1. Einführung

1.1 Das Konzept des Datenbanksystems

Datenbanken und Datenbanktechnologie spielen heute in praktisch allen Bereichen, in denen Computer überhaupt eingesetzt werden, eine wesentliche Rolle. Dies gilt für verschiedene Anwendungsgebiete wie kommerzielle Aufgaben (Rechnungswesen, Materialwirtschaft, usw.), Administration, Recht, Medizin, Bibliothekswesen, Ingenieurwesen, usw. Wo immer große Datenmengen zu verwalten sind und flexibel auswertbar sein müssen, spielen Datenbanksysteme eine zentrale Rolle. Datenbanksysteme finden sich auf den größten Rechnern ebenso wie auf Personal Computern, sie werden in manchen Anwendungen von Tausenden von Benutzern genutzt, in anderen nur von einem einzigen.

Ganz offensichtlich lässt sich eine scharfe Definition des Begriffes Datenbank gar nicht finden. Folgende Charakterisierung trifft jedoch das Wesentliche:

Begriff der Datenbank

Eine Datenbank ist eine integrierte Ansammlung von Daten, die allen Benutzern eines Anwendungsbereiches als gemeinsame Basis aktueller Information dient.

Das erste Schlüsselwort ist *integriert*. Gemeint ist damit, dass die Daten entsprechend den natürlichen Zusammenhängen in der Anwendungswelt strukturiert sind - und nicht danach, wie einzelne Anwendungen die Daten benötigen. Logische Informationseinheiten sind als solche erkennbar und die zugehörigen Daten nur einmal gespeichert. Logische Informationseinheiten sind beispielsweise die Daten über einen Angestellten, über ein Projekt, usw. Man spricht auch davon, dass eine Datenbank bestimmte Aspekte der Realwelt modelliert.

Das zweite Schlüsselwort ist *gemeinsame Basis:* Die Daten in der Datenbank können durch viele Benutzer genutzt werden. Dies kann sogar gleichzeitig geschehen (paralleler Zugriff), wobei das Datenbanksystem dafür sorgt, dass die verschiedenen Benutzer sich nicht gegenseitig stören. (Im Unterschied zur Integration spielt die gemeinsame Nutzung bei single-user Datenbanken, etwa auf PCs, natürlich keine Rolle.) Obwohl viele Anwendungen auf einer gemeinsamen Basis arbeiten, werden sie unterschiedliche Sichten auf diese Daten haben.

Integration und gemeinsame Nutzung machen eine explizite Beschreibung des Datenbankinhaltes erforderlich. Die Informationseinheiten und die Beziehungen zwischen diesen werden im *Schema der Datenbank* definiert.

Diese charakteristischen Eigenschaften einer Datenbank werden später präziser diskutiert.

Eine Datenbank ist im allgemeinen ein recht komplexes Gebilde, viele Benutzer arbeiten auf ihr (meist sogar gleichzeitig), und sie soll flexibel, d.h. vor allem auch in ungeplanter Weise, abfragbar sein. Anwendungsprogramme greifen deshalb nicht direkt auf die Datenbank zu. Vielmehr liegt die gesamte Kontrolle der Datenbank beim *Datenbankmanagementsystem (DBMS)*. Das DBMS ist ein Softwaresystem, das es ermöglicht, eine Datenbank zu definieren, Daten zu speichern, zu verändern und zu löschen, sowie Anfragen an die Datenbank zu stellen.

DBMS

Ein Beispiel für eine solche Anfrage ist „Gib mir alle Angestellten, die in Dortmund wohnen und in einer Firma außerhalb Dortmunds mit weniger als 1000 Beschäftigten arbeiten“. Es ist Aufgabe des DBMS, diese Anfrage in einzelne Datenbankzugriffe umzusetzen, und zwar so, dass die Anfrage möglichst schnell bearbeitet wird.

Das DBMS isoliert also die Datenbank von den Anwendungsprogrammen, es sorgt dafür, dass der Benutzer (Programmierer) die Details der Datenbank nicht kennen muss.

Datenbank und Datenbanksoftware bilden zusammen das *Datenbanksystem (DBS)*.

Wir hatten bereits gesagt, dass Benutzer ihre jeweils eigene Sicht auf die Datenbank erhalten. Jeder Benutzer sieht also „seinen“ Ausschnitt der Datenbank, und er erhält die Daten in der Form, die er wünscht. Die tatsächliche Organisation der Daten auf den Speichern bleibt für ihn unsichtbar.

Beispiel 1.1:

Die Datenbank enthalte Daten über Kunden und Aufträge. Herr Braun, ein Sachbearbeiter, benötigt folgende Sicht auf die Daten:

KD-NR	AUF-NR	AUF-BESCHR	AUF-DATUM
112	17	Schrauben	14.1.2009
112	20	Nägel	15.4.2009
116	4	...	...
205	48	...	...

Der Manager Weber interessiert sich allein für Auftragsvolumina, er möchte deshalb seine Daten folgendermaßen sehen:

KD-NR	KD-NAME	AUF-VOLUMEN
112	Albert	4000
116	Bernard	1000
205	Müller	400

Wie die Daten tatsächlich in der Datenbank gespeichert sind, sieht keiner dieser Benutzer - es ist für sie auch völlig unerheblich.

In Bild 1.1 ist das Konzept eines Datenbanksystems in seinen wesentlichen Grundzügen zusammengefasst.

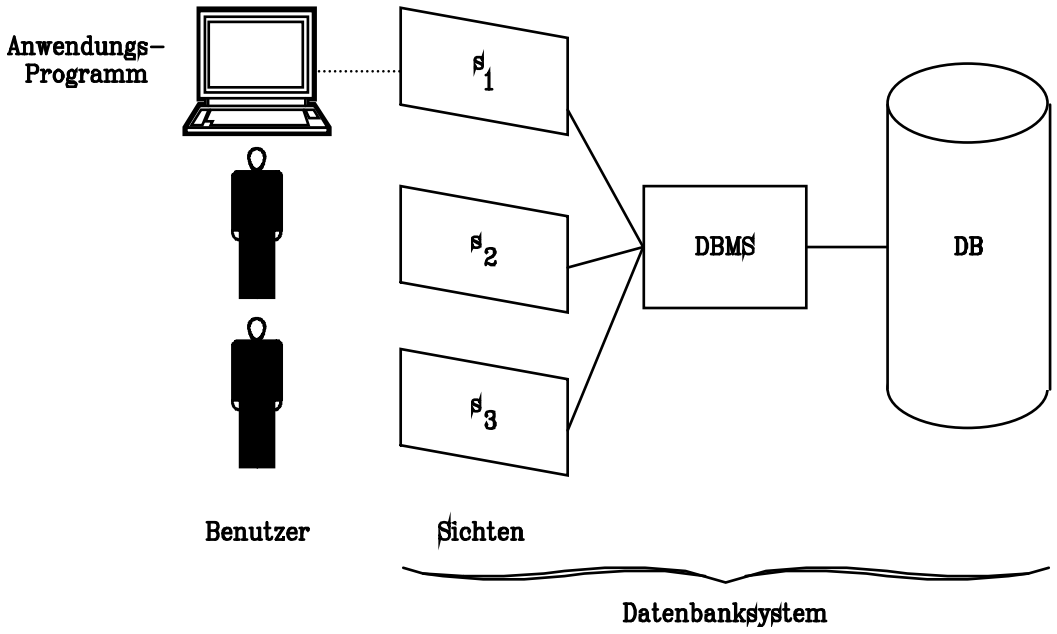


Bild 1.1: Konzept des Datenbanksystems

1.2 Datenbanksysteme und traditionelle Datenverwaltung

Datenbanksysteme wurden nicht am grünen Tisch entworfen, sondern entstanden infolge der ansteigenden Anforderungen an die Verwaltung und Auswertung großer Datenbestände. Um den dramatischen Fortschritt, der für diese Anwendungen mit Datenbanksystemen erreicht wurde, einschätzen zu können, werfen wir einen kurzen Blick auf die konventionelle Form der Verwaltung großer Datenmengen mittels *Dateisystemen* (file systems).

Dateisysteme

Typisch für die traditionelle Dateiverarbeitung ist es, dass jeder Anwendungsprogrammierer die Dateien definiert, die er für seine Anwendung braucht. Betrachten wir ein Beispiel: Im Prüfungsamt der Universität wird eine Datei über Studenten und deren Leistungen verwaltet, im Studentensekretariat wird eine Datei über Studenten und deren Zahlungen geführt. Obwohl beide Abteilungen an Daten über Studenten interessiert sind, benutzen sie separate Dateien, die im Detail auf die jeweilige Anwendung zugeschnitten sind. Auf diese Weise wird, beispielsweise, die Adresse jedes Studenten in zwei voneinander unabhängigen Dateien geführt. Das Ergebnis ist Redundanz in den Daten und ein hoher Aufwand, Änderungen von Daten nachzuhalten. Im Beispiel ist dafür zu sorgen, dass die Änderung einer Adresse tatsächlich in beiden betroffenen Dateien vollzogen wird.

Ein Blick auf Dateisysteme

Dateisysteme kurz gefasst:

Sie sollten den Übergang von der traditionellen Datenverarbeitung mit Dateien zur Datenbankphilosophie unbedingt verstehen. Falls Sie sich unter einer Datei und einem Dateisystem wenig vorstellen können: hier ist eine kurze Zusammenfassung dieser Technik. Dabei benutzen wir eine COBOL-Notation, die in den 70-

er und 80-er Jahren vor der Einführung von relationalen Datenbanken eine wichtige Rolle bei der Datenverarbeitung gespielt hat.

Satz
Datenelement
Feld
Satztyp

Grundelemente einer Datei sind der Satz und das Feld. *Sätze* sind die Zugriffs- und Verarbeitungseinheiten der Anwendungsprogramme. Ein Satz ist eine Gruppierung von *Datenelementen*, auch *Felder* genannt. Sätze vom gleichen Typ werden in einer *Satzbeschreibung* deklariert, d.h. es wird festgehalten, aus welchen Datenelementen sich die Sätze dieses Typs aufbauen.

Beispiel 1.2:

Satztyp	ANGESTELLTER	
Datenelemente	ANGNR	Angestelltennummer
	NAME	Name des Angestellten
	W-ORT	Wohnort
	GEHALT	Gehalt
	PERS	Persönliche Daten; Zahl der Kinder usw.

Datei

Eine *Datei* ist eine benannte Ansammlung von Sätzen eines oder mehrerer Satztypen. Beispielsweise würde eine Datei für die Gehaltsabrechnung je einen Satz vom Typ ANGESTELLTER für jeden Angestellten des Unternehmens enthalten.

Dateisystem
Schlüssel

Ein *Dateisystem* ist ein Softwarepaket, das den Zugriff auf einzelne Sätze in einer Datei besorgt, wenn das Anwendungsprogramm die entsprechenden Parameter liefert. Übliche Dateisysteme unterstützen nur den Zugriff über den *Schlüssel* eines Satzes, wobei der Schlüssel eine von vornherein festgelegte Kombination von Feldern ist, deren Werte den Satz identifizieren. Der Schlüssel für die Angestelltensätze ist z.B. die Angestelltennummer. Ist der Schlüsselwert eines gesuchten Satzes nicht bekannt, so muss die ganze Datei durchsucht werden. Das Dateisystem unterstützt ferner das Anlegen, Modifizieren und Löschen von Dateien.

Dateiorganisation

Die Speicherung der Sätze einer Datei kann unterschiedlich organisiert werden, üblich sind: sequentielle Organisation, index-sequentielle Organisationen, direkte Organisationen (z. B. Hash-Verfahren).

Das Dateisystem isoliert das Anwendungsprogramm in gewissem Umfang von den Hardwareeigenschaften der Speicher, so dass Änderungen in der Hardware möglich sind, ohne dass Anwendungsprogramme betroffen werden; Änderungen des Satzaufbaus oder der Dateiorganisation werden jedoch für alle Programme sichtbar, die mit der Datei arbeiten. Im Anwendungsprogramm muss die Datei, auf die zugegriffen werden soll, exakt deklariert werden. Im Programm zur Gehaltsabrechnung steht also eine Dateideklaration etwa der folgenden Form (COBOL-Notation):

```
FD      ANGESTELLTEN-DATEI
        DATA RECORD IS ANGESTELLTER;  ...  .
01      ANGESTELLTER;  ...  .
        02  ANGNR PIC 9(5);
        02  NAME PIC X(30);
        02  W-ORT PIC X(30);
```

```
02      PERS;  ...      .
02      GEHALT;        ...      .
```

Der Programmierer muss Satzaufbau und Dateiorganisation exakt festlegen, wenn er die Datei neu einrichtet; möchte er bereits bestehende Dateien verwenden, so muss er sich im Detail nach dem Aufbau derselben richten.

