Kapitel 1

Einführung

1.1 Definition

Ein **Datenbanksystem** (auch *Datenbankverwaltungssystem*, abgekürzt *DBMS = data base management system*) ist ein computergestütztes System, bestehend aus einer Datenbasis zur Beschreibung eines Ausschnitts der Realwelt sowie Programmen zum geregelten Zugriff auf die Datenbasis.

1.2 Motivation

Die separate Abspeicherung von teilweise miteinander in Beziehung stehenden Daten durch verschiedene Anwendungen würde zu schwerwiegenden Problemen führen:

- **Redundanz:**
  Dieselben Informationen werden doppelt gespeichert.
- **Inkonsistenz:**
  Dieselben Informationen werden in unterschiedlichen Versionen gespeichert.
- **Integritätsverletzung:**
  Die Einhaltung komplexer Integritätsbedingungen fällt schwer.
- **Verknüpfungseinschränkung:**
  Logisch verwandte Daten sind schwer zu verknüpfen, wenn sie in isolierten Dateien liegen.
- **Mehrbenutzerprobleme:**
  Gleichzeitiges Editieren derselben Datei führt zu Anomalien (*lost update*).
- **Verlust von Daten:**
  Außer einem kompletten Backup ist kein Recoverymechanismus vorhanden.
- **Sicherheitsprobleme:**
  Abgestufte Zugriffsrechte können nicht implementiert werden.
- **Hohe Entwicklungskosten:**
  Für jedes Anwendungsprogramm müssen die Fragen zur Dateiverwaltung erneut gelöst werden.

Also bietet sich an, mehreren Anwendungen in jeweils angepaßter Weise den Zugriff auf eine gemeinsame Datenbasis mit Hilfe eines Datenbanksystems zu ermöglichen (Abbildung 1.1).

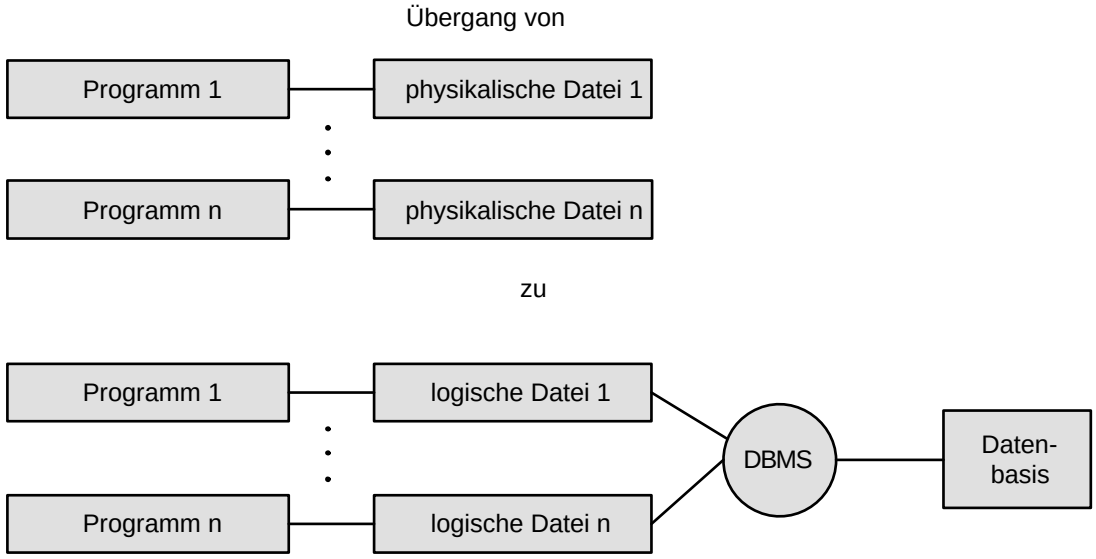

Abbildung 1.1: Isolierte Dateien versus zentrale Datenbasis

1.3 Datenabstraktion

Man unterscheidet drei Abstraktionsebenen im Datenbanksystem (Abbildung 1.2):

