



mitp

Gunter
Saake

Kai-Uwe
Sattler

Andreas
Heuer

3. Auflage

Datenbanken

Implementierungstechniken

Aufgaben und Prinzipien von Datenbanksystemen

Datenbanksysteme ermöglichen die integrierte Speicherung von großen Datenbeständen, auf die mehrere Anwendungen gleichzeitig zugreifen können. Hierbei garantiert das Prinzip der *Datenunabhängigkeit* die weitestgehende Unabhängigkeit der Datenrepräsentation von Optimierung und Änderung der Speicherstrukturen. Sie ermöglicht auch eine Reaktion auf Änderungen der Anwendungsanforderungen, ohne die logische Struktur der Daten ändern zu müssen. Diese allgemeinen Anforderungen stellen zusammen genommen hohe Anforderungen an die interne Realisierung von Datenbanksystemen.

Für Datenbanksysteme müssen daher speziell zugeschnittene Algorithmen und Datenstrukturen insbesondere für die folgenden internen Aufgaben entwickelt werden:

- Um eine effiziente Speicherung der Daten und ein schnelles Wiederauffinden zu ermöglichen, werden unterschiedliche, auf große Datenbestände optimierte, interne *Zugriffsdatenstrukturen* eingesetzt.
- Die Datenunabhängigkeit erzwingt, dass Benutzer und Anwendungsprogrammierer die vom Datenbanksystem bereitgestellten Zugriffsdatenstrukturen nicht direkt ausnutzen können – die *Zugriffsoptimierung* muss durch das Datenbanksystem erfolgen.
- Das System muss den Mehrbenutzerbetrieb kontrollieren, um unerwünschte Konflikte beim gleichzeitigen Zugriff auf Daten auszuschließen.

- Weitere Aufgaben umfassen Vorkehrungen zur Wiederherstellung der Daten nach Systemausfällen, Kooperation zwischen verteilten Datenhaltungssystemen und Unterstützung der Wartungsphase.
- Als standardkonforme Systemsoftware müssen Datenbanksysteme auf verschiedenen Rechnerarchitekturen effizient arbeiten.

Die Realisierung dieser Aufgaben in der Implementierung von Datenbanksystemen ist Inhalt dieses Buches. Der Schwerpunkt liegt dabei auf Konzepten kommerzieller, meist relationaler Datenbanksysteme, wobei aber auch weitere zukunftsweisende Entwicklungen sowie Spezialentwicklungen betrachtet werden.

In diesem ersten Kapitel werden wir zunächst die notwendigen, grundlegenden Datenbankkonzepte wiederholen. Die Wiederholung orientiert sich an der Darstellung im ersten Band dieser Buchreihe (dem „Biber-Buch“ [SSH10]). Insbesondere werden für die weiteren Kapitel des Buches Grundkenntnisse der theoretischen Grundlagen von Datenbanksystemen, insbesondere der Relationalenalgebra, und von Datenbanksprachen vorausgesetzt. Speziell werden Basiskenntnisse von SQL erwartet.

Eine detaillierte Kapitelübersicht des Buches findet sich in Abschnitt 1.2. Ferner werden wir eine weitgehend durchgängig verwendete Beispielanwendung vorstellen.

1.1 Wiederholung der Datenbank-Grundbegriffe

Dieser Abschnitt wiederholt wichtige Grundkenntnisse über Datenbanksysteme, die zum Lesen dieses Buches vorausgesetzt werden. Sollte die Darstellung in diesem Kapitel zu knapp sein oder sollten Verständnisschwierigkeiten bei den hier gegebenen Erläuterungen auftreten, so empfehlen wir zunächst das Studium der einschlägigen Kapitel des Biber-Buches [SSH10].

In diesem Abschnitt werden die Bereiche *Architekturen von Datenbanksystemen*, *Datenmodelle und Datendefinition* sowie *Anfragen und Änderungsoperationen* kurz angesprochen.

1.1.1 Architektur eines Datenbanksystems

Die Abbildung 1.1 zeigt einen Überblick über die prinzipielle Aufteilung eines Datenbankmanagementsystems in Funktionsmodule. Die Darstellung ist angelehnt an eine Aufteilung in drei Abstraktionsebenen. Die *externe Ebene* beschreibt die Sicht, die eine konkrete Anwendung auf die gespeicherten Daten hat. Da mehrere angepasste externe Sichten auf eine Datenbank existieren können, gibt die *konzeptuelle Ebene* eine logische und einheitliche Gesamtsicht

auf den Datenbestand. Die *interne Ebene* beschreibt die tatsächliche interne Realisierung der Datenspeicherung.

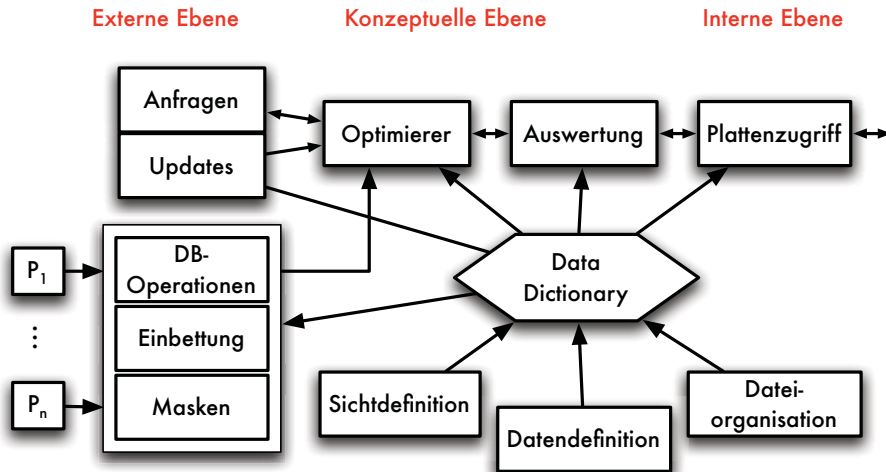


Abbildung 1.1: Vereinfachte Architektur eines DBMS

Die in Abbildung 1.1 gezeigten Komponenten können wie folgt kurz charakterisiert werden:

