Inhaltsverzeichnis

Vorwort

Der vorliegende Kurs „Datenbanken“ beschäftigt sich primär mit Datenbanken im betrieblichen Einsatz, weniger mit Online - Datenbanken. Wesentliche Schwerpunkte sind neben einer allgemeinen Einführung der Datenbank - Entwurf, die Datenbanksprache SQL (Structured Query Language), die Einbindung von SQL in Programmiersprachen, die Datenbank - Technik und Datenbank - Anwendungen. Es werden relationale, objektorientierte und multidimensionale Datenbanken behandelt.
In dem Praxisteil werden die Aufgaben zum Stoff im wesentlichen mit einem gängigen Datenbankverwaltungssystem (Microsoft Access) erarbeitet.

Einleitung zum ersten Studienbrief

Lernziel des ersten Studienbriefes ist es, das Konzept von Datenbanken und deren Nutzen darzustellen. Zunächst wird auf den Unterschied zu Dateisystemen und auf die geschichtliche Entwicklung der Datenmodelle eingegangen. Weitere Themen sind Einsatzgebiete und Architektur von Datenbanksystemen.
Danach wird das Konzept der relationalen Datenbank vorgestellt. Ein besonderer Schwerpunkt ist der Entwurf einer relationalen Datenbank, wobei die Methoden Entity Relationship Model und Normalisierung erläutert werden.

1 Einführung

Datenbanken sind Teil *eines betriebswirtschaftlichen Informationssystems,* das nach [Scheer95] als „ein System zur Aufnahme, Speicherung, Verarbeitung und Weitergabe von Informationen“ definiert wird. Es besteht aus folgenden Einzelkomponenten:

- Administrationssystem,
- Dispositionssystem,
- Management – Informationssystem und
- Planungssystem.

Diese Teilbereiche können gemäß ihrem Detaillierungsgrad in einer Pyramide dargestellt werden, wobei nach oben hin eine zunehmende Verdichtung der Daten stattfindet:

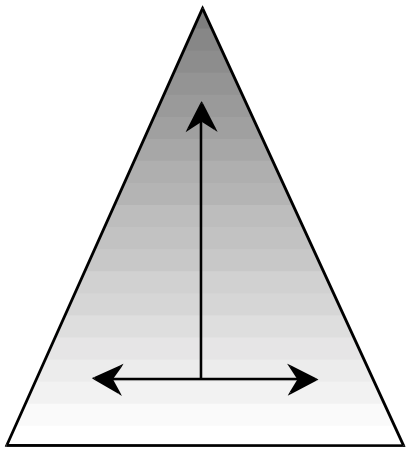


Abb. 1: Integrierte Informationssysteme

Langfristige Planungs- und Entscheidungssysteme bilden die höchste Verdichtungsstufe und geben Unterstützung für unstrukturierte und ad-hoc-Entscheidungen.
Die Analysesysteme auf der nächsten Stufe enthalten neben verdichteten internen Daten auch Daten aus externen Datenquellen. Es handelt sich dabei um Informationssysteme für die Bereiche Produktion, Investition, Beschaffung, Marketing, Vertrieb und Personal.
Berichts- und Kontrollsysteme sind Controllingsysteme, die zur Überwachung der beiden unteren Ebenen eingesetzt werden; es handelt sich um Controllingsysteme für die o. g. Bereiche.
Wertorientierte Abrechnungssysteme begleiten die mengenorientierten operativen Systeme der untersten Ebene, um die betriebswirtschaftlichen Konsequenzen sichtbar werden zu lassen. Beispiele solcher Sys-

teme sind die Buchführungen für das Lager, die Anlagen, die Kreditoren, die Debitoren und das Personal (Lohn- und Gehaltsabrechnung).
Die mengenorientierten operativen Systeme umfassen die Administrations- und Dispositionssysteme. In ihnen werden mengenorientierte Prozesse erfaßt, die eng mit der Leistungserstellung verbunden sind, z. B. Produktion, Technik, Beschaffung, Vertrieb und Personaleinsatz.
Ein *integriertes Informationssystem* zielt auf die Integration der Teilsysteme in vertikaler und (im Fall der operativen Systeme) in horizontaler Richtung. Integration kann als eine einheitliche Gestaltung funktionsübergreifender Informationssysteme und die Durchgängigkeit der unterschiedlichen Teilsysteme mit Weiterverwendung der Datenbasis interpretiert werden. Letzteres bedeutet, daß die Daten für jede Teilaufgabe nicht immer wieder eingegeben werden müssen, sondern von anderen Teilsystemen übernommen werden können.
Grob charakterisiert besteht ein (EDV-gestütztes) Informationssystem aus einer Menge von Anwendungssoftware, die auf einer Datenbasis operieren. Beispiele für solche Anwendungssysteme sind SAP R/3 von SAP/Waldorf oder Oracle Applications.
Unter dem Bereich Anwendungssoftware werden auch Anfrage- und Auswertungssysteme sowie Regeln von Expertensystemen subsummiert; ferner werden hierzu auch betriebswirtschaftliche Modelle und Methoden gezählt. Eine Ablaufsteuerung steuert betriebswirtschaftliche Teilaufgaben; eine Durchlaufterminierung z. B. kann nur dann erfolgreich durchgeführt werden, wenn die Materialwirtschaft für die Bereitstellung des Materials gesorgt hat.
Bei der Realisierung eines DV-Projekts ist zu beachten, daß die gesamte Lebenszeit eines Informationssystems den Gesetzen des Software-Lifecycles unterliegt, wobei der Lifecycle in mehrere Phasen eingeteilt wird.

Siehe dazu Anhang

„Vorgehensweise bei DV-Projekten“ und
„Sollkonzept“

Wichtig dabei ist eine vollständige und exakte Dokumentation der einzelnen Teilbereiche und damit eine Beschreibung des gesamten Informationssystems; nur dann ist eine problemlose Entwicklung und Weiterentwicklung gewährleistet. Die Beschreibung ist auch für die Auswahl von Standardsoftware oder die Schulung der Anwender erforderlich.
Es gilt die Regel, daß bei stärkerer Unterstützung der ersten Phasen die Kosten bei den nachfolgenden Phasen geringer bleiben. Investitionen in frühen Phasen zahlen sich eben doppelt und dreifach aus.

1.1 Die Datenbank im Informationssystem

Alternativ zu Datenbanken können Daten in Dateien gespeichert werden. Folgende Tabelle zeigt die wesentlichen Unterschiede bei der Benutzung von Dateien oder Datenbanken; Erklärungen im Text bzw. später.

Eigenschaft	Dateisystem	Datenbanksystem (relat.)
Datenunabhängigkeit	Keine, da Datenstruktur im Programm	Datenunabhängigkeit durch Projektion und Views sowie durch Trennung in externes, konzeptionelles und physisches Schema
Datensicherheit	Tagessicherung	Logging aller Datenänderungen
Zugriffsschutz	Auf Dateiebene	Auf Satz- und Feldebene, inhaltsbezogen
Beziehungen	-	Durch Fremdschlüssel
Redundanz	Hoch	Gering
Konsistenz	-	Erhaltung wird durch Transaktionskonzept unterstützt
Integrität	-	Referentielle Integrität; Definition von Integritätsregeln
Aktualität	Weitgehend	Ja, durch direkte Änderung
Abfragesprache	-	SQL
Schneller Zugriff	B*-Baum, Hash	B*-Baum, Hash
Zugriffsform	Einzelsatz-Zugriff	Mengenzugriff
Synchronisation im Mehrbenutzerbetrieb	Sperren auf Satzebene	Sperren auf Transaktionsebene
Tools	Wenige	Viele Tools für die Anwendungserstellung

Tab. 1: Vergleich Dateisystem mit Datenbanksystem

Die Datenbasis als Teilbereich eines Informationssystems enthält sämtliche für das Informationssystem relevante Daten. Bei der Verarbeitung dieser Daten stellt der Anwender allerdings bestimmte Anforderungen an die *Qualität* der Daten. Damit die Daten den tatsächlichen Gegebenheiten (im Unternehmen oder in der Organisation) genau entsprechen bzw. ihnen möglichst nahe kommen, sollen sie weitgehend redundanzfrei, in sich konsistent (auch nach Absturz des Computers) und korrekt (auch bei Fehleingaben des Benutzers) sein; letzteres wird mit *Integrität* der Daten bezeichnet.
Darüber hinaus wird erwartet, daß die Daten jederzeit up to date sind, d. h. daß Benutzer oder Benutzerprogramme immer auf aktuelle Daten

zugreifen. Gleichzeitig mit den Daten arbeitende Benutzer sollen sich dabei möglichst nicht behindern. Die Abfrage der Daten sollte dem Endbenutzer bzw. dem Anwendungsprogrammierer durch eine komfortable Zugriffssprache erleichtert werden.

Beispiel:

> Ändere das Lieferdatum des Produktes x der Variante y bei allen Kundenaufträgen mit mittlerer Priorität, falls das Lieferdatum nach dem Datum z liegt.

Daten existieren nicht isoliert nebeneinander, sondern stehen in *Beziehung* zueinander. Beispielsweise sind die Aufträge eines Kunden immer in Verbindung zu dem Kunden selbst zu sehen.

1.1.1 Datenspeicherung in Dateien

Eine Form der Speicherung und Verwaltung von Daten auf externen Datenträgern (meist Magnetplatte) ist die Nutzung von *Dateiverwaltungssystemen*. Diese bieten eine hardwareunabhängige Zugriffsmethode auf die Daten an, wobei der explizite Zugriff aus dem Anwendungsprogramm von der Datenorganisation (sequentiell, random, indexsequentiell) abhängt. Für den indexsequentiellen Zugriff verwalten die Dateisysteme intern Zugriffsunterstützungen, die auf bekannten Techniken wie B*-Bäumen oder Hashing basieren und einen schnellen Zugriff über Schlüssel (z. B. Kunden-Name) auf einen großen Datenbestand ermöglichen. Bei kleineren Datenmengen, bei überwiegend sequentiellem Zugriff oder bei rein administrativen Aufgaben haben sich Dateiverwaltungssysteme bewährt und sind heute noch vielfach im Einsatz.
Bei der Organisation von Daten in Dateien eines Dateiverwaltungssystemes entsteht die Problematik, daß i.d.R. für jeden Anwendungsfall ein eigener Datenbestand in Form von mehreren Dateien angelegt wird, teilweise sogar durch Kopieren aus vorhandenen Dateien. Die Folgen sind hohe Redundanzen, keine Gewährleistung der Konsistenz, d. h. kein automatischer Abgleich der Dateiinhalte nach Änderungen, keine Datenunabhängigkeit, d. h. keine Unabhängigkeit der Anwendungsprogramme von den Datenstrukturen.

1.1.2 Datenspeicherung in Datenbanken

Zur Erreichung der oben genannten Anforderungen an die Datenqualität bieten Dateiverwaltungssysteme nur unzureichende Hilfsmittel. Das bedeutet, daß die zusätzlich anfallenden Aufgaben durch das Anwendungsprogramm abgedeckt werden müssen, wobei jedes Anwendungsprogramm dann über entsprechende Prozeduren verfügen muß. Um hier eine bessere Situation zu schaffen, sind in den 60er Jahren die *Datenbankverwaltungssysteme* (DBVS) entwickelt und seither zur Lösung von neuen Anforderungen permanent weiterentwickelt wor-

den. Besonders die Darstellung von Beziehungen der Daten zueinander werden durch geeignete Hilfsmittel besonders unterstützt. Ein DBVS bearbeitet nicht nur eine einzelne Datei, sondern eine Vielzahl von Dateien, die auch Tabellen genannt werden. Dadurch sind überhaupt erst komplexere Zugriffe auf Daten mehrerer Tabellen gleichzeitig möglich. Bei solchen Zugriffen handelt es sich dann um *Mengenzugriffe*, d. h. das Ergebnis einer Datenabfrage liefert mehrere Treffer.
Durch die Möglichkeit, Beziehungen zwischen den Daten zu bearbeiten, bieten DBVSe eine geeignete Plattform für die Schaffung von integrierten Informationssystemen. Dies gilt auch für die Integration der Daten anderer Datenformate wie z. B. Grafiken, Bild- und Tondaten oder Videosequenzen. Moderne DBVSe bieten geeignete Datenstrukturen und Funktionalitäten zur Unterstützung von solchen *Multimedia-Anwendungen*.
Zunächst noch ein Hinweis zu einigen Begriffen. Mit einer *Datenbank* (DB) sind die Daten selbst gemeint, die gespeichert sind, in einem bestimmten Zusammenhang stehen und für konkrete Anwendungen oder Zugriffe zur Verfügung stehen. Das Datenbankverwaltungssysteme (DBVS) ist die Software, die zur Verwaltung und zum Zugriff der Daten auf einem Computer eingesetzt wird. Ein Datenbanksystem (DBS) schließlich ist die Zusammenfassung von DB und DBVS.
Abschließend kann festgehalten werden, daß ein Datenbanksystem durch folgende Eigenschaften charakterisiert werden kann:

- zentral in der Datenbank gespeicherte, weitgehend redundanzfreie Daten
- mehrere Benutzer mit gleichzeitigem Zugriff auf die Daten
- verschiedene Anwendungsprogramme mit unterschiedlicher Aufgabenstellung
- Erhaltung der Konsistenz und Integrität der Daten
- weitgehende Datenunabhängigkeit

1.2 Eigenschaften eines Datenbanksystems

Auf einige der wichtigsten Eigenschaften von Datenbanksystemen wird kurz eingegangen.

1.2.1 Datenunabhängigkeit

Eine wesentliche Eigenschaft zur problemlosen Erweiterung der einer Datenbasis zugrundeliegenden Datenstruktur ist die Datenunabhängigkeit: die Datenstrukturen in den Anwendungsprogrammen sollen möglichst unabhängig von den Datenstrukturen in der Datenbank sein. Dadurch ist eine flexible Erweiterbarkeit der Datenbankstruktur gegeben, ohne daß sämtliche Anwendungen, die auf die Datenbank zugreifen, angepaßt werden müssen. Bei Dateisystemen „sehen“ die Anwen-

dungsprogramme noch die Struktur der Datensatzfelder in der Datei. Wird eine solche Struktur erweitert, z. B. indem in einer Kundendatei noch ein weiteres Feld „Vertreter“ aufgenommen werden soll, müssen alle Programme, die auf die Kundendatei zugreifen, angepaßt werden. Bei der ersten Generation von DBVSen, insbesondere bei solchen, die das Hierarchische Datenmodell (Erklärung später) unterstützen, ist der Grad der Datenunabhängigkeit noch relativ gering. Dies liegt daran, daß die Verbindungen zwischen Dateien zur Darstellung der Beziehungen noch von den Programmen ausgewertet werden müssen. Deshalb sind aufwendige *Data Dictionary - Systeme* (DD) erforderlich, die festhalten, welche Programme auf welche Daten in welcher Form zugreifen. Bei Änderungen der Datenstruktur kann auf der Basis dieser Information eine Anpassung der Anwendungen vorgenommen werden. Neben der o. g. logischen Datenunabhängigkeit soll auch eine physische Datenunabhängigkeit gegeben sein. Das bedeutet, daß die Programme keine Informationen über die physische Speicherung (Satznummern, Existenz von Indexen, Blockungsfaktor) haben dürfen.

1.2.2 Datensicherheit

Da viele Benutzer auf die DB zugreifen, muß ein wirkungsvoller Datenschutzmechanismus von dem DBVS angeboten werden. Es muß möglich sein, bestimmte Daten gegenüber bestimmten Benutzern/Benutzergruppen zu verbergen oder gegen Änderungen zu schützen (Security).

1.2.3 Prüfung der Konsistenz und Integrität

Es muß möglich sein, im Schema Bedingungen formulieren zu können, die die Konsistenz und die Integrität der Daten prüfen bzw. gewährleisten. So muß man z. B. angeben können, daß ein Überziehungskredit unter 10.000,- DM bleiben muß. Oder ein Kunde, der noch Aufträge in der Auftragsdatei hat, kann nicht aus der Kundendatei gelöscht werden (Consistency, Integrity).

1.2.4 Synchronisation im Mehrbenutzerbetrieb

Wenn mehrere Benutzer dieselben Daten gleichzeitig (Timesharing-Betriebssystem) bearbeiten wollen, muß das DBVS diese Zugriffe synchronisieren. Wollen etwa zwei Benutzer den gleichen Datenwert um den Wert 1 erhöhen, um z. B. einen Platz zu reservieren, und diesen dann in die Datenbank zurückschreiben, wird die erste Buchung ohne Synchronisation nicht berücksichtigt.

1.2.5 Recovery nach DBVS-Abstürzen und Plattenfehlern

Ein Problem für die Erhaltung der Konsistenz besteht auch darin, daß das DBVS bei der Änderung von Daten abstürzen kann. Dann dürfen keine halben Aktionen übrigbleiben, die die Datenkonsistenz verletzen. Z. B. muß bei einer Kontenumbuchung jeder Abbuchung auch eine Zubuchung gegenüberstehen. Nach einem solchen Absturz, sei er hardware- oder softwarebedingt, muß das DBVS ein Recovery durchführen und dabei den Datenbestand wieder in einen konsistenten Zustand überführen. Eine besondere Art des Recovery ist nötig, wenn eine Platte, auf der der Datenbestand gespeichert ist, ausfällt (Platten-Restauration). Mittels vorher durchgeführter Sicherungen der Daten auf z. B. Magnetbänder wird die Platte wieder restauriert.

1.2.6 Zugriff auf die Daten mit SQL

Der Zugriff auf eine Datenbank über das DBVS kann mittels eines Anwendungsprogrammes oder über interaktive Tools am Bildschirm erfolgen. Anwendungsprogramme werden meist in einer konventionellen Programmiersprache wie Cobol, C/C++, Pascal/Delphi, Visual Basic oder Java entwickelt. Damit die Programme die Datenbank abfragen oder ändern können, bietet das DBVS eine Datenbanksprache als Schnittstelle an. Hier hat sich SQL (Structured Query Language) als Standard herausgebildet, der praktisch von allen marktgängigen DBVSen unterstützt wird. Eine Alternative ist eine auf die Entwicklung von Datenbankanwendungen spezialisierte *Sprache der 4. Generation* (4GL). Hier steht SQL im Vordergrund, wobei die Sprache um die gängigen Konstrukte (Variablen, Kontrollstrukturen, Fehlerbehandlung usw.) von höheren Programmiersprachen angereichert ist. Solche 4GL-Sprachen sind leider nicht standardisiert und variieren mit dem DBVS.
Die starke Verbreitung von Datenbanken liegt nicht zuletzt an der komfortablen Möglichkeit, kleinere Anwendungen oder ad-hoc-Abfragen interaktiv am Bildschirm ohne Programmierung zu erledigen. Hierfür stehen eine Vielfalt von Tools zur Verfügung, z. B. Interaktives SQL, Query by Forms bzw. Query by Example oder interaktive Masken- und Berichtsgeneratoren.

1.3 Einsatzgebiete und Anwendungsmöglichkeiten

Datenbanksysteme werden überall dort eingesetzt, wo größere Datenmengen zu verwalten sind und einem oder mehreren Benutzern zum Zugriff zur Verfügung gestellt werden. In vielen Fällen wird die Datenbank für eine spezielle Anwendung genutzt. Beispiele dafür sind Buchungssysteme, Verwaltungssysteme, PPS, Auftragsbearbeitung,

Fakturierung, Materialwirtschaft, Vertriebsunterstützung, CAD, Stückliste, Meßdatenverarbeitung, Verwaltung geografischer Daten.
Im Rahmen der gestiegenen Anforderungen an ein leistungsstarkes Informationssystem zur Unterstützung sämtlicher Vorgänge im Unternehmen ist der Trend nach sogenannten Unternehmens-Datenbanken mit Integration aller Teil-Datenbanken als Basis von Unternehmensanwendungen wie etwa SAP R/3 zu beobachten. Bei größeren Unternehmen findet man häufig auch die Situation vor, daß historisch gewachsen mehrere DBVSe auf unterschiedlichen Rechnerebenen (Großrechner, Abteilungsrechner) existieren. Die Integration dieser Datenbestände ist dabei mehr oder weniger gut gelöst.
Mit dem Aufkommen der PCs bzw. Workstations findet eine weitere Form, die persönliche Datenbank, zunehmendes Interesse. Beispiele hierfür sind Adreßlisten, Telefonverzeichnisse, offene Punkte-Listen, Termin-Daten und auch viele Anwendungen im privaten Bereich.
Eine etwas andere Form sind die Online-Datenbanken. Hier werden von Unternehmen, die sich auf das Anbieten von Informationsdiensten spezialisiert haben, Daten zu allen möglichen Themen in externen Netzwerken, meist gegen eine entsprechende Gebühr, angeboten. Durch die Verbreitung des Internet ist gerade diese Form der Informationsgewinnung stark im Vormarsch.

1.4 Modelle und Strukturen

Dieses Kapitel befaßt sich mit den Strukturen der Daten. Dabei werden bestimmte Modelle zur Beschreibung dieser Strukturen vorgestellt.

1.4.1 Datenmodelle

Datenbankverwaltungssysteme werden danach unterschieden, welche Datenmodelle sie unterstützen. Datenmodelle beschreiben die Struktur der Daten und ihre Beziehungen zueinander. Man unterscheidet folgende Datenmodelle:

- Hierarchisches Modell
- Netzwerk-Modell
- Relationales Modell

In der ersten Generation der DBVSe konnte nur das hierachische Datenmodell verwendet werden; dann folgte das Netzwerk-Modell und schließlich kam ab ca. 1970 das relationale Modell mit der Abfragesprache SQL, das inzwischen zum dominierenden Modell geworden ist.

