchlaufen werden.

Das Abarbeiten einer Liste in Rückwärtsrichtung ist dann sinnvoll, wenn auf Listenelemente zugegriffen werden soll, die sich vermutlich eher in der zweiten Hälfte der Liste befinden als in der ersten Listenhälfte. Bei solchen Elementen kann es sich z. B. um die zuletzt in eine Datei eingefügten Sätze handeln.

Übungsaufgabe 5.4

Zwischen der in der Abb. 3.27 dargestellten Realisierungsform von Beziehungen zwischen Dateien und der in dem Beispiel 5.2 demonstrierten Kettsatztechnik besteht der Zusammenhang der Identität. Die in Abb. 3.27 verwendete Verknüpfungstechnik ist nämlich nichts anderes als die Kettsatztechnik. Die Kettsätze in Abb. 3.27 sind in der Datei "Bestellungen" zusammengefasst. Anders als in Beispiel 5.2 enthält jeder Kettsatz nicht nur Zeiger, sondern zusätzlich noch eine Bestellmenge als Nutzdatum.

Übungsaufgabe 6.1

In einem Fertigungsbetrieb gibt es zu jedem selbst hergestellten Artikel eine Stückliste. Wird nun ein neu zu fertigender Artikel in die Datenbasis aufgenommen, so kann es vorkommen, dass die zugehörige Stückliste noch nicht in der Datenbank enthalten ist. Folglich kann bei der Erfassung des entsprechenden Artikelsatzes der Wert des Attributs *StklisteNr* nicht der Relation *Stkliste* entnommen werden. Dies verhindert die referentielle Integrität, welche fordert, dass in der Sohnrelation *Artikel* nur solche Werte des Attributs *StklisteNr* auftreten, welche in der Vaterrelation *Stkliste* vorhanden sind.

Übungsaufgabe 6.2

Eine primäre Konsistenzbedingung wird durch die Ausführung elementarer Datenbankoperationen nicht verletzt. Sie stellt daher stets eine Zustandsbedingung und keine Übergangsbedingung dar.

Eine sekundäre Konsistenzbedingung ist an Datenbankoperationen gebunden: Es existieren Datenbankoperationen die zur Verletzung der Konsistenzbedingung führen, und solche, die die Konsistenz wiederherstellen. Somit ist eine sekundäre Konsistenzbedingung zugleich auch eine Übergangsbedingung.

Eine starke Konsistenzbedingung kann zugleich auch eine Übergangsbedingung sein. Dies gilt z.B. für die Bedingung, dass der Familienstand eines Mitarbeiters einer Firma nicht von "verheiratet" auf "ledig" geändert werden darf.

Auch eine schwache Konsistenzbedingung kann zugleich eine Übergangsbedingung darstellen. So dürfen Firmenmitarbeiter, die einen Anspruch auf eine Firmenrente haben, nach dem Erreichen der Altersgrenze gegebenenfalls dann aus dem Personalstamm gelöscht werden, wenn sie verzoogen sind und sich ihr Aufenthaltsort und eventuelle Angehörige nicht ermitteln lassen.

Übungsaufgabe 6.3

Eine sekundäre Konsistenzbedingung lässt lediglich temporäre Inkonsistenzen einer Datenbasis zu. Löst eine elementare Datenbankoperation eine Inkonsistenz aus, so ist diese Inkonsistenz durch nachgeschaltete Datenbankoperationen zu beheben. Erst nach

der Durchführung dieser nachgeschalteten Operationen ist es angezeigt, zu prüfen, ob die Konsistenz der Datenbasis tatsächlich wieder hergestellt wurde.

Unter dem Aspekt der Datenbankkonsistenz bilden eine konsistenzverletzende Datenbankoperation und nachgeschaltete, die Konsistenz wiederherstellende Operationen, eine Folge von zusammengehörigen Operationen. Bekanntlich bezeichnet man eine solche Operationsfolge auch als Transaktion.

Übungsaufgabe 6.4

- a) ASSERT Rab ON Kunde:
 Rabatt <= 10
- b) ASSERT Grup ON Kunde:
 IF Jahresumsatz <= 50000 THEN
 Gruppe = 'A'
 ELSE IF (Jahresumsatz >= 50000) AND (Jahresumsatz <= 100000) THEN
 Gruppe = 'B'
 ELSE
 Gruppe = 'C'

Alternativ können die beiden Konsistenzbedingungen auch in einer Konsistenzbedingung zusammengefasst werden:

```
ASSERT RabGrup ON Kunde:
  Rabatt <= 10
AND
  IF Jahresumsatz <= 50000 THEN
    Gruppe = 'A'
  ELSE IF (Jahresumsatz >= 50000) AND (Jahresumsatz <= 100000) THEN
    Gruppe = 'B'
  ELSE
    Gruppe = 'C'
```

Übungsaufgabe 6.5

Das optimistische Verfahren eignet sich stets dann, wenn die Wahrscheinlichkeit für den Eintritt von Konfliktsituationen bei der Durchführung von Transaktionen sehr gering ist. Also eher bei einem Datenbankbetrieb mit geringer Transaktionsdichte. Da im Falle des Eintritts einer Konfliktsituation die betroffenen Transaktionen erneut durchzuführen sind, wäre bei einer hohen Transaktionsdichte nur ein geringer Durchsatz an Transaktionen möglich.