- Die Komponente *Dateiorganisation* beinhaltet die Definition der Dateiorganisation und Zugriffspfade auf der internen Ebene.
- Die Komponente *Datendefinition* betrifft die konzeptuelle Datendefinition, das heißt die Definition des *konzeptuellen Schemas*.
- In der Komponente *Sichtdefinition* erfolgt die Definition von Benutzersichten, also die Deklaration der Datendarstellung auf der externen Ebene.
- Die Komponente zur Definition von *Masken* beinhaltet den Entwurf von Menüs und Masken für die Benutzerinteraktion.
- Die Komponente *Einbettung* von Konstrukten der Datenbanksprache in eine Programmiersprache bildet die Schnittstelle zur Anwendungsprogrammierung (vgl. [SSH10, Kapitel 13]).
- Die Komponenten zur Bearbeitung von *Anfragen* und *Änderungen (Updates)* ermöglichen einen interaktiven Zugriff auf den Datenbestand.
- Die Komponente *Datenbankoperationen* (kurz *DB-Operationen*) realisiert die Datenbankoperationen für Anfragen und Änderungen, die von Anwendungen genutzt werden.

- Der Komponente *Optimierer* übernimmt die Optimierung der Datenbankzugriffe.
- Die Komponente des *Plattenzugriffs* realisiert die Plattenzugriffssteuerung.
- Die Komponente *Auswertung* betrifft die Auswertung der Ergebnisse von Anfragen und Änderungen.
- $P_1 \dots P_n$ sind verschiedene Datenbankanwendungsprogramme.
- Das *Data Dictionary* (oft auch Katalog oder Datenwörterbuch genannt) bildet den zentralen Katalog aller für die Datenhaltung relevanten Informationen.

Die verschiedenen Komponenten eines Datenbanksystems können dabei zu folgenden Klassen zusammengefasst werden (siehe Abbildung 1.2):

- Die *Definitionskomponenten* bieten Datenbank-, System- und Anwendungsadministratoren die Möglichkeit zur Datendefinition, Definition der Dateiorganisationsformen und Zugriffspfade sowie zur Sichtdefinition.
- Die *Programmierkomponenten* beinhalten eine vollständige Entwicklungsumgebung in einer höheren Programmiersprache, einer 4GL oder einer grafischen Sprache, die Datenbankoperationen und in den meisten Fällen auch Werkzeuge zur Definition von Menüs, Masken und andere Primitiven einer grafischen Benutzeroberfläche einbettet.
- Die *Benutzerkomponenten* umfassen die interaktiven Anfrage- und Änderungswerkzeuge für anspruchsvolle Laien und die vorgefertigten Datenbankanwendungsprogramme für den unbedarften Benutzer („parametric user“).
- Die *Transformationskomponenten* wandeln Anfrage- und Änderungsoperationen schrittweise über Optimierung und Auswertung in Plattenzugriffsoperationen um. Umgekehrt werden die in Blöcken der Platte organisierten Bytes außerdem in die externe Benutzerdarstellung (im Relationenmodell: Tabellen) transformiert.
- Der zentrale Kern des ganzen Systems ist das *Data Dictionary*, das die Daten aus den Definitionskomponenten aufnimmt und die Programmier-, Benutzer- und Transformationskomponenten mit diesen Informationen versorgt.

Gerade die Transformationskomponenten sind in der Drei-Ebenen-Architektur noch etwas ungenau beschrieben. Die im nächsten Kapitel beschriebene Fünfschichten-Architektur wird die schrittweise Transformation von Operationen und Daten genauer darlegen.

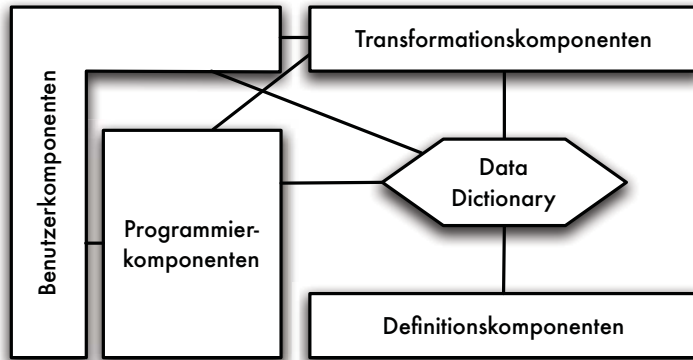


Abbildung 1.2: Klassifikation der Komponenten eines DBMS aus Abbildung 1.1

1.1.2 Neun Funktionen nach Codd

Im Laufe der Jahre hat sich eine Basisfunktionalität herauskristallisiert, die von einem Datenbankmanagementsystem erwartet wird. Codd hat 1982 diese Anforderungen in einer Liste von neun Punkten zusammengefasst [Cod82]:

1. *Integration*

Die Datenintegration erfordert die *einheitliche* Verwaltung *aller* von Anwendungen benötigten Daten. Hier verbirgt sich die Möglichkeit der kontrollierten nicht-redundanten Datenhaltung des gesamten relevanten Datenbestands.

2. *Operationen*

Auf der Datenbank müssen Operationen möglich sein, die Datenspeicherung, Suchen und Änderungen des Datenbestands ermöglichen.

3. *Katalog*

Der Katalog, auch „Data Dictionary“ genannt, ermöglicht Zugriffe auf die Datenbeschreibungen der Datenbank.

4. *Benutzersichten*

Für unterschiedliche Anwendungen sind unterschiedliche Sichten auf den Datenbestand notwendig, sei es in der Auswahl relevanter Daten oder in einer angepassten Strukturierung des Datenbestands. Die Abbildung dieser speziellen Sichten auf den Gesamtdatenbestand muss vom System kontrolliert werden.

5. *Konsistenzüberwachung*

Die Konsistenzüberwachung, auch als *Integritätssicherung* bekannt, übernimmt die Gewährleistung der Korrektheit von Datenbankinhalten und der korrekten Ausführung von Änderungen, so dass diese die Konsistenz nicht verletzen können.