Typisch für die konventionelle Datenverarbeitung auf der Basis von Dateisystemen ist es, dass die Dateien in aller Regel für eine Anwendung oder für eng zusammenhängende Anwendungen entworfen werden. Jeder Programmierer baut sich seine Dateien selbst auf, unabhängig und vielleicht sogar ohne Kenntnis der Dateien anderer Programmierer. Der Dateiaufbau ist unmittelbar an die jeweilige Verarbeitung angepasst, und in dieser Form ist die Datei auch abgespeichert. Die grundsätzliche Situation ist in Bild 1.2 zusammengefasst.

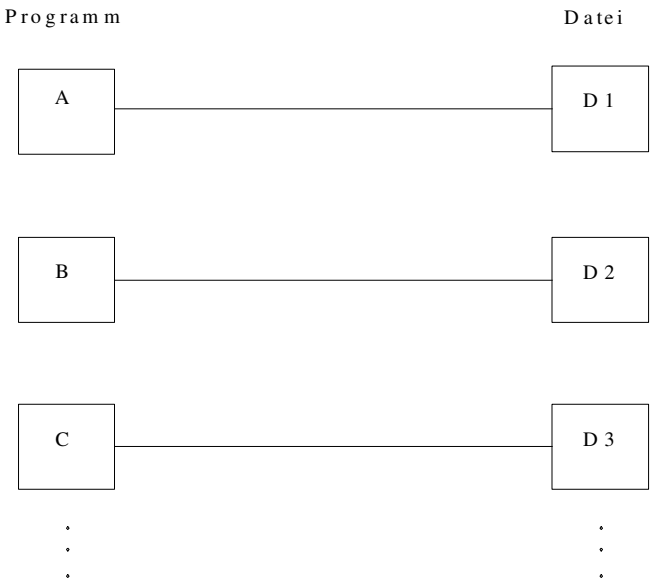


Bild 1.2: Dateisysteme: enge Kopplung zwischen Programm und Datei

Ende Blick auf Dateisysteme!

Probleme traditioneller Datenverwaltung

Die traditionelle Verarbeitung großer Datenmengen mittels Dateisystemen führt zu schwerwiegenden Problemen:

(1) **Redundanz**:
Da die Daten jeweils speziell für bestimmte Anwendungen entworfen werden, werden dieselben Daten in verschiedenen Dateien wieder auftauchen (z.B. Namen, Adressen). Redundanz führt zu Speicherverschwendung und zu erhöhten Verarbeitungskosten, vor allem bei Änderungen. Schlimmer jedoch ist es, dass diese Redundanz in der Regel nicht zentral kontrolliert wird, so dass Konsistenzprobleme auftreten.

(2) **Inkonsistenz**:
Die Konsistenz der Daten (d.h. die logische Übereinstimmung der Datei-Inhalte) kann nur schwer gewährleistet werden. Bei der Änderung einer Grö-

ße müssten alle Dateien geändert werden, die diese Größe beinhalten, und diese Änderungen müssten so miteinander abgestimmt geschehen, dass nicht verschiedene Programme zum selben Zeitpunkt unterschiedliche Werte derselben Größe sehen können.

(3) **Daten-Programm-Abhängigkeit**:
Ändert sich der Aufbau einer Datei oder ihrer Organisationsform, so müssen darauf basierende Programme geändert werden. Wird beispielsweise für eine Anwendung ein weiteres Datenelement in einem Satztyp benötigt (z.B. Telefonnummer eines Angestellten), so müssen infolge der notwendigen Neu-Definition der Datei alle Programme geändert werden, unabhängig davon, ob sie dieses neue Datenelement sehen wollen oder nicht.

(4) **Inflexibilität**:
Da die Daten nicht in ihrer Gesamtheit, sondern nur anwendungsbezogen gesehen werden, ist es in vielen Fällen sehr kompliziert, neue Anwendungen oder Auswertungen vorhandener Daten zu realisieren. Dies gilt insbesondere für Auswertungen, die Daten aus verschiedenen Dateien benötigen würden. Die Organisation nach diesem konventionellen Vorgehen ist sehr wenig anpassungsfähig an die sich verändernden Anforderungen in einem Unternehmen.

Flexibilität
ad-hoc Anfragen

Neue Philosophie: Datenbank

Diese Schwierigkeiten führten schließlich zu einer veränderten Betrachtung der Daten in einem Unternehmen. Die Daten sind nicht mehr Anhängsel der Anwendungsprogramme, sondern werden als eigenständige wesentliche Betriebsmittel des Unternehmens gesehen, die als integriertes Ganzes zu verstehen sind; alle Anwendungsprogramme arbeiten auf dieser gemeinsamen Basis aktueller Information. Die Daten werden einmal definiert und für alle Benutzer zentral verwaltet. Dies ist der entscheidende Schritt zum Konzept des Datenbanksystems.

Daten als eigenständiges Betriebsmittel eines Unternehmens

Mit dieser Philosophie wird überflüssige Redundanz automatisch vermieden, da Information über ein Objekt auch nur einmal gespeichert und vom Datenbanksystem jedem Anwendungsprogramm in geeigneter Weise zur Verfügung gestellt wird. Wo Redundanz sinnvoll ist (etwa aus Effizienzgründen), wird sie durch das DBMS zentral kontrolliert, so dass Inkonsistenzen vom System selbst verhindert werden können.

Eine ganz entscheidende Verbesserung gegenüber der Dateiverarbeitung ist die Trennung von Programm und Daten, die durch das DBMS erreicht wird. Das DBMS übergibt dem Anwendungsprogramm nicht mehr ganze Sätze, so wie sie im Speicher liegen, sondern genau die vom Anwendungsprogramm benötigten Datenelemente. Veränderungen des Satzaufbaus oder der internen Organisation der Daten bleiben somit für das Anwendungsprogramm unsichtbar, so dass bisher laufende Programme von solchen Änderungen nicht betroffen werden. Diese Trennung von Anwendungsprogramm und Daten wird als *Datenunabhängigkeit* - Unabhängigkeit des Programmes von der Organisation der Daten - bezeichnet. Auf den Aspekt der Datenunabhängigkeit werden wir später genauer eingehen.

Die Trennung von Programm und Daten: Datenunabhängigkeit

Da die Daten integriert sind, ist es sehr viel einfacher als in Dateisystemen, neue Anwendungen zu realisieren. In der Datenbank sind sämtliche verfügbaren Daten in überschaubarer Weise bekannt und können deshalb im Prinzip in beliebiger Weise ausgewertet werden. Diese Flexibilität ist Voraussetzung für Informationssysteme, in denen nicht vorausgeplante Anfragen (*ad-hoc queries*) gestellt werden können.

Integrität

Ein weiterer Vorteil von Datenbanksystemen ist bisher nicht angeklungen: die Integrität der Daten. *Integrität* der Daten bedeutet ganz allgemein Korrektheit und Vollständigkeit der abgespeicherten Daten - beides zu messen an der Realität, über die die Datenbank Daten enthält. In Dateisystemen ist der Anwendungsprogrammierer dafür verantwortlich, geeignete Prüfungen in seine Programme einzubauen, die verhindern, dass falsche Daten abgespeichert werden. Dies führt natürlich in der Praxis dazu, dass Daten, die durch verschiedene Programme erfasst werden, unter Umständen einen völlig unterschiedlichen Grad an „Sicherheit“ besitzen. Hinzu kommt, dass fehlerhafte Software und Hardware ebenfalls Fehler in den Daten erzeugen können. Die zentrale Verwaltung der Daten durch das DBMS ermöglicht es, bei Änderung von Daten Kontrollroutinen einzuschalten oder von Zeit zu Zeit mit Hilfe spezieller Prüfprogramme nach Integritätsverletzungen zu suchen. Darüber hinaus können jetzt zentral - und damit für alle Daten einheitlich - Maßnahmen implementiert werden, die im Falle von Fehlern in den Daten zu ergreifen sind.

Ein zusammenfassendes Beispiel:

Beispiel 1.3:

Betrachten wir die Software-Entwicklungsabteilung eines Unternehmens. Die Abteilung besteht aus mehreren Angestellten, die an verschiedenen Software-Projekten beteiligt sind. Der Leiter dieser Abteilung besitze zur Überwachung der Abteilung ein Programm PV zur Personalverwaltung und ein Programm TK zur Terminkontrolle der Software-Projekte. Für PV werden die Angestellten-Daten in der ANGESTELLTEN-DATEI aufsteigend sortiert nach der Angestellten-Nr. gespeichert. Für das Programm TK werden die Projekt-Daten aufsteigend sortiert nach der Projekt-Nr. in der PROJEKT-DATEI gespeichert. Zur besseren Kontrolle der Abteilung wird nach einiger Zeit eine PROJEKT-BESCHREIBUNGS-DATEI eingerichtet. Diese Datei enthält für jedes Projekt die entsprechenden Termine, die Beschreibung der Projekte, den Namen des Projektleiters und diejenigen der Projekt-Mitarbeiter. Nehmen wir an, es werde nun auch noch ein Programm zur Erstellung von Tätigkeitsberichten gewünscht. Ein Tätigkeitsbericht enthält zu jedem Angestellten zusätzlich zu seinen Daten die Beschreibung und den Endtermin der von ihm bearbeiteten Projekte. Hierfür wird aus Effizienzgründen wiederum eine eigene Datei, die TÄTIGKEITS-BERICHTS-DATEI, aufgebaut.

Man kann an diesem Beispiel sehr schön die oben angesprochenen Probleme (1) - (4) erkennen. Beispielsweise tauchen Angaben über die Angestellten, z.B. Name, in der Angestellten-Datei, Projekt-Berichts-Datei und der Tätigkeits-Berichts-Datei auf. Oder: Ändert sich der Endtermin eines Projektes, so ist diese Änderung sowohl in der Datei PROJEKT-BESCHREIBUNGEN als auch in der TÄTIGKEITS-BERICHTS-DATEI durchzuführen - offensichtlich eine Quelle für Inkonsistenzen.

Statt der verschiedenen Dateien dieses Beispiels würde man in einem Datenbanksystem die Daten über Projekte und Angestellte etwa folgendermaßen zusammenfassen:

Sätze über Angestellte
Sätze über Projekte
Beziehung ANGESTELLTER-PROJEKT

Die Angestellten-Sätze enthalten die Daten über Angestellte, die Projekt-Sätze enthalten die Daten über die Projekte. Die Beziehung ANGESTELLTER-PROJEKT setzt jeden Angestellten mit jedem Projekt in Verbindung, an dem er mitarbeitet. Diese Beziehung ist in den Daten enthalten; man kann sie sich als Zeiger vorstellen: vom Angestellten-Satz verweisen Zeiger auf die zugehörigen Projekt-Sätze, vom Projekt-Satz Zeiger auf die zugehörigen Angestellten-Sätze. Wir werden sehen, dass man diese Beziehung jedoch auf sehr unterschiedliche Art realisieren kann. Das Datenbanksystem stellt Befehle zur Verfügung, um die Beziehungen auf einfache Weise auswerten zu können.

Vorteile des Datenbankansatzes

Wir können die **Vorteile der Datenbank-Philosophie** gegenüber der konventionellen Denkweise in Dateisystemen in folgender Weise zusammenfassen:

(1) Es gibt eine gemeinsame Basis für alle Anwendungen (Vereinheitlichung).

(2) Redundanz entfällt; wo Redundanz nützlich ist, ist sie durch das DBMS kontrolliert.

(3) Wegen (2) entfallen die Konsistenzprobleme traditioneller Dateiorganisationen.

(4) Die Anwendungsprogrammierung wird vereinfacht, da der Programmierer nur die ihn interessierenden Eigenschaften der Daten, nicht aber deren spezielle Organisation auf den Speichern kennen muss.

(5) Die Abhängigkeit zwischen Programmen und Daten wird reduziert. Da das DBMS die Daten so abliefert, wie sie vom Anwendungsprogramm benötigt werden (Sicht!), führen viele Änderungen in der Datenorganisation nicht zu Änderungen der Anwendungsprogramme.

(6) Das Datenbanksystem verschafft mehr Flexibilität für die Datenauswertung. Da alle Daten über den Anwendungsbereich zusammengefasst definiert und gespeichert sind, ist es wesentlich einfacher, neue Anwendungen zu programmieren oder ungeplante Anfragen zu beantworten.