Beim *Hierarchischen Modell* sind die Datendateien hierarchisch angeordnet. Jeder Datensatz einer höheren Hierarchieebene enthält einen

Verweis auf die ihm zugeordneten Datensätze der nächst niedrigeren Ebene. Der Zugriff auf die Daten erfolgt durch „Navigieren“ des Anwenderprogrammes durch die Struktur, d. h. das Programm muß sich anhand der Verweise von Datei zu Datei durchhangeln, bis es den richtigen Datensatz gefunden hat.

Beispiel:

> Die Dateien Kunden, Aufträge und Artikel können in eine hierarchische Beziehung gebracht werden. Der Kundensatz enthält die Verweise auf die Aufträge, der Auftragssatz diejenigen auf die bestellten Artikel.

Der Vorteil des hierarchischen Datenmodells liegt in der schnellen Abarbeitung von hierarchischen Beziehungen, z. B. bei Stücklisten (Auflösung oder Teileverwendung). Nachteile sind die Datenunabhängigkeit und die fehlenden netzartigen Beziehungen.
Vertreter**:** IMS von IBM

Das *Netzwerkmodell* der Normierungsgruppe CODASYL ist eine Erweiterung des hierarchischen Modells um netzartige Strukturen. Der Zugriff aus dem Anwendungsprogramm erfolgt nach wie vor durch Navigation, s. o.
Vertreter**:** UDS von Siemens

Das *relationale Datenmodell* basiert auf dem Relationenkalkül der Algebra und unterstützt jede Form der Datenbeziehungen. Die Relationenalgebra ist in die mächtige Datenzugriffssprache SQL umgesetzt worden. Alle modernen DBVSe unterstützen das relationale Datenmodell.
Vertreter**:** DB2 von IBM, ORACLE, Informix u.v.a.m.

1.4.2 Objektorientierte Datenbank

Nach der starken Verbreitung von objektorientierten Programmiersprachen ist es nur noch eine Frage der Zeit, bis sich objektorientierte Datenbanken (OODB) am Markt etabliert haben. Um eine Aufwärtskompatibilität zu erreichen, haben viele OODBen einen relationalen und einen objektorientierten Teil; man spricht von einer objektrelationalen Datenbank. OODBen orientieren sich am Objektmodell, d. h. daß die Daten mit den Methoden, also den auf die Daten wirkenden Funktionen, eine Einheit (ein Objekt) bilden.

1.4.3 Multidimensionale Datenbank

Durch den konsequenten Einsatz von Datenbanken in Unternehmen hat sich inzwischen eine beträchtliche Datenmenge angesammelt. Für Controllingzwecke, aber auch für Analysen der Geschäftsleitung, bieten diese Daten eine hervorragende Basis, um zukünftige Entschei-

dungen vorzubereiten. Um das Tagesgeschäft nicht zu beeinträchtigen, und um die vorhandenen Daten geeignet aufzubereiten (Umwandlung, Verdichtung), wird für diese Analysen ein sogenanntes *Data Warehouse* genutzt, eine eigenständige, nur für diesen Zweck aufgebaute Datenbank. Die Struktur dieser Datenbank ist *multidimensional*, da die eigentlichen Daten (Fakten) im Zusammenhang mit bestimmten Dimensionen (Zeit, Ort, Produkt) betrachtet werden.

1.4.4 Datenbank-Architektur

Um die physische und logische Datenunabhängigkeit, d. h. die Unabhängigkeit der Anwendungsprogramme von der Struktur der Daten in der Datenbank und der Speicherform der Daten, zu erreichen, werden die gespeicherten Daten auf drei Abstraktionsebenen betrachtet. Man unterscheidet zwischen folgenden Datensichten:

- der *externen* Datensicht; hier handelt es sich um Sichtweisen auf die Datenbank, die an den Bedürfnissen der Benutzer orientiert sind (Views);
- der *konzeptionellen* (konzeptuellen oder logischen) Datensicht; sie bezeichnet die Sicht der Gesamtheit der Daten in der Datenbank und ist einheitlich und eindeutig;
- der *physischen* (internen) Datensicht; damit ist die interne Datenorganisation, d. h. die tatsächliche Speicherung der Daten gemeint.

Die Beschreibung der jeweiligen Sichten ist damit durch das externe, konzeptuelle und interne Schema gegeben.

2 Relationale Datenbank

Einen der Schwerpunkte des Kurses bildet die relationale Datenbank. Der Siegeszug des relationalen Modells begann 1970, als E.F. Codd in seiner Arbeit [Codd70] dieses Modell begründete. Die Firma Oracle war die erste, die ein geeignetes DBVS, ebenfalls mit dem Namen ORACLE, auf den Markt brachte. Oracle ist inzwischen der Marktführer im Datenbankbereich.

2.1 Das Konzept der relationalen Datenbank

Die Basis für das Speichern von Daten in einer relationalen Datenbank sind *Tabellen*. Diese können mit Karteikästen verglichen werden.

Beispiel:

> Die Tabelle KUNDE, siehe Anhang Auftrags-Datenbank, enthält die Kundendaten. Jeder Datensatz (Zeile) beschreibt dabei einen Kunden. Die einzelnen Felder enthalten Firmenname, Straße, PLZ, Ort, Telefon-Nummer usw. des Kunden.

2.1.1 Tabellen und Relationen

Eine Tabelle besteht aus *Zeilen* und *Spalten*. Eine Zeile enthält genau einen Datensatz, eine Spalte enthält pro Zeile ein Feld. Tabellennamen werden meist in der Einzahl gewählt.
Eine bestimmte Spalte der Tabelle ist gleichzeitig der *Primärschlüssel*. Der Wert jedes Feldes dieser Spalte muß in der Tabelle eindeutig sein. In manchen Fällen reicht dazu eine Spalte allein nicht aus. Dann muß eine weitere Spalte (oder sogar mehrere) dazu genommen werden, um den Primärschlüssel zu definieren. Um umfangreiche Primärschlüssel zu vermeiden, kann ein künstlicher Primärschlüssel eingeführt werden, z.B. eine Spalte mit einer laufenden Nummer pro Datensatz.

Beispiel:

> Ein Primärschlüssel für die Tabelle KUNDE könnte die Kombination der Spalten FIRMA, STRASSE und ORT sein, da Firma allein nicht eindeutig ist. Hier sollte besser eine Kundennummer (z. B. KuNr oder KKURZ) als Primärschlüssel eingeführt werden.

Bei Datenbanken greifen häufig mehrere Benutzer gleichzeitig mit eventuell unterschiedlichen Anwendungen, z. B. Auftragsbearbeitung,

Fakturierung, Vertriebsinformationssystem, auf die Daten zu. Deshalb ist es wichtig, daß die Daten aktuell, verfügbar und redundanzfrei sind.
Die Daten enthalten nicht nur sogenannte Stammdaten, z. B. Kunden oder Artikel, sondern auch Bewegungsdaten, z. B. Aufträge oder Kundenbesuche. Auch diese Daten sind in Tabellen gespeichert. Ein Datensatz beschreibt dann einen Auftrag oder einen Kundenbesuch.
Schauen wir uns einen Auftrag etwas genauer an. Ein Auftrag stammt i. d. R. von einem Kunden und enthält Leistungen, die für den Kunden erbracht wurden. Damit besteht eine *Beziehung* zwischen den Tabellen KUNDE und AUFTRAG und zwischen AUFTRAG und LEISTUNG.
Eine Beziehung wird durch einen *Fremdschlüssel* ausgedrückt. Dabei wird der Primärschlüssel der einen Tabelle als Fremdschlüssel(-Spalte) in der anderen Tabelle aufgenommen. Die Tabelle AUFTRAG bekommt also eine Spalte, die die Kundennummer KKURZ aus der Tabelle KUNDE zu dem jeweiligen Auftrag enthält. KUNDE wird dann zur *Mastertabelle* und AUFTRAG zur *Detailtabelle*. Es handelt sich um eine 1:n-Beziehung, d. h. 1 Kunde kann n Aufträge gegeben haben, während 1 Auftrag von 1 Kunden stammt. Zwischen AUFTRAG und LEISTUNG besteht eine analoge Beziehung.
Für eine Beziehung kann die Einhaltung der *referentiellen Integrität* bestimmt werden. Dann ist es in dem obigen Beispiel nicht mehr möglich, in der Detailtabelle eine Kundennummer durch Einfügen oder Ändern eines Datensatzes zu erzeugen, die nicht in der Mastertabelle enthalten ist. Umgekehrt kann auch kein Kunde gelöscht werden, der noch Aufträge hat.
Zunächst einige formale Definitionen:

Definition:

Sind $D_1, D_2, D_3, \ldots, D_n$ Mengen von Werten (Attributen), so ist

$$R \subseteq D_1 \times D_2 \times D_3 \times \ldots \times D_n$$

eine n-stellige *Relation* über den Mengen (Domains) D_i. Ein Element $r = (d_1, d_2, d_3, \ldots, d_n) \in R$ heißt *Tupel* der Relation R, wobei d_i die i-te Komponente von r ist.
(Hinweis: $\subseteq$ steht hier für die Teilmengenbeziehung,
x für das kartesische Kreuzprodukt von Mengen)

Eine Relation wird als zweidimensionale Tabelle dargestellt. Damit kommt sie dem einfachen Karteikasten sehr nahe. Die Tabelle hat Zeilen und Spalten. Eine Zeile entspricht einem Tupel, eine Spalte einer Attribut-Wertemenge. Eine andere Bezeichnung für Zeile ist auch Satz, eine andere für Spalte ist Feld.

Jede Zeile ist eindeutig innerhalb der Tabelle. Die Menge der Spalten (Attribute), deren Werte eine Zeile eindeutig bestimmen, wird als Pri-

märschlüssel der Tabelle bezeichnet. Es wird jedoch empfohlen, einen künstlichen Primärschlüssel, z. B. Kundennummer, zu verwenden. Sowohl Datenobjekte als auch Beziehungen zwischen den Objekten werden im relationalen Modell zu Tabellen. Dadurch ist die Behandlung von Beziehungen sehr einfach, nämlich wie die der Entities, geregelt. Falls in einer Tabelle eine Referenz auf den Schlüssel einer anderen Tabelle als Attribut enthalten ist, so wird dieses Attribut Fremdschlüssel genannt.

Beispiel:

> In einer Tabelle Auftrag ist die Kundennummer KKURZ enthalten, die zu einem Auftrag die entsprechende Kundennummer des Auftraggebers aus der Tabelle KUNDE enthält. KKURZ ist der Primärschlüssel von KUNDE.

2.1.2 Codd'sche Regeln

Damit das relationale Modell erreicht wird, müssen die sogen. 12 Codd'schen Regeln erfüllt werden. Hier ein Auszug der Anforderungen:

Entitäts-Integrität:
Jede Relation weist einen Primärschlüssel auf, z. B. die Kundennummer in KUNDE.

Referentielle Integrität:
Jeder Fremdschlüssel verweist auf einen Primärschlüssel aus einer anderen Relation, z. B. die Kundennummer in AUFTRAG.

Benutzerdefinierte Integrität:
Die Zulässigkeit von Werten für ein Attribut kann eingeschränkt werden, z. B. 10000 <= PLZ <= 89999 oder FERTIGIST >= FERTIGSOLL.

Auf die Relationen (Tabellen) des relationalen Modells sind bestimmte *Operationen* definiert. Dabei liegt die Relationenalgebra zugrunde:

Selektion:
Auswahl von bestimmten Zeilen mittels Auswahlkriterien. Aus der Tabelle wird eine Teilmenge von Datensätzen (Tupeln) ausgesucht, z. B. alle Kunden mit ORT = Bremen.

Projektion:
Auswahl von bestimmten Spalten der Tabelle. Die Ergebnismenge, d. h. das Ergebnis einer Abfrage, soll nur eine Teilmenge von Attributen (Spalten) haben, z. B. nur FIRMA und ORT.

Verbund (Join):
Mehrere Relationen können in einer Abfrage miteinander verknüpft werden, z. B. suche die Aufträge der Kunden, die in Bremen wohnen; die Daten werden dann aus den Tabellen AUFTRAG und KUNDE zusammengestellt.

2.1.3 Die Datenbanksprache SQL

SQL (Structured Query Language) ist die weitgehend standardisierte Sprache des relationalen Modells. Es liegen SQL-Standards von den Normierungsgremien ANSI, ISO und X/Open vor. Derzeit wird überwiegend der Sprachumfang SQL2 , siehe [SQL92], von den DBVSen angeboten. Der Vorgänger ist SQL1, siehe [SQL89]. Die Standardisierungsgremien arbeiten derzeit an einer SQL3-Version, die deutlich von der Objektorientierung (Objektorientierte Datenbank, OODB) geprägt ist.
Da praktisch alle relationalen Datenbanksysteme über SQL verfügen, werden Anwendungen, die mit SQL realisiert sind, voraussichtlich noch viele Jahre mit dieser DB-Sprache eingesetzt werden können.
SQL ist *deskriptiv*, d. h. nicht prozedural. Datenunabhängigkeit dadurch erreicht, daß in den Datenbankoperationen im Anwendungsprogramm nicht programmiert werden muß, *wie* die Daten gesucht werden, sondern nur noch *was* an Daten gewünscht wird. Wegen dieser Eigenschaft ist SQL auch zentraler Bestandteil der Sprachen der 4. Generation (4GL).
SQL ist *mengenorientiert*, d. h. das Ergebnis einer Datenbankabfrage kann aus mehreren Treffern bestehen. Man spricht dann von einer *Ergebnismenge*.
SQL ist nicht nur eine reine Abfragesprache, wie der Name vielleicht vermuten läßt. Mit SQL können alle Datenbankoperationen durchgeführt werden. SQL besteht aus drei Bereichen:

DDL Data Definition Language mit
CREATE (Anlegen von Tabellen, Sichten, Indexen, ...)
ALTER (Ändern)
DROP (Löschen)

DML Data Manipulation Language mit
INSERT (Einfügen von Zeilen)
UPDATE (Ändern)
DELETE (Löschen)
SELECT (Abfragen)

DCL Data Control Language mit
GRANT (Vergabe von Zugriffsrechten)
REVOKE (Zurücknahme von Zugriffsrechten)
COMMIT (Abschluß einer Transaktion)
ROLLBACK (Abbruch einer Transaktion)

SQL ist bei den meisten relationalen Datenbanksystemen die unmittelbare Schnittstelle zum DBVS. In einigen Fällen ist SQL als ein zusätzliches Interface realisiert, was auf der konkreten DB-Sprache aufsetzt.

2.2 Der Entwurf einer relationalen Datenbank

Beim Datenbankentwurf besteht die Aufgabenstellung darin, von der (Mini-) Welt des Unternehmens oder der Organisation zu einer geeigneten Datenbankstruktur für das verwendete Datenbankverwaltungssystem zu gelangen. Unter Beachtung der Datenunabhängigkeit sollte die Aufgabe in Teilschritten gelöst werden, wobei die Zwischenergebnisse noch gänzlich unabhängig vom verwendeten DBVS sein sollten. Im folgenden sollen die Schritte des Datenbankentwurfs kurz dargestellt werden.

1. Informationsbedarfs-Ermittlung und -Analyse
- Sammlung aller in einer Miniwelt des Unternehmens relevanten Objekte, ihrer Eigenschaften und Beziehungen.
- Überprüfung dieser Informationen auf inhaltliche Korrektheit.

2. Konzeptueller (logischer) Datenbankentwurf
- Entwurf der Tabellen und Definition der Attribute, Primärschlüssel, Fremdschlüssel, Datentypen.
- Methodik: Entity-Relationship-Model, Normalisierung.
- Festlegung weiterer Integritäts-Regeln wie Wertebereiche u. a. sowie Konsistenzregeln.

3. Physischer Datenbankentwurf
- Anlegen von Datencontainern auf den Platten des Datenbank-Computers mit Größe und Lage.
- Verteilung der Daten im Rechnernetz.
- Aufbau von Zugriffsstrukturen (Indexe).
- und viele weitere datenbanksystemabhängige Festlegungen.

Der physische Datenbankentwurf zum Anlegen der physischen Strukturen der Datenbank ist stark abhängig von dem verwendeten DBVS und der unterlagerten System- und Hardware-Architektur. U. a. müssen Speichergrößen (Hauptspeicherbereiche, Pufferpool, Plattenbereiche, LOG-Bereiche), Prozeßverwaltungselemente (Semaphore, Kommunikationselemente), Verteilungsaspekte und spezielle Verfahren wie Spiegel- oder Streifenplatten definiert werden. Einiges dazu wird in einem späteren Kapitel behandelt.

4. Externer Datenbankentwurf
- Definition von Benutzer-Sichten (Views) auf die Datenbank; Anlegen von Benutzern und Gruppen.
- Festlegung der Zugriffsrechte der Benutzer.

5. Realisierung des Entwurfs in dem konkreten DBVS

- Installation des DBVS
- Anlegen der Datenbank gemäß dem physischen Entwurf
- Anlegen der Tabellen gemäß dem logischen Entwurf
- Anlegen der Sichten, Indexe, ...

Die Strukturen des Datenbankentwurfs sind im *Data Dictionary* bzw. *Repository* beschrieben.
Im folgenden wird detailliert auf die zwei Methoden des logischen Datenbankentwurfs eingegangen, das Entity-Relationship-Model (ERM) und die Normalisierung.

2.3 Entity-Relationship-Model (ERM)

Das relationale Datenmodell definiert ein einheitliches Datenmodell. Die Interpretation der Daten bleibt jedoch dem Datenbank-Designer vorbehalten. Für den konzeptuellen Entwurf der Datenbank wird hier näher auf das *Entity-Relationship-Model (ERM)* von [Chen76] eingegangen, das sich ebenfalls als Standard auf dem Markt etabliert hat. Das ERM ist ein Top-down-Ansatz zum Entwurf, während die Normalisierung (s. u.) einem Bottom-Up-Vorgehen entspricht.
Als Beispiel betrachten wir eine Datenbank für das Projektmanagement. Zunächst werden die „starken“ und die „schwachen“ Entities ermittelt. Starke Entities sind vereinfacht solche, die eigenständig existieren können. Zu den starken gehören u. a. Mitarbeiter, Projekte und Abteilungen, zu den schwachen (abhängigen) Mitarbeiternamen, Projektnummern usw. Die schwachen Entities werden den starken zugeordnet. Ab jetzt werden nur noch die starken als Entities bezeichnet, die schwachen dagegen als Attribute der Entities.
Wir stellen so intuitiv, d. h. ohne weitere formale Regeln, eine erste Entitystruktur auf, wobei später jedes Entity zu einer Tabelle und die Attribute zu den Tabellenspalten werden.

Entity	Attribut	Erläuterung
Mitarbeiter	MaNr	Mitarbeiter-Nummer
	Name	Name des Mitarbeiters
	Vorname	Vorname des Mitarbeiters
	PLZ	Postleitzahl
	Ort	Wohnort
	Straße	Straße
	Geb-Dat	Geburtsdatum
	Ein-Dat	Eintrittsdatum
	Gehalt	Gehalt

	Qualifikation	Qualifikation des Mitarbeiters
Abteilung	AbtNr	Abteilungs-Nummer
	AbtBez	Abteilungs-Bezeichnung
	AbtLtr	Abteilungs-Leiter
Projekt	ProjNr	Projekt-Nummer
	ProjBez	Projekt-Bezeichnung
	ProjStart	Projekt-Start
	ProjEnde	Projekt-Ende
	Status	Projekt-Status
	Qualifikation	Erforderliche Qualifikation

Tab. 2: Intuitiver Entwurf einer Projektmanagement-Datenbank

Die Anforderung, daß bei den Mitarbeitern unterschiedliche Typen mit jeweils zusätzlichen speziellen Attributen vorliegen, ist noch nicht berücksichtigt:

- Techniker z. B. mit Überstundenregelung
- Manager z. B. mit variablem Gehaltsanteil
- Entwickler z. B. mit Heimarbeitszeitanteil
- Sekretärin z. B. mit Vertretung

Die unterstrichenen Attribute sind die Primärschlüssel des betreffenden Entity. Dieser erste, intuitive Entwurf wird nun weiter untersucht. Speziell die Beziehungen zwischen den Entities ist dabei von Interesse. Dabei werden die Beziehungen grafisch dargestellt, um den Entwurf besser analysieren zu können.
Für die grafische Darstellung der Entities und Beziehungen wird die Notation aus [Teor94] verwendet, siehe Anhang ERM Basisobjekte. In der Literatur sind einige andere Notationen zu finden, z. B. die „Krähenfuß“-Notation, die auch bei Oracle verwendet wird.
Bei den Beziehungen unterscheidet man zwischen ihrem Typ, der Existenz, dem Grad und weiteren Besonderheiten, die im folgenden aufgezeigt werden.

2.3.1 Typ der Beziehung

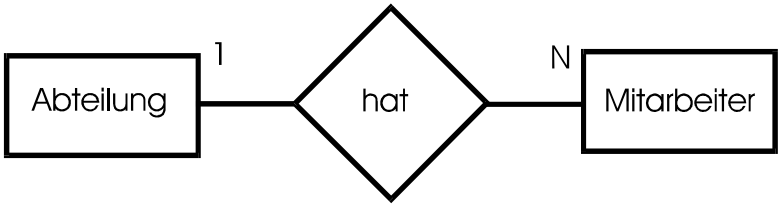
Es gibt folgende *Beziehungstypen* (auch *Kardinalität*):

Beziehungstyp 1:N:

Dieser Typ liegt vor, wenn zu einem Wert eines Entities A i. a. mehrere Werte eines anderen Entities B in Beziehung stehen, umgekehrt jedoch zu jedem Wert von B genau ein Wert von A eine Beziehung hat. Die Beziehungen werden also von beiden Seiten aus betrachtet.

Beispiel:

Eine Abteilung besteht aus mehreren (N) Mitarbeitern, während ein Mitarbeiter zu einer (1) Abteilung gehört.