6. *Datenschutz*

Aufgabe des Datenschutzes ist der Ausschluss unautorisierter Zugriffe auf die gespeicherten Daten. Dies umfasst datenschutzrechtlich relevante Aspekte personenbezogener Informationen ebenso wie den Schutz firmenspezifischer Datenbestände vor Werksspionage.

7. *Transaktionen*

Unter einer Transaktion versteht man eine Zusammenfassung von Datenbankoperationen zu Funktionseinheiten, die als Ganzes ausgeführt werden sollen und deren Effekt bei Erfolg permanent in der Datenbank gespeichert werden soll.

8. *Synchronisation*

Konkurrierende Transaktionen mehrerer Benutzer müssen synchronisiert werden, um gegenseitige Beeinflussungen, etwa versehentliche Schreibkonflikte auf gemeinsam benötigten Datenbeständen, zu vermeiden.

9. *Datensicherung*

Aufgabe der Datensicherung ist es, die Wiederherstellung von Daten etwa nach Systemfehlern zu ermöglichen.

Die Punkte 1, 2, 5 und 6 werden schwerpunktmäßig in [HS00, SSH10] behandelt, während die anderen Punkte Inhalt dieses Buches sind.

1.1.3 Datenbankmodelle und Datendefinition

Es gibt verschiedene Datenbankmodelle, die für die Datenbeschreibung auf der konzeptuellen Ebene eingesetzt werden. Kommerziell am erfolgreichsten sind dabei zurzeit ohne Zweifel die relationalen Datenbanksysteme bzw. deren Erweiterung um objektrelationale Konzepte.

Wir werden in diesem Buch daher primär die Implementierung von relationalen Datenbanksystemen als Motivation für die vorgestellten Techniken verwenden. Natürlich können diese Techniken (teils in abgewandelter Form) auch für andere Datenbankmodelle eingesetzt werden.

Konzeptionell ist eine relationale Datenbank eine Ansammlung von *Tabellen*. Hinter den Tabellen steht mathematisch die Idee einer *Relation*, ein grundlegender Begriff, der dem Ansatz den Namen gegeben hat.

Die folgenden zwei Tabellen sollen die Produkt- und Lieferantendaten eines Kaffeehändlers repräsentieren.

| PRODUKT | ProdNr | Bezeichnung | Preis | LName |
|---------|--------|-------------------|-------|--------------|
| | 1042 | Jamaica Blue | 14,90 | CoffeeShop |
| | 1043 | Arabica Black | 10,90 | Kaffeebude |
| | 1044 | New York Espresso | 18,20 | Kaffeebude |
| | 1045 | Arabica Black | 12,00 | CoffeeDealer |

| LIEFERANT | LName | Adresse |
|-----------|--------------|-----------|
| | CoffeeShop | München |
| | Kaffeebude | Berlin |
| | CoffeeDealer | Magdeburg |

Wir verwenden in diesem Abschnitt die folgenden begrifflichen Konventionen: Die erste Zeile gibt jeweils die Struktur einer Tabelle an (Anzahl und Benennung der Spalten). Diese Strukturinformation bezeichnen wir als *Relationenschema* (als Pluralform von Schema verwenden wir *Schemata*). Die weiteren Einträge in der Tabelle bezeichnen wir als *Relation* zu diesem Schema. Eine einzelne Zeile der Tabelle bezeichnen wir als *Tupel*. Spaltenüberschriften werden als *Attribut(namen)* bezeichnet.

Selbst in einem derart einfachen Datenstrukturierungsmodell wie dem relationalen ist es sinnvoll, bestimmte *Konsistenzforderungen* oder *Integritätsbedingungen* an gespeicherte Datenbanken zu stellen, die vom System gewährleistet werden müssen.

Betrachten wir die PRODUKT-Tabelle erneut. Die Einträge für Lieferanten in der LName-Spalte, in der folgenden Tabelle kursiv hervorgehoben, sollten sicher nicht beliebig gewählt werden dürfen.

| PRODUKT | ProdNr | Bezeichnung | Preis | LName |
|---------|--------|-------------------|-------|---------------------|
| | 1042 | Jamaica Blue | 14,90 | <i>CoffeeShop</i> |
| | 1043 | Arabica Black | 10,90 | <i>Kaffeebude</i> |
| | 1044 | New York Espresso | 18,20 | <i>Kaffeebude</i> |
| | 1045 | Arabica Black | 12,00 | <i>CoffeeDealer</i> |

Von jedem LName-Eintrag erwarten wir, dass er tatsächlich auf einen Lieferanteneintrag in der LIEFERANT-Tabelle verweist. Dies ist aber nur möglich, wenn diese Lieferantennamen eindeutig je eine Zeile identifizieren. Wir bezeichnen diese Eigenschaft als *Schlüsseigenschaft*.

| LIEFERANT | LName | Adresse |
|-----------|---------------------|-----------|
| | <i>CoffeeShop</i> | München |
| | <i>Kaffeebude</i> | Berlin |
| | <i>CoffeeDealer</i> | Magdeburg |

Derartig einfache Integritätsbedingungen sind im relationalen Datenbankmodell fest integriert. Wir werden darum im Folgenden jeweils Relationenschema *plus* Integritätsbedingungen betrachten.

Unter *lokalen* Integritätsbedingungen verstehen wir Bedingungen, die für genau eine Tabelle gewährleistet sein müssen. Etwa ist das Attribut *ProdNr Schlüssel* für PRODUKT, d.h. eine *ProdNr* darf nicht doppelt vergeben werden oder anders ausgedrückt: In der Spalte *ProdNr* dürfen keine zwei gleichen Werte auftauchen.

Unter *globalen* Integritätsbedingungen verstehen wir Bedingungen, die über den Bereich einer Tabelle hinausreichen. Wir sagen, dass *LName* in der Tabelle PRODUKT ein *Fremdschlüssel* bezüglich LIEFERANT ist. Dies bedeutet, dass *LName* in einem anderen Relationenschema als Schlüssel auftaucht und die Lieferantennamen in PRODUKT auch in LIEFERANT auftreten müssen. Es sei noch angemerkt, dass es in einer realen Anwendung besser wäre, künstliche Lieferantennummern als Schlüssel zu verwenden, um bei Namensänderungen nicht Referenzierungsprobleme zu riskieren.