(7) Das Datenbanksystem kann zentral die Korrektheit von Daten überprüfen (Prüfprogramme etc.).

(8) Das Datenbanksystem kann zentral Mechanismen zur Wiederherstellung einer korrekten Datenbank nach dem Auftreten von Fehlern bereitstellen.

2. Architektur eines Datenbanksystems

2.1 Drei Datenebenen

Die Diskussion der Idee eines Datenbanksystems hat gezeigt, dass die Daten eines Anwendungsbereiches oder Unternehmens auf unterschiedliche Weise gesehen werden. Ein Benutzer sieht z.B. die Daten völlig anders als der Systemprogrammierer, der die Organisation der Daten auf den Speichern festlegt.

Wir müssen *drei Datenebenen* unterscheiden:

- die logische Gesamtsicht,
- die Datenorganisation der Daten auf den Speichern: interne Sicht,
- die Sicht einzelner Benutzergruppen: externe Sichten.

Logische Gesamtsicht

In der Datenbank sind alle wichtigen Unternehmensdaten zusammengefasst (zur Verkürzung der Schreibweise sprechen wir im Folgenden von Unternehmensdaten statt von den Daten des betrachteten Anwendungsbereiches). Um die Datenbank erstellen zu können, ist eine Gesamtschau der Unternehmensdaten notwendig. Alle Daten müssen zunächst auf logischer Ebene in Form von Informationseinheiten und deren Beziehungen untereinander beschrieben werden, unabhängig von EDV-Gesichtspunkten. Diese Beschreibung der Gesamtheit der Unternehmensdaten nennen wir logische Gesamtsicht.

Interne Sicht

Die Daten müssen auf den Speichern so organisiert werden, dass die Zugriffsanforderungen der verschiedenen Benutzer möglichst effizient erfüllt werden können.

Externe Sichten

Jede Benutzergruppe sieht den Ausschnitt der Datenbank, der für sie von Bedeutung ist. Die Daten werden so dargestellt, wie es für die Benutzer wünschenswert oder natürlich ist.

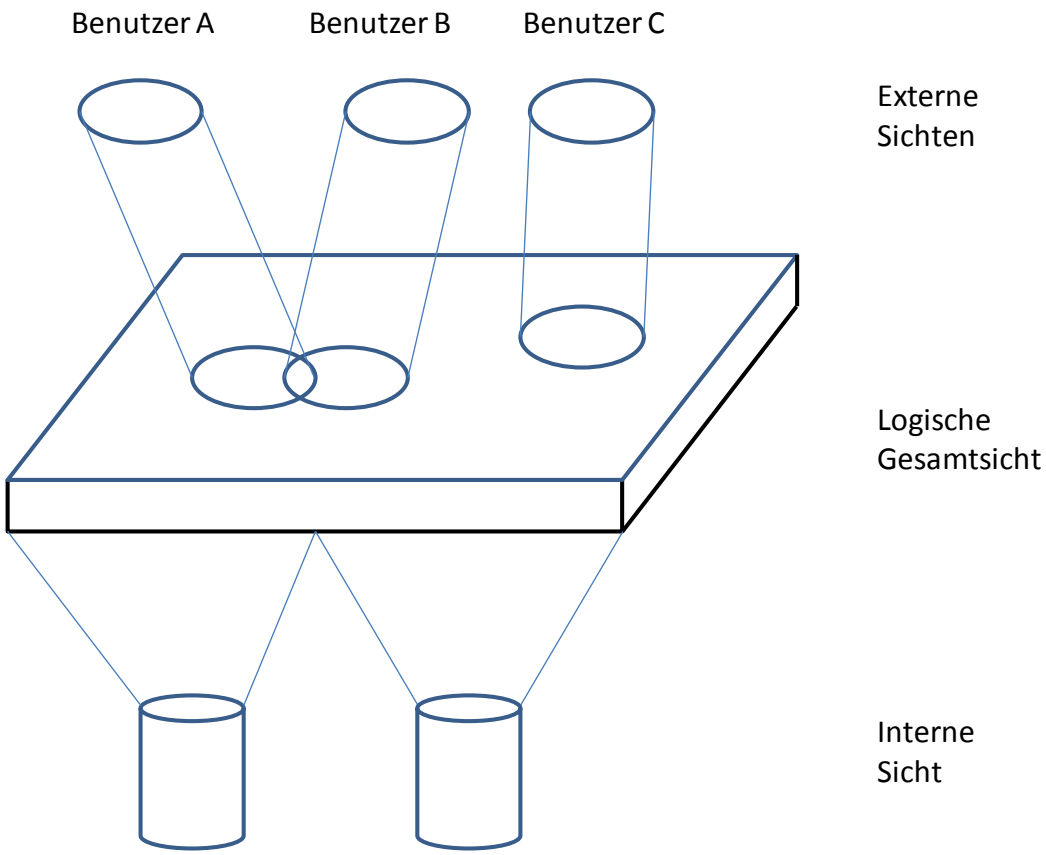


Bild 2.1: Die drei Datenebenen in einem Datenbanksystem

Die Benutzer arbeiten mit der Datenbank ausschließlich über ihre externen Sichten. Sie sehen weder die logische Gesamtsicht noch die interne Organisation der Daten. Die notwendigen Umsetzungen von einer externen Sicht in die logische Gesamtsicht und von dort in die interne Sicht erledigt das DBMS. Es benötigt dazu Beschreibungen der jeweiligen Sichten und Regeln, die die Umsetzung von einer Sicht in die andere ermöglichen.

Modelle der Datenwelt
Datenbeschreibungssprache

Jede Ebene der Daten modelliert die Unternehmensdaten auf einem anderen Abstraktionsniveau. Diese *Modelle* der Datenwelt des Unternehmens werden mit Hilfe sogenannter *Datenbeschreibungssprachen* in einer für das DBMS verständlichen Form beschrieben; diese Beschreibung heißt dann *Schema*. Es gibt also verschiedene externe Schemata, ein konzeptuelles Schema, das die logische Gesamtsicht beschreibt, und ein internes Schema.

Schema

Dieses Verständnis der Daten führt zu einer Drei-Schichten-Architektur für ein Datenbanksystem /ANS75/. Jeder der drei Datenebenen entspricht eine eigene Schnittstelle und damit eine eigene Schicht innerhalb des Gesamtsystems. Bild 2.2 zeigt diese prinzipielle Architektur. Die wesentlichen Komponenten sind das konzeptuelle Schema, das interne Schema und die externen Schemata. Die Transformationsregeln definieren die Abbildungen zwischen den Ebenen, d.h. sie legen fest, wie die Daten aus einer Schicht in die entsprechenden Daten der anderen Schicht transformiert werden. Das DBMS akzeptiert die Benutzeraufträge, die in den Begriffen des externen Schemas formuliert sind. Es stellt über die verschiedenen Datenebenen hinweg fest, welche gespeicherten Daten für die Bearbeitung

des Benutzerauftrages benötigt werden. Für die Zugriffe auf die Speicher ruft das DBMS im allgemeinen das Betriebssystem auf.

Wir wollen im Folgenden die einzelnen Komponenten der Architektur und die skizzierten Zusammenhänge etwas genauer betrachten.

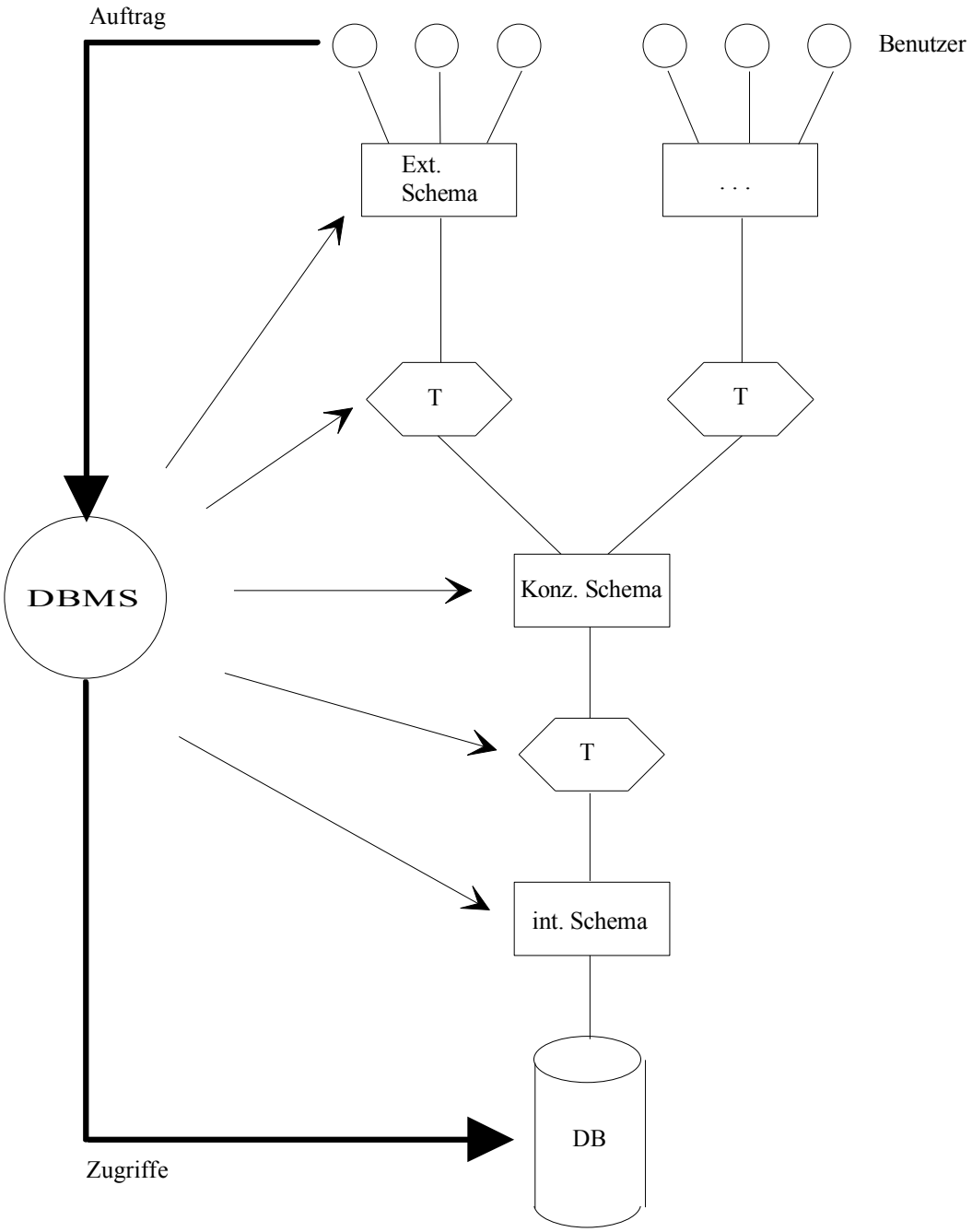


Bild 2.2: Architektur eines Datenbanksystems
(T = Transformationsregel)

2.2 Das konzeptuelle Modell

Das konzeptuelle Modell beschreibt die Gesamtheit derjenigen Daten des Unternehmens, die in der Datenbank verwaltet werden - es erfasst also die logische Gesamtsicht. Die Daten werden auf logischer Ebene beschrieben, ausgehend von den Strukturen, in denen sie in der Realität existieren, unabhängig von Gesichtspunkten der einzelnen Benutzer und Gesichtspunkten der physischen Speicherung. Das konzeptuelle Modell ist ein Modell des Unternehmens in dem Sinne, dass alle für das Unternehmen wesentlichen Daten und alle wesentlichen Beziehungen zwischen den Daten beschrieben werden. Zusätzlich zu den Informationen über Daten und Beziehungen zwischen Daten beschreibt das konzeptuelle Modell die Bedingungen, die für die Daten des Unternehmens immer gelten oder die Vorschriften für das Ändern von Daten machen. Beispielsweise können solche Bedingungen festlegen, dass das Ändern bestimmter Daten das Ändern bestimmter anderer Daten zur Folge hat. Da solche Bedingungen im realisierten Datenbanksystem die Möglichkeit zur Prüfung der Korrektheit von Daten geben, heißen solche Bedingungen auch *Integritätsbedingungen*.

