

REFERAT

13. Juni 1991

Datenbanken

Wolfgang Bock
Markus Braun
Thomas Zimmermann

Inhaltsverzeichnis

I.	Warum Datenbanken	Seite	2
II.	Das Konzept der Datenbank	Seite	3
III.	Transaktionsverarbeitung und Sperren	Seite	5
IV.	Die Drei-Ebenen-Architektur der Datenbank	Seite	6
V.	Datenbankmodelle	Seite	8
	1. Hierarchisches Modell	Seite	10
	2. Netzwerkmodell	Seite	12
	3. Relationenmodell	Seite	15
	4. Vergleich der Datenbankmodelle	Seite	18
VI.	Isolierte, zentrale und verteilte Datenbanken	Seite	19
VII.	Datenbank-Maschinen	Seite	23
VIII.	Stand und Entwicklung der Datenbanktechnik	Seite	27
IX.	Objektorientierte Datenbanken	Seite	29
X.	Literaturverzeichnis	Seite	31
XI.	Anhang		
	- Marktübersicht: Datenbanken für PC		
	- Übersicht installierter relationaler Datenbanken		

I. Warum Datenbanken

Herkömmliche Datenhaltung ist programmbezogen. Im Extremfall hat jedes Programm seine eigenen Dateien, die dann natürlich speziell auf die Belange dieses Programms zugeschnitten sind.

Programmbezogene Datenhaltung war angemessen, als sich die DV noch auf wenige Aufgaben der Massenverarbeitung beschränkte.

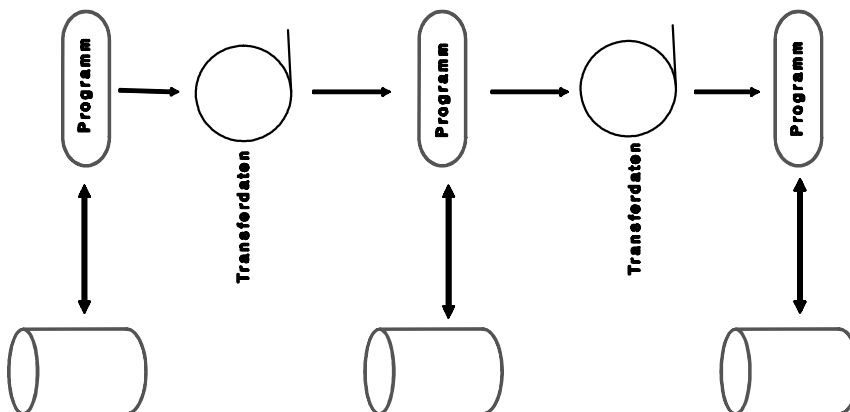
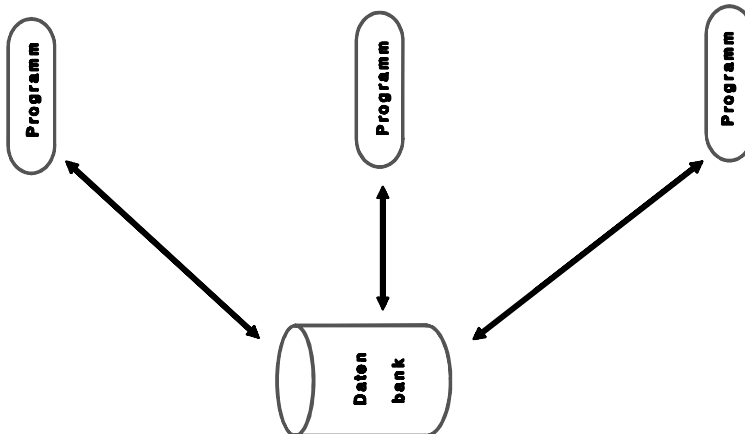
Nachdem aber immer mehr Funktionen von der DV erledigt wurden, ergaben sich folgende Probleme:

1. Redundanz (Mehrfachspeicherung)
2. Dateninkonsistenz
3. Brückenprogramme und Transferdateien
4. Hohe Durchlaufzeit der Informationen
5. Keine Mehrfachverwendung der Dateien möglich

(gleichzeitig)

6. Maßnahmen zum Schutz der Daten müssen in allen Programmen des Anwendungssystems einzeln realisiert werden

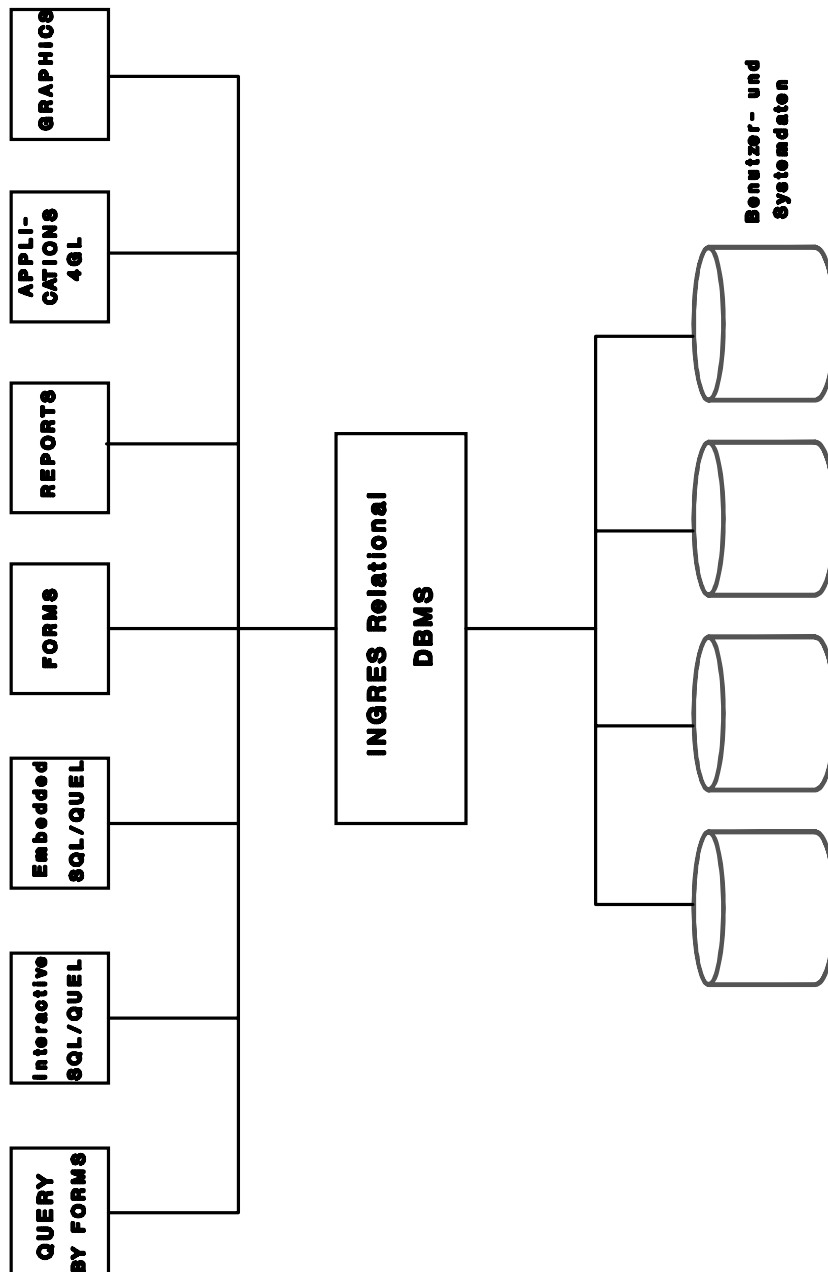
Die Lösung dieser Probleme bildet das **Konzept der Datenbank**, in der alle Daten des Anwendungssystems zusammengefaßt und nach einheitlichen Regeln abgespeichert werden (Datenintegration).



II. Das Konzept der Datenbank

Ein **Datenbanksystem (DBS)** ist die Software, die zum Betreiben einer **Datenbank** benötigt wird. Die **Datenbank** besteht aus der Software und der **Datenbasis**.

Der wichtigste Teil der Software ist das **Datenbankverwaltungssystem (Data Base Management System, DBMS)**, über das alle Anforderungen der Anwendungsprogramme an die Datenbank laufen. Befinden sich Datenbasis und DBMS oder Teile davon auf verschiedenen Rechnern eines Rechnernetzes, so spricht man von einer **verteilten Datenbank (s. Kap VI)**. Die Datenbanken der einzelnen Rechner bilden dann einen **Datenverbund**. Ein Benutzer der Datenbank hat im allgemeinen keine Kenntnis darüber, welche Daten auf welchem Rechner gespeichert sind.



Die Vorteile der Datenbank sind vor allem:

1) Verminderung der Datenredundanz und schnellere Aktualisierung der Daten

Jedes Datum wird nur einmal in der Datenbank gespeichert. Kopien von Daten werden einzig und allein aus Sicherheitsgründen gehalten und nicht etwa, weil zwei verschiedene Anwendungsprogramme das Datum in zwei verschiedenen Darstellungen benötigen. In diesem Fall transformiert das DBMS das Datum in die jeweils benötigte Form.

2) Einhaltung der Datenintegrität

Die Zentralisierung der Kontrolle erlaubt eine einfachere Überprüfung der Daten auf Korrektheit und Vollständigkeit (**Datenkonsistenz**). Irrtümliche, absichtliche oder durch Fehler hervorgerufene Verfälschungen von Daten können durch gelegentlich auszuführende Prüfprogramme einfacher entdeckt werden.

3) Verbesserter Schutz der Daten vor unberechtigtem Zugriff

Grundsätzlich fördert der einheitliche kontrollierende Zugang aller Anwender zur Datenbasis die Einhaltung von Datenschutzvorschriften. Durch die Integration und Zentralisierung der Daten verbunden mit der Möglichkeit, auf jedes beliebige Datum in Sekundenschnelle zugreifen und Querbeziehungen zwischen Daten herstellen zu können, entstehen neue Einsatzbereiche, aber auch neue Datenschutzprobleme (Rasterfahndung nach Rasterman).

4) Datenunabhängigkeit

Die Änderung der physikalischen Organisation der Daten erfordert keine Änderung der Anwendungsprogramme.

III. Transaktionsverarbeitung und Sperren

Datenbanktransaktionen sind elementare Arbeitseinheiten (z.B. bestimmte Programmteile/Befehlsfolgen), die bei einem korrekten Zustand der Datenbank beginnen und mit einem korrekten Zustand enden.

Transaktionen helfen, die Richtigkeit der Daten auf Datenbanken zu bewahren. Damit das DBMS weiß, wo eine Transaktion beginnt und wo sie endet, werden in die Programme entsprechende Befehle aufgenommen (z.B. begin-transaction und end-transaction). Muß die Transaktion dann unterbrochen werden, so werden alle Datenbankoperationen, die seit dem Befehl "begin-transaction" durchgeführt wurden, zurückgesetzt.