Zur Umsetzung der entworfenen Relationenschemata mit ihren Integritätsbedingungen in das Data Dictionary eines Datenbanksystems verwenden wir den Teil der Sprache SQL, der die *Datendefinition* ermöglicht und daher auch als DDL (Data Definition Language) bezeichnet wird.

Die folgende einfache SQL-Deklaration zeigt den prinzipiellen Aufbau von Tabellendeklarationen. Neben einem eindeutigen *Tabellennamen* werden insbesondere die Attribute mit ihren Wertebereichen aufgeführt.

```
create table PRODUKT (  
    ProdNr int not null,  
    Bezeichnung varchar(200),  
    Preis decimal(5,2),  
    LName varchar(30) )
```

Bereits in diesem einfachen Beispiel ist mit der Angabe **not null** eine einfache *Integritätsbedingung* angegeben, die für das Attribut *ProdNr* definierte Werte erzwingt. Allgemein unterstützt SQL eine Reihe von Möglichkeiten, Integritätsbedingungen etwa mit der **check**-Klausel anzugeben.

Für die interne Realisierung, zum Beispiel für die Definition von tabellenübergreifenden *Clustern*, sind Angaben für die Definition von *Primärschlüsseln* und *Fremdschlüsselbedingungen* besonders wichtig. Das folgende erweiterte Beispiel versieht die PRODUKT-Tabelle mit einem Primärschlüssel, dem Attribut *ProdNr*, und definiert eine Fremdschlüsselbeziehung zur Tabelle LIEFERANT über das Attribut *LName*.

```
create table PRODUKT (  
    ProdNr int,  
    Bezeichnung varchar(200),
```

```

Preis decimal(5,2),
LName varchar(30),
primary key (ProdNr),
foreign key (LName) references LIEFERANT (LName))

```

SQL unterstützt mit der **alter**-Anweisung auch eine einfache Form der *Schemaevolution*, auf die wir allerdings nicht weiter eingehen werden.

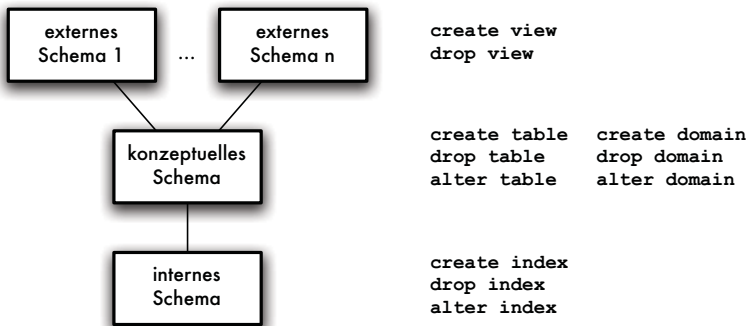


Abbildung 1.3: DDL-Anweisungen in der Drei-Ebenen-Schemaarchitektur

Abbildung 1.3 ordnet zusammenfassend die wichtigsten DDL-Anweisungen der Drei-Ebenen-Schemaarchitektur zu.

1.1.4 Anfrage- und Änderungsoperationen

Anfrage- und Änderungsoperationen sind die Grundlage für die gesamte Verwaltung von Datenbanken. Formale Grundlagen für die Anfrageoperationen sind

- die Relationenalgebra sowie
- der Tupel- oder Bereichskalkül.

Da wir in diesem Buch vor allem die Relationenalgebra als interne Darstellung von Anfragen benötigen, stellen wir noch einmal kurz deren wichtigste Anfrageoperationen Selektion, Projektion und Verbund vor.

Die *Selektion* ermöglicht es, Zeilen einer Tabelle auszuwählen. Hierbei kann ein einfaches Prädikat¹ über die Tupelwerte der zu selektierenden Zeilen angegeben werden. Die Selektion wird im Folgenden mit dem griechischen

¹Prädikate sind Bedingungen, die bei einer Auswertung die Wahrheitswerte **true** oder **false** als Ergebnis liefern.

Buchstaben σ notiert, wobei die Selektionsbedingung unten rechts am Operatorensymbol notiert wird. Ein Beispiel ist die folgende Anfrage:

$$\sigma_{\text{LName}='Kaffeebude'}(r(\text{PRODUKT}))$$

Mit der Notation $r(\text{PRODUKT})$ wird die zum Relationenschema PRODUKT gehörende, in der Datenbank gespeicherte Relation bezeichnet. Die Anfrage liefert als Ergebnis die folgende Tabelle:

| ProdNr | Bezeichnung | Preis | LName |
|--------|-------------------|-------|------------|
| 1043 | Arabica Black | 10,90 | Kaffeebude |
| 1044 | New York Espresso | 18,20 | Kaffeebude |

Während die Selektion Zeilen selektiert, werden mittels der *Projektion* Spalten ausgewählt. Die Projektion wird analog zur Selektion mit π notiert:

$$\pi_{\text{LName}}(r(\text{PRODUKT}))$$

Zur Auswahl von Spalten müssen die Attributnamen angegeben werden. Das Ergebnis dieser Anfrage ist die folgende Tabelle:

| LName |
|--------------|
| CoffeeShop |
| Kaffeebude |
| CoffeeDealer |

Wie man am Ergebnis sieht, werden bei der Projektion doppelte Tupel entfernt. Dies ist die Folge der Interpretation von Tabellen als mathematische Relationen, also als (duplikatfreie) Mengen von Tupeln.