- **Konzeptuelle Ebene**
  Hier wird, unabhängig von allen Anwenderprogrammen, die Gesamtheit aller Daten, ihre Strukturierung und ihre Beziehungen untereinander beschrieben. Die Formulierung erfolgt vom *enterprise administrator* mittels einer *DDL (data definition language)*. Das Ergebnis ist das konzeptuelle Schema, auch genannt Datenbankschema.
- **Externe Ebene**
  Hier wird für jede Benutzergruppe eine spezielle anwendungsbezogene Sicht der Daten (*view*) spezifiziert. Die Beschreibung erfolgt durch den *application administrator* mittels einer DDL, der Umgang vom Benutzer erfolgt durch eine *DML (data manipulation language)*. Ergebnis ist das externe Schema.
- **Interne Ebene**
  Hier wird festgelegt, in welcher Form die logisch beschriebenen Daten im Speicher abgelegt werden sollen. Geregelt werden record-Aufbau, Darstellung der Datenbestandteile, Dateiorganisation, Zugriffspfade. Für einen effizienten Entwurf werden statistische Informationen über die Häufigkeit der Zugriffe benötigt. Die Formulierung erfolgt durch den *database administrator*. Ergebnis ist das interne Schema.

Das *Datenbankschema* legt also die Struktur der abspeicherbaren Daten fest und sagt noch nichts über die individuellen Daten aus. Unter der *Datenbankausprägung* versteht man den momentan gültigen

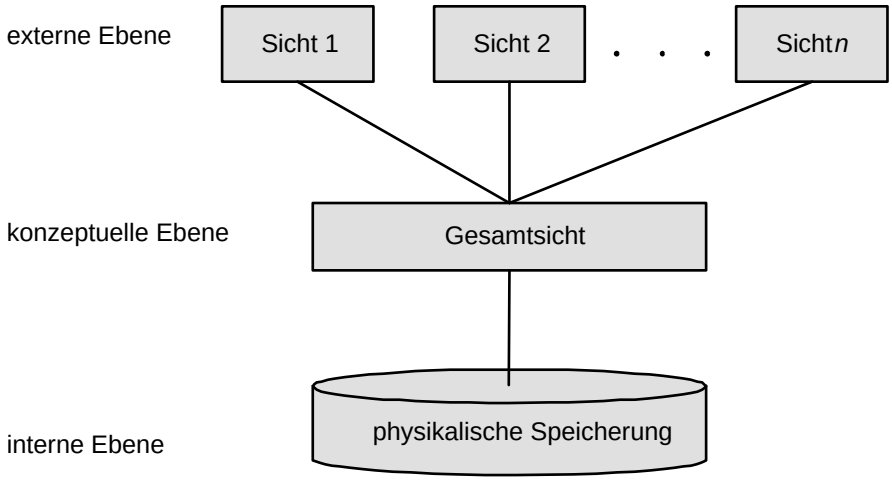

Abbildung 1.2: Drei Abstraktionsebenen eines Datenbanksystems

Zustand der Datenbasis, die natürlich den im Schema festgelegten Strukturbeschreibungen gehorchen muß.

1.4 Transformationsregeln

Die Verbindungen zwischen den drei Ebenen werden durch die *Transformationsregeln* definiert. Sie legen fest, wie die Objekte der verschiedenen Ebenen aufeinander abgebildet werden. Z. B. legt der *Anwendungsadministrator* fest, wie Daten der externen Ebene aus Daten der konzeptuellen Ebene zusammengesetzt werden. Der *Datenbank-Administrator (DBA)* legt fest, wie Daten der konzeptuellen Ebene aus den abgespeicherten Daten der internen Ebene zu rekonstruieren sind.

- **Beispiel Bundesbahn:**
  Die Gesamtheit der Daten (d. h. Streckennetz mit Zugverbindungen) ist beschrieben im konzeptuellen Schema (Kursbuch). Ein externes Schema ist z. B. beschrieben im Heft *Städteverbindungen Osnabrück.*

- **Beispiel Personaldatei:**
  Die konzeptuelle Ebene bestehe aus Angestellten mit ihren Namen, Wohnorten und Geburtsdaten. Das externe Schema `Geburtstagsliste` besteht aus den Komponenten `Name`, `Datum`, `Alter`, wobei das `Datum` aus Tag und Monat des Geburtsdatums besteht, und `Alter` sich aus der Differenz vom laufenden Jahr und Geburtsjahr berechnet.