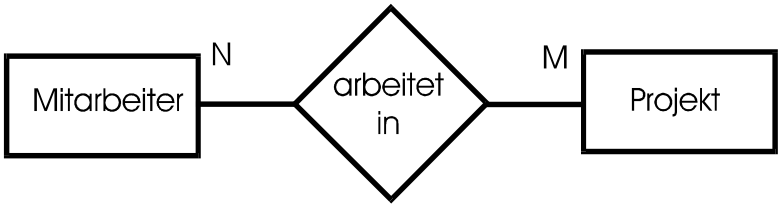


Beziehungstyp N:M:

Dieser Typ liegt vor, wenn zu einem Wert eines Entities ein oder beliebig viele Werte eines anderen Entities in Beziehung stehen und umgekehrt (many-to-many).

Beispiel:

Ein Mitarbeiter arbeitet in mehreren Projekten und zu einem Projekt gehören mehrere Mitarbeiter.

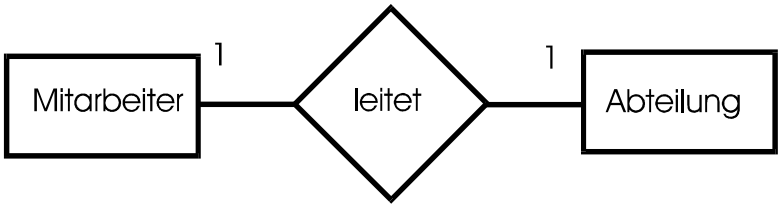


Beziehungstyp 1:1:

Dieser Typ liegt vor, wenn zu jedem Wert eines Entities A genau ein Wert eines anderen Entities B eine Beziehung hat.

Beispiel:

Der Leiter der Abteilung ist ein Mitarbeiter. Dabei soll jemand nur max. eine Abteilung leiten; für eine Abteilung soll es auch nur einen Abteilungsleiter geben.

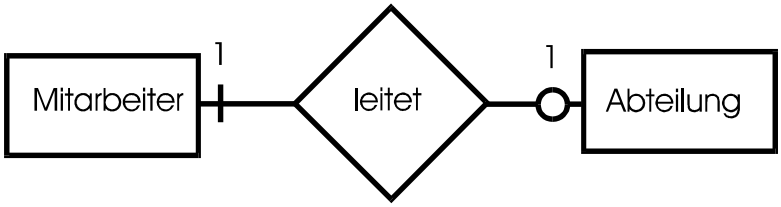


2.3.2 Existenz

Bei den Beziehungen ist noch eine weitere Eigenschaft wesentlich für den Entwurf, nämlich die Information darüber, ob die Beziehung opti-

onal (kann-Beziehung) oder obligatorisch (muß-Beziehung) ist. Diese Eigenschaft ist wiederum für beide Seiten der Beziehung zu untersuchen.

Beispielsweise gilt für die Beziehung



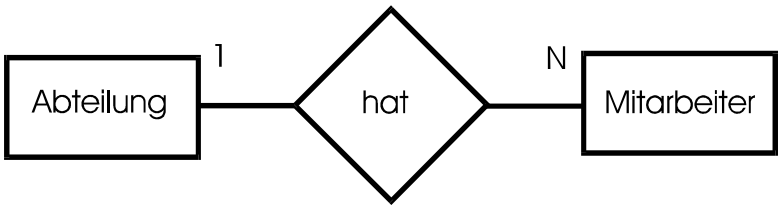
daß ein Mitarbeiter eine Abteilung leiten kann (Kreis), während eine Abteilung von einem Mitarbeiter geleitet werden muß (Strich). Falls unbekannt ist, ob es sich um eine kann- oder muß-Beziehung handelt, wird der Kreis bzw. der Strich weggelassen. Zu beachten ist die Position des kann- bzw. muß-Symbols.

2.3.3 Grad der Beziehung

Der Grad der Beziehung sagt aus, wieviel Entities an der Beziehung beteiligt sind.

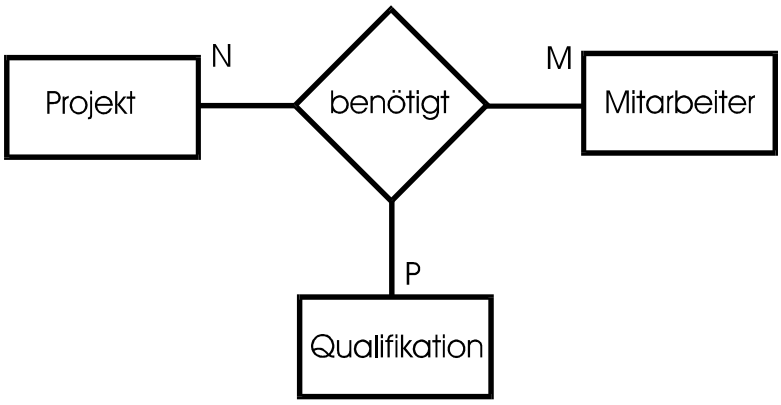
binär (2-stellig)

Es sind 2 Entities an der Beziehung beteiligt.



ternär (3-stellig)

Es sind 3 Entities an der Beziehung beteiligt.

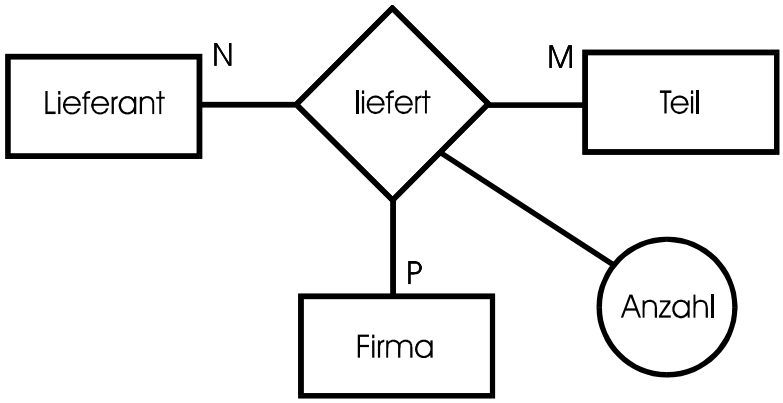


Beispiel:

> Zu einem bestimmten Projekt werden mehrere Mitarbeiter mit bestimmten Qualifikationen benötigt. Ein Mitarbeiter kann in mehreren Projekten arbeiten, braucht dazu jedoch verschiedene Qualifikationen. Eine Qualifikation kann in mehreren Projekten benötigt bzw. von mehreren Mitarbeitern erfüllt werden.

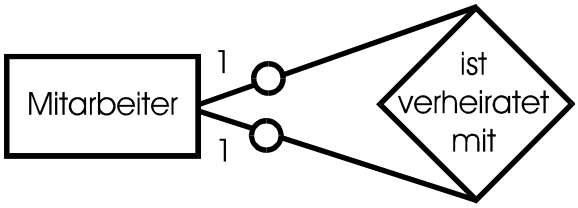
Ternäre Beziehungen sollten, wenn möglich, in binäre Beziehungen umgewandelt werden. Es gibt auch n-äre Beziehungen, d. h. Beziehungen mit n Entities.

Weiteres Beispiel für ternäre Beziehung:



rekursiv binär

Es ist nur 1 Entity an der Beziehung beteiligt, die Beziehung ist jedoch rekursiv.



oder andere Darstellung:

Weitere Beispiele:

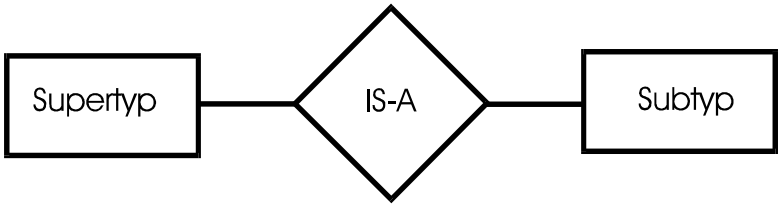
> Stückliste (Teil ist Oberteil zu Teil mit Zusatzattribut Anzahl; N:M).
> Gruppenleiter (Mitarbeiter leitet Mitarbeiter; 1:N).

Vater/Sohn (Person ist Vater von Person; 1:N).
Abteilungshierarchie (Abteilung ist übergeordnet zu Abteilung; 1:N).

2.3.4 Generalisierung/Spezialisierung (IS-A):

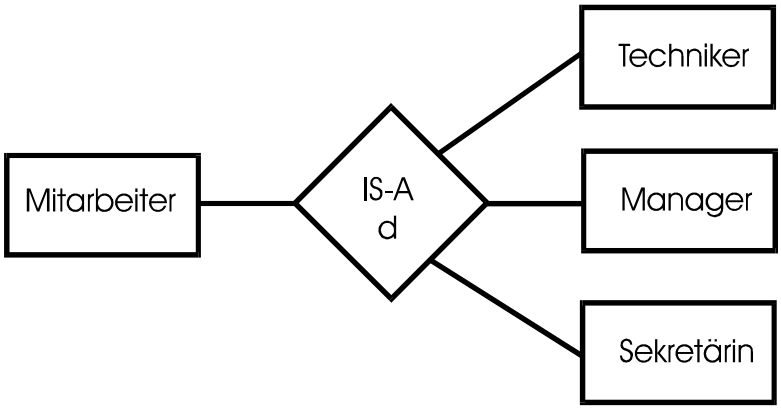
Dieser Typ liegt vor, wenn eine Teilmenge (Subtyp) weitere Attribute gegenüber der Grundmenge (Supertyp) hat. Die so entstehenden Teilmengen können die Grundmenge total oder partiell überdecken sowie disjunkt oder nicht disjunkt sein.

Darstellung:



Beispiel:

Ein Mitarbeiter kann ein Techniker, ein Manager oder eine Sekretärin sein. In allen Fällen ist er/sie zunächst ein Mitarbeiter mit den betreffenden Attributen. Zusätzlich hat jeder noch weitere Attribute, die den jeweiligen Subtyp näher beschreiben. Das d unter IS-A gibt an, daß die Teilmengen disjunkt sind.



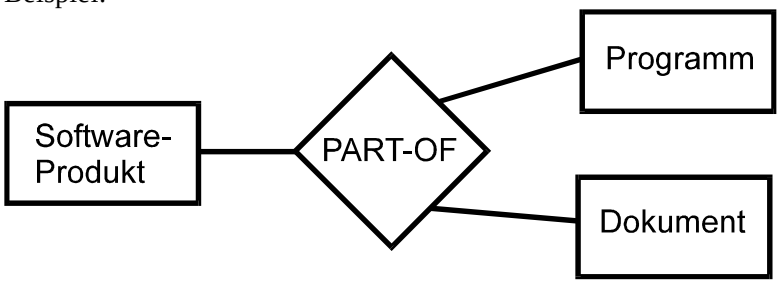
Ein Artikel besitzt nur die Attribute ArtNr, ArtBezeichnung, Einkaufs- und Verkaufspreis. Ein Lagerartikel ist ein Artikel mit zusätzlichen Attributen MinBestand und LagerOrt.
Dann gilt: Lagerartikel IS-A Artikel

Bei einer **Generalisierung** wird aus einer oder mehreren Teilmengen eine Grundmenge abgeleitet.

2.3.5 Aggregation (PART-OF):

Dieser Typ liegt vor, wenn ein Entity aus mehreren eigenständigen Entities zusammengesetzt ist.

Beispiel:



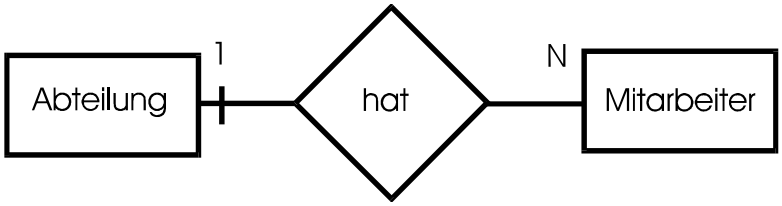
2.4 Regeln zur Bildung von Tabellen aus ERM

Aus der Art der Beziehung der Entities kann die Tabellenstruktur der Datenbank abgeleitet werden. Die Beziehung zwischen zwei Entities wird mit Hilfe von *Fremdschlüsseln* dargestellt. Es gelten folgende Regeln:

2.4.1 1:N-Beziehung

Eine 1:N-Beziehung wird in zwei Tabellen abgebildet, wenn die Teilnahme der N-Entity obligatorisch ist. Der Primärschlüssel der 1-Seite wird Fremdschlüssel in der anderen Tabelle.

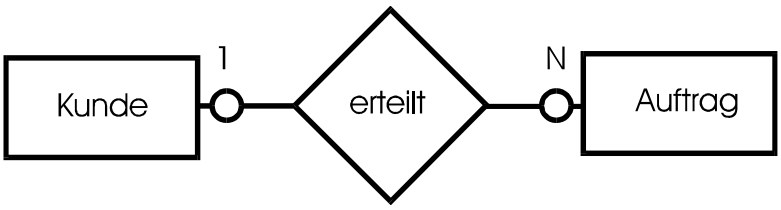
Im Fall von:



wird die AbtNr, der Primärschlüssel von Abteilung, zum Fremdschlüssel in der Tabelle Mitarbeiter.

Eine 1:N-Beziehung kann in drei Tabellen abgebildet werden, wenn die Teilnahme der N-Seite optional ist und wenn die Beziehungsausprägung relativ gering ist im Vergleich zu der Anzahl der Elemente der N-Seite. Die Beziehung wird dann in einer zusätzlichen Relation abgebildet, die nur die tatsächlichen Beziehungs-Ausprägungen enthält. Der Primärschlüssel dieser Beziehungstabelle wird in der Regel aus den Primärschlüsseln der beiden ersten Tabellen gebildet, die damit zu Fremdschlüsseln werden.

Bei folgender Beziehung



sollte eine dritte Tabelle erstellt werden, wenn es relativ wenig Aufträge mit Kundenbezug gibt, z. B. Lageraufträge:

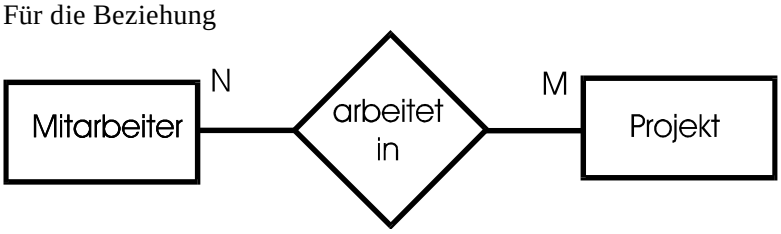
KundenAuftrag (KunNr, AuftrNr, Lieferadresse)

Im Fall von nur 2 Tabellen wären die Spalten KunNr und Lieferadresse in der Auftragstabelle für alle Aufträge ohne Kundenbezug leer.

2.4.2 N:M-Beziehung

Für eine solche Beziehung wird eine zusätzliche Beziehungs-Tabelle eingeführt (also drei Tabellen). Die Primärschlüssel der Grundtabellen werden zu Fremdschlüsseln in der Beziehungstabelle und bilden dort i. a. den neuen Primärschlüssel.

Für die Beziehung



wird eine neue Tabelle

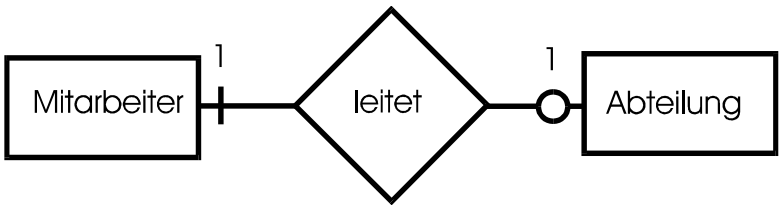
ProjektMitarbeit (ProjNr, MaNr, AnzahlStunden)

angelegt. Damit wird die N:M-Beziehung auf zwei 1:N-Beziehungen zwischen Mitarbeiter und ProjektMitarbeiter bzw. zwischen Projekt und ProjektMitarbeiter reduziert.

2.4.3 1:1-Beziehung

Eine 1:1-Beziehung wird in einer Tabelle abgebildet, wenn die Teilnahme beider Entity-Mengen obligatorisch ist. Beide Entity-Schlüssel sind Schlüsselkandidaten der Tabelle, wobei festgelegt werden muß, welcher der beiden der Primärschlüssel wird.

Eine 1:1-Beziehung wird in zwei Tabellen abgebildet, wenn die Teilnahme von nur einer der beiden Entity-Mengen obligatorisch ist. Der Primärschlüssel der obligatorischen Seite wird Fremdschlüssel in der anderen Tabelle. Bei folgender Beziehung



werden also zwei Tabellen angelegt, wobei die MaNr aus Mitarbeiter (also die MaNr des Abteilungs-Leiters) zum Fremdschlüssel in Abteilung wird.

Eine 1:1-Beziehung kann in drei Tabellen abgebildet werden, wenn die Teilnahme beider Entity-Mengen optional ist. Die Beziehung wird in einer zusätzlichen Tabelle dargestellt, die nur die tatsächlichen Beziehungs-Ausprägungen enthält. Der Primärschlüssel der dritten Tabelle setzt sich i. a. aus den Primärschlüsseln der beiden Grundtabellen zusammen, die dann auch Fremdschlüssel sind. In folgendem Beispiel

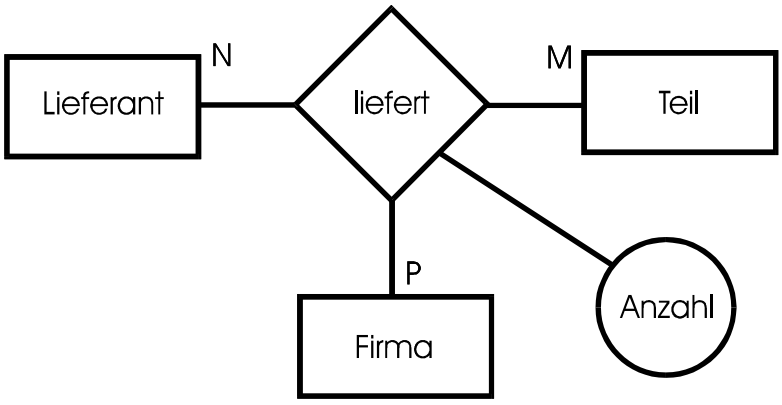
sollte eine zusätzliche Tabelle

MitarbeiterEhe (MaNr1, MaNr2, verheiratet_seit)

angelegt werden, da eine solche Ehe relativ selten ist. Diese Tabelle wäre dann die „dritte“ Tabelle, da die Mitarbeiter-Tabelle durch die rekursive Beziehung quasi doppelt gezählt werden kann.

2.4.4 Ternäre Beziehungen

Bei ternären Beziehungen werden insgesamt vier Tabellen angelegt. Zu den Grundtabellen kommt noch eine Beziehungstabelle hinzu, die sämtliche Primärschlüssel der Grundtabellen als Fremdschlüssel enthält. In folgendem Fall



wird neben den Tabellen für Lieferant, Teil und Firma eine Tabelle Lieferung für die Beziehung „liefert“ wie folgt angelegt:

Lieferung (LieferantNr, TeilNr, FirmenNr, Anzahl).

2.4.5 IS-A-Beziehung

Zusätzlich zur Tabelle für die Grundmenge wird eine weitere Tabelle für jede Teilmenge angelegt, die denselben Primärschlüssel wie die Grundmenge hat. Im Beispiel mit

Mitarbeiter IS-A Techniker

werden folgende Tabellen definiert:

Mitarbeiter (MaNr, Name, Vorname, ...)
Techniker (MaNr, Überstunden)

Totale Überdeckung bzw. Disjunktivität können durch Integritätsregeln sichergestellt werden (Teilmengenbeziehung bzw. leerer Durchschnitt, s. [Voss94], S. 138).

2.4.6 PART-OF-Beziehung

Bei solchen Beziehungen wird je eine Tabelle für den Super- und den Subtyp angelegt. Die Beziehung selbst wird nur dokumentiert, nicht jedoch per Fremdschlüssel dargestellt.

2.4.7 Kann/Muß-Beziehung

Abhängig davon, ob es sich um eine muß- oder kann-Beziehung handelt, muß die Fremdschlüsselspalte einen Wert haben oder nicht.

2.4.8 Rekursive Beziehungen

Im Fall der Stückliste (Teil ist Oberteil zu Teil mit Zusatzattribut Anzahl; N:M) werden zwei Tabellen angelegt, die wie folgt aussehen können:

Teile (TeileNr, Bezeichnung, Gewicht, ...)
Struktur (OberTeileNr, UnterTeileNr, Anzahl)

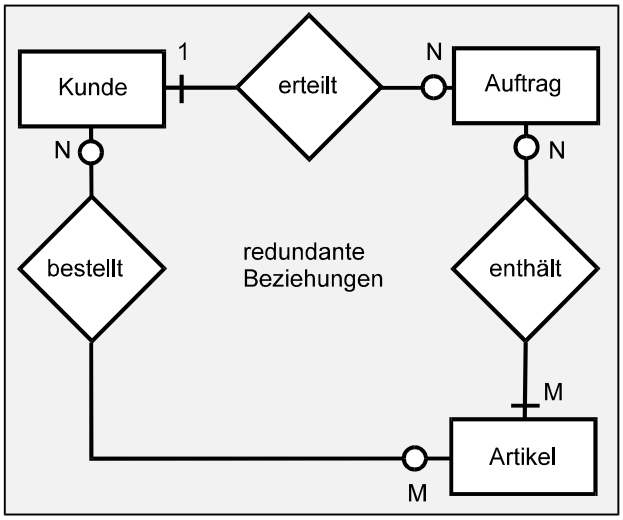
Bei der Gruppenleiter-Beziehung (Mitarbeiter leitet Mitarbeiter; 1:N) kann folgende Tabelle angelegt werden:

Mitarbeiter (MaNr, Name, Beruf, AbtNr, Gehalt, MaNrGrpLeiter)

MaNrGrpLeiter ist die MaNr des jeweiligen Gruppenleiters

2.4.9 Redundante Beziehungen

Redundante Beziehungen dürfen nicht in Tabellen umgesetzt werden, da sie sonst zu Datenredundanzen führen können. Sie werden dann besser erkannt, wenn man sämtliche Beziehungen in einem einzigen Beziehungsgraphen darstellt. Überall dort, wo ein geschlossener Beziehungskreis vorliegt, muß auf Redundanz untersucht werden.