Wir benötigen nun noch eine Operation, um zwei Tabellen miteinander zu verschmelzen. Der *Verbund* (engl. *join*) verknüpft Tabellen über gleich benannte Spalten, indem er jeweils zwei Tupel verschmilzt, falls sie dort gleiche Werte aufweisen. Er wird mit dem Symbol \bowtie notiert.

$$\pi_{\text{Bezeichnung, LName, Adresse}}(\sigma_{\text{Preis} < 15}(r(\text{PRODUKT}))) \bowtie r(\text{LIEFERANT})$$

Das Ergebnis einer Verbundoperation ist eine Tabelle, die als Schema die Vereinigung der Spaltennamen der Eingangsrelationen erhält. Die Tupel der Eingangsrelationen werden immer dann zu einem neuen Tupel verschmolzen, wenn sie in den gemeinsamen Attributen in den Werten übereinstimmen. Die obige Anfrage führt zu folgendem Ergebnis:

| Bezeichnung | LName | Adresse |
|---------------|--------------|-----------|
| Jamaica Blue | CoffeeShop | München |
| Arabica Black | Kaffeebude | Berlin |
| Arabica Black | CoffeeDealer | Magdeburg |

Auf Tabellen können weitere sinnvolle Operationen definiert werden, etwa Vereinigung, Differenz, Durchschnitt, Umbenennung von Spalten etc. Alle Operationen sind beliebig kombinierbar und bilden somit eine „Algebra“ zum „Rechnen mit Tabellen“, die so genannte *relationale Algebra* oder auch *Relationenalgebra*.

Die *Änderungskomponente* eines Datenbanksystems ermöglicht es,

- Tupel einzugeben,
- Tupel zu löschen und
- Tupel zu ändern.

Lokale und globale Integritätsbedingungen müssen bei Änderungsoperationen automatisch vom System überprüft werden.

1.1.5 Sprachen und Sichten

Als *Anfragesprache* wird oft eine Sprache zur Verfügung gestellt, die es erlaubt, aus vorhandenen Tabellen neue zu „berechnen“, die eine Antwort auf eine Fragestellung geben. Relationale Datenbanksysteme bieten eine interaktive Möglichkeit an, Datenbankabfragen zu formulieren und zu starten. Heutzutage ist die Sprache in der Regel ein Dialekt der in [SSH10] ausführlich vorgestellten Sprache SQL. SQL umfasst grob gesagt die Ausdrucksfähigkeit der Relationenalgebra und zusätzlich Funktionen (**sum**, **max**, **min**, **count** ...) zum Aggregieren von Werten einer Tabelle sowie einfache arithmetische Operationen. Die Umsetzung unserer Verbundanfrage in die SQL-Notation ergibt die folgende Anfrage:

```
select Bezeichnung, PRODUKT.LName, Adresse
from PRODUKT, LIEFERANT
where Preis < 15 and PRODUKT.LName = LIEFERANT.LName
```

Alternativ dazu existieren oft grafisch „verpackte“ Anfragemöglichkeiten für den gelegentlichen Benutzer (QBE). In diesem Buch wird allerdings ausschließlich SQL als Datenbanksprache verwendet.

In SQL wird ein Operator zum Einfügen von Tupeln in Basisrelationen, bezeichnet als **insert**, sowie Operatoren zum Löschen von Tupeln (**delete**) zum Ändern von Attributwerten (**update**) angeboten. Diese Operationen können jeweils als Ein-Tupel-Operationen (etwa die Erfassung eines neuen Kunden) oder als Mehr-Tupel-Operationen (reduziere die Preise aller Produkte um 10%) eingesetzt werden.

Die folgende Attributwertänderung ist ein Beispiel für eine Mehr-Tupel-Operation, da der Preis von mehreren Produkten auf einmal geändert wird:

```
update PRODUKT
  set Preis = Preis * 0.9
  where Preis < 10
```

Ein-Tupel-Operationen sind typisch für Einfügungen, wie das folgende Beispiel zeigt:

```
insert into PRODUKT
  values (1050, 'Espresso Roma', 14.90, 'CoffeeDealer')
```

Bei Löschungen und Attributänderungen müssen sie durch Selektionen über Schlüsselattribute erzwungen werden. Allerdings existiert auch für Einfügungen eine Mehr-Tupel-Variante, bei der die einzufügenden Tupel aus dem Ergebnis einer Anfrage kommen:

```
insert into KUNDE (
  select ...from ...)
```

Neben der interaktiven Nutzung wird SQL häufig auch in Verbindung mit Programmiersprachen verwendet, um die Entwicklung von Datenbankanwendungsprogrammen zu ermöglichen. Neben einfachen Call-Schnittstellen wie JDBC oder ODBC sowie der Einbettung in eine Wirtssprache (Embedded SQL) existieren dafür auch spezielle Spracherweiterungen wie zum Beispiel LINQ und Techniken zur Abbildung von Anwendungsobjekten auf Relationen. Eine ausführliche Vorstellung der wichtigsten Techniken ist unter anderem in [SSH10] zu finden.

1.2 Wann kommt was?

Die folgenden Kapitel vertiefen die Darstellung der Transformationskomponenten und der internen Ebene aus den Abbildungen 1.1 und 1.2 im Detail. Das anschließende Kapitel 2 betrachtet die Architektur von Datenbanksystemen näher. Hierzu wird ein *Schichtenmodell* eines Datenbanksystems eingeführt und die wichtigsten Objekte und Operationen dieser Schichten diskutiert.

Die in diesem Buch noch detaillierter behandelten Komponenten eines Datenbanksystems werden wir nun überblicksartig einführen und dazu erwähnen, in welchem Kapitel des Buches diese Techniken näher erläutert werden. Zu den behandelten Komponenten gehören der *Optimierer*, die *Dateiorganisationen und Zugriffspfade*, die *Organisation des Sekundärspeichers*, die *Transaktionsverwaltung* und die *Recovery-Komponente*.

1.2.1 Optimierer

Eine wichtige Komponente eines DBMS ist der *Optimierer*, da Anfragen unabhängig von der internen Detailrealisierung der Datenstrukturen formuliert werden sollen. Das DBMS muss Anfragen selbst optimieren, um eine effiziente Ausführung zu ermöglichen.

Die Basis für die Optimierung von Anfragen ist die *Auswertung* von einzelnen Anfrageoperationen. Die Optimierung wie auch die Auswertung von Operationen sind die ersten beiden Transformationskomponenten aus Abbildung 1.1.