Mit dem Transaktionskonzept ist das des Sperrens (Locking) eng verbunden. Das Sperren bestimmter Daten ist speziell beim konkurrierenden Zugriff mehrerer Benutzer ein Mittel, die Datenbankzugriffe zu koordinieren.

Wenn ein Programm selbst nur Daten lesen, nicht aber verändern möchte, besteht kein Grund, andere Programme davon abzuhalten, diese Daten ebenfalls zu lesen. Dementsprechend gibt es bei den meisten Datenbanksystemen (DBS) die Möglichkeit, Reservierungen entweder als exklusiv (exclusive lock) oder als teilbar (shared lock) durchzuführen.

Eine exklusive Sperrung ist sinnvoll, wenn die Daten vom Programm verändert werden sollen und kann nur erworben werden, wenn keinerlei Reservierung vorliegt. Sie sperrt alle Zugriffe von anderen Programmen auf diese Daten.

Sollen die Daten nur gelesen werden, erwirbt das Programm eine teilbare Reservierung (shared lock). Andere Programme, die selbst auch nur lesen wollen, können dann ebenfalls teilbare Reservierungen erwerben.

Die Sperrung kann Hand in Hand mit Beginn und Ende der

Transaktion durchgeführt werden. Um andere Transaktionen nicht zu verzögern, versucht man aber meist, die Zeit der Sperrung möglichst gering zu halten.

IV. Die Drei-Ebenen-Architektur der Datenbanken

Bei konventioneller Datenhaltung ist es Aufgabe der Softwareentwickler, neben dem Inhalt der Dateien auch ihre Organisationsform festzulegen. Werden Änderungen im Programm notwendig, dann werden Programm und Dateiorganisation parallel vom Softwareentwickler angepaßt.

In der Datenbanktechnik sind solche Eingriffe nicht mehr erwünscht, und zwar vor allem aus zwei Gründen:

1. Änderungen an Daten, die auch von anderen Programmen benutzt werden (Mehrfachnutzung), könnten die Funktionsfähigkeit dieser Programme beeinträchtigen, und
2. die gemeinsame Datenhaltung für viele Anwendungen ist viel komplizierter als die für ein einziges Anwendungsprogramm. Daher wird der Programmierer im DBS von den technischen Details der Datenhaltung entlastet und diese Aufgaben dem DBS übertragen. Der Anwender soll nur noch formulieren müssen, welche Daten er benötigt und nicht, wie und wo diese Daten zu finden sind.

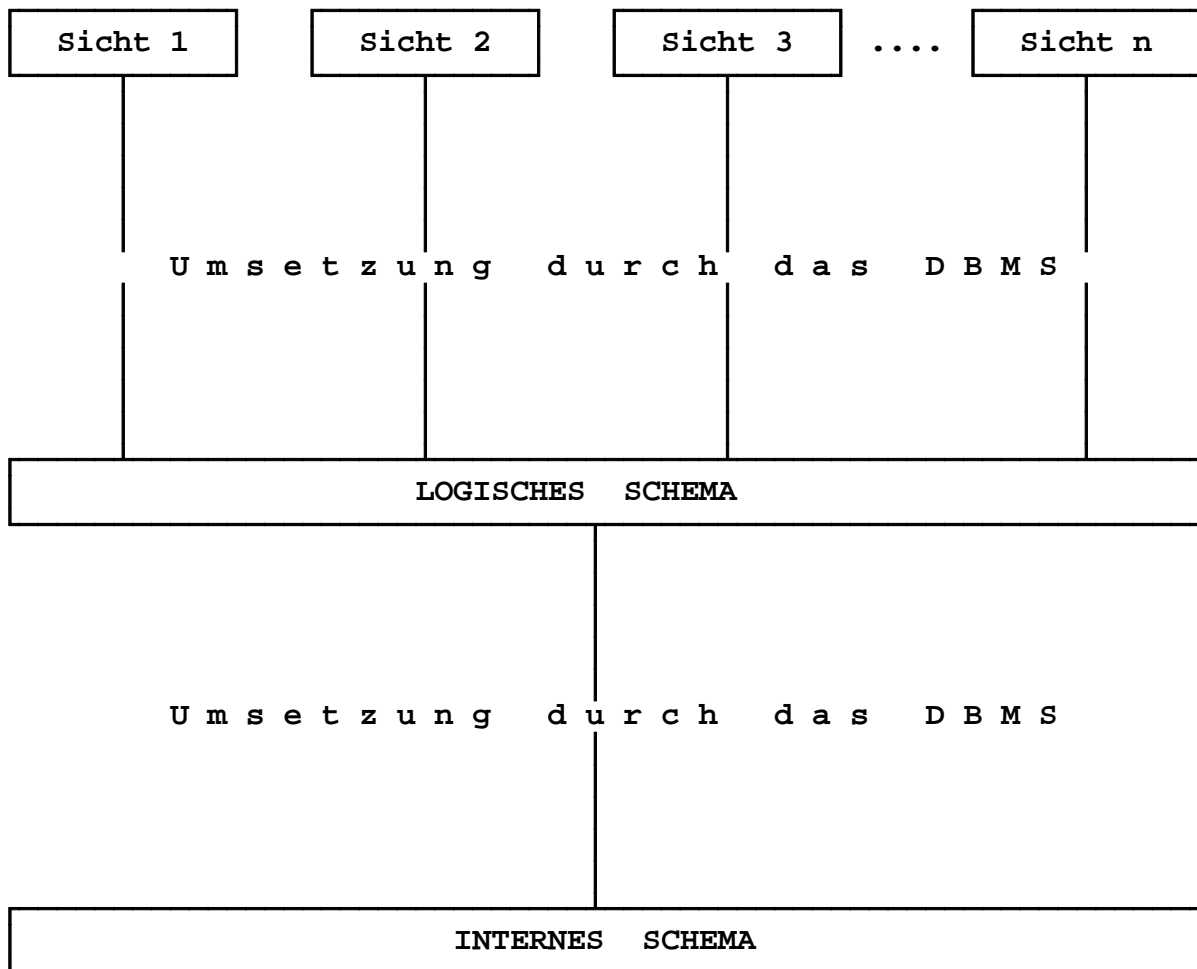
Eine heute viel praktizierte Lösung ist die **3-Ebenen-Architektur**. Dabei betrachtet man die Daten unter drei verschiedenen Aspekten: Die unterste Ebene ist die der physischen Speicherung (**internes Schema**). Von außen (vom Anwender) muß nicht bekannt sein, wie und wo die Daten gespeichert sind. Mit "Schema" sind die Festlegungen gemeint, die sich hier auf die physische Speicherung beziehen. Auf der mittleren Ebene wird die gesamte Datenbank inhaltlich betrachtet. Der Aufbau dieser Ebene ist gemeint, wenn von einem bestimmten Datenbankmodell die Rede ist. Hier erfolgt die Sinngebung der Daten: aus Bits und Bytes werden Kunden, Aufträge und Mitarbeiter. Die Festlegungen, die Inhalt und Aufbau dieser Ebene betreffen, werden **logisches Schema** genannt.

Die oberste Ebene stellt die Verbindung zwischen Datenbank und Anwender her. Jeder Anwendung ist ein **externes Schema** zugeordnet, das angibt, welchen Ausschnitt aus der Gesamtheit der Daten, also

der logischen Ebene, sie bearbeiten darf.

Der Anwender sieht von der Datenbank nur diesen Ausschnitt; er wird deshalb auch als **Sicht** bezeichnet.

Die Brücke zwischen den physischen Daten und den Anwendersichten wird vom DBMS geschlagen. Es klärt bei Datenanforderungen von Programmen, welche Teile der logischen Ebene benötigt werden und wie sie zur verlangten Datensicht zusammengesetzt werden können. Andere Komponenten des DBMS lösen dann die Aufgabe, diese Daten auch physisch aufzufinden und bereitzustellen (in Zusammenarbeit mit dem Betriebssystem).



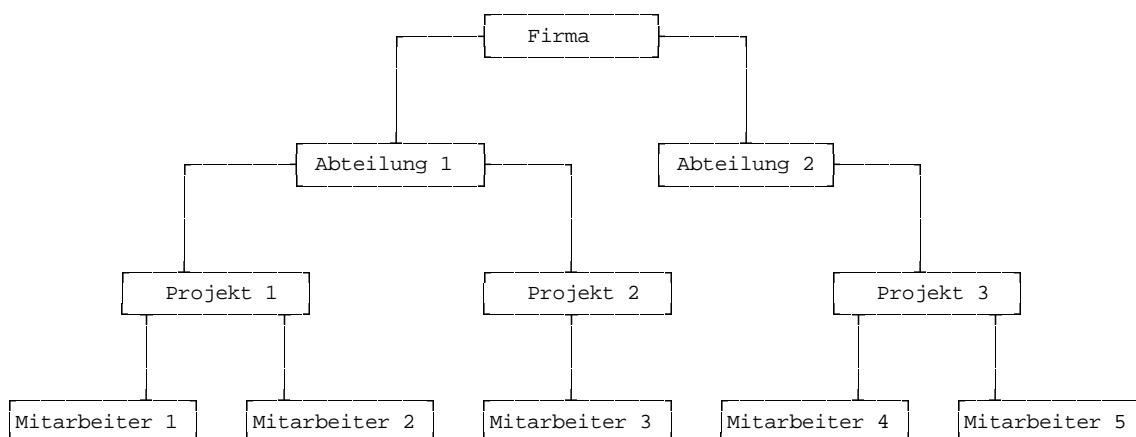
V. Datenbankmodelle

Wenn die Daten in einem gemeinsamen Pool gespeichert sind, ist es nicht mehr günstig, sie nach den Bedürfnissen eines einzelnen Programms zu gliedern. Man benötigt statt dessen eine programmneutrale Struktur (BRW: zweckneutrale Grundrechnung), die es erlaubt, jedes an diesen Pool angeschlossene Programm mit den Daten zu versorgen, die es benötigt.