Integritätsbedingungen

Grundsätzlich ist zugleich mit der Modellierung der Daten des Unternehmens auch festzulegen, welche Operationen mit den Daten ausgeführt werden dürfen. Mit den semantischen Einheiten des konzeptuellen Modells müssen Operationen verbunden werden. Jeder Zugriff auf die Datenbank muss dann in diesen „kontrollierten“ Operationen formuliert werden (Man denke an das Konzept des abstrakten Datentyps!). Beispielsweise könnte man in einer Banken-Anwendung festlegen: wird bei Konto a der Betrag x subtrahiert, so muss x bei einem anderen Konto addiert werden. Dies würde dazu führen, dass die Anwendungsprogramme nicht mehr auf einzelne Kontensätze zugreifen und diese verändern, sondern dass sie eine Funktion TRANSFER(a,b,x) aufrufen, die bewirkt, dass der Betrag x von a nach b transferiert wird. Die Änderung eines Kontos a ist damit also nur möglich, wenn zugleich ein anderes Konto b entsprechend geändert wird.

Bei den heute verbreiteten relationalen Datenbanksystemen gibt es diese Spezifikation von Operationen im konzeptuellen Schema der Datenbank nicht. Alle diese Systeme bieten sogenannte *generische* Operationen an, also Operationen, die auf alle Datentypen in der Datenbank anwendbar sind. Dies sind unter anderem speichern, lesen, löschen und modifizieren. Der Zugriff auf die Datenbank erfolgt mittels einer *Datenmanipulationssprache*, die diese Operationen zur Verfügung stellt.

Datenmanipulationssprache

Anders bei der jüngsten Generation von DBMS, den sogenannten *Objektorientierten Datenbanken*: Bei diesen werden zusammen mit der Struktur der Daten auch die zulässigen Operationen auf diesen definiert. Anwendungsprogramme müssen dann auf diesen Daten mit den vorgegebenen Operationen arbeiten.

Die Modellbildung setzt eine detaillierte Analyse der realen Welt voraus. Das konzeptuelle Modell wird deshalb nicht von EDV-Fachleuten erstellt, sondern von Leuten, deren Augenmerk auf der rein logischen Ebene des gesamten Informationssystems des Unternehmens liegt und die die Gesamtheit der Aufgabenstellungen des Informationssystems und damit die Bedeutung der Daten kennen, ohne sich für die Realisierung der Aufgaben im Einzelnen zu interessieren.

DBS und Informationssystem

Beachten Sie den Unterschied zwischen den Begriffen „Datenbanksystem“ und „Informationssystem“. Leider werden beide Begriffe oft synonym verwendet. Das Informationssystem eines Unternehmens ist die Gesamtheit aller Instanzen und Prozesse, die die für das Unternehmen wichtige Information von außen aufnehmen, verarbeiten und entsprechende Information nach außen wieder abgeben. Das Datenbanksystem ist damit nur ein Teil des Informationssystems, nämlich derjenige Teil, in dem die Informationsbasis des Unternehmens verwaltet wird.

Das konzeptuelle Modell stellt einen relativ stabilen Bezugspunkt im Informationssystem eines Unternehmens dar. Bei genügend sorgfältiger Analyse und Planung wird sich die Gesamtschau der Daten sehr viel langsamer ändern als die Anforderungen einzelner Anwendungen. Auf der Ebene der einzelnen Anwendungen werden sich laufend die Wünsche bezüglich der Sichten auf die Daten ändern; in gleicher Weise werden die sich verändernden Anforderungen laufend Änderungen der physischen Datenorganisation notwendig machen. Das konzeptuelle Modell erlaubt es, sowohl spezielle Sichten von Benutzergruppen auf die Daten zu ändern oder neue Sichten einzuführen, als auch auf der Ebene der physischen Datenorganisation Änderungen vorzunehmen, ohne dass bestehende Programme verändert werden müssen.

Vorteile des konzeptuellen Modells

Zusammenfassend können wir sagen, dass man mit dem konzeptuellen Modell folgende Vorteile gewinnt:

- Das konzeptuelle Modell stellt einen stabilen Bezugspunkt für alle Anwendungen dar. Es ändert sich nur, wenn sich die Vorstellungen über das Unternehmen ändern.

- Das konzeptuelle Modell stellt eine einheitliche Dokumentation wesentlicher Aspekte des Unternehmens dar.

- Das konzeptuelle Modell bietet die Möglichkeit, den Gebrauch der Daten an zentraler Stelle zu kontrollieren.

- Das konzeptuelle Modell schafft die wesentliche Voraussetzung für Datenunabhängigkeit der Anwendungsprogramme (Dieser Punkt wird später ausführlicher diskutiert).

Datenbeschreibungssprache (DDL)

Datenbankadministrator

Das konzeptuelle Modell wird mit Hilfe einer geeigneten *Datenbeschreibungssprache* (data definition language, DDL) im *konzeptuellen Schema* beschrieben. Wir nennen die Person oder die Instanz, die für die Erstellung von Schemata zuständig ist, *Datenbankadministrator*. In einem größeren Unternehmen werden natürlich nicht alle Schemata von einer Person oder einer Personengruppe erstellt. Die ANSI-SPARC-Vorschläge /ANS75/ unterscheiden deshalb verschiedene Administratoren für die verschiedenen Datenebenen. Wir begnügen uns hier mit dem Begriff des Datenbankadministrators, der also als Sammelbegriff zu verstehen ist.

Ein Datenmodell zur Formulierung des konzeptuellen Modells der interessierenden Miniwelt wird in Abschnitt 2.8 vorgestellt. Zur Veranschaulichung möchten wir hier lediglich ein Beispiel geben:

Um die Miniwelt der FernUniversität zu modellieren, würde man etwa folgende Objekttypen unterscheiden: Kurse, Professoren, Studenten, Räume, Fachbereiche, etc. Zu jedem dieser Objekttypen würde man die interessierenden Attribute beschreiben, beispielsweise für Studenten die Attribute „Anschrift“, „Matrikelnummer“, „Geburtsdatum“, usw. Schließlich würde man Beziehungstypen zwischen den Objekttypen festhalten, etwa die beiden Beziehungstypen zwischen Professoren und Kursen „hat_geschrieben“ und „bietet_an“. Diese Objekttypen und Beziehungstypen würden dann in einer DDL exakt beschrieben werden.

Überlegungen, wie die entsprechenden Objekte abgespeichert oder wie die Beziehungen in der Datenbank realisiert werden, spielen auf dieser Ebene keine Rolle.

2.3 Das interne Modell

Nach der Erstellung des konzeptuellen Schemas muss festgelegt werden, in welcher Form die jetzt logisch beschriebenen Daten auf den Speichern abgelegt werden und welche Zugriffsmöglichkeiten auf diese Daten bestehen sollen. Zur Ausführung dieser Aufgabe benötigt der Datenbankadministrator statistische Informationen über die Häufigkeit von Anwendungen, von Zugriffen auf Objekte, über Zeitbeschränkungen für Anwendungen, Verteilungen von Werten für Attribute, usw. Der Datenbankadministrator muss eine *physische Datenorganisation* entwickeln, so dass die im konzeptuellen Schema beschriebenen Objekte ableitbar sind, und zwar so, dass die Aufgaben der Benutzergemeinschaft insgesamt „gut“ erfüllt werden können. Die gewählte physische Datenorganisation wird im *internen Schema* beschrieben. Das interne Schema enthält also alle Informationen über den Aufbau der abgespeicherten Daten, die Speicherung der Daten, die Zugriffspfade, usw.

physische Datenorganisation

internes Schema

Das interne Schema muss demnach genaue Festlegungen zu folgenden Punkten enthalten:

- Repräsentation von Attributwerten
- Aufbau gespeicherter Sätze
- Zugriffsmethoden auf Sätze
- zusätzliche Zugriffspfade (Indexe, Verkettungen, usw.)

Einige dieser Fragen werden im Kapitel 7 behandelt werden.

In den kommerziell verfügbaren Datenbanksystemen sind immer eine Reihe von Entwurfsentscheidungen vorweggenommen. Beispielsweise stehen häufig für die Realisierung von Beziehungen nur ganz bestimmte Möglichkeiten zur Verfügung, oder es gibt nur eine beschränkte Auswahl von Zugriffsmöglichkeiten auf Sätze, usw.

Transformationsregeln

In den *Transformationsregeln konzeptuelles/internes Schema* wird die Beziehung zwischen den physisch abgespeicherten Feldern, Sätzen usw. und den Objekten des konzeptuellen Schemas definiert. Dazu werden Vorschriften benötigt, die für jedes Objekt des konzeptuellen Schemas angeben, wie die dazugehörige Information aus den abgespeicherten Daten erhalten wird. Muss zur Verbesserung des Systemverhaltens die physische Datenorganisation verändert werden, so müssen

diese Transformationsregeln ebenfalls verändert werden. Das konzeptuelle Schema bleibt von Änderungen auf der internen Ebene unberührt.

2.4 Externe Modelle

Die integrierte Modellierung der Unternehmensdaten im konzeptuellen Modell bedeutet nicht, dass jeder Benutzer nunmehr mit dieser Gesamtschau der Daten arbeiten muss. Vielmehr soll jeder Benutzer nach wie vor nur die für ihn relevanten Daten sehen, diese Daten sollen ihm in einer ihm vertrauten Form geboten werden. Mit anderen Worten: jeder Benutzer soll *seine eigene Sicht (View)* auf die Daten haben.

eigene Sicht (view)

Objekttypen und Beziehungstypen externer Modelle müssen nicht mit denen des konzeptuellen Modells identisch sein, sie müssen lediglich inhaltlich im konzeptuellen Modell enthalten sein.

Beispiele für externe Sichten wurden schon im Abschnitt 1.1 angegeben: die unterschiedlichen Sichten des Sachbearbeiters und des Managers. Sind in diesem Beispiel KUNDE und AUFTRAG zwei Objekttypen des konzeptuellen Modells, so ist die Sicht des Managers offenbar sehr verschieden vom konzeptuellen Modell: es werden Daten von den Objekten des Typs KUNDE und des Typs AUFTRAG benötigt, wobei zusätzlich die einzelnen Volumina der Aufträge eines Kunden aufaddiert werden müssen.

Auf der Ebene der externen Modelle müssen dem Benutzer Sprachmittel an die Hand gegeben werden, die es ihm erlauben, auf Daten in der Datenbank zuzugreifen und Daten in der Datenbank zu verändern. Die Sprachmittel sind bei den gängigen relationalen Systemen durch die Datenbanksprache gegeben. In einer entsprechenden Umgebung können solche Sprachen eigenständig benutzt werden, teilweise auch mit der Unterstützung von graphischen Benutzeroberflächen. Außerdem kann man die Elemente der Datenbanksprache in gängigen Programmiersprachen (Java, C, C++, usw.) aufrufen und damit auf Datenbanken zugreifen und deren Inhalte verändern.

Beispielsweise kann die Änderung eines Kundeneintrags durch folgendes SQL-Statement erfolgen:

```
UPDATE KUNDENSATZ
SET KONTOSTAND = KONTOSTAND + 100
WHERE KUNDEN-NR = 1012
```

Es kann also dazu führen, dass sich komplexe Abfragen (in der Abfragesprache formuliert) in einem Anwendungsprogramm (etwa in Java) finden.

Idealerweise sollte natürlich Programmiersprache und Datenmanipulationssprache integriert sein, um es dem Programmierer zu ersparen, dass er zwei unterschiedliche Sprachen - nämlich Programmiersprache und DML - beherrschen muss (dies wäre auch aus anderen Gründen wichtig, die wir im Zusammenhang mit objektorientierten Datenbanken kurz diskutieren werden). Solche Integrationsversuche gibt es für Pascal und andere Sprachen, die mit Konstrukten zur Manipulation von

Relationen erweitert werden (z. B. Pascal/R). Konsequent auf diesem Weg gehen die objektorientierten Datenbanken, bei denen von vornherein zwischen Programmiersprache und DML kein Unterschied besteht.

Die Beschreibung der externen Modelle erfolgt in den externen Schemata. Bezugspunkt für die externen Modelle ist das konzeptuelle Modell. In den Transformationsregeln externes/konzeptuelles Schema muss angegeben werden, auf welche Weise Entities und Beziehungen des konzeptuellen Schemas zusammengesetzt werden.