Auswertung von Anfragen

Um die Auswertung von Anfragen durchzuführen, müssen einige Basisoperationen auf den verwendeten Datenstrukturen implementiert werden. Im Relationenmodell sind die notwendigen Operationen durch die Operatoren der relationalen Algebra vorgegeben, auch wenn die tatsächliche Anfragesprache, etwa SQL, anderen Konzepten folgt. Kritische Operationen sind insbesondere die Selektion, die Projektion und der Verbund.

Die Implementierung muss derart erfolgen, dass die jeweilige Operation auf interner Ebene (interne Datensätze und Zugriffspfade) effizient ausführbar ist und eine interne Optimierung durch Auswahl geeigneter Zugriffspfade für den aktuellen Zugriff erfolgen kann.

Wo kommt das im Buch?

Kapitel 7 behandelt *Basisalgorithmen für Datenbankoperationen* und deren Komplexität. Von besonderer Wichtigkeit sind hierbei Algorithmen für die Berechnung von Verbunden und deren Aufwandsabschätzung.

Optimierung von Anfragen

Ziel der Optimierung von Anfragen ist normalerweise eine möglichst schnelle Anfragebearbeitung. Dies kann erreicht werden, indem möglichst wenige Seitenzugriffe bei der Anfragebearbeitung benötigt werden und in allen Operationen so wenig Seiten (Tupel) wie möglich zu berücksichtigen sind. Ein offensichtliches Teilziel muss dabei sein, Zwischenergebnisse klein zu halten.

Wie kann man diese Ziele erreichen? Eine anzuwendende Heuristik ist es, Selektionen so früh wie möglich auszuführen, um kleine Zwischenergebnisse zu haben. Auch können etwa Selektion und Projektion im Sinne des Pipelining zusammengefasst werden, um in einem Verarbeitungsschritt ausgeführt werden zu können. Allgemein können redundante Operationen, Idempotenzen sowie Operationen, die leere Relationen erzeugen, entfernt werden. Gleiche Unteranfragen können identifiziert werden, so dass sie nur einmal ausgeführt werden müssen. Diese intuitiv einsichtigen Vorgehensweisen müssen nun in die Implementierung eines Optimierers einfließen.

Allgemein wird der Optimierungsprozess in zwei Phasen aufgeteilt, die *logische Optimierung* und die *interne Optimierung*.

- Die *logische Optimierung* nutzt nur algebraische Eigenschaften der Operationen, also *keine* Informationen über die realisierten Speicherstrukturen und Zugriffspfade. Eine typische Operation ist die Entfernung redundanter Operationen (etwa beim Verbund). Für eingeschränkte Aufgaben kann eine exakte Optimierung vorgenommen werden, etwa die Bestimmung der minimalen Anzahl von Verbunden mittels der *Tableau-Technik* [Mai83, Ull88, Ull89].

In der Regel werden statt exakter Optimierung heuristische Regeln eingesetzt, etwa beim Verschieben von Operationen derart, dass Selektionen möglichst früh ausgeführt werden. Diese heuristische Optimierung war oben bereits als *algebraische Optimierung* eingeführt worden. Weitere Regeln werden im Folgenden kurz skizziert.

- Die *interne Optimierung* nutzt dann Informationen über die vorhandenen Speicherstrukturen aus. Etwa kann bei Verbunden die Reihenfolge der Verbunde nach Größe und Unterstützung der Relationen durch Zugriffspfade festgelegt werden. Bei Selektionen kann die Reihenfolge der Anwendung von Bedingungen nach der Selektivität von Attributen und dem Vorhandensein von Zugriffspfaden optimiert werden. Des Weiteren wird in der internen Optimierung die Implementierungsstrategie einzelner Operationen ausgewählt.

Ein realer Optimierer wendet meist eine Kombination dieser beiden Phasen an. Dazu werden mithilfe der konzeptuellen Optimierung alternative Ausführungspläne aufgestellt. Die interne Optimierung erfolgt dann *kostenbasiert*: Eine Kostenfunktion schätzt den Aufwand jeder einzelnen Operation ab. Die Kosten von Ausführungsplänen müssen dann global minimiert werden.

Wo kommt das im Buch?

Kapitel 8 präsentiert die Grundkonzepte von *Optimierern* für Datenbankanfragen. Die verschiedenen Phasen der Optimierung werden detailliert beschrieben.

1.2.2 Dateioorganisation und Zugriffspfade

Diese Komponente enthält die Mechanismen zur Speicherung von Daten in speziellen *Dateiorganisationen* und das Wiederfinden von Daten durch geeignete Zugriffspfade. Außerdem wird auf der internen Ebene die Komponente *Plattenzugriffsteuerung*, also die Umwandlung von Befehlen, die auf Sätzen von Dateien arbeiten, in Plattenzugriffoperationen der Betriebssystemebene (Zugriff

auf Seiten bzw. Blöcke) sowie die Auswertung der Operationen durch elementare Operationen auf Sätzen von Dateien angeordnet.

Einordnung der Dateiorganisation

Die Aufgabe der Dateiorganisation ist insbesondere die Abbildung der Strukturen auf der konzeptionellen Ebene in Strukturen der internen Ebene und umgekehrt. Abbildung 1.4 verdeutlicht diese Abbildungen.

| Konzeptionelle Ebene | | Interne Ebene | | Platte |
|-----------------------------|---|----------------------|---|---------------|
| Relationen | → | Dateien (Files) | → | |
| Tupel | → | Sätze (Records) | → | Blöcke |
| Attributwerte | → | Felder | → | |

Abbildung 1.4: Abbildungsstufen der Datenorganisation

Die Konstrukte des Datenmodells der konzeptionellen Ebene, also Relationen, müssen auf interne Datenstrukturen abgebildet werden, etwa Felder und Sätze von Dateien. Diese Strukturen wiederum müssen auf Blöcke der Festplatte („Seiten“) abgebildet werden.