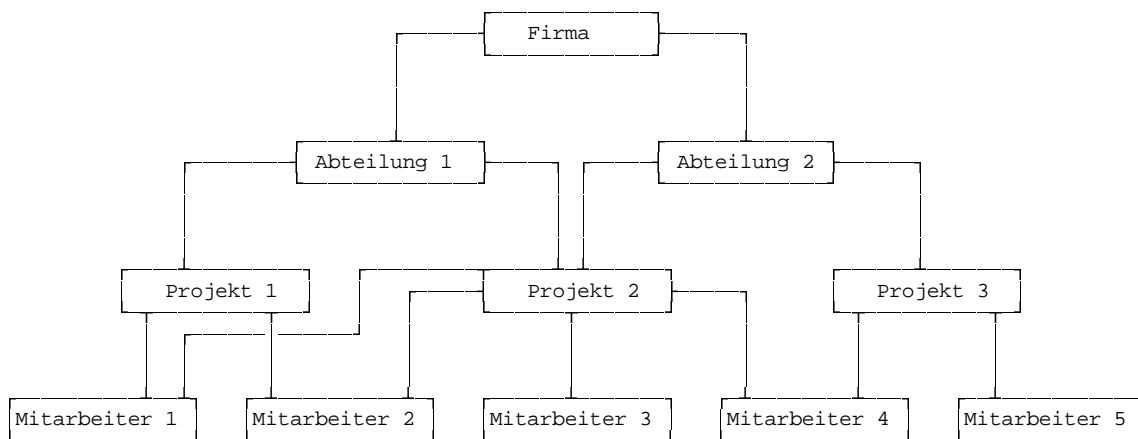
In der konventionellen Datenhaltung ordnet man die Daten in Dateien, Datensätze und Datenfelder ein. Genauso wird durch die Beschreibungsmittel des Datenbanksystems ein Rahmen für die mögliche Gliederung der Datenbasis vorgesehen. Es werden heute im wesentlichen drei Arten von Datenbanksystemen mit recht unterschiedlichen Beschreibungsmöglichkeiten benutzt: hierarchische DBS, Netzwerksysteme und relationale Systeme.

Das Datenmodell, das einem DBS zugrunde liegt, beeinflusst vorwiegend die logische Ebene. Die Art und Weise, wie die Datenbasis inhaltlich gegliedert ist, wird von diesem Modell bestimmt. Allerdings hat das Datenmodell auch Auswirkungen auf die Gestaltungsmöglichkeiten der physischen Ebene. Bei Netzwerksystemen und insbesondere beim hierarchischen System IMS (von IBM) sind die Modellierungsmöglichkeiten auch untrennbar an das Vorhandensein bestimmter Arten von Zugriffsorganisationen gebunden.

Struktur eines hierarchischen Datenmodells:



Struktur eines Netzwerkdatenmodells:



Struktur eines relationalen Datenmodells:

Relation "Mitarbeiter"

Pers-Nr.	Name	Gehalt
1427	Meier	3.217,33
8219	Schmidt	1.425,87
2624	Müller	2.438,21
.	.	.
.	.	.

Relation "Mitarbeiter in Abteilung"

Pers-Nr.	Abteilung
1427	3-1
8219	2-2
2624	3-1
.	.
.	.

1. Hierarchisches Modell

Es gibt heute nur noch ein nennenswertes hierarchisches DBS, das allerdings viel benutzt wird: das Information-Management-System (IMS) der IBM. Wenn man über hierarchische DBS spricht, spricht man über IMS.

Hierarchische Datenbanksysteme sind eine konsequente Weiterentwicklung der konventionellen Datenhaltung, bei der die Dateien normalerweise hierarchisch gegliedert sind. Diese Gliederung wurde grundsätzlich beibehalten. Die für DBS typische Mehrfachverwendung der Daten wird dadurch ermöglicht, daß an die Stelle von Duplikaten der Nutzdaten nur Verweise, sogenannte **Zeiger**, auf die bereits an anderer Stelle gespeicherten Daten treten.

Die Datenbankstruktur und Datenbankorganisation wird in einer sog. Data Base Description (DBD) beschrieben.

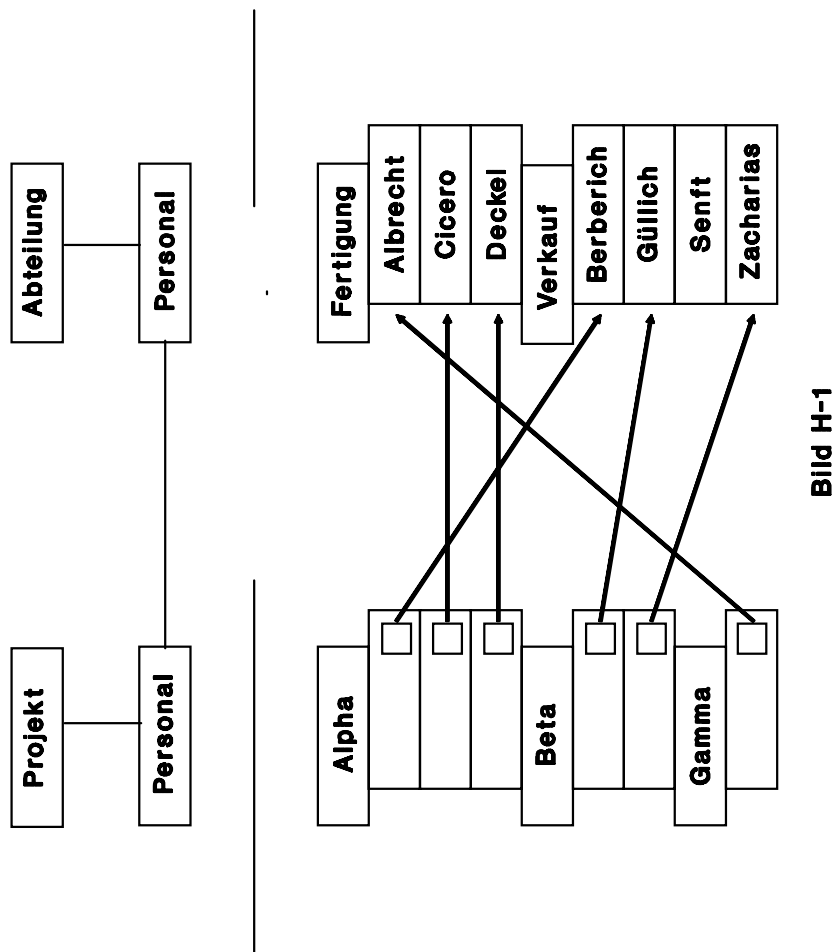
Die Sicht eines Anwenderprogramms auf die Datenbank kann bis auf Datenfeldebene herab beschränkt werden. Was ein Programm von einer IMS-Datenbank sehen darf, wird in sog. Program Specification Blocks (PSB) beschrieben. Dort steht, welche Datenbanken das Programm benutzen darf, welche Datentypen benutzt werden können und welche Verarbeitungsfunktionen (Lesen, Schreiben ...) bei den einzelnen Datenbanken erlaubt sind. (Bei IMS wird der Datenbank-Begriff in einer anderen Form benutzt: Das gesamte "Informationssystem" setzt sich aus einzelnen, zusammengehörenden Datenbanken zusammen, z.B. Personalinformationssystem setzt sich u.a. aus den Datenbanken Personal-DB, Ausbildungs-DB, Gehalts-DB ... zusammen.)

Von mehreren Hochsprachen (z.B. COBOL, PL/I) aus können Verarbeitungsfunktionen aufgerufen werden. Die entsprechende Schnittstelle stellen die DL/I-Aufrufe zur Verfügung. (DL/I = Data Language I - IMS - Datenmanipulationssprache (DML))

Bild H-1 zeigt, wie die Mitarbeiterdaten gleichzeitig innerhalb der Projektdaten-Hierarchie verwendet werden können, wenn sie dort durch Zeiger ersetzt werden. Die gesamte Datenbank besteht aus einer oder mehrerer solcher Hierarchien, die häufig mehr als zwei Stufen hoch sind.

Bild H-1 suggeriert, daß die Daten auch physisch in der hierarchischen Reihenfolge gespeichert sind. Dies ist tatsächlich möglich, allerdings ist diese Speicherungsform nur eine von mehreren, unter denen der Anwender wählen kann. Gemeinsam ist ihnen allen, daß sie eine hierarchische Ordnung zwischen den einzelnen Objekten der Speicherung herstellen.

Das Prinzip der Verzeigerung scheint auf den ersten Blick einfach und leicht zu handhaben. Leider ist das nicht so. IMS ist im Laufe der Zeit immer mehr erweitert und "verfeinert" worden, so daß es heute ein sehr kompliziertes System ist.



Ca. seit 1988 haben die relationalen Datenbanken den Leistungs-nachteil gegenüber IMS wettgemacht, der bis dahin verhinderte, daß sie trotz einfacherer Bedienung, 4GL-Unterstützung u.a. Vorteile, IMS nicht vom Markt verdrängen konnten.

Noch 1987 schrieb das Londoner Marktforschungsinstitut Input dem hierarchischen Datenbankmodell einen Anteil von 78 % am Gesamtmarkt für Datenbanken zu. IBM selbst ist mit DB2 und SQL/DS auf relationale Produkte umgeschwenkt, die DL/I - Datenbanksprache für IMS soll nicht weiter gepflegt werden, sodaß ein stetiger Rückgang von IMS am Gesamtmarkt zu erwarten ist.

Da IMS langsam aber sicher vom Markt verdrängt werden wird und sehr kompliziert aufgebaut/zu bedienen ist, gehen wir nicht näher auf dieses Modell ein, sondern nennen nur einige allgemeine Grundsätze dieses Produkts.