  Im internen Schema wird festgelegt, daß es eine Datei `PERS` gibt mit je einem record für jeden Angestellten, in der für seinen Wohnort nicht der volle Name, sondern eine Kennziffer gespeichert ist. Eine weitere Datei `ORT` enthält Paare von Kennziffern und Ortsnamen. Diese Speicherorganisation spart Platz, wenn es nur wenige verschiedene Ortsnamen gibt. Sie verlangsamt allerdings den Zugriff auf den Wohnort.

1.5 Datenunabhängigkeit

Die drei Ebenen eines DBMS gewähren einen bestimmten Grad von *Datenunabhängigkeit*:

- **Physische Datenunabhängigkeit:**
  Die Modifikation der physischen Speicherstruktur (z. B. das Anlegen eines Index) verlangt nicht die Änderung der Anwenderprogramme.
- **Logische Datenunabhängigkeit:**
  Die Modifikation der Gesamtsicht (z. B. das Umbenennen von Feldern) verlangt nicht die Änderung der Benutzersichten.

1.6 Modellierungskonzepte

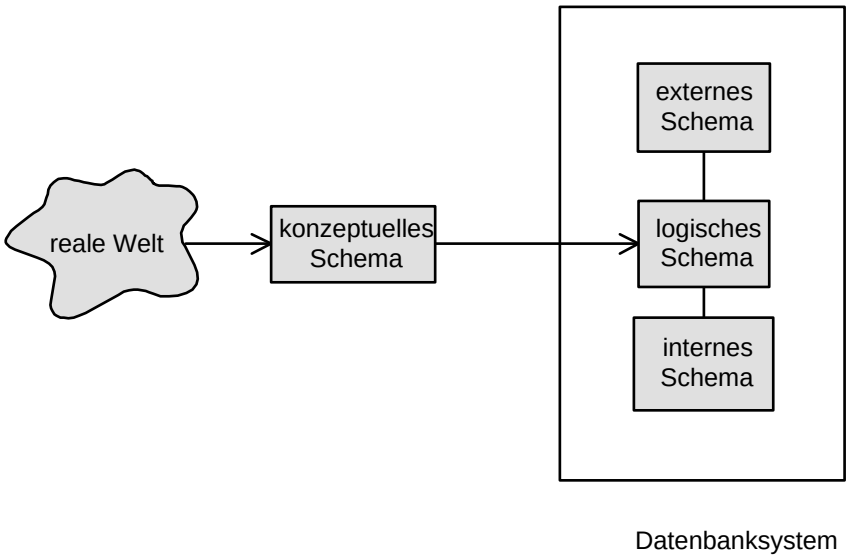

Abbildung 1.3: 2-stufige Modellierung

Das konzeptuelle Schema soll sowohl die reale Welt unabhängig von DV-Gesichtspunkten beschreiben als auch die Grundlage für das interne Schema bilden, welches natürlich stark maschinenabhängig ist. Um diesen Konflikt zu lösen, stellt man dem konzeptuellen Schema ein sogenanntes “logisches” Schema zur Seite, welches die Gesamtheit der Daten zwar hardware-unabhängig, aber doch unter Berücksichtigung von Implementationsgesichtspunkten beschreibt. Das logische Schema heißt darum auch implementiertes konzeptuelles Schema. Es übernimmt die Rolle des konzeptuellen Schemas, das nun nicht mehr Teil des eigentlichen Datenbanksystems ist, sondern etwas daneben steht und z. B. auch aufgestellt werden kann, wenn überhaupt kein Datenbanksystem zum Einsatz kommt (1.3).

Zur Modellierung der konzeptuellen Ebene verwendet man das **Entity-Relationship-Modell**, welches einen Ausschnitt der Realwelt unter Verwendung von *Entities* und *Relationships* beschreibt :

- **Entity:**
  Gegenstand des Denkens und der Anschauung (z. B. eine konkrete Person, ein bestimmter Ort)

- **Relationship:**
  Beziehung zwischen den entities (z. B. wohnen in)

Entities werden charakterisiert durch eine Menge von Attributen, die gewisse Attributwerte annehmen können. Entities, die durch dieselbe Menge von Attributen charakterisiert sind, können zu einer Klasse, einem Entity-Typ, zusammengefaßt werden. Entsprechend entstehen Relationship-Typen.