2.5 Das Datenbankmanagementsystem

Das Datenbankmanagementsystem (DBMS) ist die Software, die jeglichen Zugriff auf die Daten der Datenbank übernimmt. Der Benutzer formuliert seine Zugriffswünsche in den Begriffen seines externen Modells. Das DBMS übernimmt die notwendigen Umsetzungen, ermittelt die Objekte, die von den Speichern zu lesen sind, und übergibt die gewünschten Daten in der entsprechenden Form an das Anwendungsprogramm. Neben dieser Durchführung der Datenzugriffe hat das DBMS eine Reihe weiterer wesentlicher Aufgaben, die wir im Anschluss diskutieren werden. Betrachten wir zunächst die Vorgänge beim Zugriff auf die Datenbank etwas genauer. Logisch gesehen laufen die folgenden Schritte ab, wenn ein Anwendungsprogramm ein Objekt lesen will (vgl. Bild 2.2):

1. Das DBMS empfängt den Befehl des Anwendungsprogrammes, ein bestimmtes Objekt eines externen Modells zu lesen.

2. Das DBMS holt sich die benötigten Definitionen des entsprechenden Objekttyps aus dem zugehörigen externen Schema.

3. Mit Hilfe der Transformationsregeln externes/konzeptuelles Schema stellt das DBMS fest, welche konzeptuellen Objekte und Beziehungen benötigt werden.

4. Mit Hilfe der Transformationsregeln konzeptuelles/internes Schema stellt das DBMS fest, welche physischen Objekte zu lesen sind, es ermittelt die auszunützenden Zugriffspfade.

5. Das DBMS übergibt dem Betriebssystem die Nummern der zu lesenden Speicherblöcke.

6. Das Betriebssystem übergibt die verlangten Blöcke an das DBMS in einem Systempuffer.

7. Mit Hilfe der Transformationsregeln stellt das DBMS aus den vorhandenen physischen Sätzen das verlangte externe Objekt zusammen.

8. Das DBMS übergibt das externe Objekt dem Anwendungsprogramm in seinen Arbeitsspeicher.

9. Das Anwendungsprogramm verarbeitet die vom DBMS übergebenen Daten.

Beachten Sie, dass der Zugriff auf ein externes Objekt (wie hier skizziert) zum Zugriff auf mehrere konzeptuelle Objekte führen kann, und dass für jedes konzeptuelle Objekt möglicherweise Zugriff auf mehrere interne Objekte notwendig wird. Nicht jeder Befehl wird in der obigen geradlinigen Folge ablaufen. So wird das DBMS häufig größere Datenmengen durchsuchen müssen, ehe es die geforderten Daten findet. In diesem Falle muss das DBMS nach Schritt 6 prüfen, ob die gesuchten Daten gefunden sind; wenn nicht, wird das Betriebssystem erneut aufgerufen, um weitere in Frage kommende Speicherblöcke abzuliefern, usw. Diese Situation tritt beispielsweise dann ein, wenn zum Auffinden eines Objektes Dateien zu durchsuchen oder Listen zu durchlaufen sind.

Diese Skizze der Arbeitsweise des DBMS ist natürlich sehr vereinfacht. Insbesondere ist hier der Prozess *interpretativ* dargestellt, d.h. so, als würden die ganzen Analysen, Schemainterpretationen etc. zur Laufzeit vorgenommen. Tatsächlich kann der skizzierte Ablauf auf sehr unterschiedliche Weise realisiert werden. Dies gilt vor allem für den Prozess des *Bindens*. Um den Befehl eines Anwendungsprogramms auszuführen, müssen die Objekte des externen Modells ausgedrückt werden durch Objekte des konzeptuellen Modells und schließlich durch Objekte des internen Modells. Sobald der Befehl, der sich auf ein externes Objekt bezieht, ersetzt ist durch Befehle, die sich auf das konzeptuelle Modell beziehen, sind die entsprechenden Daten des Anwendungsprogrammes an das konzeptuelle Modell „gebunden“, entsprechend für das interne Modell. Ändert sich nach dem vollständigen Binden des Anwendungsprogrammes das interne Modell, so läuft das Anwendungsprogramm fehlerhaft - es muss neu (an das veränderte interne Modell) gebunden werden.

Binden

Das Binden eines Programmes kann zu verschiedenen Zeitpunkten stattfinden. Die beiden Enden des Spektrums sind

Bindezeitpunkt

- Binden zur Übersetzungszeit
- Binden zum Zeitpunkt des Zugriffs auf die Daten (Laufzeit).

Im Bereich der Programmiersprachen würde man im ersten Fall von Compilation, im zweiten von Interpretation sprechen.

Beispiel 2.1:

Ein Anwendungsprogramm benötigt die Summe der Gehälter aller Angestellten in einer bestimmten Abteilung. Die Angestelltensätze zu jeder Abteilung seien intern durch eine Liste verbunden. Binden zur Übersetzungszeit bedeutet nun, dass das übersetzte Programm die Befehle zum Durchlaufen dieser Liste (Verfolgen von Zeigern an bestimmten Positionen) enthält. Ändern wir nun die Organisation der Daten und ersetzen die Liste z.B. dadurch, dass zusammengehörige Sätze einfach physisch aneinandergereiht werden, so muss das Programm neu übersetzt werden. Wird Binden zur Laufzeit angewandt,

so enthält das übersetzte Programm Kommandos etwa der Form „Lies den nächsten Angestelltensatz von Abteilung x“. Erst nach Empfang dieses Kommandos ermittelt das DBMS, wie auf den nächsten Angestelltensatz zuzugreifen ist. Das Programm wird hier also erst zur Laufzeit an die Daten (die Organisation der Daten) gebunden.

Wird das Binden zur Übersetzungszeit durchgeführt, so muss das Programm neu übersetzt werden, wenn sich am konzeptuellen oder internen Modell etwas ändert. Größtmögliche Unabhängigkeit des Anwendungsprogrammes von der Organisation der Datenbank wird erreicht, wenn das Binden erst zum Zeitpunkt des Zugriffs auf die Daten stattfindet. In diesem Falle kann das konzeptuelle wie das interne Modell zu jedem Zeitpunkt geändert werden, ohne dass das Anwendungsprogramm davon berührt wird. Diese Form des Bindens ist jedoch sehr zeitaufwendig (teuer), denn bei jedem Zugriffswunsch des Anwendungsprogrammes muss zur Laufzeit der gesamte oben skizzierte Bindevorgang erneut ablaufen.

Die gegebene Beschreibung der Arbeitsweise des DBMS ist sehr prinzipieller Natur. Im Folgenden skizzieren wir die Abläufe auf einer mehr technischen Ebene. Dabei betrachten wir den Ablauf bei der Bearbeitung eines DML-Befehls, der sich auf ein Objekt, einen Satz, bezieht. Im Zusammenhang mit relationalen Datenbanken betrachten wir später die Abläufe bei mengenorientiertem Zugriff.

Wir nehmen an, dass aus einem Anwendungsprogramm heraus der Satz des Angestellten Nummer 12 gelesen werden soll. Der Satztyp für Angestellte auf der externen Ebene sei ANG (ANGNR, NAME, GEHALT), d.h., ein Satz vom Typ ANG enthält die Felder ANGNR, NAME, GEHALT, deren Typ natürlich wiederum spezifiziert sein muss. Folgender Befehl (SQL) wird also abgesetzt:

```
SELECT ANGNR, NAME, GEHALT
FROM ANG
WHERE ANGNR=12
```

Auf der konzeptuellen Ebene sei der Satztyp für die Angestellten

ANG (ANGNR, BERUF, NAME, ADRESSE, GEHALT, GEB-DAT, POS).

Die Abarbeitung des DML-Befehls geht nun wie folgt vor sich (vgl. Bild 2.3):

(1) Übergabe des DML-Befehls an das DBMS.

(2) Interpretation des DML-Befehls durch das DBMS; Ermittlung der zugehörigen konzeptuellen Beschreibung.

(3) Ermittlung der Speicherungsstruktur und der möglichen Zugriffspfade auf den gesuchten Satz (internes Schema). Optimierung der Abfrage: Festlegung der Folge der Abarbeitungsschritte.

(4) Das DBMS ermittelt die Datenseite (Page), auf der der gesuchte Satz gespeichert ist. Es prüft, ob sich diese Datenseite bereits im Systempuffer befindet. Falls nicht, muss auf die Datenbank zugegriffen werden; sonst weiter mit (8).

(5) Auswahl einer Seite im Systempuffer, die durch die benötigte Seite überlagert werden kann. Falls die zu ersetzende Seite verändert wurde, muss sie in die Datenbank geschrieben werden.

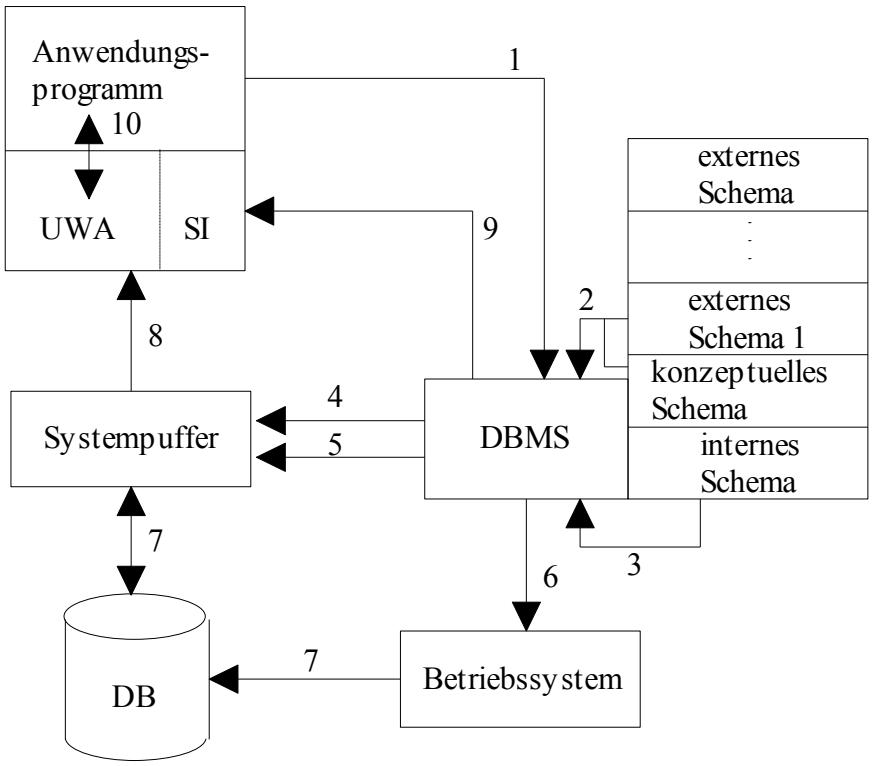
(6) Das DBMS ruft das Betriebssystem für zwei E/A-Vorgänge auf:
- Schreiben der zu ersetzenden Seite in die Datenbank (entfällt gegebenenfalls)
- Einlesen der gesuchten Seite.

(7) Das Betriebssystem führt die physischen E/A-Aufträge durch und speichert die angeforderte Seite an der vorgegebenen Adresse im Systempuffer.

(8) Das DBMS liest den gesuchten Satz aus dem Systempuffer, transformiert ihn in die durch das externe Schema definierte Form und überträgt ihn in den Arbeitsbereich (user work area - UWA) des Anwendungsprogrammes. In der UWA ist ein Speicherbereich für diesen Satztyp reserviert.

(9) Das DBMS hinterlegt Status-Information über den Ausgang der Operation in einem speziellen Bereich der UWA. Diese Status-Information ist dem Anwendungsprogramm zugänglich.

(10) Das Anwendungsprogramm verarbeitet den Satz. (Wir betrachten hier nicht die Abläufe auf der Sprachebene, wenn - wie in diesem Beispiel angedeutet - SQL in ein Programm in einer klassischen Programmiersprache eingebettet ist; s. unter relationale Datenbanken).



UWA = User Working Area
SI = Status-Information

Bild 2.3: Funktionsweise des DBMS

Die genaue Verteilung der Aufgaben zwischen Betriebssystem und DBMS ist in den verschiedenen Systemen unterschiedlich. Oft verfügt das Betriebssystem über einen File-Manager, der den Zugriff auf Sätze übernimmt. Vielfach sind jedoch diese File-Manager für die Belange eines DBMS nicht geeignet, so dass das DBMS selbst diese Funktion mit übernimmt. Für unsere Betrachtung der prinzipiellen Abläufe ist dies jedoch nicht von Bedeutung.

Weitere Aufgaben des DBMS

Neben der Aufgabe, Zugriffswünsche von Anwendungsprogrammen auszuführen, hat das DBMS folgende weitere Aufgaben:

- **Datendefinition**

Das DBMS muss Datendefinitionen (externe Schemata, das konzeptuelle Schema, das interne Schema, die zugehörigen Transformationsregeln) in den zugehörigen DDLs akzeptieren und interpretieren können. Es erzeugt aus der Quellform (Source) eine interne Darstellung.