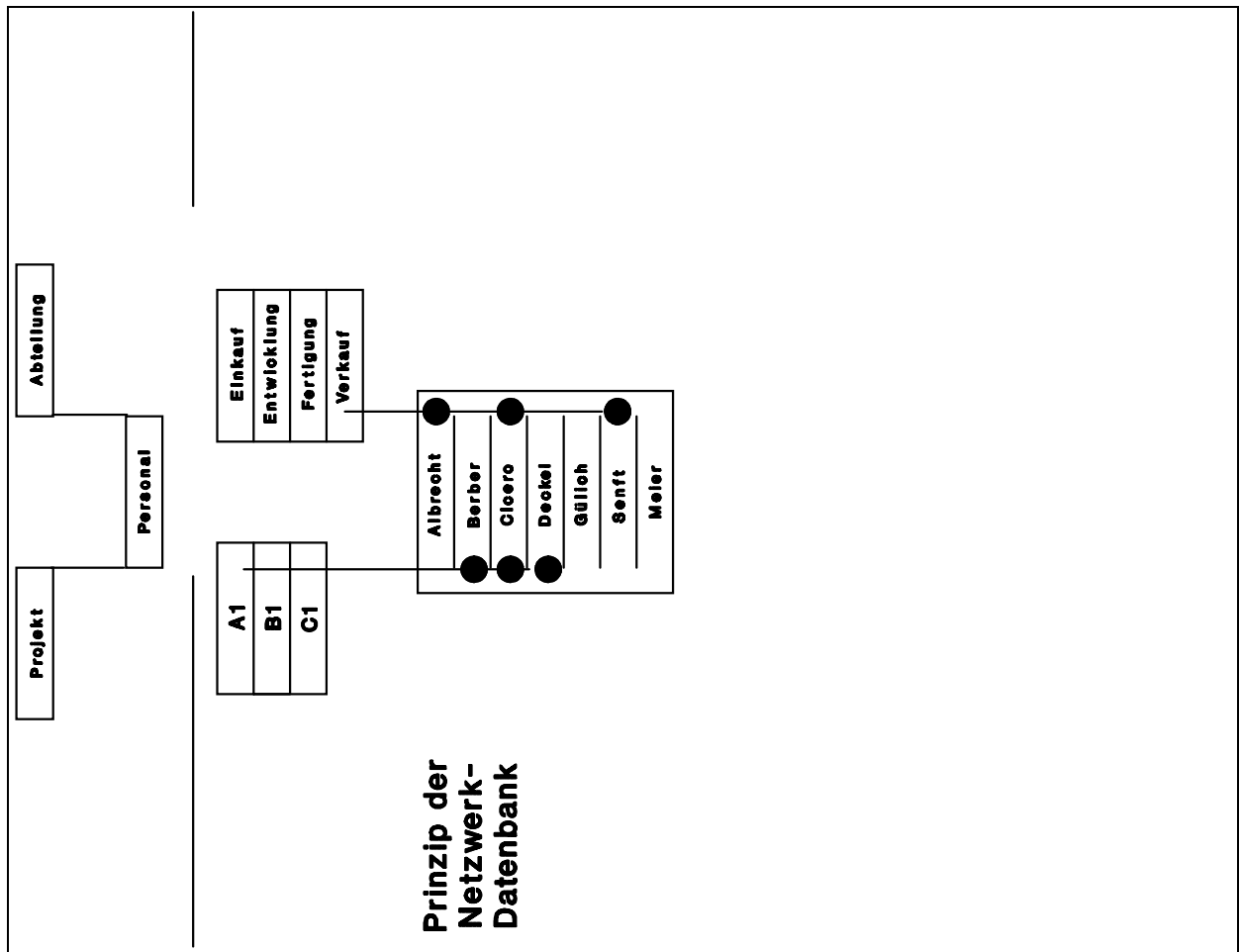
Eine IMS-Datenbank kann aus mehreren logischen und physischen Datenbanken bestehen. Zusammengehörnde Datenfelder werden in sogenannten Segmenten zusammengefaßt; auf die einzelnen Datenfelder kann nur über die Segmente und auf die Segmente nur vom Wurzelsegment (Root) aus und dann der Hierarchie folgend zugegriffen werden. Es reicht also nicht, die Datenfeld-Namen zu kennen, sondern man muß auch wissen, "wo in der Hierarchie", d.h.

in welchem Segment, sich die betreffenden Datenfelder befinden.

2. Netzwerkmodell

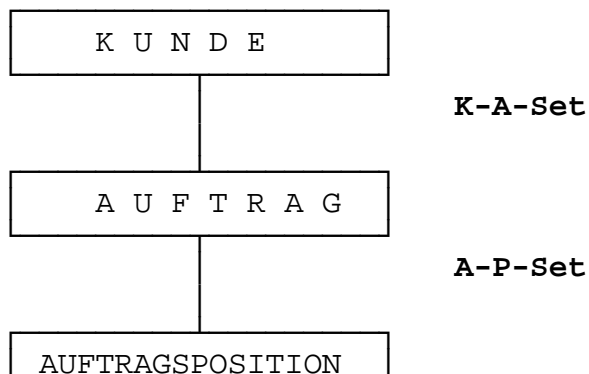
Der Ansatz eines Netzwerkmodells wurde von der **Codasyl Data Base Task Group (DBTG)** entwickelt (=> deswegen auch Codasyl-Datenbanksystem genannt). Netzwerkdatenbanken sind im Großcomputerbereich weitverbreitet, hingegen auf Mikrocomputern kaum zu finden. Sie werden, wie IMS, vorwiegend für administrative Anwendungen eingesetzt. Bekannte Systeme sind IDMS (Integrated Database Management System) von Cullinet Software und UDS (Universelles Datenbanksystem) von Siemens.

Die DBTG löste sich von der strengen Hierarchie und konnte deshalb auf den Platzhalter-Trick des hierarchischen Modells verzichten. Das hierarchische Modell geht in das Netzwerkmodell über, wenn man zulässt, daß ein und dieselbe Satzart in verschiedenen Unterordnungsbeziehungen verwendet werden kann. Die Trennung zwischen den Hierarchien, die für IMS kennzeichnend ist, gibt es beim Netzwerkmodell nicht. Die Hierarchien sind sämtlich in einem Netz enthalten, das aus Satzarten, wie Abteilung, Projekt und Personal, und Über- bzw. Unterordnungsbeziehungen zwischen diesen Satzarten besteht.



Die im Bild N-2 benutzte Darstellung des Datenbankaufbaus in Form eines Bachmann-Diagramms ist üblich. Die Satzarten werden durch Kästchen repräsentiert, die Pfeile weisen jeweils von der übergeordneten zur untergeordneten Satzart. In der Codasyl-Terminologie heißt ein solches Unterordnungsverhältnis **Set**, die übergeordnete Satzart wird als **Ownersatzart** bezeichnet, die untergeordnete als **Membersatzart**. Die konkrete Zuordnung untergeordneter Sätze zu einem übergeordneten Satz heißt **Setausprägung** (Set Occurrence).

Bachmann-Diagramm:



Physisch können die Verbindungen zwischen Owner- und Membersätzen auf unterschiedliche Weise festgehalten sein. Bild N-3 zeigt die klassische Form der Zeigerkette. Vom Owner zeigt ein Zeiger auf den ersten Membersatz, von diesem wird auf den nächsten Membersatz verwiesen usw. Neben der Zeigerkettenanbindung gibt es noch diverse andere Formen. Gewöhnlich kann der Datenbankverwalter je nach den Zugriffserfordernissen zwischen verschiedenen Organisa-tionen wählen.

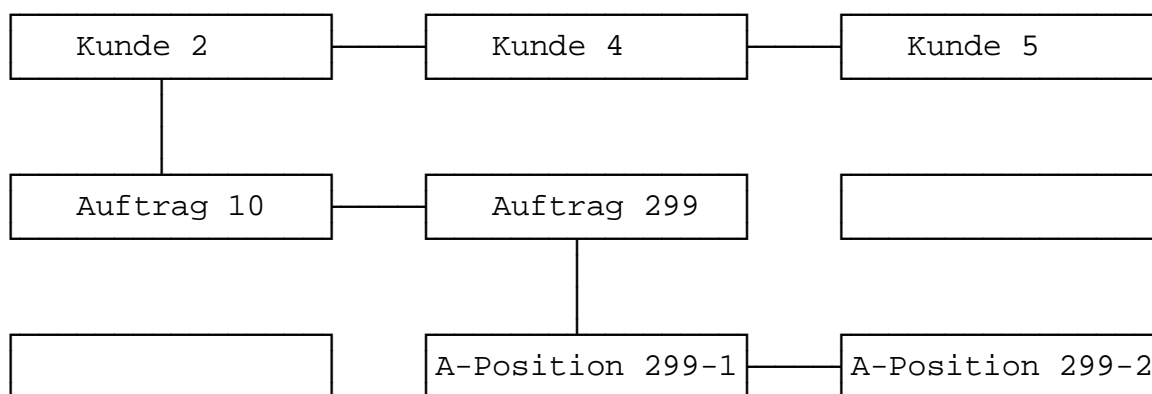


BILD N-3

Auch bei Netzwerkdatenbanken ist die übliche Verarbeitungsart die über konventionelle Programme. Sie erfolgt wie bei IMS Satz für Satz (eine Segmentsausprägung bei IMS entspricht einem Satz der Netzwerkdatenbank), jedoch viel flexibler. Alle Sätze können bei geeigneter Datendefinition wahlfrei oder sequentiell verarbeitet werden. Die sequentielle Verarbeitung ist innerhalb der Satzart oder innerhalb eines Sets möglich. Dabei kann auch von einem Set auf den anderen gewechselt werden.

Im Bild 6-4 ist dargestellt, wie zuerst der Ownersatz "Kunde 2" und dann die Membersätze "Auftrag 10" und "Auftrag 299" innerhalb des K-A-Sets gelesen werden können und anschließend innerhalb des A-P-Sets die Einzelpositionen des Auftrags 299.

Im Unterschied zu IMS kann der Einstieg grundsätzlich bei jeder Satzart erfolgen. Auftragssätze können also nicht nur über die Kundensätze erreicht werden, sondern auch direkt.

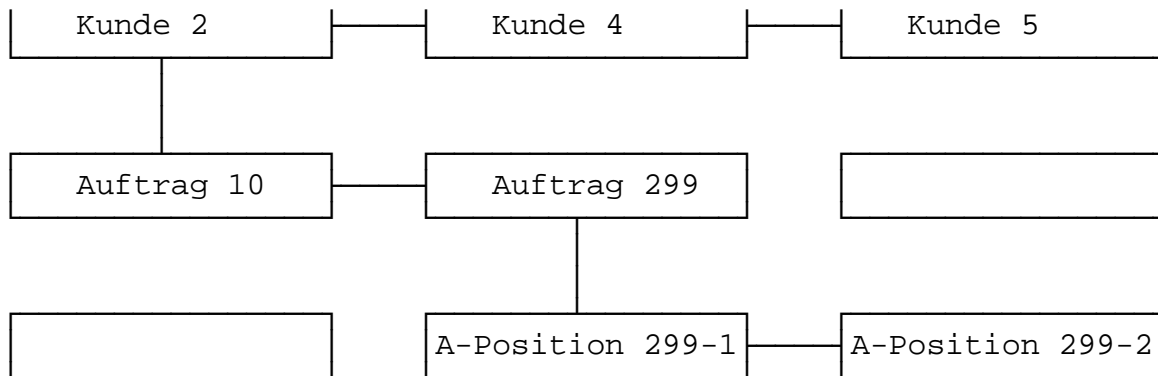


BILD N-4

Der Versuch, auch für Netzwerkdatenbanken Zugangsmöglichkeiten für Endbenutzer zu schaffen ist zwar nicht gänzlich mißlungen wie bei IMS, hat aber auch nicht zu solch flexiblen und anwenderfreundlichen Lösungen geführt wie bei relationalen Datenbanksystemen.

Sowohl für IDMS als auch für UDS werden Frontends für den Endbenutzer angeboten. Sie werden jedoch im Vergleich zum satzweisen Zugriff aus Anwendungsprogrammen heraus wenig benutzt.