- **Beispiel:**
  Entity-Typ `Studenten` habe die Attribute `Mat-Nr.`, `Name`, `Hauptfach`.
  Entity-Typ `Orte` habe die Attribute `PLZ`, `Name`.
  Relationship-Typ `wohnen in` setzt `Studenten` und `Orte` in Beziehung zueinander.

Die graphische Darstellung erfolgt durch Entity-Relationship-Diagramme (E-R-Diagramm). Entity-Typen werden durch Rechtecke, Beziehungen durch Rauten und Attribute durch Ovale dargestellt. Abbildung 1.4 zeigt ein Beispiel.

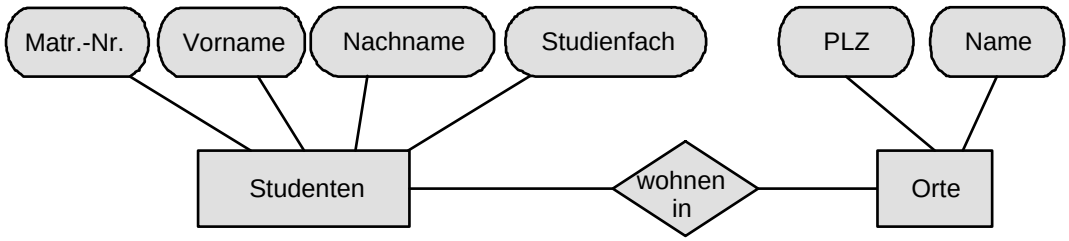

Abbildung 1.4: Beispiel für E-R-Diagramm

Zur Formulierung des logischen Schemas stehen je nach zugrundeliegendem Datenbanksystem folgende Möglichkeiten zur Wahl:

- Das hierarchische Modell (z. B. IMS von IBM)
- Das Netzwerkmodell (z. B. UDS von Siemens)
- Das relationale Modell (z. B. Access von Microsoft)
- Das objektorientierte Modell (z. B. O_2 von O_2 Technology)

Das hierarchische Modell (basierend auf dem Traversieren von Bäumen) und das Netzwerkmodell (basierend auf der Navigation in Graphen) haben heute nur noch historische Bedeutung und verlangen vom Anwender ein vertieftes Verständnis der satzorientierten Speicherstruktur. Relationale Datenbanksysteme (basierend auf der Auswertung von Tabellen) sind inzwischen marktbeherrschend und werden teilweise durch Regel- und Deduktionskomponenten erweitert. Objektorientierte Systeme fassen strukturelle und verhaltensmäßige Komponenten in einem Objekttyp zusammen und gelten als die nächste Generation von Datenbanksystemen.

1.7 Architektur

Abbildung 1.5 zeigt eine vereinfachte Darstellung der Architektur eines Datenbankverwaltungssystems. Im oberen Bereich finden sich vier Benutzerschnittstellen:

- Für häufig zu erledigende und wiederkehrende Aufgaben werden speziell abgestimmte Anwendungsprogramme zur Verfügung gestellt (Beispiel: Flugreservierungssystem).
- Fortgeschrittene Benutzer mit wechselnden Aufgaben formulieren interaktive Anfragen mit einer flexiblen Anfragesprache (wie SQL).
- Anwendungsprogrammierer erstellen komplexe Applikationen durch "Einbettung" von Elementen der Anfragesprache (embedded SQL)
- Der Datenbankadministrator modifiziert das Schema und verwaltet Benutzerkennungen und Zugriffsrechte.

Der DDL-Compiler analysiert die Schemamanipulationen durch den DBA und übersetzt sie in Metadaten.

Der DML-Compiler übersetzt unter Verwendung des externen und konzeptuellen Schemas die Benutzer-Anfrage in eine für den Datenbankmanager verständliche Form. Dieser besorgt die benötigten Teile des internen Schemas und stellt fest, welche physischen Sätze zu lesen sind. Dann fordert er vom Filemanager des Betriebssystems die relevanten Blöcke an und stellt daraus das externe entity zusammen, welches im Anwenderprogramm verarbeitet wird.