Besteht hingegen eine hohe Wahrscheinlichkeit für das Eintreten von Konfliktsituationen, wie z.B. bei einem Datenbankbetrieb mit hoher Transaktionsdichte, so empfiehlt sich eher der Einsatz eines Sperrverfahrens. Die in Sperrverfahren verwendeten Synchronisationsmechanismen führen auch im Falle einer hohen Transaktionsdichte zu einer

weitgehenden Vermeidung von Konfliktsituationen und ermöglichen somit einen entsprechend hohen Durchsatz.

Übungsaufgabe 6.6

Setzt eine Transaktion eine Schreibsperre auf ein Sperrobjekt, so bedeutet dies, dass das Sperrobjekt geändert werden soll. Damit keine undefinierten Datenbankzustände eintreten, sollen während dieses Änderungsvorgangs keine anderen Transaktionen auf das Sperrobjekt zugreifen. Vielmehr sind Transaktionen, die den alten Wert des Sperrobjekts - als den Wert vor der Änderung - zu verarbeiten haben, vor der hier betrachteten Transaktion abzuwickeln, und Transaktionen, die den neuen Wert des Sperrobjekts - also den Wert nach der Änderung - zu verarbeiten haben, nach der hier betrachteten Transaktion. Genau dies wäre aber nicht sichergestellt, falls Transaktionen auf ein durch eine Schreibsperre belegtes Sperrobjekt weitere Sperren zum Zweck des Lesens oder Schreibens setzen dürften.

Übungsaufgabe 6.7

Die Transaktion *abgverb* setzt erst dann Schreibsperren auf die Relationen *Artikel* und *ArtikelBew*, wenn die Bedingung für die Mutation dieser Relationen auch tatsächlich erfüllt ist. Anders als bei der Transaktion *abgangverbuchen* können daher andere Transaktionen vor dem Setzen der Schreibsperren lesend auf diese Relationen zugreifen. Die Transaktion *abgverb* trägt damit zu einer höheren Parallelität des Datenbankbetriebs bei.

Dem genannten Vorteil steht jedoch auch ein Nachteil gegenüber: Setzen in der Tat mehrere andere Transaktionen ebenfalls Lesesperren auf die Relation *Artikel*, so kann diese Relation von der Transaktion *abgverb* erst dann mit einer Schreibsperre belegt werden, wenn die anderen Transaktionen ihre Lesesperren auf *Artikel* wieder aufgehoben haben. Andere Transaktionen können also die Abwicklung der Transaktion *abgverb* verzögern.

Übungsaufgabe 6.8

Bei beiden angegebenen Grenzfällen werden die Sperrobjekte länger gesperrt als unbedingt erforderlich. Die Folge ist eine Verringerung der Parallelität des Datenbankbetriebs.

Übungsaufgabe 6.9

Zuerst muss der Logfile geschrieben werden. Auf diese Weise wird sichergestellt, dass die alten Werte eines geänderten Datenobjekts im Falle eines Transaktionsfehlers verfügbar sind. Die alten Objektwerte werden für das Zurücksetzen der betroffenen Transaktion benötigt.

Übungsaufgabe 6.10

Bei den Transaktionen der Gruppe (3) ist zwischen Abfragen und Mutationen zu unterscheiden. Da Abfragen die Konsistenz einer Datenbank nicht verletzen, müssen Sie lediglich neu gestartet werden. Dagegen können abgebrochene Mutationen Inkonsistenzen zur Folge haben. Vor Ihrem Neustart sind sie daher zurückzusetzen.

Übungsaufgabe 6.11

Transaktionen welche lediglich wiederholt werden, benutzen die After Image-Einträge aus dem Logfile, um korrekte Werte in die Datenbasis einzutragen.

Neu gestartete Transaktionen müssen dagegen die zu schreibenden Informationen neu ermitteln.

Übungsaufgabe 6.12

Identifikationskarten:

Von Vorteil ist die handhabungsfreundliche Art der Sicherung des Zugangs über eine Magnetkarte. Von Nachteil ist die Möglichkeit der Entwendung oder der Kopie einer solchen Magnetkarte.

Physische Benutzermerkmale:

Die Stimm- und die Fingerabdruckkontrolle verursachen noch weniger Handhabungsaufwand als die Identifikationskarten und sie identifizieren einen Datenbankbenutzer im Normalfall zweifelsfrei. Andererseits sind diese Verfahren sehr aufwendig und damit kostspielig. Außerdem funktioniert die Stimmerkennung schon bei leichter Erkältung nicht mehr einwandfrei.

Passwortverfahren:

Die Passwortverfahren sind die am wenigsten aufwendigen Verfahren der Identitätskontrolle. Allerdings ist bei ihrer Verwendung die Gefahr des Missbrauchs durch Dritte auch am größten. Dem erhöhten Missbrauchsrisiko kann durch einen regelmäßig vorgenommenen Passwortwechsel entgegengewirkt werden.

Übungsaufgabe 6.13

GRANT READ, INSERT, UPDATE ON Personal TO Hempel