3. Relationenmodell

Während der Einsatz von IMS und Netzwerksystemen vorwiegend auf den administrativen Bereich beschränkt ist, sind relationale Systeme universell anwendbar. Der einfache Aufbau der logischen Ebene erleichtert die Arbeit mit der Datenbank für den professionellen Programmierer, erlaubt daneben aber auch, Schnittstellen für Endbenutzer anzubieten. **Wegen der Klarheit des Konzepts gehört dem Relationenmodell die Zukunft, zumindest die der nächsten zehn Jahre.**

Relationale Datenbanken sind für Großcomputer und für Mikrocomputer erhältlich. Als verteilte Systeme oder in Netzen eingesetzt, können sie auch beide Hardwarekategorien gleichzeitig versorgen.

Bekannte Produkte sind DB2, und SQL/DS von der IBM, Oracle, Ingres, Informix, RdB (DEC), Sybase, Sesam (Siemens) und das bislang ausschließlich für Mikrocomputer angebotene dBase.

In einer relationalen Datenbank werden die Daten in Tabellen

(Relationen) abgelegt. Die Spalten dieser Tabellen heißen Attribute, Merkmale oder auch Felder. Die Zeilen werden auch als Tupel bezeichnet (Kunstwort für n-Tupel). In Anlehnung an die konventionelle Datenhaltung werden die Tabellen auch als Datenbankdateien bezeichnet, die Tupel als Sätze. Man muß sich aber darüber im klaren sein, daß eine Tabelle einer Datenbank keineswegs als eine separate Datei auf dem Speicher abgelegt sein muß. Auch kann eine konventionelle Datei eine hierarchische Struktur haben, während eine Relation keine gegliederten Spalten oder Wiederholungsmerkmale enthalten darf ("flache Tabellen").

Für jede Tabelle muß ein Attribut oder eine Kombination von Attributen als Schlüssel definiert werden. Jeder Schlüsselwert darf nur einmal in der Tabelle vorkommen, so daß anhand dieses Wertes die zugehörige Zeile eindeutig identifiziert werden kann. Die Schlüssel dienen auch dazu, Beziehungen zwischen Tupeln verschiedener Relationen zu dokumentieren.

<u>Abteilung</u>							
AbtName		KOSTNR	...				
Einkauf		3785	...				
Entwicklung		4307	...				
Fertigung		5212	...				

<u>Projekt</u>					<u>Personal</u>			
PName	Standort	Start	...	Name	PName	Abtname	...	
A1	3	1.1.87	...	Albrecht	A1	Fertigung	...	
B1	1	3.5.86	...	Berber	C1	Verkauf	...	
C1	1	2.7.88	...	Cicero	C1	Fertigung	...	

Die Fremdschlüssel üben beim relationalen System die Funktion aus, die in IMS und in der Netzwerkdatenbank durch physische Nachbar-schaft oder durch Zeiger wahrgenommen wird. **Die Beziehungen zwischen den Sätzen sind in den Datensätzen selbst festgehalten und nicht durch die Zugriffsorganisation.** Dies macht in relationalen Systemen die logische Ebene weitgehend von der physischen Ebene unabhängig.

Die in relationalen Datenbanken gespeicherten Daten können nicht nur tupelweise verarbeitet werden, sondern auch mit Hilfe von Operationen, die sich auf ganze Tabellen beziehen. Beispielsweise können mit der Operation **Join** (Verbund) zwei oder mehr Tabellen miteinander verknüpft werden.

<u>Name</u>	<u>PName</u>	<u>AbtName</u>	...	<u>AbtName</u>	<u>KOSTNR</u>	...
Albrecht	A1	Fertigung	...	Fertigung	5212	...
Berber	C1	Verkauf	...	Verkauf	7763	...

Weitere Operationen, die in relationalen Systemen gewöhnlich zur Verfügung stehen, sind

Projektion:

Spalten einer Tabelle, die nicht gebraucht werden, werden weggelassen. (Nur bestimmte Spalten ausgewählt)

Selektion:

Es werden die Tupel (Zeilen) einer Tabelle ausgewählt, die einer bestimmten Bedingung genügen. (Bsp.: Umsatz > 100.000)

Vereinigung: (Union)

Zusammenfassung von Tabellen, die Attribute desselben Formats besitzen und damit "vereinigungsverträglich" sind.

(Bsp.: Zusammenfassung von Kunden und Lieferanten zur Tabelle Geschäftspartner)

Durchschnitt:

Ergebnis sind die Zeilen, die in zwei Tabellen gleichzeitig vorkommen. (Bsp.: Alle Lieferanten, die gleichzeitig Kunden sind)

Differenz:

Aus zwei Tabellen werden die Zeilen ermittelt, die in der einen vorhanden sind, in der anderen aber nicht.

(Bsp.: Abgleich O-P-Tabelle mit A-P-Tabelle)

Das Prinzip bei der Anwendung der Operationen ist stets dasselbe: Die Daten, die der Anwender braucht, sind normalerweise in den Tabellen der Datenbank nicht in der gerade gefragten Zusammensetzung enthalten (die Gliederung ist ja anwenderneutral). Durch ein oder mehrere Relationenoperationen kann sich der Anwender aus den Basistabellen eine Ergebnistabelle ableiten, die gerade die Information enthält, die er braucht.

Die Liste der genannten Operationen ist nicht vollständig. Endbenutzer müssen für ihre Datenbankabfragen keineswegs alle Relationenoperationen kennen. Sie bedienen sich normalerweise einer Abfragesprache wie SQL (Structured Query Language), in der die meisten Probleme einfach zu formulieren sind.

SQL ist nicht nur eine Sprache, in der Endbenutzer schnell und leicht Informationen abrufen können, die auf ihre jeweiligen Probleme zugeschnitten sind, sondern sie ist eine **universelle Datenbanksprache**. Der Datenbankverwalter benutzt sie zur Einrichtung der Datenbank und zur Definition der Tabellen, der Anwendungsprogrammierer kann mit ihrer Hilfe aus COBOL- oder PL/I-Programmen heraus mit der Datenbank arbeiten, oder sogar nur

mit SQL und einigen Zusatzbefehlen komplette Anwendungen programmieren. Der Wert einer solchen einheitlichen Oberfläche für alle Benutzer ist nicht zu unterschätzen.

Gegen relationale Systeme wird häufig eingewandt, sie seien langsamer als IMS oder Netzwerksysteme. Man muß mit solchen Aussagen vorsichtig umgehen. Erstens sind die relationalen Datenbanken noch recht jung, während die meisten Systeme der beiden alten Daten-bankmodelle bereits viele Verbesserungen erfahren haben, zweitens können sie vieles, wofür die alten Modelle nicht geeignet sind, beispielsweise Endbenutzerabfragen. Die momentane Überlegenheit der alten Modelle bei administrativen Anwendungen ist nicht auf eine modellimmanente Schwäche der relationalen Systeme zurückzuführen, sondern darauf, daß diese noch nicht voll ausgebaut und optimiert sind.

4. Vergleich der Datenbank-Modelle

	hierarchische	netzförmige	relationale
Anwendungsspektrum	nur Administrationssysteme	nur Administrationssysteme	alle Standardanwendungen, IDV
4GL-Unterstützung	nein	nein	ja
Manipulation komplexer Objekte	nein	nein	nein
Schnelligkeit	ausgeschöpft	ausgeschöpft	teilweise noch zu verbessern
Programm-, Daten-Unabhängigkeit	nein	nicht vollkommen	ja
Zahl der Objektarten	begrenzt	unbegrenzt	unbegrenzt
Ausbaumöglich-			

keiten	kaum	umständlich	leichter
Verteilte Datenhaltung	nein	nein	teilweise

Das Ziel der integrierten Datenverarbeitung, was heute mehr und mehr an Bedeutung gewinnt, ist mit einer hierarchischen Datenbank nicht zu erreichen. Alle Benutzersichten müssen bei der Konzeption der Datenbank bekannt sein, da der Zugriff nur über "geplante Wege" erfolgen kann. Erweiterungen und Änderungen einer solchen Datenbank sind sehr kompliziert, da, fast wie bei der herkömmlichen Datenverarbeitung mit Dateien, ein Zusammenhang zwischen den Anwendungen und der Struktur der Datenbank besteht. IMS, und damit die hierarchischen Datenbank-Modelle, werden sicher noch einige Zeit bei großen, unflexiblen Anwendern weiterverwendet werden, doch bis zum nächsten Jahrtausend ausgestorben sein.

Netzwerkdatenbanken sind bei COBOL-Anwendern sehr beliebt, da sich die Datenbeschreibungs- und Datenmanipulationssprache an das Konzept dieser Programmiersprache anlehnt. Das Arbeiten mit Netzwerkdatenbanken ist für Endbenutzer zwar leichter als bei IMS, die Grundlage bilden aber immer noch Sprachen der 3. Generation, die einige Erfahrung und Ausbildung voraussetzen.

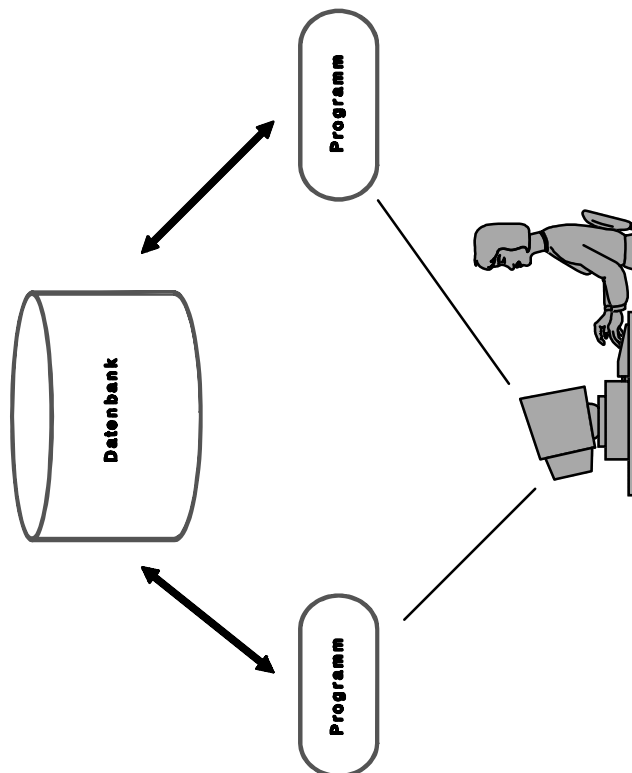
Relationale Datenbanken sind heute Standard. Einsatzmöglichkeiten von Sprachen der 4. Generation (SQL), vielfältige Zusatzprogramme, wie Query by Example, graphische Auswertungshilfen u.ä. sind in den meisten Datenbankprodukten enthalten. Die nächsten Jahre dienen sicher nur dazu, die Möglichkeiten und Leistungsdaten der bestehenden relationalen Datenbanken zu erweitern - ein neues Datenbank-Konzept, das das relationale in nächster Zeit ablösen könnte, ist nicht in Sicht.