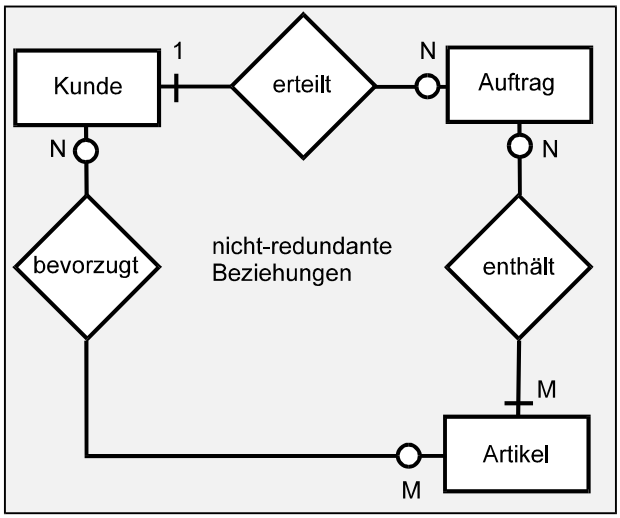


Abb. 2: Redundante und nicht-redundante Beziehungen

2.4.10 ERM und Tabellenstruktur

Das vollständige ERM der Projektdatenbank sieht wie folgt aus:

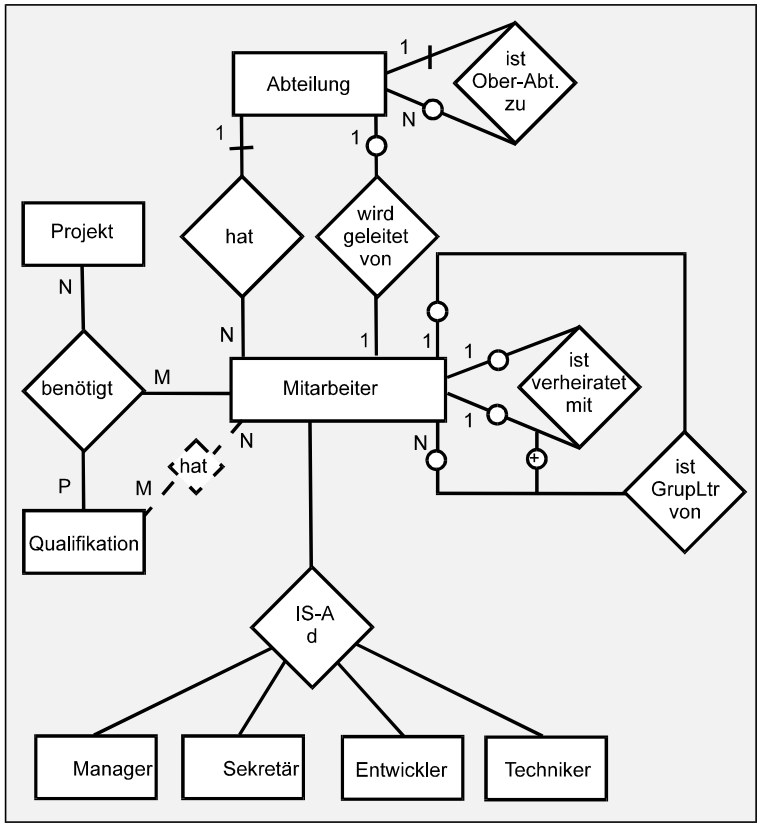


Abb. 3: Vollständiges ERM

Aus dem ERM ergibt sich damit folgende Tabellenstruktur. Die Fremdschlüssel sind unterstrichen (gestrichelt).

Tabelle	Primärschlüssel	Attribute
Abteilung	AbtNr	Bezeichnung, Aufgabe, übergeordnAbtNr (1), AbtLtr
Mitarbeiter	MaNr	Name, Vorname, AbtNr
Qualifikation	Qualifikation	
Projekt	ProjNr	ProjBez, Start, Ende, Status
ProjektMitarbeit	ProjNr, MaNr,	Summe_Stunden
	Qualifikation	
MaEhen	Mann, Frau (2)	verheiratet_seit, Anzahl_Kinder
Gruppe	MaNr, GrpLtr (3)	seit_wann
Manager	MaNr	Manager_seit, var_Gehalt, ...
Techniker	MaNr	Spezialgebiet, Q_Stufe, ...
MaQualifikation	MaNr, Qualifik.	Q-Stufe

Tab. 3: Vollständige Tabellenstruktur

(1): übergeordnAbtNr ist die AbtNr der übergeordneten Abteilung
(2): Mann und Frau sind jeweils MaNr aus Mitarbeiter
(3): GrpLtr ist die MaNr des Gruppenleiters aus Mitarbeiter

2.5 Normalisierung

Als Bottom-up-Methode zum Datenbank-Entwurf hat die Normalisierung als wesentliches Ziel die Beseitigung von Redundanzen und ungewollten Nebeneffekten („Anomalien“) bei Datenänderungen wie Einfügen, Ändern und Löschen von Datensätzen.
Bei der Normalisierung werden die Relationen durch Auslagerung von Attributen in kleinere Einheiten aufgeteilt, oder die Attribute werden in bereits bestehende Relationen verlagert bzw. einfach gestrichen. Es gibt keinen Zwang zur Normalisierung, sondern nur Regeln, wie normalisiert werden soll. Normalisierung ist anwendungsabhängig und nicht immer eindeutig. Je weiter normalisiert wird, desto eher können durch die dabei erzeugten vielen kleinen Relationen bei den Datenbank-Anwendungen Performance-Engpässe auftreten, weil die Anwendungen häufig Verbunde benötigen, um die Informationen wieder in logischen Einheiten zusammenzufassen.

Beispiel für eine *Löschanomalie*:

Eine Tabelle ARTIKEL enthält folgende Spalten:

ArtNr, ArtBez, Preis, LagNr, LagOrt, LagStr

Es werden also die Lagerinformationen LagOrt (Lagerort) und LagStr (Lagerstrasse) bei den Artikeln gespeichert. Neben der erheblichen Redundanz kann noch folgendes passieren. Wenn zufällig alle Artikel eines bestimmten Lagers gelöscht werden, ist auch die Information über die Lageradresse in der Datenbank gelöscht!
Am Ende der Normalisierung können Tabellen mit gleichen Primärschlüsseln zu einer Tabelle zusammengefaßt werden. ERM und Normalisierung erzeugen dieselbe Tabellenstruktur. Deshalb kann beim Datenbank-Entwurf zunächst die ERM-Methode benutzt werden, um dann das Ergebnis mit der Normalisierung zu überprüfen.
In der Literatur werden bis zu 12 (und mehr) Normalformen beschrieben. Es werden hier jedoch nur die 1. bis 4. Normalform behandelt.
Es gelten folgende Regeln für die Normalisierung von Tabellen. Dabei gilt generell, daß die n – te Normalform die n-1 – te beinhaltet.

2.5.1 „1. Normalform“

Eine Tabelle ist in der 1. Normalform (1NF), wenn sie sich als lineare Tabelle darstellen läßt, d. h. es gibt keine Tabelle in der Tabelle. Jedes Feld ist ein einfacher Wert. Dies kann immer durch Auslagerung von unterstrukturierten Feldern zu anderen/eigenständigen Tabellen erreicht werden. Der Bezug auf diese neue Tabelle wird durch Fremdschlüssel hergestellt.

Beispiel:

> Tabelle KUNDE mit
> KuNr, Firma, AufNr, Ort, ...

Falls der Kunde mehrere Aufträge erteilt hat, enthält die Spalte AufNr (Auftragsnummer) eine Liste von Werten, was einer Tabelle entspricht.
Lösung: Streichen des Attributs AufNr in der Tabelle KUNDE und Einführen eines neuen Attributs KuNr in der Tabelle AUFTRAG als Fremdschlüssel.

2.5.2 „2. Normalform“

Eine Tabelle ist in 2NF, wenn sie in 1NF ist und jedes Nicht-Schlüsselattribut von **allen** Schlüsselattributen abhängig ist. Mit Schlüssel ist hier der Primärschlüssel gemeint.
Dies wird erreicht, indem die nicht abhängigen Nicht-Schlüsselattribute in eigene Tabellen ausgelagert werden.

Beispiel:

> Eine Tabelle POSITION enthält die Auftragspositionen mit folgender Struktur:
> ArtNr, AufNr, Menge, Preis

Der Preis ist nur von dem Schlüsselattribut ArtNr (Artikelnummer) abhängig, sofern kein auftragsspezifischer Preis gilt.
Lösung: Auslagern des Attributs Preis in die Tabelle ARTIKEL.

2.5.3 „3. Normalform“

Eine Tabelle ist in 3NF, wenn sie in 2NF ist und jedes Nicht-Schlüsselattribut von allen Schlüsselattributen nicht transitiv abhängig ist. Kurz: Die Nicht-Schlüsselfelder müssen unabhängig voneinander sein (Mit Schlüssel ist wieder der Primärschlüssel gemeint).

Die 3NF wird wieder durch Auslagerung erreicht.

Beispiel:

Eine Tabelle ARTIKEL mit
ArtNr, ArtBez, Preis, LagNr, LagOrt, LagStr

LagOrt (Lagerort) und LagStr (Lagerstraße) sind von LagNr (Lagernummer) und damit transitiv von ArtNr (Artikelnummer) abhängig.
Lösung: Auslagern der Attribute LagOrt und LagStr in eine Tabelle LAGER.

2.5.4 „4. Normalform“

Eine Tabelle ist in 4NF, wenn sie in 3NF ist und keine aus der Datenbank abgeleiteten Felder, z. B. durch Berechnung, enthält. Dabei muß auch untersucht werden, ob sich ein Feld nicht aus Feldern anderer Tabellen ableiten läßt.

Beispiel:

Eine Tabelle RECHNUNG mit
RechNr, RechDat, AufNr, NettoWert, MWSt, BruttoWert

Der BruttoWert kann mittels NettoWert und der MWSt berechnet werden.
Lösung: Streichen der Spalte BruttoWert.

Aus der Tabellenstruktur vor der Normalisierung wird durch Normalisieren die verbesserte Struktur. Dabei sind die Fremdschlüssel wieder gestrichelt.

2.5.5 Tabellen vor und nach der Normalisierung

Im den beiden folgenden Tabellen wird die Struktur vor und nach der Normalisierung dargestellt:

Tabelle	Primärschlüssel	Attribute
KUNDE	KuNr	Firma, AufNr, Ort
AUFTRAG	AufNr	AufDat, LiefDat
ARTIKEL	ArtNr	ArtBez, LagNr, LagOrt, LagStr
POSITION	ArtNr, AufNr	Menge, Preis
RECHNUNG	RechNr	RechDat, AufNr, NettoWert, MWSt, BruttoWert

Tab. 4: Tabellen vor der Normalisierung

Tabelle	Primärschlüssel	Attribute
KUNDE	KuNr	Firma, Ort
AUFTRAG	AufNr	KuNr, AufDat, LiefDat
ARTIKEL	ArtNr	ArtBez, Preis, LagNr
POSITION	ArtNr, AufNr	Menge
RECHNUNG	RechNr	RechDat, AufNr, MWSt
LAGER	LagNr	LagOrt, LagStr

Tab. 5: Tabellen nach der Normalisierung

2.5.6 Non-First-Normal-Form

Die 1NF ist eine notwendige Voraussetzung für das relationale Modell für Standardanwendungen. Bei sogenannten Non-Standardanwendungen in Bereichen wie CAD oder Büro führt die 1NF zu unakzeptablen Performance-Problemen. Deshalb ist der Begriff *Non-First-Normal-Form***,** NFNF oder NF^2, eingeführt worden, um auch Tabellenstrukturen, die nicht in 1NF sind, zu verarbeiten.

2.6 Vorgehen beim Entwurf

Beim Entwurf werden zusammenfassend folgende Aktivitäten durchgeführt:

1. Definition der Aufgabenstellung
Zunächst wird die zu lösende Aufgabenstellung klar umrissen. Dabei können größere Vorhaben in mehrere kleine Zwischenschritte aufgeteilt werden.

2. Informationsbeschaffung
Es werden alle für die Anwendung benötigten Informationen gesammelt. Dabei werden alle Daten unstrukturiert aufgeschrieben, z. B. „Firmenname“, „Kunde“, „Auftragsdatum“, „Artikelnummer“, „Artikelbezeichnung“, „Preis“, „Firmenadresse“, ...

3. Bestimmen der Entities mit ihren Attributen
Als nächstes werden intuitiv die für die zu lösende Aufgabenstellung benötigten Entities festgelegt. Dabei werden die Daten der Informationsbeschaffung strukturiert, indem zusammengehörige Daten zusammengefaßt und einem Oberbegriff zugeordnet werden. Also „Firmenname“ und „Firmenadresse“ zu dem Oberbegriff „Kunde“, „Artikelnummer“, „Artikelbezeichnung“ und „Preis“ zu „Artikel“, „Auftrags-

datum“ zu „Auftrag“ usw. Die Oberbegriffe sind die starken Entities, die anderen die schwachen. Bei der Strukturierung ist zu beachten, daß:

- jedes Attribut eines Entities einen direkten Bezug zu diesem Entity hat,
- alle benötigten Informationen als Entities bzw. Attribute auftauchen und
- keine abgeleiteten oder berechneten Attribute existieren.

4. Bestimmen des Primärschlüssels
Nun wird dasjenige Attribut bestimmt, dessen Wert innerhalb des Entities eindeutig ist. Falls kein solches existiert, werden mehrere geeignete Attribute zum Primärschlüssel zusammengefaßt oder ein künstlicher Primärschlüssel z. B. als laufende Nummer angelegt.

5. Ermitteln der Beziehungen
Mit Hilfe des ERM werden die Beziehungen zwischen den bisher definierten Entities festgestellt.

6. Ableiten der Tabellenstruktur aus dem ERM
Aus dem ERM werden die Tabellenstrukturen einschließlich der Fremdschlüssel abgeleitet.

7. Überprüfung des Entwurfs mit Hilfe der Normalisierung der so gewonnenen Tabellen (bis 4NF). Damit können logische Fehler bei der ERM-Methode festgestellt werden.

8. Festlegung der Datentypen und Integritätsregeln
Festlegung der Datentypen und Aufstellen der Integritätsregeln der Attribute und der Regeln für die referentielle Integrität.

9. Test des Entwurfs
Nun wird der Entwurf auf Fehler untersucht. Dazu wird die Datenbank als Prototyp erstellt und es werden einige Testdaten in die Tabellen eingegeben. Anschließend wird geprüft, ob alle gewünschten Informationen aus den Tabellen abrufbar sind. Dazu sollte mit Prototypen von Formularen und Berichten experimentiert werden. In dieser Phase ist eine Änderung des Entwurfs und der Tabellendefinitionen noch problemlos durchführbar. Wenn dagegen größere Datenmengen erfaßt und viele Formulare, Berichte, Abfragen und Makros/Programme (siehe weiter unten) angelegt sind, hat eine Änderung des Entwurfs eine Vielzahl von weiteren Änderungen in den Bestandteilen der Anwendungen zur Folge.

10. Anlegen von Benutzersichten und Zugriffsrechte
Zum Schluß werden Benutzersichten (Views) angelegt und Zugriffsrechte vergeben.

2.7 Ein Beispiel: Auftragsbearbeitung

Für die nachfolgenden Kapitel wird eine *Auftragsbearbeitung* als Fallstudie entwickelt. Dabei soll die Auftragsbearbeitung eines EDV-Beratungsunternehmens per Datenbankanwendung unterstützt werden. Das Unternehmen erbringt Dienstleistungen in Form von Beratungen, Schulungen, Gutachten und Artikel für Zeitschriften und Bücher. Die Abrechnung soll per Stundensatz oder gemäß einer auftragsspezifischen Abrechnungseinheit erfolgen.
Automatisiert werden soll per Datenbankanwendung:

- die Verwaltung der Kundenadressen,
- die Verwaltung der Auftragsdaten,
- die Verwaltung der erbrachten Leistungen
- die Erstellung von Übersichten über:
  - den vorhandenen Auftragsbestand,
  - zu erwartende Einnahmen,
  - erbrachte, noch nicht abgerechnete Leistungen,
  - schon abgerechnete Leistungen,
  - Kundenumsätze;
- die Erstellung von Rechnungen über erbrachte Leistungen.

Die Rechnungen sollen jederzeit rekonstruierbar sein; zur Verfolgung von Zahlungseingängen wird eine Rechnungsausgangsliste benötigt. Ein Rechnungsmuster ist im Anhang dargestellt.

2.7.1 Intuitiver Entwurf der Auftrags-Datenbank

In einem ersten Ansatz können folgende Entities für die Auftrags-Datenbank festgestellt werden:

- KUNDE
- AUFTRAG
- LEISTUNG
- AUFTRAGSBESTANDLISTE
- AUFTRAGSWERTLISTE

Die beiden Listen werden für Ausgaben der Anwendung benötigt. Den Entities werden folgende Attribute zugeordnet:

KUNDE

Name	Bedeutung
KKURZ	Kundenkürzel, Primärschlüssel
FIRMA	Name der Firma
ZUSATZ	evtl. Zusatz zum Firmennamen
STRASSE	
LKZ	Länderkennzeichen

PLZ	
ORT	
KTELEFON	Telefonnummer (Firmenzentrale)
KINFO	Kundeninformation (freier Text)
UMSATZ	aus fakturierten Aufträgen kumulierter Umsatz
VORNAME	der Kontaktperson
NACHNAME	der Kontaktperson
ANREDE	„Herr“, „Frau“
KONTELEFON	Durchwahl der Kontaktperson
FUNKTION	Stellung in der Firma
KONINFO	Kurzinfo bzgl. der Kontaktperson
ABTEILUNG	der Kontaktperson

AUFTRAG

Name	Bedeutung
AKURZ	Auftragskürzel; Primärschlüssel
KKURZ	Kundenkürzel (Auftraggeber)
PNR	Personalnummer der Kontaktperson
ADATUM	Datum der Auftragserteilung
FERTIGIST	tatsächliches Fertigstellungsdatum
FERTIGSOLL	geplanter Fertigstellungstermin
ATEXT	Kurzbeschreibung des Auftrags
AEINHEIT	Berechnungseinheit (Tag/Seite/Stunde/Exemplar/Stück/Festpreis)
AANZ	Volumen, geplante Anzahl Einheiten
ASATZ	Preis für eine Einheit
ANETTO	Nettovolumen = AANZ*ASATZ
MWSATZ	Mehrwertsteuersatz (0,0 oder 0,7 oder 0,15)
STATUS	geplant/Vertrag/erledigt/Ablage

LEISTUNG

Name	Bedeutung
LNR	Leistungsnummer; Schlüssel
AKURZ	Auftragskürzel des zugehörigen Auftrags
LDATUM	Datum der Leistungserbringung
LTEXT	Kurzbeschreibung der Leistung
LEINHEIT	Leistungseinheit
LANZ	Anzahl Leistungseinheiten
LSATZ	Leistungssatz; Nettobetrag pro Einheit
LNETTO	Nettobetrag pro Leistung = LANZ*LSATZ
MWSATZ	Mehrwertsteuersatz
RNR	Rechnungsnummer; wird automatisch erzeugt, wenn die Leistung berechnet wird

AUFTRAGSBESTANDSLISTE

Name	Bedeutung
AKURZ	Auftragskürzel
ATEXT	Kurzbeschreibung des Auftrags
FIRMA	Firmenname des Auftraggebers
AEINHEIT	Berechnungseinheit
AANZ	Anzahl Berechnungseinheiten
FERTIGSOLL	geplanter Fertigstellungstermin
STATUS	geplant/Vertrag/erledigt/Ablage

AUFTRAGSWERTLISTE

Name	Bedeutung
AKURZ	Auftragskürzel
KKURZ	Kundenkürzel (Auftraggeber)
ADATUM	Datum der Auftragserteilung
STATUS	nur die Werte „geplant“ und „Vertrag“
FERTIGSOLL	geplanter Fertigstellungstermin
VOLUMEN	berechnet: AANZ*ASATZ
MWSTEUER	berechnet: VOLUMEN*MWSATZ
BRUTTO	berechnet: VOLUMEN+MWSTEUER

2.7.2 Entwurf der Auftrags-Datenbank

Ein detailliertes ERM mit anschließender Normalisierung müßte sich jetzt anschließen. Es werden jedoch nur einige Hinweise zu den Besonderheiten des Entwurfs gemacht.
Die Auftragswert- und die Auftragsbestandsliste sind Ausgaben, die per Benutzeranforderung aus den Daten der Datenbanktabellen erzeugt werden. Zur Ermittlung eines Ausgabewertes kann direkt auf eine Tabellenspalte zugegriffen werden (z. B. ATEXT aus der Tabelle AUFTRAG) oder der Wert wird aus Tabellenwerten berechnet (z. B. MWSTEUER = AANZ*ASATZ*MWSATZ, alle aus AUFTRAG).

Die drei Tabellen KUNDE, AUFTRAG und LEISTUNG werden zu Datenbanktabellen. Es ist jedoch sofort zu erkennen, daß die Tabelle KUNDE nicht in der 1NF ist, falls es zu einem Kunden mehrere Kontaktpersonen geben kann. Deshalb wird eine zusätzliche Tabelle PERSON mit folgenden Attributen benötigt:

PERSON
PNR, VORNAME, NACHNAME, ANREDE

PNR ist der Primärschlüssel; die Attribute von PERSON sind dann in KUNDE nicht mehr enthalten.