Die Datendefinitionen müssen übrigens auch für den Benutzer der Datenbank abfragbar sein, d.h. das DBMS sollte nicht nur Anfragen über die „rohen“ Daten beantworten können, sondern auch über die Definition der Daten in der Datenbank (welche Objekttypen gibt es, was sind die Attribute, ...). Die Definitionen der Daten werden oft auch als *Meta-Daten* bezeichnet.

Meta-Daten

Die Datendefinitionen werden im *Katalog* oder auch *Data Dictionary* des DBMS zusammengefasst (s. Abschnitt 2.6).

- **Integrität der Datenbank**

Das DBMS sollte in möglichst großem Umfang in der Lage sein, Integritätsverletzungen zu verhindern (es verwehrt beispielsweise Änderungen von Daten, die vorgegebenen Integritätsbedingungen widersprechen)

- **Datensicherung (Recovery)**

Nach dem Auftreten von Fehlern muss das DBMS in der Lage sein, die Datenbank wieder in einen konsistenten Zustand zu versetzen. Fehler dieser Art sind Abbrüche von Anwendungsprogrammen, Systemzusammenbruch, Plattenfehler, etc.

- **Koordination gleichzeitig auf der Datenbank arbeitender Benutzer**

Auf großen Datenbanken arbeiten im allgemeinen viele Benutzer gleichzeitig. Das DBMS muss dafür sorgen, dass sich die parallel arbeitenden Programme nicht gegenseitig stören oder infolge unkoordinierter Parallelarbeit die Integrität der Datenbank zerstören.

- **Schutz der Daten gegen unberechtigten Zugriff**

Hierzu gehören alle technischen Maßnahmen zum *Datenschutz*, d.h. zum Schutz der Daten gegen Missbrauch jeglicher Art. Das DBMS muss dafür sorgen, dass nur befugte Benutzer Daten lesen, löschen und verändern können. Da in einer Datenbank sehr viele Daten integriert vorhanden sind, spielen Datenschutzmaßnahmen in Datenbanksystemen eine besondere Rolle. Ein einfaches Beispiel mag dieses verdeutlichen: in einem Unternehmen dürfen verschiedene Personen Information über Angestellte und deren Gehälter sehen (Abteilungsleiter, Lohnbüro, Revision, usw.), das Recht zu Änderungen muss aber einem sehr kleinen Personenkreis vorbehalten bleiben. Andere Personen dürfen lediglich die Gehälter alleine sehen, nicht aber, welcher Angestellte welches Gehalt verdient. Wieder andere Personen dürfen nichts über die Gehälter erfahren.

2.6 Weitere Komponenten eines Datenbanksystems

Tools

Tools = Werkzeuge

Zu den verschiedenen DBMS bieten Hersteller und unabhängige Häuser (third-party software vendors) unterschiedliche Tools oder Werkzeuge an, die die Entwicklung von Anwendungen auf Datenbanken erleichtern. Dabei gibt es sowohl Tools, die den Anwendungsprogrammierer unterstützen, als auch solche, die es dem Endbenutzer ermöglichen, Anwendungen zu erzeugen, ohne konventionell zu programmieren. Ein Beispiel hierfür sind Abfragesysteme, die es auch dem Nicht-

Fachmann erlauben, ad-hoc Anfragen an das System zu richten, meist unterstützt mit einer graphischen Oberfläche.

Typische Tools sind

- Abfragesysteme
- Report-Writer
- Spreadsheets und Business Graphics Tools
- Tools für den Datenbankentwurf
- 4GL Entwicklungsumgebungen. 4GL (*fourth generation language*, Datenbanksprachen der vierten Generation) integrieren imperative Sprachkonzepte mit SQL. Sie unterstützen typischerweise die interaktive Programmierung von Menüs, Masken und Formularen mit dahinterliegenden Prozeduren (Auslöseregeln und Aktionen).
- CASE Tools (computer aided software engineering) für den Entwurf von Datenbankanwendungen

Die verfügbaren Tools bestimmen wesentlich den Aufwand für die Erstellung und die Pflege von Anwendungen. Sie sind deshalb wichtige Faktoren bei der Auswahl eines DBMS.

Utilities

Utilities („nützliche Sachen“) sind Hilfsprogramme für den Datenbankadministrator. Ebenso wie bei den Tools werden Utilities sowohl von den Datenbankherstellern selbst als auch von unabhängigen Häusern angeboten. Typische Utilities sind:

- Laderoutinen (für das erstmalige Laden der Datenbank),
- statistische Routinen,
- Routinen zur Fehleranalyse,
- Routinen zur Reorganisation (der Daten auf den Speichern),
- Kopier- und Archivierungsroutinen.

Data Dictionary und Repository

Das Data Dictionary (Katalog) hat zwei Funktionen:

- Es dient dem DBMS zur Speicherung der Daten zur Verwaltung der Datenbank (Schema-Informationen, Sichten, Zugriffsrechte, Informationen zur Optimierung von Anfragen wie etwa Statistiken usw.).
- Es dient dem Anwendungsprogrammierer zur Suche nach Informationen über gespeicherte Daten und deren Struktur (Schema-Informationen) sowie zur Analyse bei Leistungsproblemen.

Damit ist das Data Dictionary eine Datenbank für sich selbst. Entsprechend sind Data Dictionaries bei modernen Systemen im gleichen Datenmodell realisiert wie die Datenbank selbst (also z.B. ebenfalls in Form von Tabellen oder Relationen) und können damit in gleicher Weise abgefragt werden (also z.B. mittels SQL).

Repositories

Für die Entwicklung und Wartung von datenbankgestützten Informationssystemen sind sehr viel mehr Informationen von Bedeutung als die Datendefinitionen. Entsprechend werden Data Dictionaries heute mehr und mehr zu *Repositories*, in denen alle wesentlichen Informationen über die Daten, Programme und Benutzer des Informationssystems gespeichert werden. Insbesondere kommen Entwurfs- und Anwendungsinformationen hinzu. Welche Informationen ein Data Dictionary oder ein Repository verwaltet und welche Funktionen es anbietet, ist von Hersteller zu Hersteller sehr verschieden. Ein modernes Data Dictionary System wird etwa folgende Informationen verwalten:

- Beschreibungen der Daten
- Angaben zu den Beziehungen zwischen den Daten
- Beschreibungen der Programme (Transaktionen)
- Angaben darüber, welche Programme welche Daten nutzen
- Konsistenzbedingungen
- Angaben über Zugriffsbefugnisse
- Entwurfsdaten (grafische konzeptuelle Modelle, Dokumentation der Entwurfsschritte usw.)
- Verantwortlichkeiten
- Entwurfsdokumente, Quell-Code zu Anwendungsprogrammen

Jeder Mitarbeiter in einem Software-Entwicklungsprojekt muss sich an die im Repository abgelegten Vereinbarungen halten; neue, global wirkende Vereinbarungen müssen im Repository hinterlegt werden. Nur mit Steuerinstrumenten dieser Art sind große Projekte handhabbar.

Repositories können eigenständige Produkte sein, werden jedoch häufig auch als integraler Bestandteil von DBMS angeboten.

2.7 Datenunabhängigkeit

Wir hatten bereits diskutiert, dass Datenunabhängigkeit eine der praktisch wichtigsten Konsequenzen des Datenbankansatzes ist: Anwendungsprogramme bleiben unberührt von Änderungen auf der internen und der konzeptuellen Ebene, solange die Bedeutung der Daten, mit denen das Anwendungsprogramm arbeitet, durch diese Änderungen nicht betroffen wird. Dies verbessert geradezu dramatisch die Wartbarkeit und die Entwicklungsmöglichkeiten des gesamten Informationssystems. Die dreischichtige Betrachtungsweise der Daten ist der Schlüssel

zur Datenunabhängigkeit. Änderungen innerhalb einer Ebene können in gewissem Umfang von den übrigen Ebenen ferngehalten werden, indem man die Änderungen durch die zwischengelagerten Transformationsregeln auffängt. Man unterscheidet logische Datenunabhängigkeit und physische Datenunabhängigkeit.

physische Datenunabhängigkeit

Physische Datenunabhängigkeit bedeutet Isolierung der Anwendungsprogramme von Änderungen der physischen Datenorganisation. Diese Form der Datenunabhängigkeit wird in der Drei-Schichten-Architektur dadurch gewährleistet, dass Änderungen des internen Modells das konzeptuelle Modell unberührt lassen: die Änderungen werden durch Modifikation der Transformationsregeln aufgefangen. Da das konzeptuelle Modell unberührt bleibt, bleiben auch die externen Modelle und damit alle Benutzerprogramme von Änderungen der Datenorganisation unberührt. So führen also Änderungen von Dateiorganisationen, das Umstrukturieren von Sätzen (z. B. Zerlegen von großen Sätzen in separate Teilsätze), das Anlegen von Indexen, usw. nicht mehr zu Änderungen bestehender Programme.

logische Datenunabhängigkeit

Logische Datenunabhängigkeit bedeutet Isolierung der Anwendungsprogramme von Änderungen des konzeptuellen Modells. Beispielsweise können neue Anwendungen dazu führen, dass im konzeptuellen Modell Objekttypen zusätzliche Attribute erhalten, dass neue Beziehungstypen eingeführt werden müssen, usw. In allen diesen Fällen müssen die bestehenden externen Modelle nicht verändert werden, d.h. die vorhandenen Programme bleiben unberührt. Lediglich die Transformationsregeln externes/konzeptuelles Modell müssen möglicherweise abgeändert werden.

Heutige DBMS bieten unterschiedliche Grade an Datenunabhängigkeit. Praktisch alle Systeme weisen jedoch noch kleinere oder größere Schwächen in diesem Punkte auf.

Der Begriff der Datenunabhängigkeit muss auch unter dem Gesichtspunkt des Bindens gesehen werden. Im Falle des Bindens zur Übersetzungszeit bleibt das Anwendungsprogramm nach Änderungen des internen oder konzeptuellen Schemas unverändert, muss aber neu übersetzt werden. Man spricht von *statischer Datenunabhängigkeit*. Im Falle des Bindens zur Zugriffszeit ist auch diese Abhängigkeit aufgehoben, man spricht von *dynamischer Datenunabhängigkeit.*

statische und dynamische Datenunabhängigkeit

Die Forderung nach Datenunabhängigkeit bezieht sich primär darauf, dass Programme bei Änderungen in der Datenbank nicht umgeschrieben werden müssen. Neuübersetzung ist oft vertretbar. Dynamische Datenunabhängigkeit ist mit hohen Kosten verbunden, bedingt sowohl durch die notwendigen Zugriffe auf die Schemata und Tranformationsregeln (E/A-Vorgänge) als auch durch die Interpretation der entsprechenden Einträge zur Laufzeit (CPU-Zeit).

2.8 Ein Datenmodell für die konzeptuelle Ebene

Wir hatten gesehen, dass das konzeptuelle Modell der Eckpunkt in der Entwicklung eines datenbankgestützten Informationssystems ist. Um die Vorstellung über die konzeptuelle Ebene etwas zu konkretisieren, gehen wir am Schluss dieser Kurseinheit auf eine Methode zur Modellierung ein.

Datenmodell

Die Beschreibung der Datenwelt des Unternehmens erfolgt in den Begriffen eines *Datenmodells*, d.h. es sind gewisse Konstrukte oder Bauelemente vorgegeben, mit deren Hilfe das spezielle konzeptuelle Modell erstellt werden muss. Das Datenmodell muss mächtig genug sein, um alle wichtigen Aspekte der Realwelt beschreiben zu können, zugleich muss es möglich sein, in einfacher Weise eine effiziente Implementierung auf der internen Ebene abzuleiten.

Die heute standardmäßig benutzten relationalen Datenbanksysteme bieten mit relationalen Modell kein befriedigendes Datenmodell bzw. die dazugehörige Datenbeschreibungssprache für das konzeptuelle Modell. Neuere Modelle sind das *objektrelationale* und das *objektorientierte Datenmodell*. Diese Datenmodelle sind ausreichend, um für die üblichen Realweltsituationen in Unternehmen Datenbanken aufbauen und betreiben zu können; sie sind jedoch nicht reich genug, um alle wichtigen und nützlichen Informationen über die betrachtete Realwelt festhalten zu können.

Praktisch bedeutet dies, dass die Realwelt zunächst mittels eines Datenmodelles beschrieben wird, das nicht unmittelbar vom Datenbanksystem unterstützt wird. Aus der semantisch „reichen“ Beschreibung der Realwelt wird dann ein semantisch wesentlich „ärmeres“ konzeptuelles Modell für das Datenbanksystem abgeleitet (z.B. Relationen oder Netze).