VI. Isolierte, zentrale und verteilte Datenbanken

1. Isolierte Datenbanken

Unter einer isolierten Datenbank versteht man im Normalfall ein DBS, das die Daten eines einzigen Benutzers auf einem Ein-Platz-Mikrocomputer verwaltet. Es erfüllt zwar nicht die Aufgabe einer umfassenden Schnittstelle, erspart jedoch durch die meist umfangreichen Tools dem Endbenutzer den größten Teil der sonst anfallenden Programmierarbeiten. Auch kann hier wegen der Ein-Platzanwendung auf komplizierte und platzraubende Transaktionsüberwachungs- und Sperrmechanismen verzichtet werden.

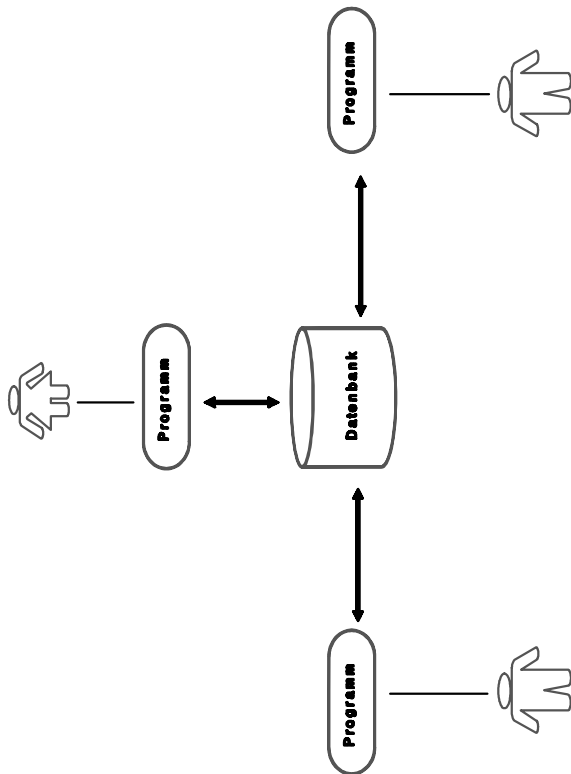
In Ausnahmefällen findet man jedoch auch noch isolierte Datenbanken für administrative Zwecke auf Großrechnern. Manchmal wurde diese Isolation mit Bedacht erstrebt, um eine größtmögliche Datensicherheit für geheime oder sensitive Daten zu erreichen. Jedoch entstand und entsteht eine Isolation meistens durch Unwissenheit und unsachgerechte Planung. Ein solches DBS verfehlt seine Funktion, da es im allgemeinen Datenaustausch ohne Bedeutung ist.



Isolierte Datenbank

2. Zentrale Datenbanken

Die zentrale Datenbank ist heute noch der Normalfall. Die Datenbank dient einer Reihe von Anwendungen als gemeinsamer Datenpool. Jeglicher Datenaustausch geschieht über die Datenbank, die in diesem Fall eine umfassende Schnittstelle darstellt:



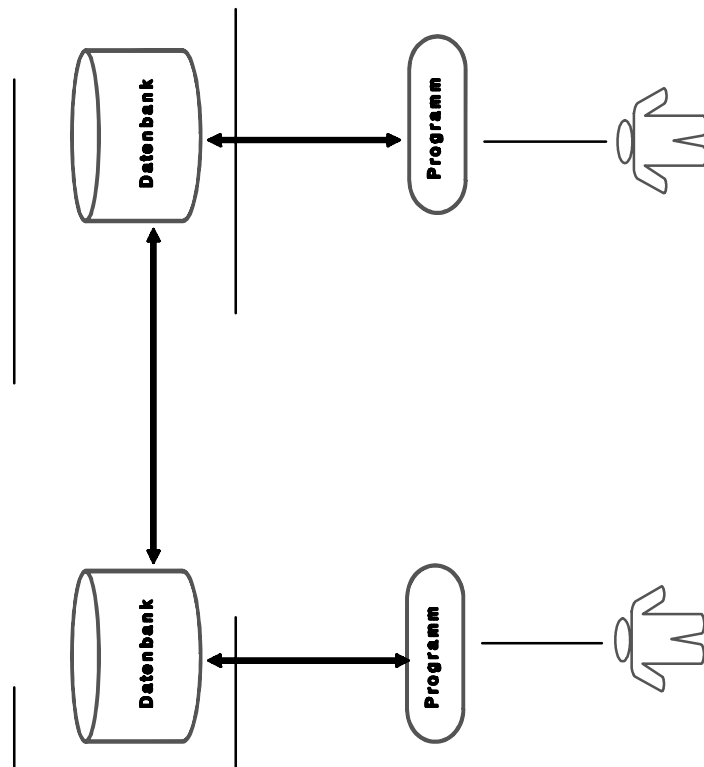
Zentrale Datenbank

In der Praxis findet man eine solch optimale Konstellation allerdings nur selten. Rechner und Datenbanksoftware sind heute zwar schon sehr leistungsfähig, aber doch überfordert, wenn sie alle Anwendungen eines großen Betriebes abwickeln sollen. Die Programme werden in diesem Fall verschiedenen Datenbanken zugeordnet. Somit gibt es in vielen Betrieben mehrere zentrale Datenbanken.

Die herkömmlichste Anwendungsform der zentralen Datenbank ist die auf dem Großrechner. Von verschiedenen Terminals aus können diverse Programme benutzt werden, die alle mit der DB arbeiten. Sollen auch andere Rechner auf die zentrale Datenbank zugreifen können, so kann diese in ein Netz eingebunden werden. Im Netzbetrieb wird die DB dann von einem Rechner (Datenbank-Server) verwaltet. Um Engpässe wegen der Datenübertragung zu vermeiden, werden Programme, die während ihrer Abarbeitung viele Zugriffe auf die Datenbank durchführen müssen, auf dem Server selbst installiert. Eine weitere Möglichkeit, verschiedene Rechnerarten auf eine gemeinsame Datenbank zugreifen zu lassen, ist die, verteilte Datenbanken einzusetzen.

3. Verteilte Datenbanken

Von verteilten Datenbanken spricht man, wenn die Datenbasis auf mehr als eine Datenbank aufgeteilt ist und für den Anwender die Möglichkeit besteht, mit jeder dieser Datenbanken Daten auszutauschen:



Verteilte Datenbanken

Normalerweise befinden sich diese Datenbanken auf verschiedenen Rechnern, so daß die Datenverteilung auch eine örtliche Verteilung ist. Zwar ist sie keine zwingende Voraussetzung, jedoch ist gerade dies ein Argument für verteilte Datenbanken. Denn der Hauptzweck dieser Systeme ist es, Daten dort besonders schnell verfügbar zu machen, wo sie besonders häufig und schnell gebraucht werden, andererseits aber die übrigen Programme nicht abzuschneiden. Der Benutzer weiß in der Regel nicht, welche Daten auf welche Rechner "verteilt" sind; er arbeitet hier also wie mit einer zentralen Datenbank.

Die Datenbanksysteme, die zu verteilten Systemen zusammengeslossen werden können, unterscheidet nach Art des Zusammenschlusses in:

- a) homogene Systeme
Zusammenschluß von typgleichen Datenbanken
(auf dem Markt bereits zu erwerben: Ingres Star, Oracle Star)
- b) heterogene Systeme
Zusammenschluß von unterschiedlichen Datenbanktypen. Ihre Entwicklung gestaltet sich wegen der zu meist unterschiedlichen Anwenderschnittstelle sehr schwierig.

Gründe für den Einsatz der verteilten Verarbeitung:

- Dezentralisierte Organisationsformen

Wenn ein Unternehmen oder eine Verwaltung dezentrale Verantwortliche kennt, äußert sich dies auch oft in deren Bedürfnis, über lokale Datenbestände verfügen zu können. Eine solche lokale Autonomie kann zusätzlich auch mit Datenschutzbestrebungen begründet sein.

- Sicherheitsüberlegungen, Verfügbarkeit

In Organisationen, die regional verteilt sind, vielleicht sogar über große Distanzen, soll bei allfälligen Problemen in einem fremden Netzknoten wenigstens ein reduzierter lokaler Betrieb möglich bleiben; dazu sind lokale Datenbestände nötig.

- Flexibilitätswünsche

An dezentralen Systemen bedeutet die Zufügung oder Wegnahme eines weiteren Netzknotens meist einen geringeren Eingriff als wesentliche Änderungen am einzigen System einer zentralen Datenbank.

- Leistungsanforderungen

Die Datenmenge oder das Transaktionsvolumen können die Leistungsfähigkeit eines einzelnen Rechners übersteigen.

- Kostenaspekte

In regional verteilten Organisationen konzentrieren sich die Datenzugriffe an jedem einzelnen Standort häufig auf einen bestimmten Teil der Daten. Durch entsprechende Dezentralisierung können Datenübertragungskosten eingespart werden.

- Koordinationsbemühungen

Diese Begründung ist komplementär zum Argument der Dezentralisierung der Verantwortung. Vielfach entstehen heute nämlich kleine, isolierte, unabhängige Datenbanken, z.B. auf Arbeitsplatzrechnern, irgendwo im Betrieb (z.B. Planungsstellen). Sollen diese in sinnvoller Weise wieder in den größeren Rahmen eines unternehmensweiten Datensystems "zurückgeholt" und eingegliedert werden, so ist dies nur möglich mit dem Angebot einer gewissen lokalen Autonomie. Zentralistische Lösungen sind dafür untauglich.

VII. Datenbankmaschinen

Hat ein Rechner einzig und allein die Aufgabe, ein oder mehrere Datenbanksysteme zu verwalten, spricht man von einer Datenbankmaschine bzw. von einem Datenbankrechner.