Zwischen KUNDE und PERSON ist eine N:M-Beziehung gegeben. Dies resultiert daraus, daß ein Kunde mehrere Kontaktpersonen haben

kann und daß z. B. durch Firmenwechsel innerhalb eines Projektzeitraums (Beginn und Ende eines Auftrags) eine Kontaktperson zu mehreren Kunden gehören kann. Diese N:M-Beziehung wird dadurch aufgelöst, daß eine zusätzliche Beziehungsmenge KONTAKT durch Herauslösung bestimmter Attribute aus KUNDE erstellt wird:

KONTAKT

PNR, KKURZ, KONTELEFON, FUNKTION, KONINFO, ABTEILUNG

PNR und KKURZ sind Fremdschlüssel und verweisen auf die Person bzw. den Kunden. Beide zusammen bilden innerhalb KONTAKT den Primärschlüssel, da nur sie beide allein ein Tupel von einem anderen eindeutig unterscheiden.

Die Tabelle KUNDE sieht dann wie folgt aus:

KUNDE

KKURZ, FIRMA, ZUSATZ, STRASSE, LKZ, PLZ, ORT, KTELEFON, KINFO, UMSATZ

Durch die Tabelle LEISTUNG wird ein Mehrfachauftreten von Leistungen innerhalb der Tabelle AUFTRAG vermieden. Die Tabelle LEISTUNG ist auch erforderlich, um bei Auftragsänderungen die Historie zum ursprünglichen Auftrag festzuhalten.
Die o. g. Tabellen sind bereits alle in der 2NF. Ein Verstoß gegen die 2NF wäre z. B. folgender:

> Nachname der Person in KONTAKT; der Nachname ist funktional abhängig von dem 1. Schlüsselattribut PNR, aber nicht funktional abhängig von dem 2. Schlüsselattribut KKURZ.

Die o. g. Tabellen sind auch alle in der 3NF. Ein Verstoß gegen die 3NF wäre z. B. folgender:

> Nachname der Person in AUFTRAG; der Nachname ist nicht direkt abhängig von dem Schlüsselattribut AKURZ in AUFTRAG sondern nur transitiv abhängig über das Attribut PNR.

Gegen die 4NF wird mit den Attributen KUNDE.UMSATZ, AUFTRAG.ANETTO und LEISTUNG.LNETTO verstoßen, weil es sich bei diesen Werten um aus anderen Attributwerten berechenbare Werte handelt. Deshalb werden diese Attribute wieder aus den jeweiligen Tabellen entfernt.
In unserem Modell wird der STATUS in AUFTRAG in eine eigene Tabelle AUFSTAT ausgelagert. Diese Zusatztabelle hat den Vorteil, daß man nicht von den Statusbezeichnungen abhängig ist. Sie können problemlos geändert werden, ohne daß die Tabelle AUFTRAG davon betroffen ist. Damit wird also ein Beitrag zur 3NF geleistet. Das Er-

gebnis des Entwurfs ist die im Anhang dargestellte Auftrags-Datenbank.

3 Datenbanksprache SQL

In diesem Kapitel wird die Datenbanksprache SQL (Structured Query Language) vorgestellt. Zunächst werden die Kommandos zur Datendefinition, zum Anlegen und Löschen von Tabellen und zum Ändern der Tabellenstruktur vorgestellt. Danach folgt der große Block der Kommandos zur Datenmanipulation, wozu auch die Datenbank-Abfrage gehört. Nach den einfachen Abfragen und Kommandos zum Einfügen, Ändern und Löschen von Daten werden komplexe Abfragen mit Zugriff auf mehrere Tabellen, mit Datengruppierungen und mit Unterabfragen erörtert. Das Anlegen von Datensichten (Views) wird ebenfalls bearbeitet. Zum Abschluß werden die Kommandos der Datenkontrolle (Zugriffsrechte und Transaktionen) sowie die Verfahren zur Nutzung von SQL in Programmiersprachen behandelt.

3.1 Datendefinition

Nach dem Entwurf steht als nächster Schritt das Anlegen der Datenbank und der Tabellen an. Das Anlegen der Datenbank erfolgt entweder bei der Installation des Datenbankverwaltungssystems oder ist interaktiv über Menüfunktionen möglich. In SQL stehen Anweisungen zum Anlegen, Ändern oder Löschen einer Tabelle (genauer: einer Tabellenstruktur) oder einer View zur Verfügung.
In diesem und den folgenden Kapiteln wird die jeweilige ANSI-SQL-Notation, in einigen Fällen aber auch die Oracle-Notation angegeben. Der Sprachumfang von SQL ist bei den DBVSen speziell im Bereich der Datentypen nicht immer einheitlich.

3.1.1 Anlegen einer Tabelle

Mit folgendem Kommando CREATE TABLE wird eine Tabelle angelegt:

```
CREATE TABLE  Tabellenname
(Spaltenname1     Datentyp [NOT NULL] [DEFAULT deftyp],
Spaltenname2      Datentyp,
 ,,,)
```

Es sind folgende *Datentypen* möglich ([Date89]):

```
CHARACTER         [(length)]
NUMERIC           [(precision [, scale])]
DECIMAL           [(precision [, scale])]
```

INTEGER
SMALLINT
FLOAT [(precision)]
DOUBLE PRECISION
REAL

CHARACTER, DECIMAL und INTEGER können mit CHAR, DEC und INT abgekürzt werden. Precision gibt die Gesamtanzahl Stellen, scale die Anzahl Nachkommastellen an. Die Vielfalt der Datentypen existiert deshalb, um den verschiedenen Programmiersprachen gerecht zu werden. Die Details zu precision und length sind i. d. R. implementationsabhängig.
Mit *NOT NULL* wird festgelegt, daß das entsprechende Attribut für alle Zeilen der Tabelle einen Wert haben muß. NULL ist nicht dasselbe wie die Zahl 0, sondern gilt für alle Datentypen und bedeutet soviel wie nichts oder leer, NOT NULL also = „etwas“.
Falls NOT NULL nicht angegeben ist, kann mit *DEFAULT* ein Standardwert für eine leere Spalte angegeben werden. Das heißt, daß beim Einfügen einer Zeile, die keinen Wert in dieser Spalte aufweist, dieser Standardwert automatisch eingetragen wird.
Mit dem Anlegen einer Tabelle können einige *Integritätsregeln* festgelegt werden.
Mit *UNIQUE* (Spaltenliste) wird angegeben, daß die betreffenden Spalten in der Tabelle eindeutige Werte haben. Eine Einfügeoperation einer Zeile mit einem Spaltenwert, der bereits in der Tabelle vorhanden ist, wird dann nicht zugelassen. Dies wird auch mit candidate key definition bezeichnet.
Mit *PRIMARY KEY* (Spaltenliste) wird der *Primärschlüssel* der Tabelle festgelegt. Der Primärschlüssel ist automatisch vom Typ UNIQUE.

Mit

FOREIGN KEY (Spaltenliste)
REFERENCES Basistabelle [(Spaltenliste)]

werden *Fremdschlüssel* definiert. Beim Einfügen einer Zeile in eine Tabelle wird überprüft, ob der Spaltenwert einen korrespondierenden Spaltenwert in der Referenz-Tabelle hat (*referential integrity*). Die angegebenen Spalten in der Referenz-Tabelle müssen dort entweder Primärschlüssel oder vom Typ UNIQUE sein. Falls in der Zeile REFERENCES keine Spalten angegeben werden, wird davon ausgegangen, daß die Spaltennamen in beiden Tabellen übereinstimmen.
Als Referenztabelle kann auch dieselbe Tabelle angegeben werden.

Beispiel:

Eine Tabelle PERSONAL mit

PNR, Name, Vorname, Vorgesetzter

enthält in der Spalte Vorgesetzter die PNR (Personalnummer) des betreffenden Vorgesetzten der Person; der Vorgesetzte ist selbst wieder eine Person.

Mit *CHECK* (search-condition) wird festgelegt, welche Werte bzw. welchen Wertebereich die betreffende Spalte haben darf. Es können u. a. Bereichsangaben (PLZ BETWEEN 10000 AND 89999), konkrete Werte (ANREDE IN ('Herr', 'Frau')) oder Werterelationen (KuNr <= 4711) in der CHECK-Klausel enthalten sein. Diese können dann wieder mit AND und OR verknüpft werden. Es sind also alle Varianten der search-condition = *Auswahlbedingung* (wird später erklärt) möglich.
Die vorgenannten Klauseln können entweder direkt an die Spaltendefinition angefügt werden, sofern sie sich nur auf diese Spalte beziehen, oder als ergänzende Klauseln am Ende der Tabellendefinition notiert werden.

Beispiele für CREATE TABLE in der ORACLE-Notation:

```
CREATE TABLE KUNDE
(KuNr       INT             NOT NULL  PRIMARY KEY,
Name        CHAR    (20)    NOT NULL,
Vorname     CHAR    (15),
PLZ         INT             CHECK (PLZ BETWEEN 10000
                            AND 89999),
Ort         CHAR    (30),
Str         CHAR    (30),
KRED_LIM    DEC     (7,2)   DEFAULT 10000 NOT NULL,

CHECK (KRED_LIM BETWEEN 0 AND 50000),
CHECK (ORT IN ('Iserlohn', 'Hagen', 'Lüdenscheid')))

CREATE TABLE ARTIKEL
(ArtNr      INT             NOT NULL  PRIMARY KEY,
ArtBez      CHAR    (20)    NOT NULL,
EkPreis     DEC     (8,2)   NOT NULL,
VkPreis     DEC     (8,2),
LagNr       INT             REFERENCES LAGER (LagNr),
MinBest     NUM     (6)     CHECK (MinBest >= 0))

CREATE TABLE LAGER
(LagNr      INT             NOT NULL  PRIMARY KEY,
...)

CREATE TABLE LAGERUNG
(ArtNr      INT             NOT NULL,
LagNr       INT             NOT NULL,
LagMen      NUM     (6)     NOT NULL,
```

```
PRIMARY KEY (ArtNr, LagNr),
FOREIGN KEY (ArtNr) REFERENCES ARTIKEL(ART_NR),
FOREIGN KEY (LagNr) REFERENCES LAGER,
CHECK (LAGERUNG.LagMen >= ARTIKEL.MinBest AND LA-
GERUNG.LagMen <= 1000))
```

3.1.2 Ändern einer Tabellenstruktur

Um eine existierende Tabellenstruktur zu ändern, z. B. zu erweitern, ist im SQL-Standard (noch) keine Möglichkeit vorgesehen. In vielen DBVSen gibt jedoch adäquate Konstrukte wie ALTER TABLE.

3.1.3 Laden einer Tabelle

Nachdem die Tabellen angelegt und mit einigen Testdaten überprüft wurden, können größere Datenmengen aus vorhandenen Datenquellen geladen werden. Zum Laden solcher Massendaten bieten die Datenbanksysteme Tools an, z. B. LOAD, IMPORT usw. Dabei können die Daten aus ASCII-, Text- oder anderen Datei-Formaten geladen werden.

3.2 Einfache Abfragen

Abfragen werden in SQL mit dem SELECT-Kommando formuliert. Zunächst soll die allgemeine Form von SELECT [Date89] vorgestellt werden:

SELECT	Angabe der gewünschten Spalten
INTO	Angabe der Zielvarablen (*)
FROM	Angabe der Tabellenquellen
WHERE	Auswahlbedingungen
GROUP BY	Gruppenbildung von gleichen Werten in einer Spalte
HAVING	Auswahlbedingung bestimmter Gruppen
UNION	Vereinigung von Ergebnismengen
ORDER BY	Sortierreihenfolge des Ergebnisses

(*) nur bei Programmiersprachen-Einbettung

Im folgenden wird der SELECT-Befehl detailliert erläutert.
Wir starten mit einfachen Abfragen, bei denen Daten in nur einer Tabelle gesucht werden.

3.2.1 SELECT mit Projektion

Für eine Projektion auf eine Tabelle lautet der SELECT-Befehl:

```
SELECT      [ALL | DISTINCT] Spaltenname1,,,
FROM        Tabellenname
ORDER BY    Spaltenname1,,,[ASC | DESC]
```

wobei die Parameter folgende Bedeutung haben:

ALL	Es sollen alle Werte, auch mehrfach vorkommende, angezeigt werden (Default).
DISTINCT	Es sollen nur verschiedene Werte bzgl. dieser Spalte angezeigt werden.
ASC	Sortierreihenfolge aufsteigend (Default)
DESC	Sortierreihenfolge absteigend

Anstelle der Spaltennamen, deren Reihenfolge nicht mit derjenigen der Tabelle übereinstimmen muß, kann auch ein * für die Auswahl aller Spalten verwendet werden.

Beispiele für Projektionen:

Selektiere / Suche alle Kunden:

```
SELECT      *
FROM        KUNDE;
```

Suche Kundenkürzel und Firmenname aller Kunden, sortiert nach Firmenname:

```
SELECT      KKURZ, FIRMA
FROM        KUNDE
ORDER BY    FIRMA;
```

Suche die unterschiedlichen Postleitzahlen aller Kunden:

```
SELECT      DISTINCT PLZ
FROM        KUNDE;
```

Mit dem Schlüsselwort DISTINCT werden Werte, die mehrfach auftreten, nur einfach gezählt. Ohne DISTINCT würden alle Postleitzahlen angezeigt, unabhängig davon, wie oft sie auftreten.
Suche Ort und Firmenname der Kunden, sortiert nach ORT; falls ORT doppelt, dann zusätzlich sortiert nach Firmenname:

```
SELECT      ORT, FIRMA
FROM        KUNDE
ORDER BY    ORT, FIRMA;
```

3.2.2 SELECT mit Selektion

Eine Selektion ist eine Auswahl von Zeilen aus einer Tabelle nach bestimmten Kriterien; sie wird durch die WHERE-Klausel geregelt:

WHERE Auswahlbedingung

WHERE beinhaltet eine Auswahlbedingung. Aus den Tabellenzeilen, die die in der WHERE-Klausel angegebene Bedingung erfüllen, wird gemäß der angegebenen Spaltenliste eine Ergebnismenge gebildet. Beim interaktiven SQL am Bildschirm wird diese Ergebnismenge angezeigt.
Innerhalb der WHERE-Bedingung können folgende *Vergleichsoperatoren* zum Vergleich von Werten (desselben oder eines kompatiblen Datentyps) benutzt werden:

= (gleich), < (kleiner als), > (größer als), <= (kleiner oder =), >= (größer oder =), <> (ungleich)

- alphanumerische Konstanten sind mit " " oder ' ' anzugeben
- Groß- und Kleinbuchstaben sind unterschiedlich
- ANSI- und ASCII-Vergleich liefern evtl. unterschiedliche Werte

Mehrere Vergleichsoperationen können mittels *logischer Operatoren* miteinander verknüpft werden:

Operator	Beschreibung
NOT	Verneinung der Bedingung
AND	Logisches UND
OR	Logisches ODER
XOR	Logische Antivalenz
EQV	Logische Äquivalenz (zeichenweiser Bitvergleich)
IMP	Logische Implikation (Wenn-Dann-Beziehung)

In einem Vergleich können mathematische Ausdrücke mit *, /, +, -, \ (Div), ^ (Potenz) und Mod verwendet werden.
Die Prioritätsreihenfolge der Operatoren ist genau definiert und der Online-Hilfe des DBVS zu entnehmen. Ggfs. sind Klammern zu benutzen.
Neben den o. g. Operatoren ist der &-Operator noch von Bedeutung. Er wird benutzt, um zwei Zeichenfolgen miteinander zu verbinden, z. B. "Hans" & " " & "Hugo" ergibt "Hans Hugo".

Beispiele für Selektionen:

Hinweis: Die Angaben in der Klammer, z. B. (sqlv10), sind die Namen der Abfragen, wie sie in der MS Access-Datenbank auftrag.mdb gespeichert sind.

Suche alle Kunden mit Kundenkürzel > 105:

```
(sqlv10)
SELECT      KKURZ, FIRMA, LKZ, ORT
FROM        KUNDE
WHERE       KKURZ > 105;
```

Suche alle Kunden mit Kundenkürzel > 105 und Länderkennz. = „D“:

```
(sqlv11)
SELECT      KKURZ, FIRMA, LKZ, ORT
FROM        KUNDE
WHERE       KKURZ > 105 AND LKZ = "D";
```

Suche die Aufträge mit KKURZ = 106 und Auftragsstatus-Nr = 3 oder 4 (erledigt oder Ablage):

```
(sql110b)
SELECT      KKURZ, AKURZ, ATEXT, AANZ, ASATZ, ASTNR
FROM        AUFTRAG
WHERE       KKURZ = 106 AND (ASTNR = 3 OR ASTNR = 4);
```

Suche die Aufträge mit KKURZ = 106 und (Auftragsstatus-Nr = 3 oder 4):

```
(sql111a)
SELECT      KKURZ, AKURZ, ATEXT, AANZ, ASATZ, ASTNR
FROM        AUFTRAG
WHERE       KKURZ = 106 AND (ASTNR = 3 OR ASTNR = 4);
```

Suche die Aufträge mit Fertigsoll bis zum 31.12.97:

```
(sql107)
SELECT      KKURZ, AKURZ, ATEXT, AANZ, ASATZ, FER-
            TIGSOLL
FROM        AUFTRAG
WHERE       FERTIGSOLL <= #12/31/97#;
```

Hinweis: In SQL-Anweisungen in Access Basic müssen US-/englische Datumsformate verwendet werden. Im QBE-Entwurfsbereich von Access können jedoch die internationalen Datumsformate verwendet werden, z. B. <= #31.12.1997# (wobei # von Access automatisch ergänzt wird).

3.2.3 SELECT mit BETWEEN, IN und LIKE

SQL bietet einige Möglichkeiten, um Vergleiche von Werten einfach zu formulieren.

Mit dem Operator BETWEEN Wert1 AND Wert2

kann folgende Bedingung

```
WHERE     FERTIGIST >= #01/01/97# AND
          FERTIGIST <= #12/31/97#;
```

auch wie folgt spezifiziert werden:

```
WHERE     FERTIGIST BETWEEN #01/01/97# AND
          #12/31/97#;
```

Mit dem Operator IN (Wert1, Wert2,,,) kann eine Liste von Werten zum Vergleich benutzt werden (entspricht einem mehrfachen OR). Mit dem Operator LIKE für alphanumerische Zeichenketten kann nach Zeichenmustern gesucht werden. Dabei können Platzhalterzeichen verwendet werden.

Zeichen	Steht für
? bzw. _	Ein beliebiges einzelnes Zeichen
* bzw. %	Null oder mehrere beliebige Zeichen
#	Eine beliebige einzelne Ziffer (0-9)
[ZeichListe]	Ein beliebiges Zeichen aus der ZeichListe, z. B. "[AD]*": alle Werte, die mit A oder D beginnen. "[A-D]*": alle Werte, die mit A, B, C oder D beginnen.
[!ZeichListe]	Ein beliebiges Zeichen außerhalb von ZeichListe

Um herauszufinden, ob in einer Spalte kein Wert eingetragen ist, muß

WHERE Spaltenname IS NULL formuliert werden.

Die Verneinung von BETWEEN, LIKE und IN wird durch ein vorangestelltes NOT realisiert (z. B. Spaltenname NOT LIKE Wert); für IS NULL lautet die Verneinung dagegen IS NOT NULL

Beispiele:

Suche die Aufträge mit Fertig-Ist-Datum zwischen 1.1.97 und 31.12.97

```
(sql113)
SELECT    KKURZ, AKURZ, ATEXT, AANZ, ASATZ, FER-
          TIGIST
FROM      AUFTRAG
WHERE     FERTIGIST BETWEEN #01/01/97# AND
          #12/31/97#;
```

Suche die Aufträge der Kunden 106 und 110

(sql114)

```
SELECT      KKURZ, AKURZ, ATEXT, AANZ, ASATZ, FER-
            TIGIST
FROM        AUFTRAG
WHERE       KKURZ IN (106,110)
ORDER BY    KKURZ;
```

Suche die Aufträge, die im Auftragstext die Zeichenkette "DIAG" enthalten

(sql115)

```
SELECT      KKURZ, AKURZ, ATEXT, AANZ, ASATZ, FER-
            TIGIST
FROM        AUFTRAG
WHERE       ATEXT LIKE "*DIAG*";
```

Suche die Aufträge, die noch nicht fertig sind

(sql116c)

```
SELECT      KKURZ, AKURZ, ATEXT, AANZ, ASATZ, FER-
            TIGIST
FROM        AUFTRAG
WHERE       FERTIGIST IS NULL;
```

3.2.4 SELECT mit Funktionen, Ausdrücken, virtuellen Spalten

Bei der Formulierung eines Queries können statistische Werte abgefragt werden. Dazu stehen folgende *Funktionen* zur Verfügung, die sich jeweils auf die Ergebnismenge der Abfrage (bzw. der Gruppe, siehe unten) beziehen. Im Fall einer Abfrage darf dann jedoch nur **eine** Spalte selektiert werden.

COUNT	ermittelt die Anzahl der Treffer inkl. der NULL-Werte
SUM	ermittelt die Summe der numerischen Werte
AVG	ermittelt den Durchschnitt der numerischen Werte
MAX	ermittelt den maximalen Wert aller Werte
MIN	ermittelt den minimalen Wert aller Werte
StdAbw	ermittelt die Standard-Abweichung in der Stichprobe
StdAbwG	dito in der Grundmenge
Varianz	ermittelt die Varianz in der Stichprobe
VarianzG	dito in der Grundmenge

In Ergebnisspalten können sowohl komplexe *Ausdrücke* zum Vergleich herangezogen werden, z. B.

```
WHERE       MWSATZ = 15/100      (anstelle MWSATZ = 0.15)
```

als auch Ergebnisse durch Ausdrücke berechnet werden, in denen unterschiedliche Attribute als Elemente des Ausdrucks enthalten sein können, z. B.

```
WHERE     AANZ*ASATZ <= 5000
```

AANZ*ASATZ kann als Auftragswert interpretiert werden.