Diese Abbildung ist im Allgemeinen nicht bijektiv (also eine 1:1-Zuordnung), da etwa die Tupel einer Relation in verschiedener Reihenfolge abgespeichert werden können, ohne dass dieses auf der konzeptionellen Ebene sichtbar ist. Die Abbildung unterliegt einer Reihe von Einflussfaktoren, genannt seien hier etwa die gewählten Organisationsformen der internen Schemata, Fragen der Speicherung und Adressierung sowie der Codierung der Datentypen etc.

Speicherung und Zugriff auf Datensätze

Betrachten wir speziell das Problem der Speicherung der Tupel einer gegebenen Relation. Natürlich könnten wir die Tupel in willkürlicher Reihenfolge in einer Liste anordnen, aber dies würde uns keinerlei Vorteil bei Anfrageoperationen bringen.

Darum werden die Datensätze oft in Abhängigkeit vom Wert des *Primärschlüssels* in einer Datei gespeichert, um so einen schnelleren Zugriff zumindest über Primärschlüssel zu ermöglichen. Verbreitet sind zwei Alternativen:

- Die Tupel können *geordnet* abgelegt werden, etwa in Verbindung mit einem Suchbaum. Das Einfügen und Suchen erfolgt dann mit logarithmischem Aufwand, und Verbundoperationen über den Primärschlüssel werden beschleunigt.
- Eine *gestreute* Speicherung mittels einer Hash-Funktion führt bei direktem Zugriff über den Primärschlüssel zu konstantem Aufwand, ermöglicht aber keinen Durchlauf in Sortierreihenfolge.

Beide Verfahren erreichen einen schnellen Zugriff über den Primärschlüssel. Die Zugriffsunterstützung für Zugriffe über den Primärschlüssel mit einer eigenen Zugriffsstruktur, die die Datensätze selber organisiert, wird *Primärindex* genannt.

In der Regel ist auch ein schneller Zugriff über andere Attributmengen wünschenswert, die nicht unbedingt der Schlüsseleigenschaft genügen müssen. Dies wird trotzdem oft als Zugriff über *Sekundärschlüssel* bezeichnet.

Zugriffsstrukturen, die den Zugriff über Attributwerte unterstützen, werden als *Index* bezeichnet (als Pluralform von *Index* wird *Indexe* benutzt). Gemäß obiger Aufteilung unterscheiden wir zwischen Primär- und Sekundärindexen.

Die Speicherung der Datensätze mittels Primärindexen erfordert einen hohen Aufwand beim Einfügen und Ändern von Datensätzen einer Relation, da zum Beispiel die sequentielle Ordnung bewahrt bleiben muss. Dies ist insbesondere bei sehr großen Relationen mit einer Reorganisation der Speicherung verbunden. Einige Datenbanksysteme verzichten daher im Normalfall auf Primärindexe, so dass Datensätze einfach „hinten“ an eine Datei anhängt werden können, und bieten als Zugriffsunterstützung nur Sekundärindexe im obigen Sinne an.

Einordnungskriterien für Zugriffsverfahren

Als erstes Kriterium unterscheiden wir den Zugriff über *Primärschlüssel* von dem über *Sekundärschlüssel*, da im Fall eines Sekundärschlüssels eine direkte Suche mit einem Parameterwert mehr als ein Tupel als Ergebnis liefern kann. Dies hat sowohl Einfluss auf die verwendeten Datenstrukturen als auch auf den Ergebnistyp einer Suchanfrage.

Als zweites Kriterium können wir beim Zugriff über eine Attributkombination differenzieren, ob die Zugriffsstruktur flexibel bezüglich der Reihenfolge der Attribute ist. Wir bezeichnen die unterschiedlichen Attribute der Kombination als *Dimensionen* des Suchraums. Folgerichtig unterscheiden wir *eindimensionale* (Zugriff unterstützt für eine feste Feldkombination) und *mehrdimensionale* (Zugriff unterstützt für eine variable Feldkombination) Verfahren.

Wo kommt das im Buch?

Kapitel 3 beschäftigt sich mit der *Verwaltung des Hintergrundspeichers*. Nach der Diskussion der *Speicherhierarchie* werden insbesondere spezielle Speichermedien wie Disk-Arrays vorgestellt. Desweiteren wird die Abbildung der logischen Datenbankobjekte auf Seiten und Blöcke des Speichers vorgestellt.

Im folgenden Kapitel 4 wird die *Pufferverwaltung* eingeführt. Hier geht es um die Bereitstellung der Seiten des Hintergrundspeichers in bestimmten Bereichen des Hauptspeichers. Da Letzterer zwar schnell, aber vom Umfang her sehr beschränkt ist, müssen Strategien entwickelt werden, nur die wichtigsten Seiten länger in diesem Hauptspeicherbereich zu halten.

Das anschließende Kapitel 5 behandelt die Basisverfahren zur Speicherung von Datenbankrelationen. Verschiedene Baumverfahren wie B-Bäume werden ebenso diskutiert wie einfache Hash-Verfahren und Partitionierungsmechanismen.

Aufbauend auf den klassischen Verfahren diskutiert das Kapitel 6 Zugriffsverfahren für spezielle Anwendungen oder Datenbankmodelle. Das Kapitel beginnt mit Varianten von Hashverfahren und stellt mehrdimensionale Speicherungsverfahren vor. Anschließend werden spezielle Datenstrukturen für neue Anwendungsgebiete behandelt, so für Geoinformationssysteme, Multimediadatenbanken und Data-Warehouse-Anwendungen.

1.2.3 Transaktionen

Wir hatten den Begriff der Transaktion in [SSH10] ausschließlich im Zusammenhang mit dem Problem der *Integritätssicherung* betrachtet. In diesem Zusammenhang ist eine Transaktion als *Einheit* der Konsistenzbewahrung bezeichnet worden: Eine Transaktion muss einen (bezüglich den Integritätsbedingungen) konsistenten Zustand in einen ebenfalls konsistenten Zustand überführen.

Allerdings können fehlerhafte Datenbankzustände trotz Garantie semantischer Integrität durch die einzelnen Transaktionen auftreten. Neben Systemfehlern ist hierbei die gegenseitige Beeinflussung von Transaktionen im *Mehrbenutzerbetrieb* zu berücksichtigen.