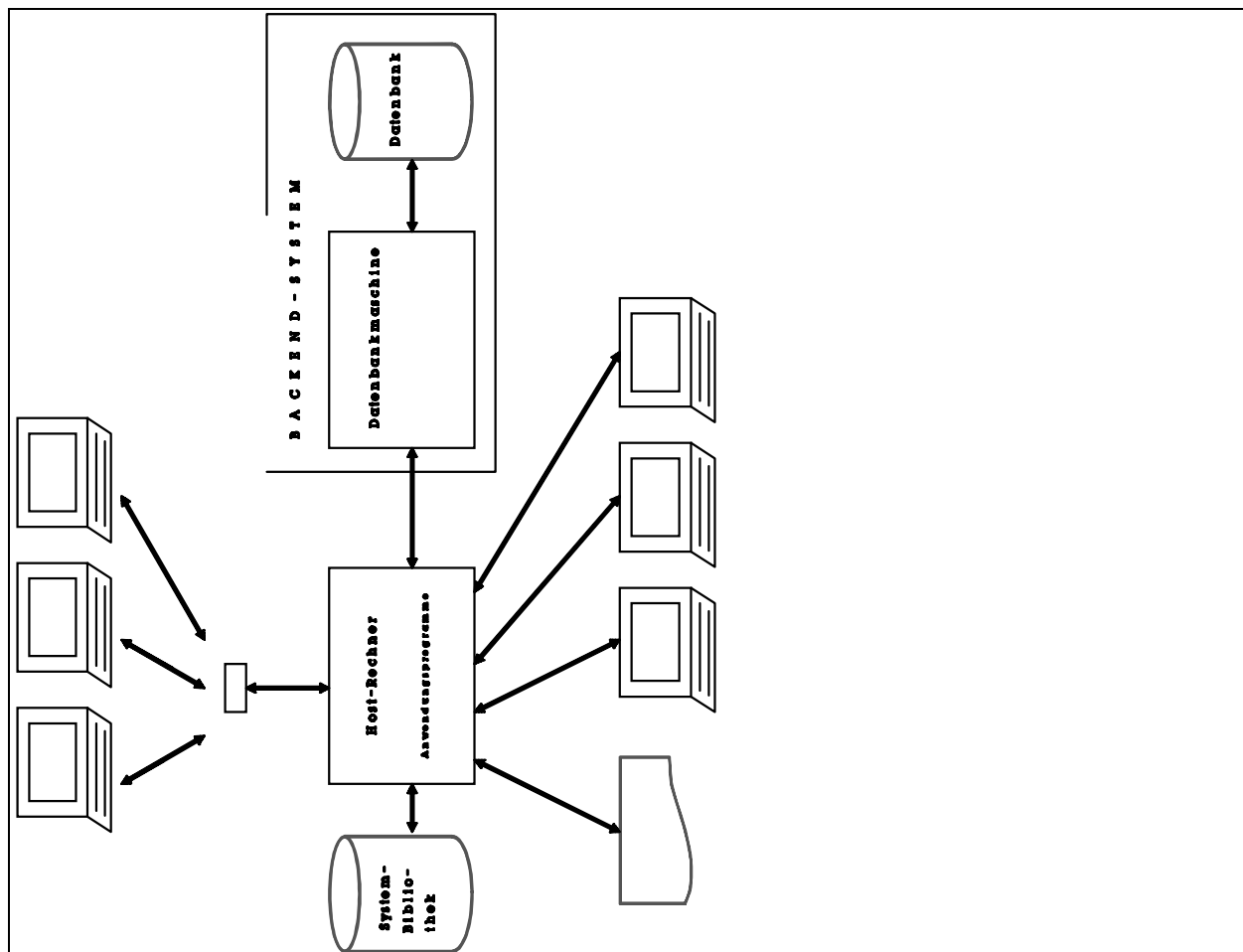
Im Gegensatz zu einem Universalrechner besteht das Betriebssystem nur aus dem Programm, das zur Verwaltung der Datenbanken notwendig ist und den Programmen, die für einen Anschluß an ein Datennetz notwendig sind.

Hardware-Architektur einer Datenbankmaschine

Aus der Komplexität eines Betriebssystems für einen Universalrechner heraus, und dem Wunsch die Kanäle und die Zentraleinheit eines Universalrechners zu entlasten, ergab sich die Frage, ob es nicht möglich ist, eine eigene Datenbankmaschine zu entwickeln. Durch die spezielle Architektur des Rechners sollte die Leistungsfähigkeit für die Verarbeitung von Datenbanken gesteigert werden.

Das bedeutet, das die DBMS-Funktionen sowohl hardware- als auch softwaremäßig in dem Rechner enthalten sind.

Die Hardware-Architektur einer Datenbankmaschine entspricht der eines Minicomputers oder der eines Prozessrechners. CPU, Hauptspeicher und die verschiedenen Controller sind an einem gemeinsamen Bus angeschlossen.



Eine praktikable Hauptspeicherkonfiguration besteht aus zwei bis sechs MB, mehreren Platten und einem Ethernetanschluß.

Da die Datenbankmaschine nur für die Verwaltung von Datenbanksystemen konzipiert ist, kann sie ohne einen anderen Rechner, dem Host, nicht arbeiten. Als Host können PC's, Minicomputer oder Großrechner verwendet werden. Der Anschluß erfolgt über das schon erwähnte Ethernet.

Ein derartiges System, bei dem die externen Eingaben nicht mehr von mehreren Terminals oder anderen Eingabestationen kommen, sondern diese an den Host-Rechner angeschlossen sind, der seinerseits die Verbindung zur Datenbankmaschine herstellt, wird im allgemeinen Backend-System genannt.

Die Standardkonfiguration eines solchen Backend-Systems besteht somit aus einem Host-Rechner, an dem mehrere periphere Einheiten angeschlossen sein können und einen Backend-Rechner, welcher über eine eigene Peripherie zur Datenspeicherung verfügt.

Software-Architektur einer Datenbankmaschine

Die Software einer Datenbankmaschine besteht aus einem überwiegend speicherresident gehaltenen Betriebssystem, das nur die zur Datenbankverwaltung erforderlichen Funktionen enthält.

Spool-Systeme, Batch-Manager usw. findet man in dieser Software daher nicht.

Besondere Effizienz erzielt man dadurch, daß das Betriebssystem der Datenbankmaschinen in der Regel nur relationale Datenbanken unterstützt.

Die konsistente Verwendung eines relationalen Konzepts wirkt sich auch auf die Systemsteuerung aus :

Das gesamte Betriebssystem der Datenbankmaschine wird über Systemrelationen verwaltet. Der Systemverwalter kann somit eine relationale Datenbanksprache wie z.B. SQL anwenden. Diese Datenbanksprache läuft wie z.B. auch die Programme zur Bildschirmsteuerung auf dem Host und verkehrt mit der Datenbankmaschine über ein kompliziertes Protokoll.

Vom Benutzer formulierte Befehle, beispielsweise in SQL, werden in kompilierter Form an die Datenbankmaschine übertragen, das Ergebnis der Abfrage sowie eventuelle Fehlermeldungen werden an den Host-Rechner zurückgegeben.

Das bedeutet, daß der gesamte Änderungsprozeß sich vor Ort auf der Datenbankmaschine vollzieht, ohne die Kapazität des Host-Rechners in Anspruch zu nehmen.

Von besonderem Interesse ist noch die Fähigkeit des Betriebssystems zwei Magnetplatten als sog. Spiegelplatten zu betreiben.

Beide Platten sind dann völlig identisch. Schreib-Befehle werden auf beiden Platten ausgeführt, gelesen wird demgegenüber von der gerade am wenigsten belasteten Platte. Der Vorteil ist, beim Ausfall einer Platte kann das Datenbanksystem unterbrechungsfrei mit den Daten der anderen Platte weiterarbeiten.

Funktionsweise einer Datenbankmaschine

1. Der Host-Rechner empfängt eine Transaktionsanweisung.
2. Er analysiert, welche Aufgaben von der Datenbankmaschine durchgeführt werden können.
3. Er erstellt Aufgabenanforderungen für die Datenbankmaschine.
4. Er sendet die Aufgabenanforderungen an die Datenbankmaschine.
5. Er wartet auf die Ergebnisse der Anforderung.
6. Er empfängt die Ergebnisse von der Datenbankmaschine.
7. Er verarbeitet die Aufgabenergebnisse nach Bedarf.
8. Er gibt die Antwort auf die Transaktionsanforderung an den Benutzer aus.

Leistungsgewinne :

Benötigen die Schritte 3 bis 6 weniger Zeit als wenn sie vom Hauptprozessor selbst durchgeführt werden, so ist die Datenbankmaschine schneller als der Prozessor des Host-Rechners. Aber auch wenn die Ansprechzeit nicht besser ist, kann eine Datenbankmaschine erhebliche Vorteile mitsichbringen, da der Hauptprozessor während diesem Vorgang andere nützliche Arbeiten ausführen kann.

Nutzung einer Datenbankmaschine

Datenbankmaschinen erlauben eine vielfältige Nutzung:

- Der User kann unter DOS mit einem PC oder im Time-Sharing-Mode eines Großrechnerbetriebsystems z.B. ein Programm namens SQL aufrufen. Danach kann er interaktiv SQL-Kommandos eingeben.
- In der Regel wird vom Hersteller ein Anwendungsgenerator mitgeliefert. Dieser erlaubt, ausgehend von einer relationalen Datenbank, automatisch Bildschirmmasken für die Dateneingabe, Datenänderung und Datenabfrage zu erstellen.
- Für einige Programmiersprachen, wie z.B. Cobol und Pascal, werden Runtime-Routinen oder Precompiler angeboten. Damit können Anwendungsprogramme direkt mit der Datenbankmaschine kommunizieren. Außerdem ist es in der Regel möglich Kommandos der angewandten Datenbanksprache aus einem Anwendungsprogramm abzusetzen und die Ergebnisse weiterzuverarbeiten.
- In das Betriebssystem des Host können Befehle integriert werden, die eine Schnittstelle zwischen der Datenbankmaschine und dem Hostrechner darstellen. Die wichtigsten Befehle bewirken :
 - * Laden einer Relation einer Datenbank von einer sequentiellen Datei des Hosts aus.
 - * Entladen einer Relation einer Datenbank in eine sequentielle Datei des Hosts.
 - * Sichern einer Datenbank auf den Host oder ein Band des Datenbankrechners. Analog: Zurückladen einer gesicherten Datenbank von diesen Medien

Mögliche Vorteile durch den Einsatz einer Datenbankmaschine

- Im Host-Rechner wird Hauptspeicher eingespart, da das DBMS und die Datenbank in das Backend-System ausgelagert wird. Es wird lediglich Speicherplatz für ein Interface zum Backend-System und ein Interface für das Kommunikationssystem benötigt.
- Es kann ein höherer Gesamtdurchsatz an Daten erwartet werden.
- Die Aufwendungen für eine Datenbankmaschine dürften geringer sein, als dies für eine analoge Leistungssteigerung beim Host-Rechner erforderlich wäre.
- Die Zuverlässigkeit des Systems steigt, da sich beide Systeme gegenseitig kontrollieren können.
- Die Datensicherheit wird sich erhöhen, da der Zugriff auf die Daten nur über das Backend-System erfolgen kann.

Stand der Datenbankmaschinen heute

Zur Zeit ist eine Reihe spezialisierter Datenbankmaschinen verfügbar. Es bleibt jedoch für Hardwarelösungen schwierig, mit den fortschrittlichen Algorithmen, die schnell in die Software eingeführt werden können, Schritt zu halten, so daß es schwierig ist, das Verhalten gut ausgelegter Softwaresysteme zu schlagen. Um eine hohe Produktivität von Systemen zu erhalten, die Datenbankmaschinen einschließen, müssen die Belastungen der verschiedenen Komponenten sorgfältig ausgeglichen werden, so daß kein Gerät ständig der Engpaß für alle anderen ist. Verschiedene Transaktionen haben natürlich verschiedene Bauteilnutzungsmuster, so daß eine übermäßige Optimierung leicht wirkungslos wird.