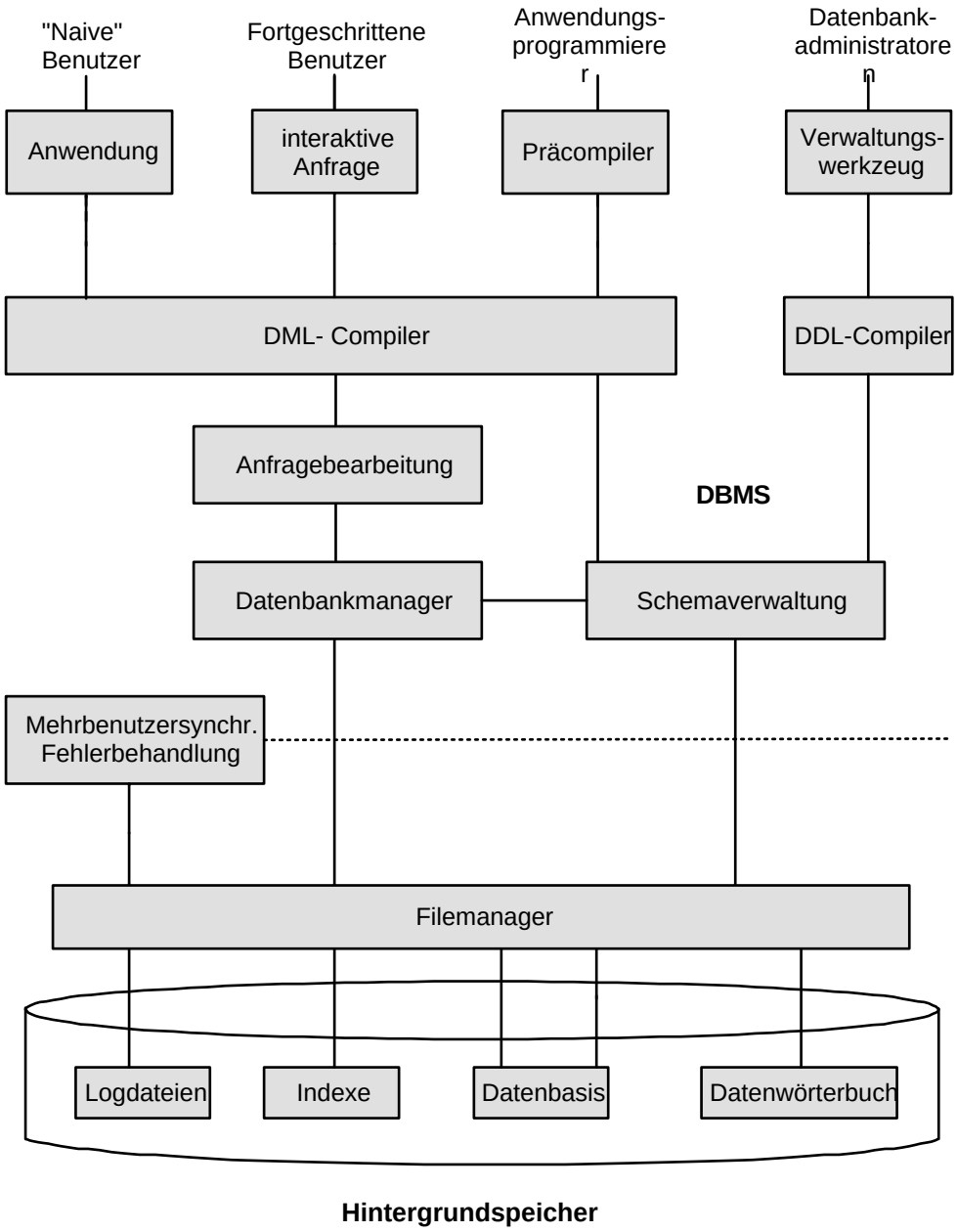

Abbildung 1.5: Architektur eines DBMS