Beispiele:

Wieviele Kunden sind in der Datenbank?

```
SELECT    COUNT (*)
FROM      KUNDE;
```

Welches ist das größte Kundenkürzel?

```
SELECT    MAX (KKURZ)
FROM      KUNDE;
```

Bei vielen Datenbanksystemen können bei der Ausgabe eines interaktiven SQL-Kommandos die Spaltenüberschriften mit AS geändert werden (*Aliasnamen* für Spalten),
z. B.

Ermittle Auftragskürzel, Kundenkürzel und Auftragsdatum der Aufträge:

```
SELECT    AKURZ AS Auftragskürzel, KKURZ AS Kundenkürzel, ADATUM AS Auftragsdatum
FROM      AUFTRAG;
```

Es können auch zusätzliche Spalten (*Virtuelle Spalten*) in der Ergebnismenge auftreten, die nicht in der Tabelle enthalten sind; sie werden durch Berechnung o. ä. gebildet, z. B.

Ermittle die Auftragswerte der Aufträge:

```
SELECT    AKURZ, AANZ*ASATZ AS Auftragswert
FROM      AUFTRAG;
```

Ermittle die Mehrwertsteuerbeträge der Aufträge:

```
SELECT    AKURZ, KKURZ, AANZ*ASATZ*MWSATZ AS Mehrwertsteuer
FROM      AUFTRAG;
```

Zwei oder mehrere Spalten können zu einer (virtuellen) Ergebnisspalte zusammengefaßt werden:

```
SELECT      PLZ & " " & ORT AS Wohnort
FROM        KUNDE;
```

Während in Access das AS gefordert wird, ist im Standard-SQL kein AS mehr nötig.

Beispiele:

Die gezeigten Tabellen sind die jeweiligen Ergebnismengen, die nach dem Ausführen der Abfrage auf dem Bildschirm angezeigt werden.

(sql129)

```
SELECT      AKURZ, FERTIGIST, FERTIGSOLL,
            FERTIGIST - FERTIGSOLL AS Verspätung (in Ta-
            gen)
FROM        AUFTRAG;
```

AKURZ	FERTIGIST	FERTIGSOLL	Verspätung
109	01.05.96	01.05.96	0
111		30.11.98	
112	13.04.97	28.02.97	44
113	04.02.98	15.12.97	51
114		12.07.98	
115	02.09.97	30.07.97	34
116			

(sql122b)

```
SELECT      VORNAME, NACHNAME, PNR AS Personalnum-
            mer
FROM        PERSON
WHERE       PNR < 10
ORDER BY    NACHNAME;
```

VORNAME	NACHNAME	Personalnummer
Karl-Heinz	Adam	3
Werner	Assmann	7
Karl	Auffahrt	2
Ewald	Beck	4
Klaus	Bettag	1
	Bomhoff	9
	Henning	8
	Kummer	6
Adolf	Schultz	5

(sql122a)

```
SELECT      KKURZ, AKURZ, ATEXT, AANZ*ASATZ AS
            Netto, AANZ*ASATZ*(1+MWSATZ) AS Brutto
FROM        AUFTRAG
ORDER BY    AANZ*ASATZ;
```

KKURZ	AKURZ	ATEXT	Netto	Brutto
110	116	Einführung	800	800
109	109	Serienbrief-Einweisung	1.500	1.740
110	113	Kunden-Verwaltung	5.600	5.600
106	115	Datenbank-Entwurf DIAG	6.500	7.540
101	114	ORACLE-Rezension	9.000	9.630
110	112	Offertenverwaltung	9.600	9.600
100	111	Mitarbeiterschulung	18.000	20.880

Mit Format (AANZ*ASATZ,"#.###") AS Netto wird eine zusätzliche Formatangabe zur Darstellung des Tausenderpunktes benutzt.

(sql130)

```
SELECT      AKURZ, FERTIGIST, FERTIGSOLL, FERTIGIST-
            FERTIGSOLL AS [Sollzeit der Fertigstellung in Ta-
            gen], Date() AS heute, Date()-FERTIGIST AS [Tage
            seit Fertigstellung]
FROM        AUFTRAG
ORDER BY    Date()-FERTIGIST;
```

AKURZ	Fert.IST	Fert.SOLL	Sollzeit Fer-tigstell.	heute	Tage seit Fertig-stellung
116				17.09.97	
114		12.07.98		17.09.97	
111		30.11.98		17.09.97	
113	04.02.98	15.12.97	51	17.09.97	-140
115	02.09.97	30.07.97	34	17.09.97	15
112	13.04.97	28.02.97	44	17.09.97	157
109	01.05.96	01.05.96	0	17.09.97	504

(sql120)

```
SELECT      VORNAME & " " & NACHNAME AS Name, PNR
FROM        PERSON
WHERE       PNR < 10
ORDER BY    NACHNAME;
```

Name	PNR
Karl-Heinz Adam	3
Werner Assmann	7
Karl Auffahrt	2
Ewald Beck	4
Klaus Bettag	1
Bomhoff	9
Henning	8
Kummer	6
Adolf Schultz	5

3.3 Gruppierungen

Die Mächtigkeit von SQL läßt auch komplexe Abfragen zu. Hierunter fallen Abfragen mit Gruppierungen und statistischen Funktionen, Abfragen, die sich auf mehr als eine Tabelle beziehen, sowie Abfragen mit geschachtelten Unterabfragen. In diesem Kapitel werden auch SQL-Kommandos für die Änderung von Daten behandelt.
Mit *GROUP BY* wird eine Gruppenbildung bei SELECT durchgeführt. Dabei werden jeweils Teilmengen von Ergebniszeilen zu einer Gruppe zusammengefaßt, wenn sie in einer ausgewählten Spalte (der Gruppenspalte) gleiche Attributwerte besitzen.
Gruppenbildungen werden i. a. vorgenommen, um Werte wie Summe, Maximum o.ä. für die ganze Gruppe zu berechnen und dann in einer Zeile für die Gruppe auszugeben. Betrachten wir dazu ein Beispiel:

Ermittle die Anzahl der Aufträge mit verschiedenen Mehrwertsteuersätzen. Die Menge der Aufträge zu einem MWST-Satz bildet dabei jeweils eine Gruppe:

(sql136)
```
SELECT      MWSATZ, COUNT(*)  As Häufigkeit
FROM        AUFTRAG
GROUP BY    MWSATZ;
```

MWSATZ	Häufigkeit
0	3
0,07	1
0,16	3

Die Gruppenspalte MWSATZ muß als Suchspalte hinter SELECT angegeben werden. Es wird automatisch nach der Gruppenspalte sortiert. Als Ergebnisspalten (hinter SELECT) sind nur Gruppenspalten, arithmetische Funktionen und virtuelle Spalten zugelassen.
Man kann auch mehrere Spalten als Gruppenspalten definieren:

(sql137b)
```
SELECT      MWSATZ, LSATZ, Count(*) AS Anzahl
FROM        LEISTUNG
GROUP BY    MWSATZ, LSATZ;
```

MWSATZ	LSATZ	Anzahl
0	20	1
0	1.200	3
0	1.400	1
0,07	125	1
0,07	300	2
0,16	10	1
0,16	75	1
0,16	125	4
0,16	1.500	2

Durch die zusätzliche Gruppenspalte LSATZ wird das Ergebnis erweitert; es werden auch mehr Zeilen angezeigt. Das liegt daran, daß jetzt jeweils eine Gruppe bezüglich der Kombination MWSATZ und LSATZ gebildet wird.

Wichtig:

> Hinter GROUP BY müssen mindestens dieselben Tabellenspalten wie hinter SELECT aufgeführt sein!

Sortiert wird zunächst nach der 1. Gruppenspalte, dann nach der 2. usw. Mit *HAVING* wird eine Selektion hinsichtlich einer Gruppeneigenschaft vorgenommen. Dabei sind nur arithmetische Funktionen zugelassen.

Als Beispiel wollen wir wissen, wieviele Kontaktpersonen pro Kunde existieren; dabei interessieren uns nur Ergebnisse mit Anzahl > 2:

```
(sqlv12)
SELECT      KKURZ, COUNT(*) As Anzahl
FROM        KONTAKT
GROUP BY    KKURZ
HAVING      COUNT(*) > 2;
```

KKURZ	Anzahl
100	3
101	4
107	3
109	3

Welche kumulierten, geplanten Umsätze (Summe aus AANZ*ASATZ) sind pro Kunde vorhanden? Hier wird die Gruppe durch den jeweiligen Kunden (KKURZ in AUFTRAG) gebildet; die Anzahl der Aufträge des jeweiligen Kunden soll auch angezeigt werden:

```
(sql134)
SELECT      KKURZ, SUM(AANZ*ASATZ) As Planumsatz,
            COUNT(AKURZ) As Auftragsanzahl
FROM        AUFTRAG
GROUP BY    KKURZ;
```

KKURZ	Planumsatz	Auftragsanzahl
100	18.000	1
101	9.000	1
106	6.500	1
109	1.500	1
110	16.000	3

Von welchem Kunden liegt mehr als 1 Auftrag vor?

(sql139)-Variante
```
SELECT      KKURZ, COUNT(*) As Anzahl Aufträge
FROM        AUFTRAG
GROUP BY    KKURZ
HAVING      COUNT(*) > 1;
```

KKURZ	Anzahl Aufträge
110	3

Es sollen die pro MWSt-Satz kumulierten Leistungsergebnisse (Summe LANZ*LSATZ aus LEISTUNG) derjenigen Leistungen ermittelt werden, deren MWSATZ ungleich 0 ist. Da in der HAVING-Klausel immer eine Funktion (hier MIN) angegeben werden muß, kann die Abfrage wie folgt formuliert werden (mit WHERE MWSATZ > 0 würde dasselbe Ergebnis erzielt werden):

(sql140b)
```
SELECT      MWSATZ, SUM(LANZ*LSATZ) As Nettoumsatz,
            COUNT(MWSATZ) As Anzahl
FROM        LEISTUNG
GROUP BY    MWSATZ
HAVING      MIN(MWSATZ) > 0;
```

MWSATZ	Nettoumsatz	Anzahl
0,07	9.125	3
0,16	25.175	8

3.4 Daten einfügen, ändern, löschen

Um einzelne Datensätze mittels SQL in eine Tabelle einzufügen, wird folgendes Kommando verwendet:

```
INSERT INTO  Tabellenname [(Spaltenname1,,,)]
Quelle;
```

Quelle kann dabei eine Liste von Werten sein, die mit *VALUES* eingeleitet wird, oder es können Werte aus anderen Tabellen sein, die über ein SELECT aus diesen Tabellen ermittelt werden.

Es gibt folgende Restriktionen bei INSERT:

R1 Falls in der Quelle ein SELECT auftritt, darf die dort im FROM-Teil genannte Tabelle nicht mit der Zieltabelle des INSERT übereinstimmen.

R2 Falls in VALUES Parameter (bei der Programmiersprachenein-bettung, siehe späteres Kapitel) benutzt werden, dürfen diese nicht in Ausdrücken auftreten.

Mit dem folgenden SQL-Kommando werden Tabellenzeilen gelöscht:

```
DELETE
FROM        Tabellenname
[WHERE      Suchbedingung];
```

Falls die WHERE-Klausel nicht angegeben ist, werden sämtliche Zeilen in der Tabelle gelöscht. Es gilt sinngemäß die Restriktion R1 des INSERT-Kommandos.

Mit dem SQL-Kommando

```
UPDATE      Tabellenname
SET         Spaltenname = Feldwert,,,
[WHERE      Suchbedingung];
```

können Feldwerte in einer Tabelle geändert werden. Es gilt sinngemäß die Restriktion R1 des INSERT-Kommandos. Im Standard-SQL kann das UPDATE-Kommando wie bei INSERT ein SELECT als Quelle haben.

Beispiele für INSERT, DELETE und UPDATE:

```
(sql92)
INSERT INTO KUNDE ( KKURZ, FIRMA, ZUSATZ, STRASSE,
LKZ, PLZ, ORT, KTELEFON, KINFO)
VALUES (113, "Stadtwerke", "Fuhrpark", "Kantstr. 4", "D", "28283",
"Bremen", "0421-667788", "Verwaltung");
```

Da hier alle Spalten betroffen sind, kann man auch einfach schreiben:

```
INSERT INTO KUNDE
VALUES (113, "Stadtwerke", "Fuhrpark", "Kantstr. 4", "D", "28283",
"Bremen", "0421-667788", "Verwaltung");
```

Der Löschbefehl für die eingefügte Zeile lautet:

```
DELETE
FROM        KUNDE
WHERE       KKURZ = 113;
```

Falls eine weitere Tabelle AKTAUFTR für die aktuellen Aufträge (AUFSTAT = 2 = Vertrag) mit den relevanten Informationen angelegt wird, können aus der Tabelle AUFTRAG alle Aufträge mit AUFSTAT = 2 in diese Tabelle eingefügt werden (Syntax ist Access-spezifisch):

(sqlv13)

```
CREATE TABLE      AKTAUFTR
    (AKURZ        NUMERIC,
    KKURZ         NUMERIC,
    FERTIGSOLL    DATE,
    ATEXT         CHARACTER(25),
    ASTNR         NUMERIC);
```

(sqlv14)

```
INSERT INTO  AKTAUFTR
    SELECT   AKURZ, KKURZ, FERTIGSOLL, ATEXT, ASTNR
    FROM     AUFTRAG
    WHERE    ASTNR = 2;
```

ergibt in AKTAUFTR folgenden Inhalt:

AKURZ	KKURZ	F.SOLL	ATEXT	ASTNR
114	101	12.07.98	ORACLE-Rez.	2
116	110		Einführung	2

Bei den folgenden Änderungen soll der Leistungssatz (LSATZ) der noch nicht abgerechneten Leistungen (RNR Is Null) um 10% erhöht werden; dann sollen die Aufträge 113 und 115 in den Status Ablage (4) übergehen:

```
UPDATE      LEISTUNG
SET         LSATZ = LSATZ*1.10
WHERE       RNR Is Null;

UPDATE      AUFTRAG
SET         ASTNR = 4
WHERE       AKURZ IN (113, 115);
```

3.5 Daten in mehreren Tabellen suchen

Um Daten aus mehr als einer Tabelle mit nur einer Abfrage zu bekommen, kann man mit einem *Verbund* (Join), mit einer *Unterabfrage* (Subquery) oder mit einer *Vereinigung* (Union) von Ergebnismengen arbeiten.

3.5.1 SELECT mit Tabellenverbund (join)

Bei einem Verbund werden Spalten aus verschiedenen Tabellen in einer Ergebnismenge zusammengefaßt. Grundlage des Verbunds ist das kartesische Produkt der verwendeten Tabellen. Durch eine Verbundbedingung (Joinbedingung) wird eine Teilmenge des kartesischen Produkts gebildet und ausgegeben. Mit dieser Bedingung werden die Attributwerte einer bestimmten Spalte einer Tabelle mit denen einer

anderen Spalte einer zweiten Tabelle verglichen. Dabei handelt es sich i. a. um Primär- oder Fremdschlüsselspalten.

3.5.2 Gleichheitsverknüpfung (Equijoin)

An folgendem Beispiel soll die Wirkungsweise eines Verbunds erklärt werden. Es sollen alle Aufträge, die geplant sind oder für die es einen Vertrag gibt (also ASTNR = 1 oder 2), inkl. der Firmennamen der Kunden, angezeigt werden. Das Problem besteht darin, daß die Auftragsdaten in der Tabelle AUFTRAG, die Firmennamen jedoch in der Tabelle KUNDE stehen.

```
(sqlv15)
SELECT      AUFTRAG.AKURZ, AUFTRAG.KKURZ, KUN-
            DE.FIRMA, AUFTRAG.ASTNR
FROM        AUFTRAG, KUNDE
WHERE       AUFTRAG.KKURZ = KUNDE.KKURZ
AND         ASTNR IN (1, 2);
```

AKURZ	KKURZ	FIRMA	ASTNR
111	100	Versicherungspartner	1
114	101	Computer Courier	2
116	110	Bacher Elektronik AG	2

Da verschiedene Tabellen angesprochen werden, wird bei jedem Spaltennamen die betreffende Tabelle mittels eines vorangestellten, durch einen „Punkt“ getrennten Tabellennamen angegeben, z. B. AUFTRAG.AKURZ. Falls alle Spalten ausgegeben werden sollen, wird dies durch

Tabellenname**.***

notiert. Unbedingt notwendig ist der vorangestellte Tabellenname jedoch nur bei solchen Spalten, die in dem Tabellenverbund in mehr als einer Tabelle mit gleichem Namen vorkommen.
In der FROM-Klausel werden die beteiligten Tabellennamen angegeben.
Die Bedingung WHERE AUFTRAG.KKURZ = KUNDE.KKURZ ist die *Verbundbedingung*. Mit ihr wird die Verbindung zwischen den beiden Tabellen über das Attribut KKURZ hergestellt, das in AUFTRAG ein Fremdschlüssel und in KUNDE der Primärschlüssel ist. Durch das „=“ in der Verbundbedingung wird diese zu einer *Gleichheitsverknüpfung* (Equijoin**)**.

Die oben genannte Schreibweise gilt für SQL1 bzw. SQL89. In SQL2 bzw. SQL92 (auch in Access) wird die Verbundbedingung wie folgt formuliert:

FROM KUNDE INNER JOIN AUFTRAG ON KUNDE.KKURZ = AUFTRAG.KKURZ

Um kürzere SQL-Kommandos zu erzeugen, können *Aliasnamen* für die Tabellennamen verwendet werden. Der Aliasname wird in der FROM-Klausel angegeben:
FROM Tabellenname1 As Alias1, Tabellenname2 As Alias2
Er kann dann auch zur Qualifizierung der Attributnamen benutzt werden.

(sqlv16)
```
SELECT    A.AKURZ, A.KKURZ, K.FIRMA, A.ASTNR
FROM      AUFTRAG AS A, KUNDE AS K
WHERE     A.KKURZ = K.KKURZ
AND       A.ASTNR IN (1, 2);
```

AKURZ	KKURZ	FIRMA	ASTNR
111	100	Versicherungspartner	1
114	101	Computer Courier	2
116	110	Bacher Elektronik AG	2

Um nicht nur die Auftragsstatusnummer (ASTNR) anzuzeigen, sondern auch den Statustext, muß zusätzlich auf die Tabelle AUFSTAT zugegriffen werden. Damit haben wir einen Verbund mit 3 Tabellen:

(sqlv17)
```
SELECT    A.AKURZ, A.KKURZ, K.FIRMA, A.ASTNR,
          A1.ASTXT
FROM      AUFTRAG AS A, KUNDE AS K, AUFSTAT AS A1
WHERE     A.KKURZ = K.KKURZ
AND       A.ASTNR = A1.ASTNR
AND       A1.ASTNR IN (1, 2);
```

AKURZ	KKURZ	FIRMA	ASTNR	ASTXT
111	100	Versicherungspartner	1	geplant
114	101	Computer Courier	2	Vertrag
116	110	Bacher Elektronik AG	2	Vertrag

In der obigen Abfrage müssen 2 Verbundbedingungen spezifiziert werden: KKURZ für den Verbund zwischen den Tabellen AUFTRAG und KUNDE, und ASTNR für den Verbund zwischen AUFTRAG und AUSTAT. Die Ergebnismenge muß nicht alle Spaltennamen enthalten, die in den Verbundbedingungen enthalten sind, wie folgendes Beispiel zeigt:

Welcher Kunde hat einen Vertrag? Ausgabe des Firmennamens und des Auftragsstatustextes.

(sqlv18)

```
SELECT    K.FIRMA, A1.ASTXT AS Auftragsstatus
FROM      AUFTRAG AS A, KUNDE AS K, AUFSTAT AS A1
WHERE     A.KKURZ = K.KKURZ
AND       A.ASTNR = A1.ASTNR
AND       A1.ASTXT = 'Vertrag';
```

FIRMA	Auftragsstatus
Computer Courier	Vertrag
Bacher Elektronik	Vertrag

3.5.3 Reflexionsverknüpfung (Selfjoin)

Bei einem Join kann auch eine Tabelle mit sich selbst verknüpft werden (*Reflexionsverknüpfung*).
Gegeben sei folgende Tabelle ANG, die die Angestellten einer Abteilung umfassen soll:

AngNr	Name	Beruf	AbtNr	VorgesNr	Gehalt
374	KÄMP	SEKR	3	401	3.800
112	MANG	PROGR	1	205	4.300
400	KLEF	SEKR	3	198	3.800
205	WIND	ORGAN	1		5.100
307	MILL	PROGR	2	205	5.400
117	SEEL	ING	3	401	4.100
198	FELD	KAUFM	3	401	4.100
401	OTTO	KAUFM	3	205	4.700

Die VorgesNr ist die AngNr derjenigen Person, die Vorgesetzter der gerade betrachteten Person ist. Wir wollen nun wissen, welcher Angestellte mehr als sein Vorgesetzter verdient. Wir bilden dazu einen Verbund der Tabelle ANG mit sich selbst. Dabei stellen wir uns vor, die Tabelle ANG läge 2x identisch vor. Wir bilden nun einen Verbund zwischen den beiden Kopien von ANG.