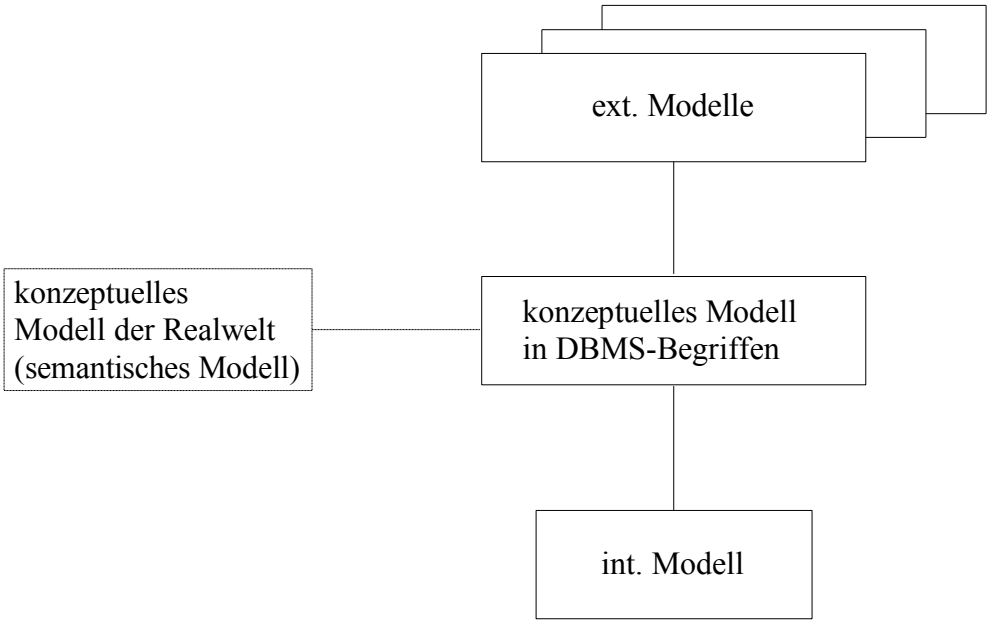


Bild 2.4 : Semantisches konzeptuelles Modell und konzeptuelles Modell in Begriffen des gegebenen DBMS

In der Literatur wurde eine ganze Reihe von Modellen für die konzeptuelle Ebene vorgeschlagen. Wegen ihrer gegenüber gängigen Datenmodellen von Datenbanksystemen sehr viel reicheren Modellierungsmöglichkeiten werden solche Modelle häufig *semantische Datenmodelle* genannt (weil sie erlauben, mehr Information über die *Bedeutung* der Daten festzuhalten).

Wir betrachten im Folgenden eines der bekanntesten Modelle für die konzeptuelle Ebene, das Entity-Relationship-Modell. Dieses Modell ist zugleich praktisch das

wichtigste: es liegt vielen Tools für Datenbankentwurf und Anwendungsentwicklung (4GL) zugrunde.

Das Entity-Relationship-Modell (ER-Modell)

Der ursprüngliche Vorschlag für ein Entity-Relationship-Modell stammt von Chen /CHEN76/. Das Modell wurde vielfach erweitert, insbesondere wurden Abstraktionsmechanismen hinzugefügt.

Das ER-Modell kennt folgende Basiskonstrukte

- Entity-Typ (entity type)
- Beziehungstyp (relationship type)
- Entity-Typen und Beziehungstypen haben Attribute
- Zu jedem Typ gibt es dann beliebig viele Instanzen, also Entities und Beziehungen.

Entities

Entity

Die Modellbildung besteht darin, dass wir gewisse Dinge der realen Welt als Objekte, sogenannte *Entities*, ansehen, zwischen denen gewisse Beziehungen bestehen. Es ist Aufgabe des Datenbankadministrators, im Einzelfalle zu beschließen, was eine Entity ist. Als Entity wird man im allgemeinen solche Dinge ansehen, die in der realen Welt allein für sich identifizierbar sind. In einem Unternehmen wären Entities etwa die einzelnen Mitarbeiter, die einzelnen Projekte, die verschiedenen Abteilungen, die einzelnen Artikel, usw.

Entity-Typ

Natürlich betrachtet man beim Modellieren nicht einzelne Entities, sondern *Entity-Typen*. Ein Entity-Typ repräsentiert die Menge aller Entities, die die gleichen charakteristischen Eigenschaften besitzen. Mit der obigen Aufzählung der Entities haben wir eine solche Typbildung schon eingeschlossen: Entity-Typen sind gerade MITARBEITER, PROJEKTE, usw.

Typ und zugehörige Menge von Instanzen

Beachten Sie, dass bei dieser (für die Datenbankwelt typischen) Denkweise ein Typ zugleich als Behälter aller seiner Instanzen gesehen wird; also: der Typ MITARBEITER beschreibt nicht nur die charakteristischen Eigenschaften eines Mitarbeiters (Struktur) sondern steht zugleich für die Menge aller Mitarbeiter von diesem Typ. Es stehen also Entity-Typen zugleich für Mengen von Entities, und wir schreiben deshalb auch $e \in E$ statt „Entity e vom Typ E“ (Es ist durchaus vorstellbar, Typ und Menge von Objekten zu trennen; dieses Konzept wird jedoch lediglich in wenigen Prototypen objektorientierter Datenbanken verfolgt. Wir gehen an dieser Stelle hierauf nicht ein.).

Beziehungen

Beziehungen

Entities können untereinander in Beziehung stehen. Beispielsweise kann die Entity „Kurt Müller“ in der Beziehung „ist-Vater-von“ zu der Entity „Peter Müller“ stehen. Ein anderes Beispiel ist die Beziehung zwischen einem bestimmten Mitarbeiter und einem Projekt, an dem er mitarbeitet. Genauso wie bei Entities betrachten wir nicht einzelne Beziehungen, sondern *Beziehungstypen*. Beispielsweise existiert für die Entity-Typen ANGESTELLTER und PROJEKT der Beziehungstyp IST-MITARBEITER-AN.

Beziehungstyp

Besteht also ein Beziehungstyp B zwischen den Entity-Typen E_1 und E_2, so bedeutet dies, dass Entities vom Typ E_1 in Beziehung B zu Entities vom Typ E_2 stehen können. B steht also wiederum für eine Menge von Beziehungen. Ein Beziehungstyp kann mehrere Entity-Typen umfassen; wir schreiben

$B (E_1, E_2, ... E_k)$.

Jedes k-Tupel $(e_1, e_2,..., e_k)$ in der Menge B beschreibt eine konkrete Beziehung zwischen den Entities $e_1, ..., e_k$. Die Entity-Typen in der Liste $E_1, ..., E_k$ müssen nicht unterschiedlich sein.

Beispiel 2.2:

Die Tatsache, dass Personen miteinander verheiratet sein können, wird erfasst durch den Beziehungstyp VERHEIRATET(PERSON,PERSON).

Beispiel 2.3:

Die Tatsache, dass Lieferanten bestimmte Bauteile für bestimmte Projekte liefern, können wir erfassen durch die Beziehung

B (LIEFERANT, BAUTEIL, PROJEKT).

Ein Tupel (L1, B1, P1) von B besagt, dass L1 das Bauteil B1 für Projekt P1 liefert.

In der praktischen Anwendung ist die Mehrzahl der Beziehungen zweistellig. Man beachte, dass Beziehungen nicht gerichtet sind; d.h. gilt $B(E_1, E_2)$, so erfasst B für $e_1 \in E_1$ alle zugehörigen $e \in E_2$, liefert aber ebenso für $e_2 \in E_2$ alle zugehörigen $e \in E_1$.

Attribute

Attribut

Wertebereich

Jeder Entity-Typ besitzt *Attribute*. Attribute des Entity-Typs Angestellter sind z.B. Name, Personalnummer, Beruf, Gehalt usw. Jedes Attribut kann Werte aus einem bestimmten *Wertebereich* annehmen. Der Wertebereich für das Attribut GEHALT ist beispielsweise die Menge der reellen Zahlen x, $0 < x < 10000$. Für einen bestimmten Angestellten gilt etwa:

Name:	Meyer, Hans
Personalnummer:	107
Beruf:	Systemanalytiker
Gehalt:	3000
.	

Die Entity „Angestellter Meyer“ wird durch die Kombination seiner Attributwerte beschrieben.

Attribute für Beziehungstypen

Analog zu den Entity-Typen können auch einem Beziehungstyp Attribute zugeordnet sein. Wenn in einem Unternehmen ein Angestellter gleichzeitig an mehreren Projekten mitarbeiten kann, so wird es beispielsweise sinnvoll sein, dem Beziehungstyp IST-MITARBEITER-AN das Attribut PROZENT-DER-ARBEITSZEIT zuzuordnen. Für die Beziehung „Meyer bearbeitet Projekt P10“ besagt der Wert dieses Attributes dann beispielsweise, dass Meyer mit 50% seiner Arbeitszeit am Projekt P10 arbeitet.

Schlüssel, Primärschlüssel

Wir hatten gesagt, dass Entities voneinander unterscheidbare Einheiten sind. Es gibt also ein Attribut oder eine Menge von Attributen für jeden Entity-Typ, deren Werte jede Entity identifizieren. Ein solches Attribut oder eine solche Menge von Attributen heißt *Schlüssel* für diesen Entity-Typ. Ein Schlüssel für den Entity-Typ ANGESTELLTER ist beispielsweise die Attributmenge (NAME, GEBURTSDATUM). Gelegentlich sind solche Schlüssel ungeeignet, beispielsweise müsste man in vielen Anwendungen zu (NAME, GEBURTSDATUM) weitere Attribute hinzufügen, um einen Schlüssel zu erhalten. Aus diesem Grunde werden oft künstliche Attribute eingeführt, die als Schlüssel verwendet werden sollen. Typisch ist etwa die Verwendung von Personalnummern in Betrieben.

Primärschlüssel

Gibt es mehrere Schlüssel für einen Entity-Typ, so wird im allgemeinen einer von ihnen als *Primärschlüssel* ausgezeichnet.

Komplexität von Beziehungstypen

Von großer praktischer Bedeutung sind Aussagen über die Art des Beziehungstyps: kann eine Entity vom Typ E_1 nur mit einer oder mit vielen Entities vom Typ E_2 in Beziehung stehen, usw. Im ER-Modell werden deshalb Beziehungstypen durch ihre Komplexität charakterisiert: Die einfachste Form der Beziehung zwischen zwei Entity-Typen E_1 und E_2 ist die *1:1-Beziehung*. Jeder Entity $e \in E_2$ ist höchstens eine Entity $e' \in E_1$ zugeordnet, jeder Entity $e \in E_1$ höchstens eine Entity $e' \in E_2$. Ein Beispiel für einen 1:1-Beziehungstyp ist VERHEIRATET.

1:1-Beziehung

Häufiger ist der Beziehungstyp der Art n:1. Jede Entity e vom Typ E_2 steht in Beziehung mit beliebig vielen (keinem, einem oder mehreren) Entities vom Typ E_1, aber jede Entity vom Typ E_1 steht in Beziehung zu höchstens einer Entity vom Typ E_2. Ein solcher Beziehungstyp heißt *n:1 von E_1 nach E_2*. Gehört jeder Mitar-

n:1-Beziehung

beiter einer Firma zu höchstens einer Abteilung, so ist der Beziehungstyp IST-ANGEHÖRIGER-VON vom Typ *n:1* von MITARBEITER zu ABTEILUNG.

n:m-Beziehung

Bei einem *n:m-Beziehungstyp* können einer Entity vom Typ E_1 beliebig viele Entities vom Typ E_2 zugeordnet sein und einer Entity vom Typ E_2 können beliebig viele Entities vom Typ E_1 zugeordnet sein. Dies ist die komplizierteste Beziehung, sie kann nicht in allen Datenmodellen unmittelbar ausgedrückt werden. Ein Beispiel für einen *n:m*-Beziehungstyp ist IST-MITARBEITER-AN zwischen Projekten und Mitarbeitern. Ein Mitarbeiter kann an mehreren Projekten mitarbeiten, an einem Projekt arbeiten meist mehrere Mitarbeiter mit.