Der Mehrbenutzerbetrieb ist oft durch den gleichzeitigen Zugriff mehrerer Benutzer auf dieselben Daten gekennzeichnet (bei Zugriff auf verschiedene Daten tritt in der Regel keine Beeinflussung auf). Anders formuliert: Mehrere Programme laufen simultan und greifen auf dieselbe Datenbank zu. Wir haben es hier also mit *nebenläufigen, konkurrierenden Prozessen* zu tun.

Die sogenannte *Ablaufintegrität* der Transaktionen muss durch eine Mehrbenutzersynchronisation abgesichert werden. Diese Absicherung ist als *Concurrency Control* bekannt.

Wo kommt das im Buch?

Kapitel 9 stellt die Modellbildung für Datenbanktransaktionen und deren theoretische Grundlagen vor. Zentral sind hierbei die Serialisierbarkeitsbegriffe. Neuere Entwicklungen wie geschachtelte Transaktionen und die Ausnutzung semantischer Informationen werden in eigenen Abschnitten ausführlich betrachtet. Das folgende Kapitel 10 diskutiert ausgehend von den vorgestellten Transaktionsmodellen konkrete Algorithmen zur Verwaltung und Synchronisation von Transaktionen im Mehrbenutzerbetrieb.

1.2.4 Recovery und Datensicherheit

Ein wichtiger Aspekt von Datenbanksystemen ist die *Wiederherstellung*, das sogenannte *Recovery*, der Datenbestände nach Systemfehlern und ähnlichen Ereignissen, die normalerweise einen Datenverlust zur Folge haben.

Betreffend die Datenspeicherung können wir zwei Arten von Speichern unterscheiden: Der *instabile* Speicher ist der Hauptspeicher sowie besonders genutzte flüchtige Speichermedien wie Cache und Puffer. Gemeinsam ist diesen Medien, dass der Speicherinhalt bei Stromausfall oder Systemabstürzen verloren geht. Der *stabile* Speicher umfasst Speichermedien wie Platte, Band oder CD-ROM.

Beim instabilen Speicher geht der Inhalt durch *Systemfehler* verloren, beim stabilen Speicher kann der Inhalt durch *Mediafehler* verlorengehen.

Typische Maßnahmen gegen Systemfehler sind das Führen von Logbuch und Änderungsjournal, Wiederherstellungsprotokolle, Einsatz eines Schattenspeichers etc.; Maßnahmen gegen Mediafehler sind Erstellung von Backups, Archivierung der Datenbestände, Führen von Spiegelplatten.

Wo kommt das im Buch?

Kapitel 11 beschäftigt sich mit Verfahren zur Wiederherstellung von Datenbeständen nach Fehlersituationen.

1.3 Vertiefende Literatur

Die meisten Lehrbücher über Grundlagen vertiefen die in diesem Abschnitt vorgestellten Grundkonzepte von Datenbanken. Zu nennen sind hier bei deutschsprachigen Werken insbesondere die Bücher von Saake, Sattler und Heuer [SSH10], Kemper und Eickler [KE09], Vossen [Vos94, Vos08] sowie die Übersetzung des Buchs von Elmasri und Navathe [EN09].

Im englischsprachigen Bereich empfehlen wir insbesondere die Bücher von Elmasri und Navathe [EN10], Silberschatz, Korth und Sudarshan [SKS97], Ramakrishnan [Ram98] und Ullman [Ull88, Ull89]. Eine Einführung gibt das Buch von Garcia-Molina, Ullman und Widom [GUW08].

Neben den erwähnten Datenbanklehrbüchern gibt es eine Reihe von Büchern speziell zu SQL. Date und Darwen stellen den SQL-Standard in [DD97] vor. Auch Melton und Simon präsentieren den SQL-Standard [MS02]. Türker gibt in [Tür03] eine umfassende deutschsprachige Einführung zu den aktuellen Standards SQL:1999 und SQL:2003.

Die Grundprinzipien der Anwendungsprogrammierung werden von Neumann in [Neu96] ausführlich diskutiert. Melton behandelt die Erweiterung von

SQL um gespeicherte Prozeduren und operationale Sprachmittel [Mel98]. Saa-ke und Sattler beschreiben in [SS03] verschiedene Techniken der Datenbank-anwendungsentwicklung mit Java.

1.4 Übungen

Dieser Abschnitt enthält einige einführende Aufgaben, die dazu gedacht sind, das Verständnis der Grundbegriffe zu testen. Sollten bei der Beantwortung dieser Fragen Schwierigkeiten auftreten, empfehlen wir zunächst die Lektüre der entsprechenden Kapitel in den erwähnten Lehrbüchern, bevor mit diesem Buch fortgefahren wird.

Übung 1-1 Ein Teil der Beispieltabellen beschreibt eine Bibliotheksdatenbank. Geben Sie für jeden der neun Punkte von Codd ein Beispiel für ein konkretes Problem an, das entstehen könnte, wenn die Datenbank diese Punkte nicht erfüllen würde.

Übung 1-2 Geben Sie für eine einfache SQL-Anfrage (Kombination eines Joins mit Selektionen) je einen unoptimierten (direkt der SQL-Notation folgenden) und einen „geschickteren“ Anfrageplan an. Wie unterscheiden diese sich im Aufwand? Welche Indexe könnten sinnvoll als Anfrageunterstützung eingesetzt werden?

Übung 1-3 Diskutieren Sie die Eignung zweier Ihnen aus dem Informatikgrundwissen bekannter Datenstrukturen (etwa lineare Listen oder Binärbäume) als Speicherstruktur für Relationen. Beachten Sie dabei, dass die Transporteinheit zwischen Hauptspeicher und Platte jeweils ein ganzer Block ist.

Übung 1-4 Geben Sie ein Beispiel für zwei verschränkt ablaufende Banküberweisungen (jeweils erst Kontostand lesen, und danach neuen Kontostand zurückschreiben) an, bei dem aufgrund fehlender Synchronisation ein Geldbetrag verloren geht.

Ergänzen Sie den Ablauf einer Überweisung um explizite Sperr- und Entsperroperationen, die diesen Effekt verhindern.