(sqlv19)

```
SELECT    ANGESTELLTE.Name AS Angestellter, ANGE-
          STELLTE.Gehalt AS A-Gehalt, VORGES.Name AS
          Vorgesetzter, VORGES.Gehalt AS V-Gehalt
FROM      ANGESTELLTE, ANGESTELLTE AS VORGES
WHERE     ANGESTELLTE.VorgesNr = VORGES.AngNr
AND       ANGESTELLTE.Gehalt > VORGES.Gehalt;
```

Als Ergebnis bekommen wir:

Angestellter	A-Gehalt	Vorgesetzter	V-Gehalt
MILL	5.400	WIND	5.100

3.5.4 Inklusionsverknüpfung (Outerjoin)

Neben der bisher betrachteten Gleichheitsverknüpfung (Innerjoin) gibt es noch die *Inklusionsverknüpfung* (Outerjoin). Eine Gleichheitsverknüpfung erfaßt nur Zeilen, bei denen mindestens eine Zeile mit gleichem Attributwert bei der Gleichsetzung der Verbundspalten gefunden wird. Die Inklusionsverknüpfung behandelt auch Tabellenwerte, für die kein Vergleichswert (in der anderen Tabelle) existiert, z. B. Kunden, die keinen Auftrag haben.
Man unterscheidet zwischen einer linksseitigen (left) und einer rechtsseitigen (right) Inklusionsverknüpfung.
Beispiel für eine Inklusionsverknüpfung (aus gkurs.mdb):

```
SELECT    nachname AS Sachbearbeiter, präsent AS Präsent
FROM      Sachbearbeiter LEFT JOIN SA-Präsente ON Sachbe-
          arbeiter.nr = SA-Präsente.nr;
```

Es werden alle Sachbearbeiter angezeigt, auch diejenigen, die keine Präsente bekommen haben.

3.5.5 SELECT mit Unterabfrage

Bei einem SELECT mit *Unterabfrage* (Subquery) wird die Abfrage an eine Haupttabelle gestellt, wobei die Vergleichsbedingung eine Nebentabelle zum Vergleich heranzieht. Haupt- und Nebentabelle können identisch sein. Aus der Nebentabelle können ein oder mehrere Vergleichswerte entnommen werden.

3.5.6 SELECT mit Subquery mit einem Vergleichswert

Die Grundstruktur eines SELECT mit Subquery mit einem Vergleichswert sieht wie folgt aus:

```
SELECT    Spalte1,,,
FROM      Tabelle1
WHERE     Spaltex =
          (SELECT    Spaltex
          FROM       Tabelle2
          WHERE      );
```

Dabei wird zunächst die Unterabfrage abgearbeitet. Mit dem daraus ermittelten Wert (nur 1 Wert erlaubt!) wird dann die Hauptabfrage bearbeitet; also Abarbeitung von unten nach oben.

Beispiel: Suche alle Firmen, die denselben Firmensitz wie Kunde 101 haben.

```
(sql164)
SELECT    FIRMA, ORT
```

```
FROM        KUNDE
WHERE       ORT =
            (SELECT     ORT
            FROM        KUNDE
            WHERE       KKURZ = 101);
```

FIRMA	ORT
Versicherungspartner	Bremen
Computer Courier	Bremen
Deutsche Unterhaltung	Bremen
JUFI GmbH	Bremen

Es gelten folgende Einschränkungen für solche SELECTs:

- in der Unterabfrage darf nur eine Spalte im SELECT angegeben werden,
- die Selektion in der Unterabfrage darf nur einen Wert liefern; (für mehr als 1 Wert siehe nächstes Kapitel)
- DISTINCT ist in der Haupt- und Nebentabelle erlaubt
- ORDER BY ist in der Nebentabelle nicht erlaubt

Die Unterabfrage kann selbst wieder eine Unterabfrage besitzen; dazu weitere Beispiele:

Welche Leistung ist zuletzt abgeliefert worden?

```
(sql166a)
SELECT      LNR, LTEXT
FROM        LEISTUNG
WHERE       LDATUM =
            (SELECT     MAX(LDATUM)
            FROM        LEISTUNG);
```

LNR	LTEXT
16	Rezension SQL*FORMS

Welcher Kunde hat Aufträge mit einem Auftragswert, der über dem Durchschnitt aller Aufträge liegt?

```
(sql176)
SELECT      KKURZ, AANZ*ASATZ As Auftragswert
FROM        AUFTRAG
WHERE       AANZ*ASATZ >
            (SELECT     AVG(AANZ*ASATZ)
            FROM        AUFTRAG);
```

KKURZ	Auftragswert
100	18.000
110	9.600
101	9.000

Welcher Kunde hat den Auftrag mit dem höchsten Wert? (Anzeige der Firma, der PLZ und der Kundennummer):

```
(sql165)
SELECT    FIRMA, PLZ, KKURZ
FROM      KUNDE
WHERE     KKURZ =
          (SELECT   KKURZ
          FROM      AUFTRAG
          WHERE     (ASATZ*AANZ) =
                    (SELECT   MAX(ASATZ*AANZ)
                    FROM      AUFTRAG));
```

FIRMA	PLZ	KKURZ
Versicherungspartner	28719	100

3.5.7 SELECT mit Subquery mit mehr als einem Vergleichswert

Um mit mehr als einem Vergleichswert aus einer Nebentabelle arbeiten zu können, muß eine weitere Form des SELECT mit Subquery benutzt werden. Hierbei sind die Schlüsselwörter ANY, ALL, IN und EXISTS im Vergleichsoperator zulässig, wobei auch NOT IN und NOT EXISTS zugelassen sind.
Die allgemeine Form für solche SELECTS lautet (nur die WHERE-Bedingung):

```
WHERE     Spaltenname Vergleichsoperator [ANY | ALL]
          (Unterabfrage)
oder
WHERE     Spaltenname IN
          (Unterabfrage)
oder
WHERE     Spaltenname EXISTS
          (Unterabfrage)
```

Mit dem IN-Operator (entspricht einem „= ANY“) in der Hauptabfrage werden alle Zeilen gesucht, bei denen der betreffende Spaltenwert einem der Ergebniswerte der Unterabfrage entspricht.

Welche Aufträge haben die Bremer Kunden gegeben? Anzeige der Kundennummer, des Auftragskürzels und des Auftragstextes.

```
(sql170)
SELECT      KKURZ, AKURZ, ATEXT
FROM        AUFTRAG
WHERE       KKURZ IN
```

```
        (SELECT     KKURZ
        FROM        KUNDE
        WHERE       ORT = 'Bremen');
```

KKURZ	AKURZ	ATEXT
109	109	Serienbrief-Einweisung
100	111	Mitarbeiterschulung
101	114	ORACLE-Rezension

Der ANY-Operator vergleicht einen Wert mit jedem Wert der Ergebnisliste. Dabei sucht

> ANY nach allen Werten, die größer sind als irgendein Wert der Ergebnismenge der Unterabfrage,
< ANY nach allen Werten, die kleiner sind als ...

Welche Kunden haben einen höheren geplanten Auftragswert als irgendein anderer Kunde aus Bremen?

```
(sql168)
SELECT      AUFTRAG.KKURZ AS Kunde, AANZ*ASATZ AS
            Auftragswert
FROM        AUFTRAG
WHERE       ASATZ*AANZ > Any
            (SELECT     ASATZ*AANZ
            FROM        AUFTRAG, KUNDE
            WHERE       AUFTRAG.KKURZ = KUNDE.KKURZ
            AND         KUNDE.ORT = 'Bremen')
            ORDER BY AANZ*ASATZ DESC;
```

Kunde	Auftragswert
100	18.000
110	9.600
101	9.000
106	6.500
110	5.600

Mit dem ALL-Operator wird nach denjenigen Werten gesucht, die alle Ergebniswerte der Unterabfrage über- (bei > ALL) bzw. untertreffen (bei < ALL).

Suche alle Kunden, die einen größeren geplanten Auftragswert haben als alle Auslandskunden (LKZ <> 'D').

```
(sql169)
SELECT      FIRMA, PLZ, K.KKURZ, AANZ*ASATZ AS Auftrags-
            wert
FROM        KUNDE AS K, AUFTRAG AS A
WHERE       K.KKURZ = A.KKURZ
AND         AANZ*ASATZ > All
```

```
            (SELECT    ASATZ*AANZ
            FROM       AUFTRAG AS A, KUNDE AS K
            WHERE      K.KKURZ = A.KKURZ
            AND        LKZ <> 'D')
ORDER BY AANZ*ASATZ;
```

FIRMA	PLZ	KKURZ	Auftragswert
Versicherungspartner	28719	100	18.000

Mit dem EXISTS-Operator wird die Existenz von Werten abgefragt; dazu ein

Beispiel:

Welche Kunden haben keine Aufträge vorliegen?

```
(sql171)
SELECT     KKURZ, FIRMA, PLZ
FROM       KUNDE
WHERE      NOT EXISTS
           (SELECT    *
           FROM       AUFTRAG
           WHERE      KUNDE.KKURZ = AUFTRAG.KKURZ);
```

KKURZ	FIRMA	PLZ
102	MOTRON AG	6043
103	Eutez Finanz	40545
104	Deutsche Unterhal-	28283
105	Busy Software	
107	Therm AG	4133
108	Finkelstein	
111	Alemann GmbH	26127
112	Donakil GmbH	26123

Noch ein Beispiel zum IN-Operator. Betrachten wir folgende Tabelle:

ArtNr	ArtBez	Datum	Preis
1	A	01.04.95	10,00 DM
1	A	01.05.95	8,00 DM
2	B	01.02.95	20,00 DM
2	B	01.04.95	18,00 DM
3	C	01.03.95	30,00 DM

Es soll nun von jedem Artikel der jüngste Einkaufspreis ermittelt werden:

```
SELECT     A1.ArtBez, A1.Datum, A1.Preis
FROM       Artikel AS A1
WHERE      A1.Datum IN
           (SELECT    MAX(A2.Datum)
```

```
FROM        Artikel AS A2
WHERE       A1.ArtBez = A2.ArtBez
GROUP BY ArtBez);
```

ArtBez	Datum	Preis
A	01.05.95	8,00 DM
B	01.04.95	18,00 DM
C	01.03.95	30,00 DM

Die Verbundbedingung WHERE A1.ArtBez = A2.ArtBez ist erforderlich, da sonst auch das Datum 01.04.95 für den Artikel A geliefert wird, denn die Unterabfrage bringt ja auch diesen Wert für den Artikel B.

3.5.8 UNION, INTERSECT, MINUS

UNION ist der Operator für die *Vereinigung* von Ergebnismengen. Die Grundform für UNION lautet:

```
SELECT      Spaltenname,,,
FROM  Tabellenname
WHERE       Klausel
UNION
SELECT      Spaltenname,,,
FROM  Tabellenname
WHERE       Klausel
```

Beispiel: Welche Kundenkürzel gibt es in AUFTRAG und KUNDE?

```
(sql174)
SELECT      KKURZ
FROM  AUFTRAG
UNION
SELECT      KKURZ
FROM  KUNDE;
```

KKURZ
100
101
102
103
104
105
:

Für UNION gilt folgende Restriktion:

Es muß die gleiche Anzahl Spalten projiziert werden, wobei die Projektionsspalten in den verschiedenen Abfrageblöcken denselben Datentyp haben müssen.

Doppelte Reihen, also Tupel, die in allen projizierten Attributen denselben Wert haben, werden automatisch zu einer Zeile zusammengezogen. DISTINCT wirkt deswegen nicht.

Beispiel:

```
SELECT    KuNr, KuName
FROM      KUNDE
UNION
SELECT    LiefNr, LiefName
FROM      LIEFERANT
```

Neben UNION können verwendet werden:

INTERSECT: Durchschnitt von Ergebnismengen
MINUS: Differenz von Ergebnismengen

3.6 Strategien zur Formulierung von SELECT

Wegen der differenzierten Möglichkeiten der Datenmanipulations-Befehle sollte die interaktive Arbeit mit SELECT, INSERT, UPDATE und DELETE stets gut vorbereitet werden. Komplexe SELECT's können sehr schnell selbst leistungsstarke Datenbankserver lahmlegen, INSERT, UPDATE und DELETE können kurzfristig große Datenbestände verändern oder löschen! Hier einige Tips in Form von Fragestellungen für das Vorgehen beim Formulieren von SELECT-Kommandos:

1. Wie soll die Ergebnisliste aussehen?

2. Welche Tabelle wird benutzt? (zunächst ohne Verbund)
   Bietet es sich an, mit Aliasnamen für Tabellen zu arbeiten? Im Spaltenausdruck für die Projektion muß der Tabellen- oder Aliasname (mit As) als Präfix vorangestellt werden.

2.1 Welche Spalten sollen angezeigt werden?

2.2 Welche virtuellen Spalten kommen vor?
   Es besteht die Möglichkeit,
   - (reihenbezogene) Berechnungen
   - (reihenbezogene) Stringoperationen (z. B. Verkettung)
   - (Gruppen-) Funktionen

   zu verwenden. Wenn keine Gruppenfunktionen benötigt werden, schließt sich Frage 3 an.

2.3 Welche (Gruppen-) Funktionen kommen vor?
   Um das Ergebnis einer (Gruppen-) Funktion anzuzeigen, darf die Ergebnistabelle entweder nur einen Wert haben oder es muß

später eine GROUP BY - Anweisung folgen. Bei der Verwendung von GROUP BY darf nur auf die Attribute projiziert werden, auf die gruppiert wird, oder auf solche, die mit Gruppenfunktionen zu virtuellen Spalten gemacht werden.

3. Werden mehrere Tabellen benutzt? (Verbund)
Wird mehr als eine Tabelle angegeben, entsteht ein kartesisches Produkt. Alle Reihen der einen werden mit allen Reihen der anderen Tabellen kombiniert (multipliziert, aneinandergehängt). Mit der Verbundbedingung werden die korrespondierenden Zeilen der Basistabellen ausgewählt. Es ist festzulegen, ob eine Gleichheits- oder Inklusionsverknüpfung erforderlich ist.

4. Welche Selektionsbedingungen liegen vor?

4.1 Sind die Vergleichswerte der Selektion Konstanten oder Spaltenausdrücke bezogen auf dieselbe Zeile (der Haupttabelle)?

4.1.1 Bezieht sich die Bedingung auf einzelne Reihen?
Die Bedingung muß in der WHERE-Klausel deklariert werden.

4.1.2 Bezieht sich die Bedingung auf eine Gruppe von Reihen?
Es muß GROUP BY mit HAVING-Klausel verwendet werden, s.u.

4.2 Stammen die Vergleichswerte aus einem oder mehreren anderen Zeilen derselben oder anderer Tabellen?

4.2.1 Soll der Vergleichswert aus einer anderen Zeile ermittelt werden oder aus der Berechnung einer Gruppenfunktion entstammen, wird eine Subquery formuliert, die nur einen Ergebniswert in der Nebentabelle findet.

4.2.2 Soll der Vergleichswert aus mehr als einer anderen Zeile ermittelt werden, wird eine Subquery mit ANY, ALL, IN oder EXISTS formuliert.

5. Welche Gruppenbildung wird benötigt?
Werden Eigenschaften untersucht, die nicht oder nicht allein den Tabellenschlüssel bilden, können durch GROUP BY mehrfach auftretende gleiche Attributwerte zu einer Gruppe zusammengefaßt werden. Das wirkt sich restriktiv auf die Projektion aus.

6. Wird eine bestimmte Ordnung der Liste erwünscht?
Nur durch eine ORDER BY - Klausel ist gewährleistet, daß die Ergebnistabelle sortiert wird. Bei Verwendung einer Gruppenbildung erübrigt sich eine Sortieranweisung.

3.7 Views

Eine *View* ist eine virtuelle Tabelle, d. h. sie ist nicht physikalisch vorhanden. Mit einer View wird für einen bestimmten Benutzer eine für ihn geeignete Sicht auf die Daten bereitgestellt.
Views werden mittels SELECT-Kommandos definiert. Views können auch nicht nur zum Lesen von Daten, sondern auch zum Ändern des Datenbestands unter bestimmten Voraussetzungen verwendet werden.
In Views können insbesondere auch abgeleitete Daten, z. B. Rechenergebnisse (NETTO = LANZ*LSATZ, Umsatz pro Kunde, usw), übernommen werden.
Mit Views wird der *Datenschutz* besonders unterstützt, da für jeden Benutzer nur die für ihn zugreifbaren Daten über Views zugänglich gemacht werden können. Ebenfalls sind die VIEWs ein Beitrag zur *Datenunabhängigkeit*; falls die Basistabellen geändert werden müssen, brauchen die Views davon nicht betroffen zu sein.

Die Syntax des View-Kommandos lautet:

```
CREATE VIEW     Viewname [(Spaltenname,,,)]
AS
SELECT-Kommando
[WITH CHECK OPTION];
```

Für Viewnamen gelten dieselben Regeln wie für Tabellennamen. Sind keine Spaltennamen angegeben, werden die im SELECT-Kommando angegebenen Spalten für die View-Definition verwendet. Virtuelle Spalten benötigen immer einen Spaltennamen.
Das SELECT-Kommando darf kein ORDER BY enthalten. WITH CHECK OPTION bedeutet, daß beim Ändern (INSERT, UPDATE, DELETE) in der View automatisch darauf geachtet wird, daß die View-Definition im SELECT-Kommando durch die Änderung nicht verletzt wird.

Mit

```
DROP VIEW       Viewname;
```

wird die View gelöscht.

Neben Views können auch *Synonyme* für Tabellen definiert werden:

```
CREATE SYNONYM XYZ FOR ABC;
DROP SYNONYM XYZ;
```

Als Beispiel soll eine View erzeugt werden, die nur die Kundenadressen enthält:

(sql185)

```
CREATE VIEW     KADR
```

```
AS
SELECT      KKURZ, FIRMA, STRASSE, LKZ, PLZ, ORT
FROM        KUNDE;
```

SELECT * FROM KADR; bringt folgende Ausgabe:

KKURZ	FIRMA	STRASSE	LKZ	PLZ	ORT
100	Versiche-	Ringstr. 42	D	28719	Bremen
101	Computer	Heerstr. 101	D	28307	Bremen
102	MOTRON	Buggena-	CH	6043	Adligens-
103	Eutez Fi-	Luegallee 7	D	40545	Düssel-
104	Deutsche	Postfach	D	28283	Bremen
105	Busy Soft-	BOX 27969	USA		San Diego
106	Ringwerke	Postfach	D	35001	Marburg
107	Therm AG	Zehntenstr.	CH	4133	Pratteln
108	Finkelstein	25 East	USA		Chicago
109	JUFI GmbH	Stolzen Str.	D	28207	Bremen
110	Bacher E-	Spinne-	CH	9008	St. Gallen
111	Alemann	Kreienweg	D	26127	Oldenburg
112	Donakil	Karlsdamm	D	26123	Oldenburg

Beispiel:

Es soll eine View erzeugt werden, die nur die Kundenadressen der Bremer Kunden (Ort = Bremen) enthält:

```
(sql187)
CREATE VIEW     KADR28
AS
SELECT          KKURZ,
                FIRMA,
                STRASSE,
                LKZ,
                PLZ,
                ORT
FROM            KUNDE
WHERE           LKZ = 'D'
AND             ORT = 'Bremen';
```

SELECT * FROM KADR28; zeigt an:

KKURZ	FIRMA	STRASSE	LKZ	PLZ	ORT
100	Versiche-	Ringstr. 42	D	28719	Bremen
101	Computer Cou-	Heerstr. 101	D	28307	Bremen
104	Deutsche Un-	Postfach	D	28283	Bremen
109	JUFI GmbH	Stolzen Str.	D	28207	Bremen

Beispiel:

Wie sql187, jedoch mit CHECK OPTION:

(sql189)

```
CREATE VIEW    KADR28CK
AS
SELECT         KKURZ,
               FIRMA,
               ZUSATZ,
               STRASSE,
               LKZ,
               PLZ,
               ORT
FROM           KUNDE
WHERE          LKZ = 'D'
AND            ORT = 'Bremen'
WITH           CHECK OPTION;
```

Beispiel:

Ändern in der View KADR28CK:

(sql191)

```
UPDATE      KADR28CK
SET         STRASSE = 'Marschweg 136'
WHERE       KKURZ = 109;
```

Der Versuch, den ORT in der View KADR28CK auf 'Hamburg' zu setzen, weil der Kunde nach Hamburg umgezogen ist, wird fehlschlagen, da der ORT wegen „AND ORT = 'Bremen'„ auf diesen Wert festgelegt ist.

Mit Hilfe der CHECK OPTION können diskrete Werte für Views als zugelassene Werte definiert werden.