Schwache Entity-Typen

Es kommt vor, dass eine Entity nicht durch ihre eigenen Attribute allein identifiziert werden kann, sondern dass die eindeutige Identifizierung die Ausnutzung einer Beziehung erfordert. Ein Beispiel: Über die Kinder der Angestellten eines Unternehmens sollen die Attribute NAME und ALTER abgespeichert werden. Die Kinder sind eigene Entities, zur Identifizierung ist jedoch die Personalnummer des Vaters notwendig. Mit anderen Worten: die Identifizierung ist möglich über den Beziehungstyp VATER-VON, der den Entity-Typ ANGESTELLTE mit dem Entity-Typ KIND verbindet.

schwache Entity- und Beziehungstypen

Entity-Typen, die - wie der Typ KIND - nur in Beziehung mit anderen Entity-Typen identifizierbar sind, nennt man *schwache Entity-Typen.* Beziehungstypen, an denen ein schwacher Entity-Typ teilnimmt, heißen *schwache Beziehungstypen*.

Graphische Darstellung

Das ER-Modell wird graphisch in folgender Weise dargestellt:

1. Rechtecke repräsentieren Entity-Typen. Neben dem Namen des Entity-Typs können seine Attribute angegeben werden. Wo der Platz in den Rechtecken nicht ausreicht, werden die Attribute als Kreise gezeichnet und mit ungerichteten Kanten mit den zugehörigen Rechtecken verbunden.

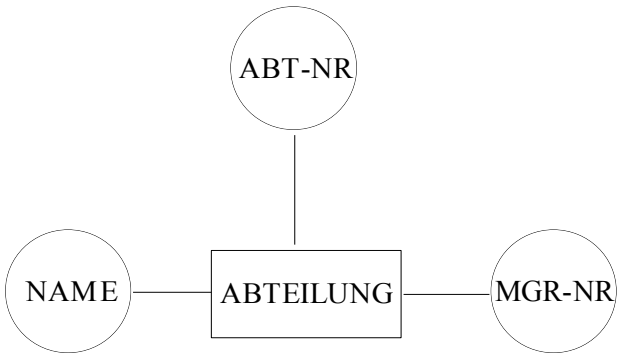


Bild 2.5:
Entity-Typ ABTEILUNG mit den Attributen ABT-NR (Abteilungsnummer), NAME, MGR-NR (Managernummer)

2. Rauten repräsentieren Beziehungstypen. Sie werden durch ungerichtete Kanten mit den zugehörigen Entity-Typen verbunden.

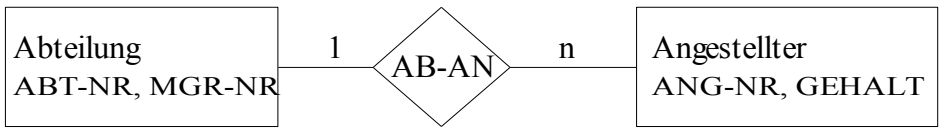


Bild 2.6:
Beziehungstyp AB-AN zwischen ABTEILUNG und ANGESTELLTER

Durch die Angabe von 1, n oder m an der jeweiligen Kante kann die Komplexität des Beziehungstyps ausgedrückt werden. Attribute können wie bei Entity-Typen in die Raute hineingeschrieben oder als Kreise dargestellt werden.

3. Doppelt umrandete Rechtecke bezeichnen schwache Entity-Typen.

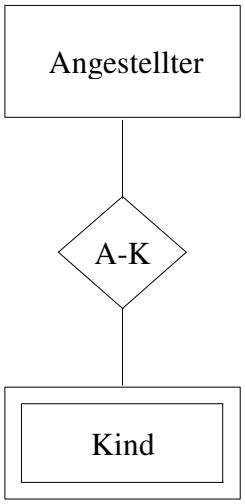


Bild 2.7: Schwacher Entity-Typ KIND und schwacher Beziehungstyp A-K

Bild 2.8 zeigt ein Beispiel für ein kleines konzeptuelles Modell in ER-Notation. Die Attribute der Entity-Typen sind nicht eingezeichnet. Der Beziehungstyp AB-AN gibt die Zugehörigkeit von Angestellten zu Abteilungen an. Der Beziehungstyp PROJ-LEIT erfasst die Leitung von Projekten; PROJ-LEIT ist 1:n von ANGESTELLTER zu PROJEKT, d.h. ein Angestellter kann mehrere Projekte leiten, während jedes Projekt nur einen Leiter besitzt. Der Beziehungstyp ANG-PROJ drückt die Zugehörigkeit von Angestellten zu Projekten aus; es handelt sich um eine n:m Beziehung. Der Beziehungstyp L-B-P verbindet drei Entity-Typen miteinander: ein Lieferant liefert ein bestimmtes Bauteil für ein bestimmtes Projekt.

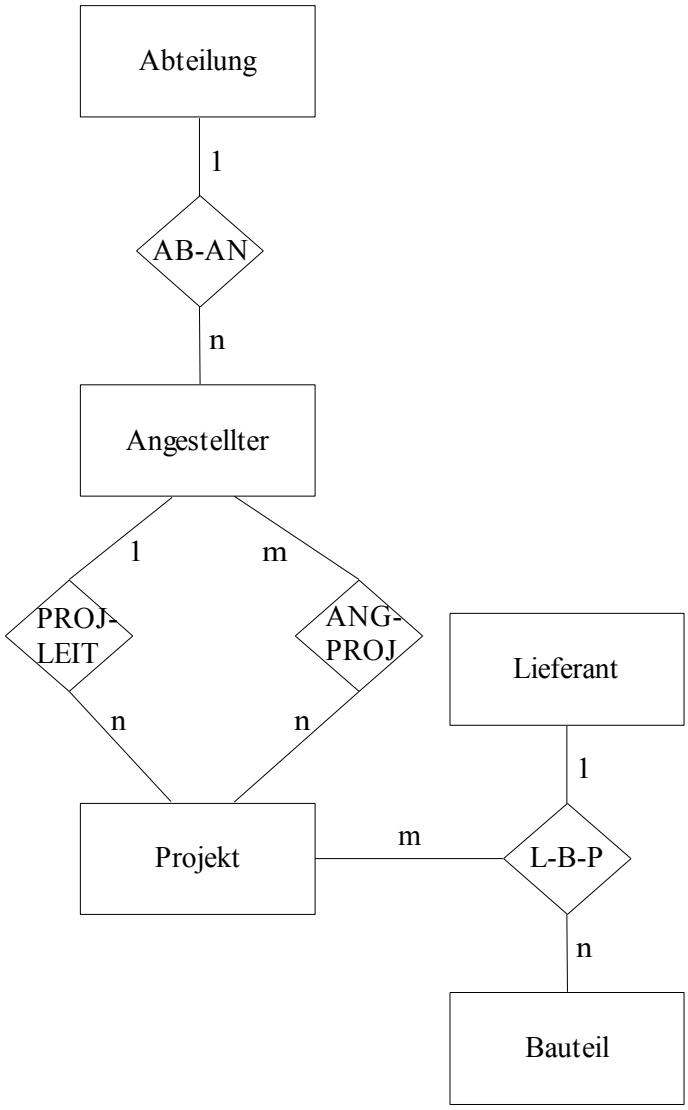


Bild 2.8: Ausschnitt aus einem konzeptuellen Modell in ER-Darstellung
(ohne Attribute)

Im konzeptuellen Schema werden die Entity-Typen und Beziehungstypen in programmiersprachen-ähnlicher Form beschrieben:

```
KONZ SCHEMA NAME = UNTERNEHMEN
    ENTITYTYP ANGESTELLTER
    ANG-NR  INT(4)
    NAME  CHAR(V)
    ADRESSE  CHAR(V)
    .
    .
    .
ENTITYTYP ABTEILUNG
    .
    .
    .
BEZIEHUNGSTYP AB-AN: ABTEILUNG, ANGESTELLTER
    TYP 1:n
BEZIEHUNGSTYP ANG-PROJ: ANGESTELLTER, PROJEKT
    TYP n:m
    .
    .
    .
```

Das konzeptuelle Schema erhält den Namen UNTERNEHMEN, für jedes Attribut der Entity-Typen und Beziehungstypen wird der Wertebereich spezifiziert. INT(4) steht für maximal vierstellige Integerzahl; CHAR(V) für Zeichenkette variabler Länge (nicht in allen kommerziellen Systemen möglich).

Erweiterungen des ER-Modells

Das klassische ER-Modell kann einige Aspekte der Daten auf konzeptueller Ebene nur unvollkommen beschreiben. Deshalb wurden verschiedene Erweiterungen eingeführt, von denen wir im Folgenden jedoch nur Generalisierung und Spezialisierung betrachten werden. Es gibt eine Reihe weiterer Modellierungskonzepte, die jedoch nicht allgemein akzeptiert sind oder genutzt werden.

Spezialisierung und Generalisierung (*is-a relationship*)

Entity-Mengen enthalten häufig Teilmengen, die aus Anwendungssicht von eigenem Interesse sind. Beispielsweise wollen wir unter den Mitarbeitern einer Firma zwischen Angestellten und freien Mitarbeitern unterscheiden. Freie Mitarbeiter und Angestellte sind in diesem Falle *Spezialisierungen* des Typs Mitarbeiter. Falls wir den Datenbankentwurf mit den Entity-Typen Angestellter und Freier Mitarbeiter begonnen hätten, so würden wir umgekehrt die beiden Typen zum Typ Mitarbeiter *generalisieren*, um in bestimmten Anwendungen nicht immer die Fallunterscheidung machen zu müssen. Offensichtlich gilt: Angestellte ebenso wie Freie Mitarbeiter sind Mitarbeiter, wir sagen deshalb: ANGESTELLTER *is-a* MITARBEITER bzw. FREIER MITARBEITER *is-a* MITARBEITER (Bild 2.9).

is -a

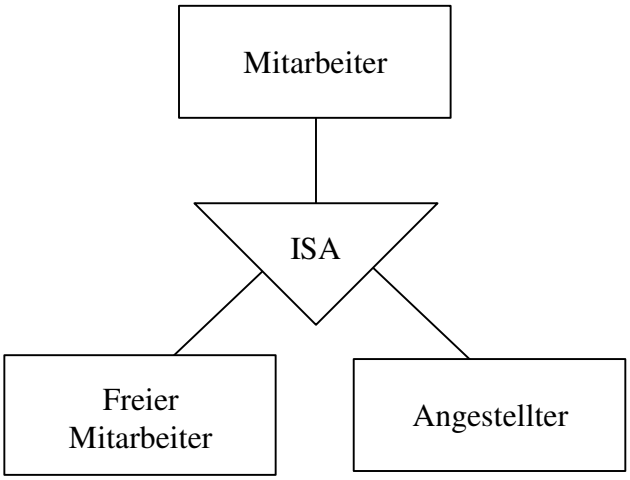


Bild 2.9: ER-Diagramm mit *is-a*-Relationship

Vererbung

Da jeder Angestellte (Freie Mitarbeiter) zugleich ein Mitarbeiter ist, hat er natürlich auch die Attribute des Mitarbeiters; wir sprechen von *Vererbung der Attribute*. Darüber hinaus kann der spezialisierte Entity-Typ weitere Attribute haben. Im Beispiel könnte für Freier Mitarbeiter in einem Attribut festgehalten werden „fortlaufend beschäftigt"/„nur zeitweise beschäftigt".

Im Falle der Generalisierung ist die Sichtweise umgekehrt: dort werden gemeinsame Eigenschaften in einem übergeordneten Typen zusammengefasst. Im Beispiel würden also die anfänglichen Entity-Typen Freier Mitarbeiter und Angestellter ihre gemeinsamen Attribute verlieren und an den übergeordneten Typ Mitarbeiter abgeben. Im Ergebnis erhalten wir dieselbe Struktur.

Partition

Das hier gewählte Beispiel ist eine *Partition:* Die Spezialisierung führt nämlich zu disjunkten Entity-Typen. Falls wir beispielsweise die Angehörigen einer Universität spezialisieren zu Mitarbeitern und Studierenden, so wäre diese Spezialisierung nicht disjunkt: ein Studierender kann zugleich studentische Hilfskraft sein, also auch Mitarbeiterstatus besitzen.

Die Partition unseres Beispiels ist *total*, es ist nämlich jeder Mitarbeiter entweder Angestellter oder Freier Mitarbeiter (andere Typen von Mitarbeitern gebe es nicht). Anders wäre die Situation, wenn wir beispielsweise beim Entity-Typ PKW nur die Spezialisierungen US-Automobile und Europäische Automobile herausgreifen würden; dies ist zwar eine Partition, sie ist aber nicht total (japanische Autos gehören keiner Spezialisierung an), sondern *partiell*.