Bis jetzt hatten spezialisierte Datenbankmaschinen noch nicht die Wirkung die von ihren Förderern erwartet wurde. Es scheint jedoch nur wenig Zweifel zu geben, daß Systeme mit mehreren Prozessoren, also auch Datenbankmaschinen, üblich werden.

VIII. Stand und Entwicklung der Datenbank-Technik

Allgemeines

Lange Zeit konnte das relationale Datenmodell den Spitzenreiter des Marktes, die hierarchische Datenbank, nicht von seiner Rolle als Marktführer verdrängen. Hier tritt nun jedoch eine Trendwende ein. Wer heute ein DBMS-System kauft, wählt relational.

Obwohl die ersten relationalen DBMS-Systeme zum Anfang der 80er Jahre auf den Markt kamen, blieb ihnen der entscheidende Durchbruch lange Zeit verwehrt. Dieses war in der geringen Performance der relationalen DBMS-Systeme begründet. Sie hielt viele potenzielle Kunden davon ab, sich für eine relationale Datenbank zu entscheiden.

Durch entscheidene Performance-Verbesserungen die fast alle Datenbankanbieter durchgeführt haben, vor allem IBM, hat sich das Blatt jedoch zugunsten der relationalen Datenbanken gewendet.

Zur Entwicklung

Unbestrittener Ausgangspunkt der Datenbanktechnik waren in den sechziger Jahren große, zentral zu betreibende Verwaltungssysteme, wie etwa Flugreservationssysteme, Personaldateien und Bankbuchhaltungen. Lange waren technische Anwendungen in der Datenbankwelt eher ungewohnt. Die Techniker "bastelten" ihre Datenorganisationen allerdings oft auch selber, obwohl Standard-DBMS wie IMS für ihre Anwendung bestens geeignet wären.

Um 1970 entstand ein sehr starker Druck auf die EDV-Abteilungen. Es wurden integrierte Dateisysteme für eine ganzheitliche Sicht der Datenverarbeitung einer Unternehmung oder Verwaltung gefordert.

Ziele dieser Entwicklung, die sich bis heute fortsetzen, sind:

- Eine große Datenbasis soll einem heterogenen Benutzerkreis zur Verfügung stehen.
- Wenn ein autorisierter Benutzer ein Datum ändert, sollen alle anderen unmittelbar danach mit dem geänderten Datum arbeiten.
- Die Datenbasis soll leicht umstrukturiert, insbesondere erweitert, werden können.
- Transaktionen sollen entweder ganz oder gar nicht ausgeführt werden, um so die Konsistenz der Daten zu gewährleisten.
- Differenzierte Zugriffsregelungen sollen z.B. aus Gründen des Datenschutzes zuverlässig implementiert werden können.

Heutzutage herrscht bei den Datenbanken ein Trend zu mehr Benutzerfreundlichkeit. Dieser Trend ist jedoch nicht das Ende der Entwicklung.

Weitere Schlagwörter sind die 4 und 5 Generation der Programmiersprachen.

Ein Datensystem der 4 Generation erlaubt flexible Datenabfragen und Speicherung ohne klassische Programmierarbeit in einer höheren Sprache.

Ein Informationssystem der 5 Generation enthält Fachwissen von Experten in flexibler Form und macht dieses dem Anwender in verständlicher Art zugänglich.

Die Zeit der 4 Generation ist mit dem Angebot von Systemen mit selbständigen Datenmanipulationssprachen wie etwa SQL bereits konkret angebrochen. Es gibt entsprechende Datenbanksysteme auf Großrechnern sogar wie auf Arbeitsplatzrechnern.

Man kann heute sehr leicht seine eigene Datenbank organisieren und benutzen, und zwar ohne Programmierung in beispielsweise Cobol oder Pascal.

Informationssysteme der 5 Generation sollen das Bedürfnis nach mehr Flexibilität, nach früher Nutzbarkeit und Erklärungs-komponenten decken.

Die Bedeutung der Daten selbst

Es ist eine Tatsache, daß bei vielen Anwendungen die Kosten der Daten alle anderen Kosten einer Computerlösung bei weitem übertreffen.

Diese Bedeutung der Daten wird in Zukunft noch zunehmen.

Einige Gründe hierfür:

- Die Menge und Qualität der Daten

Aus verschiedensten Gründen werden laufend mehr Texte und Daten produziert (z.B. Anzahl der Fachartikel und Referate).

Diese können natürlich nicht mehr alle von den Fachleuten gelesen werden. Man benötigt eine separate Aufarbeitung als Dienstleistung spezieller Dokumentationsdienste, welche ihrerseits diese aufgearbeiteten Referenzdaten anbieten.

- Informationsgehalt der Daten

Bei den Informationssystemen der 5 Generation wird das Fachwissen von Experten, z.B. Rechtsanwälten, angesprochen, das systematisch gespeichert und anderen Personen verfügbar gemacht werden soll, damit der Endbenutzer seine Probleme mit Expertenwissen lösen kann.

Solche Daten sind von hohem Informationsgehalt und gleichzeitig in manchen Fällen von relativ kurzer Lebensdauer, wie z.B. Marktdaten. Das macht sie gleichzeitig teuer und wertvoll.

Andererseits darf jedoch davon ausgegangen werden, daß sich im Laufe der Zeit eine Erneuerung unserer Art des Umganges mit Daten und Informationen ergeben wird.

In einer zukünftigen Informationsgesellschaft wird sich für das Fachwissen ein offener Informationsmarkt etablieren, ähnlich wie er heute in den Massenmedien für einfache oder allgemein interessante Informationen, wie Sport und aktuelle Tagesthemen, bereits

existiert.

Damit wird die Voraussetzung für einen Markt- und Handelswert der Information geschaffen, was wiederum zu einer Kostensenkung beitragen dürfte.

IX. Objektorientierte Datenbanken

Die Aufgaben, die von den heutigen Datenbanken unterstützt werden, stammen vorwiegend aus dem kaufmännischen Bereich und haben zwei Eigenschaften gemeinsam :

- sie lassen sich in kurze Transaktionen zerlegen,
- sie hantieren mit einfachen strukturierten Datenobjekten.

Man spricht auch von Standardanwendungen und Non-Standardanwendungen (NSA). Die NSA sind dadurch definiert, daß sie dringend einer Datenbankunterstützung bedürfen, aber mit den herkömmlichen DBMS nicht befriedigend gelöst werden können. Sie werden von den heutigen Datenbanksystemen so schlecht versorgt, daß sich deren Anwendung noch verbietet.

Non-Standardanwendungen sind :

- Entwurfsanwendungen (CAD, Case)
- Prozeßsteuerung (CAM)
- Bild und Sprachverarbeitung
- Arbeitsplan- und NC-Programmerstellung (CAP)

Hieraus ergibt sich :

- a) Die Notwendigkeit komplexe Objekte dem DBMS als eine Einheit bekanntzumachen, damit sie von diesem auch als Einheit manipuliert und nahe beieinander gespeichert werden können.
- b) Der Bedarf an Manipulationsmöglichkeiten, die speziell auf die Gestaltungsobjekte der Anwendung zugeschnitten sind (Objektorientierung).

Ein Datenbanksystem, das komplexe Objekte manipulieren soll, muß über geeignete Beschreibungsmittel verfügen, also ein Datenmodell das komplexe Objekte beschreiben kann. Aber auch die primitiven Objekte müssen berücksichtigt werden, weil diese Bausteine komplexer Objekte sein können.

Klassische Datenmodelle sind satzorientiert. Im Relationenmodell nehmen einfache Felder (Attribute) Werte primitiver, vordefinierter Datentypen auf, die zu "flachen" Tupeln (Sätzen) zusammengefaßt werden.

Diese Tupel bzw. homogenen Mengen davon (Relationen) bilden somit Einheiten zur Repräsentation von Umweltsachverhalten.

In einem solchen Datenmodell müssen komplexere Umweltobjekte im Zuge des Datenbankentwurfs zwangsläufig in mehrere Sätze zerlegt werden, d.h. in eine "1 : n Abbildung".

Die Nachteile dieser Vorgehensweise liegen auf der Hand:

- Es entsteht ein hoher Aufwand und eine große Fehleranfälligkeit beim Entwurf selbst.
- Es kommt zu einer ineffizienten internen Verwaltung der Daten,

da bei der Abbildung das Wissen über die Zusammengehörigkeit verloren geht, und somit auch nicht zu einer Optimierung der physischen Abspeicherung und des Zugriffs benutzt werden kann.

In objektorientierten Datenbanksystemen sollen die Struktur und das Verhalten von Umweltobjekten weitgehend 1 : 1 erfaßbar sein. Damit können Anwendungssachverhalte wesentlich genauer in der Datenbank repräsentiert werden und für die Schnittstelle zum Anwendungsprogramm sind günstigere Lösungen möglich. Die Konsistenz der Datenbank sowie die Effektivität des Gesamtsystems können verbessert werden.

Objektorientierte Datenbanken vereinen folgende Entwicklungslinien in sich :

Die Erweiterung der herkömmlichen Datenbanksysteme um Konzepte der objektorientierten Systementwicklung und Programmierung, sowie der sogenannten semantischen Datenmodelle auf der einen Seite und die Anreicherung objektorientierter Programmiersprachen, z.B. Smalltalk, um Datenbankeigenschaften auf der anderen.

Wenn objektspezifischen Operationen auch für die interaktive Benutzung als Endbenutzerschnittstelle zur Verfügung stehen, dann ersetzen sie die Anwendungsprogramme weitgehend. Die Funktionen wandern in die Datenbank.

Objektorientierte Datenbanksysteme stellen einen noch vergleichsweise jungen Zweig der Datenbankforschung und -entwicklung dar. Bislang fehlen sowohl theoretische Grundlagen als auch eine allgemein anerkannte systematische Sichtweise. Andererseits wird weltweit in verschiedenen Forschungs- und Entwicklungseinrichtungen, aber auch in mehreren Softwarehäusern an entsprechenden Datenbanksystemen gearbeitet.

Beispiele:

- XSQL (eine Erweiterung des relationalen Systems SQL/DS; IBM San Jose)
- VBASE (käufliches Produkt der Firma Ontologic, USA)
- POSTGRES (Weiterentwicklung des relationalen Systems INGRES, University of California/Berkeley, USA)

Ganz wesentlich für den Erfolg objektorientierter Datenbanksysteme wird es sein, inwieweit ihre Implementierung die notwendige Effizienz gewährleistet. Die momentan beobachtbare gemeinsame Arbeit von Sprach- und Datenbankspezialisten auf diesem Gebiet kann dem Erreichen dieses Zieles nur dienlich sein.