Beispiel:
In einer View sollen die Mehrwertsteuersätze auf die Werte 0, 0,07 und 0,16 eingeschränkt werden:

(sql192)

```
CREATE VIEW    AUF_MWST
AS
SELECT         *
FROM           AUFTRAG
WHERE       MWSATZ IN (0, 0.07, 0.16)
WITH           CHECK OPTION;
```

Beispiel:
Die folgende View enthält eine virtuelle Spalte mit einem berechneten Wert:

(sql194)

```
CREATE VIEW   LEISTNET  (AKURZ, LNR, LTEXT, LDATUM,
              NETTO)
AS
SELECT        AKURZ, LNR, LTEXT, LDATUM,
              LANZ*LSATZ
FROM          LEISTUNG
```

SELECT * FROM LEISTNET; liefert:

AKURZ	LNR	LTEXT	LDATUM	NETTO
111	1	Schulungsmaterial	20.05.98	3.375
111	2	Kopien/Folien	21.05.98	300
111	3	DBASE-IV-Schulung	25.05.98	13.500
112	4	Systemanalyse	15.01.96	12.000
112	5	Datenbankentwurf	07.02.96	9.600
112	6	Programmspezifikation	13.03.96	12.000
113	7	Programmspezifikation	13.12.97	49.000
109	8	Serienbrief-Einweisung	01.05.96	1.500
115	9	Porto	20.06.97	20
115	10	Handbuch	20.06.97	125
115	11	Datenbank-Entwurf DIAG	22.06.97	250
115	12	Datenbank-Entwurf DIAG	02.07.97	1.250
115	13	Datenbank-Entwurf DIAG	02.08.97	2.500
115	14	Datenbank-Entwurf DIAG	01.09.97	2.500
114	15	Rezension SQL*PLUS	10.06.98	6.000
114	16	Rezension SQL*FORMS	29.06.98	3.000

Ein Änderungsversuch (INSERT, UPDATE, DELETE) auf dieser View wie folgt:

(sql196a)

```
UPDATE        LEISTNET
SET           NETTO = 10000
WHERE         LNR = 13;
```

wird auch fehlschlagen, da die View die virtuelle Spalte NETTO mit einem berechneten Wert enthält.
Es können auch Views mit GROUP BY erzeugt werden; dazu ein Beispiel:

(sql198)

```
CREATE VIEW   AUFTRUMS(AKURZ, UMSATZ, STAND)
AS
SELECT        AKURZ, SUM(LANZ*LSATZ),
              MAX(LDATUM)
FROM          LEISTUNG
GROUP BY      AKURZ;
```

SELECT * FROM AUFTRUMS; zeigt an:

AKURZ	UMSATZ	STAND
109	1.500	01.05.96
111	17.175	25.05.98
112	33.600	13.03.96
113	49.000	13.12.97
114	9.000	29.06.98
115	6.645	01.09.97

Mit folgenden SELECTs kann z. B. auf die View zugegriffen werden:

```
(sql201a)
SELECT      *
FROM        AUFTRUMS
WHERE       UMSATZ > 10000;
```

AKURZ	UMSATZ	STAND
111	17.175	25.05.98
112	33.600	13.03.96
113	49.000	13.12.97

```
(sql202a)
SELECT AVG(UMSATZ) AS [Mittlerer Umsatz]
FROM AUFTRUMS;
```

Mittlerer Umsatz
19.486,67

Ebenso kann in der View-Definition DISTINCT verwendet werden:

```
(sql202b)
CREATE VIEW    AUFDIST
AS
SELECT         DISTINCT AKURZ
FROM           LEISTUNG;
```

SELECT * FROM AUFDIST; zeigt dann:

AKURZ
109
111
112
113
114
115

Folgendes Beispiel zeigt eine View mit Verbund:

(sql204)

```
CREATE VIEW   KUNDAUF(AKURZ, KKURZ, FIRMA, ORT,
              ATEXT, ASTNR)
AS
SELECT        AUFTRAG.AKURZ, AUFTRAG.KKURZ, KUN-
              DE.FIRMA,
              KUNDE.ORT, AUFTRAG.ATEXT, AUF-
              TRAG.ASTNR
FROM          AUFTRAG, KUNDE
WHERE         AUFTRAG.KKURZ = KUNDE.KKURZ;
```

SELECT * FROM KUNDAUF ORDER BY AKURZ; liefert:

AKURZ	KKURZ	FIRMA	ORT	ATEXT	ASTNR
111	100	Versiche-	Bremen	Mitarbei-	1
114	101	Computer	Bremen	ORACLE-	2
115	106	Ringwerke	Marburg	Daten-	3
109	109	JUFI GmbH	Bremen	Serien-	4
112	110	Bacher E-	St. Gallen	Offerten-	4
113	110	Bacher E-	St. Gallen	Kunden-	3
116	110	Bacher E-	St. Gallen	Einführung	2

Auf Join-Views können generell keine Änderungsoperationen (INSERT, UPDATE, DELETE) durchgeführt werden, da ja zwei oder mehr Basistabellen betroffen wären und nicht immer klar ist, was gelöscht werden soll; so wird z. B.

(sql205)

```
DELETE FROM KUNDAUF WHERE AKURZ = 112;
```

nicht erfolgreich ausgeführt werden.
Ein weiteres Beispiel mit Join-View; es sollen die Umsätze der Firmen in einer View dargestellt werden:

(sql207)

```
CREATE VIEW   KUNDUMS(KKURZ, UMSATZ, STAND)
AS
SELECT        KKURZ, SUM(LANZ*LSATZ),
              MAX(LDATUM)
FROM          LEISTUNG, AUFTRAG
WHERE         LEISTUNG.AKURZ = AUFTRAG.AKURZ
GROUP BY      KKURZ;
```

SELECT * FROM KUNDUMS; zeigt dann:

KKURZ	UMSATZ	STAND
100	17.175	25.05.98
101	9.000	29.06.98
106	6.645	01.09.97
109	1.500	01.05.96

110	82.600	13.12.97

Bei Änderungsoperationen auf Views gelten bestimmte, teilweise allgemeingültige oder herstellerabhängige Restriktionen**.** Z. B. darf die die View definierende Abfrage folgendes nicht beinhalten:

- DISTINCT
- berechnete (virtuelle) Spalten
- Tabellenverbund (join)
- Subquery
- GROUP BY, HAVING

3.8 Datenkontrolle

Zur Datenkontrolle, d. h. zur Definition von Zugriffsrechten und zur Transaktionssteuerung, bietet SQL ebenfalls Kommandos an.

3.8.1 Zugriffsrechte

Eines der Wesensmerkmale einer Datenbank ist, das viele Benutzer auf sie zugreifen. Um die Benutzer auseinander halten zu können, hat jeder einen Benutzernamen und ein Password, mit dem er Zugang zur Datenbank hat. Um sicherzustellen, daß keine unberechtigten Zugriffe auf bestimmte Daten erfolgen, verfügen die Datenbanksysteme über ein umfangreiches *Zugriffsrechte*-Konzept. Dabei sind Rechte einerseits für Personen (Benutzer, User) und andererseits für Objekte (Tabellen, Spalten, Views, Formulare, Programme usw) festlegbar, wobei Rechte für die unterschiedlichen Operationen (Anlegen, Lesen, Ändern, Löschen, usw.) auf die Objekte vergeben werden können.
Grundsätzlich gilt das *Besitzer-Konzept*, d. h. daß jemand, der eine Tabelle angelegt hat, deren Besitzer ist und damit alle Rechte darauf hat. Diese Rechte kann er an andere Benutzer weitergeben oder ihnen wieder entziehen. Bei der Weitergabe der Rechte kann er auch das Recht der erneuten Weitergabe für die erhaltenen Rechte mitgeben.
Es wird häufig zwischen einer privaten Datenbank, die nur einem Benutzer gehört, und der öffentlichen (PUBLIC) Datenbank unterschieden. Letztere ist diejenige, auf die alle Benutzer zugreifen können..
Neben den normalen Benutzern der Datenbank gibt es noch einen besonderen Benutzer, den *Datenbank-Administrator* (DBA). Dieser hat alle Rechte auf die Datenbank, insbesondere auf den public-Teil. Sowohl der DBA wie auch die anderen DB-Benutzer müssen „angelegt“ werden, d. h. es müssen Benutzernamen (Login-Namen), Kennwörter (Passwords), Gruppennamen und andere Informationen eingegeben werden. Jeder Benutzer muß sich dann mit seinem Benutzernamen und -Kennwort in das Datenbanksystem einwählen (Login).
Unter SQL steht folgendes Kommando für die Rechtevergabe zur Verfügung:

GRANT und REVOKE

```
GRANT       Privileg,,, | ALL [PRIVILEGES]
ON          Tabellenname,,,
TO          Benutzername,,, | PUBLIC
            [WITH GRANT OPTION]
```

mit den Privilegien:

- SELECT
- INSERT
- UPDATE
- DELETE
- ALTER
- INDEX

Falls WITH GRANT OPTION angegeben ist, kann der Rechte-Empfänger diese Rechte an andere weitergeben.
Mit ALL werden alle Privilegien vergeben; anstelle eines Tabellennamens kann auch ein View-Name angegeben werden.

Beispiel:

(sql254a)

```
GRANT       SELECT, UPDATE, INSERT
ON          KUNDE
TO          Müller
WITH GRANT OPTION;
```

Mit

```
REVOKE      Privileg,,, | ALL [PRIVILEGES]
ON          Tabellenname,,,
FROM        Benutzername,,, | PUBLIC
```

werden Zugriffsrechte wieder entzogen; dabei werden auch automatisch alle weitergereichten Rechte gelöscht.

Beispiel:

```
REVOKE      ALL
ON          KUNDE
FROM        Müller;
```

Speziell die UPDATE-Berechtigung kann weiter auf bestimmte Spalten eingeschränkt werden:

```
GRANT       UPDATE (Spaltenname,,,)
ON          KUNDE
TO          Müller;
```

Um ansonsten die Rechte auf bestimmte Spalten und Zeilen einzuschränken, können geeignete Views definiert werden, auf die dann die entsprechenden Rechte vergeben werden; z. B.

```
CREATE VIEW    ABGESCHL(AKURZ, STATUS, NETTO)
AS
SELECT         AKURZ, ASTNR, AANZ*ASATZ
FROM           AUFTRAG
WHERE          ASTNR >= 3;
```

Es können für Tabellen- oder Viewnamen auch *Synonyme* verwendet werden. Jeder Benutzer kann solche Synonyme vergeben, die dann nur für ihn gelten. Er ist damit auch der Eigner des Synonyms, so daß nur er und der DBA das Synonym wieder löschen können. Mit

```
CREATE     SYNONYM  Synonym-Name
FOR        Tabellenname | Viewname;
```

wird ein Synonym definiert.

3.8.2 Transaktionen

Änderungen des Datenbestands in der Datenbank sind häufig umfangreicher und nicht auf einen einzigen UPDATE begrenzt.

Beispiel:

> In der Auftrags-Datenbank soll eine Rechnung erstellt werden, in der alle Leistungen eines Auftrags aufgeführt sind. Diese Leistungen erhalten in der Spalte RNR eine neue Rechnungsnummer. Zusätzlich soll der betreffende Auftragsstatus in der Tabelle AUFTRAG von 2 (Vertrag) auf 3 (erledigt) übergehen.

Wir haben es hier mit mehreren Datenbank-Zugriffen zu tun, die sämtlich durchführt werden müssen, wenn die Aktion „Rechnung erstellen“ ordnungsgemäß abgeschlossen werden soll. Die Menge der DB-Zugriffe, die zu dieser Aktion gehören, wird mit Datenbank-*Transaktion* bezeichnet. Eine korrekte Transaktion muß also „ganz oder gar nicht“ durchgeführt werden. Man bezeichnet diese Eigenschaft der Transaktion mit *Atomarität* (atomicy). Damit führt eine Änderungs-Transaktion einen konsistenten Zustand der Datenbank in einen neuen konsistenten Zustand über, d. h. die Konsistenz (Consistency) der Daten bleibt erhalten.

Es können bei einer DV-Anlage jedoch Probleme auftreten, die ein vollständiges Abschließen einer Transaktion verhindern. Dazu gehören:

- Programmierfehler in der Anwendung
- Systemabstürze durch Hardwarefehler oder Stromausfall
- defekte Platten
- Fehleingaben des Benutzers

Das DBVS muß trotzdem sicherstellen, daß die Atomarität der Transaktion gewährleistet wird. Damit es „weiß“, aus welchen SQL-Kommandos die Transaktion besteht, muß ihm das über spezielle SQL-Kommandos zur Transaktionssteuerung, die wie Transaktionsklammern wirken, mitgeteilt werden.
Die SQL-Kommandos lauten

- COMMIT für den Start einer neuen Transaktion
- COMMIT für den Abschluß einer laufenden Transaktion
- ROLLBACK für das Zurücksetzen einer laufenden Transaktion

Mit COMMIT wird also die laufende Transaktion ordnungsgemäß beendet und eine neue gestartet. Folgende Tabelle zeigt die Unterschiede der Transaktionssteuerung für Standard-SQL und MS Access auf:

Aktion	SQL-Kommando	MS Access
Beginn einer Transaktion (BOT)	COMMIT WORK	CommitTrans
Ende einer Transaktion (EOT)	COMMIT WORK	CommitTrans
Zurücksetzen einer Transaktion	ROLLBACK [WORK]	Rollback

Tab. 6: Transaktionssteuerung

Falls einer der o. g. Fehler während der Abarbeitung einer Transaktion auftritt, wird beim erneuten Hochfahren des Datenbanksystems wie folgt verfahren:

- Zum Zeitpunkt des Absturzes offene Transaktionen werden zurückgesetzt, d. h. die von dieser Transaktion betroffenen Daten werden auf den Zustand vor der Transaktion gebracht.
- Abgeschlossene Transaktionen werden ggfs. erneut wiederholt und wieder abgeschlossen. Dies ist deshalb nötig, weil die Effekte abgeschlossener Transaktionen evtl. noch nicht in der Datenbank, d. h. auf dem Speichermedium (z. B. Platte) realisiert, sondern nur im Pufferpool des Hauptspeichers verblieben sind. Bei einem Stromausfall sind diese Daten natürlich verloren. Die Transaktion garantiert also auch die *Dauerhaftigkeit* (Duration, oder Persistenz/Persistency) der durch sie getätigten Datenbankänderungen.

ROLLBACK kann verwendet werden, wenn der Benutzer sich nach Start einer Transaktion entschließt, die Aktion nicht weiter durchzuführen.

3.9 SQL-Kommandos in Programmen

SQL-Kommandos können nicht nur direkt am Bildschirm ausgeführt werden. Es besteht auch die Möglichkeit, sie in einer Programmiersprache für den Datenbankzugriff zu verwenden. In der Praxis haben sich zwei Varianten der Integration etabliert: *Embedded SQL* (ESQL)) über ein Precompiler-Konzept und CLI (Call Level Interface) über Bibliotheksfunktionen.
Zu letzterem gehört die Einbindung von SQL bei MS Access in Visual Basic for Applications.
Im folgenden wird das ESQL-Konzept näher betrachtet, da es sich hier um eine standardisierte Schnittstelle handelt.
Die SQL-Kommandos werden praktisch 1:1 in das Programm übernommen. Ein Precompiler übersetzt zunächst diese SQL-Kommandos und erzeugt daraus Funktionsaufrufe für die DBVS-Laufzeitbibliothek, bevor der Compiler das Programm (in COBOL, C usw.) übersetzt.
Bei der Einbettung sind zwei Besonderheiten zu beachten:

- Die Daten werden in Programm-Variable kopiert.
- Der Mengenzugriff wird mit Hilfe eines Zwischenspeichers (Cursor) in Einzelsatzverarbeitung umgesetzt.

3.9.1 Datenübergabe bei Einzelsatzzugriff

Bei einem SELECT auf einen einzelnen Datensatz erfolgt die Übergabe der Daten einer Zeile in die Programmvariablen mittels einer INTO-Klausel:

```
SELECT      Spaltenname,,,
INTO        Variablenname,,,
FROM        Tabellenname
[WHERE      ...]
```

Dabei wird jeweils der Inhalt des ersten/zweiten/... Spaltennamens in die erste/zweite/... Variable übergeben. Die Datentypen müssen übereinstimmen. Die o. g. einfache Form des SELECT kann in einem Programm nur dann verwendet werden, wenn die Ergebnismenge aus maximal einer Zeile besteht.

Beispiel:

```
SELECT      KKURZ, FIRMA
INTO        V_KKURZ, V_FIRMA
FROM        KUNDE
WHERE       KKURZ = 100;
```

Bei INSERT, UPDATE oder DELETE von einzelnen Zeilen werden anstelle der explizit angegebenen Feldwerte (z. B. in der VALUES-Klausel) Variablen genannt, die die betreffenden Feldwerte enthalten, z. B.

```
INSERT
INTO        KUNDE
VALUES      (M_kkurz, M_firma, M_zusatz, ...);
```

3.9.2 Cursor-Konzept für SELECT mit Mengenzugriff

Programmiersprachen der 3. Generation sind i. a. so ausgelegt, daß sie bei einem Dateizugriff nur immer einen Datensatz zu einer Zeit verarbeiten können. Das gilt auch für Datenbankzugriffe. SQL-SELECTs liefern jedoch i. a. eine Ergebnismenge, also mehrere Datensätze. Um diese Datenmenge im Programm zu verarbeiten, ist das Cursor-Konzept eingeführt worden.
Die Ergebnismenge des SELECT wird zunächst in einem sogen. Cursor, einem Zwischenspeicher, abgelegt. Von dort können die Ergebniszeilen einzeln und nacheinander vom Programm abgeholt werden.

Es steht folgendes Kommando zur Verfügung:

```
DECLARE     Cursorname CURSOR
FOR
SELECT      ...
```

SELECT steht für eine beliebige Abfrage. Cursornamen müssen gültige Namen sein. Ein so definierter Cursor ist für die Dauer der SQL-Sitzung nutzbar.

Beispiel:
Es soll ein Cursor für einen SELECT auf KUNDE definiert werden, wobei der Ort in einer Variablen v_ort übergeben wird.

```
DECLARE     C_KUNDE CURSOR
FOR
SELECT      KKURZ, FIRMA
FROM        KUNDE
WHERE       ORT = v_ort;
```

Der SELECT im DECLARE Cursor wird erst bei dem Kommando

OPEN Cursorname

ausgeführt.

Bei vielen DBVSen wird in einer Systemvariablen SQLCODE ein Rückgabewert zur Fehlersituation abgelegt, z. B.

SQLCODE = 0: Kein Fehler
SQLCODE = -1: Fehler beim SELECT aufgetreten

Analog liefert einer Systemvariable SQLCNT die Anzahl Datensätze der Ergebnismenge. Die Datensätze werden in der vom SELECT ausgegebenen Ordnung im Cursor zwischengespeichert und vom Programm anschließend mittels

```
FETCH       Cursorname
INTO        Variablenname,,,
```

in einer Schleife einzeln abgeholt, z. B.

```
FETCH       C_KUNDE
INTO        v_kkurz, v_firma;
```

Falls versucht wird, über die Anzahl der Ergebniszeilen (SQLCNT) hinweg mit FETCH weitere Zeilen zu lesen, liefert SQLCODE einen Fehlerwert.

Der Cursor wird mit CLOSE Cursorname deaktiviert.

3.9.3 Cursor-Konzept für UPDATE, DELETE

Um Zeilen zu ändern (UPDATE) oder zu löschen (DELETE), kann auch das Cursor-Konzept genutzt werden. Dazu ein Beispiel

(sql308):

```
DECLARE       aufcur CURSOR
FOR
SELECT        *
FROM          AUFTRAG
WHERE         ASTNR < 4
FOR UPDATE OF KKURZ, PNR, ADATUM, FERTIGIST, ...,
              ASTNR;
```

Nach dem OPEN aufcur kann mit dem Cursor wie mit einer Tabelle gearbeitet werden. Mit der Klausel FOR UPDATE OF ist es möglich, bei Bedarf die Tabelle AUFTRAG zu ändern, wobei wegen der WHERE-Klausel nur Zeilen mit ASTNR < 4 geändert werden können. Es ist auch nicht möglich, den Primärschlüssel AKURZ zu ändern. Bei einer formularbasierten Anwendung werden typischerweise bestimmte Datensätze erst auf den Bildschirm geholt, dann teilweise

vom Benutzer aktualisiert und zum Abschluß in die Datenbank zurückgeschrieben. Für solche Anwendungen kann das Cursorkonzept wie folgt eingesetzt werden:
Zunächst werden aus dem o. g. Cursor aufcur eine Zeile per FETCH abgeholt und die Werte in der Bildschirmmaske angezeigt. Dann ändert der Benutzer bestimmte Datenfelder, wobei sich diese Änderungen zunächst in den Programmvariablen abspielen. Anschließend wird vom Programm mittels

```
UPDATE      AUFTRAG
SET         KKURZ=v_kkurz,
            PNR=v_pnr
WHERE       CURRENT OF aufcur;
```

die betreffende Zeile in der Tabelle AUFTRAG geändert. CURRENT OF gibt an, daß es sich um die gerade bearbeitete Zeile handelt. Mit

```
DELETE      FROM Tabellenname
WHERE       CURRENT OF Cursorname;
```

können Zeilen nach dem gleichen Verfahren in der Tabelle gelöscht werden.

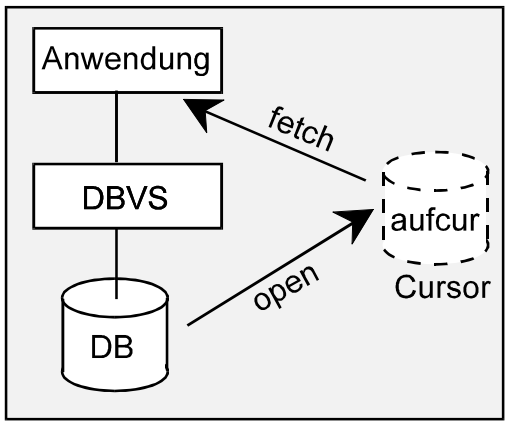


Abb. 4: Das Cursorkonzept

4 Datenbank – Technik

Bedingt durch die Besonderheiten von Datenbankanwendungen (viele Benutzer bzw. Programme greifen parallel auf eine Datenbank zu), sind Konzepte für

- die *Daten-Sicherheit* (Security),
- die *Zugriffs-Parallelität* (Concurrency)
- die *Daten-Konsistenz* (Consistency) und
- den schnelle Zugriff (*Performance*)

erforderlich und werden von den DBVSen auch unterstützt.
Zu dem Bereich der Daten-Sicherheit gehören die Themen Zugangsberechtigung (Login, Password), Sichten (Views) und Zugriffsrechte (GRANT), die bereits behandelt wurden.
Im folgenden werden die Zugriffs-Parallelität auf Basis von Sperrverfahren, die Erhaltung der Daten-Konsistenz auf Basis des Transaktionskonzeptes, die Wiederherstellung von konsistenten Daten nach Fehlern mittels Recovery-Verfahren und der schnelle Datenzugriff mit Hilfe von Indizes vorgestellt.

4.1 Transaktionen und Sperren

Um die *Konsistenz* der Daten zu gewährleisten, ist das Transaktionskonzept in SQL eingeführt worden. Eine *Transaktion* ist dabei eine Folge von Datenbankzugriffen mit folgenden Eigenschaften:

Atomarität:
Das DBVS stellt (mittels Logging-Technik, s. u.) sicher, daß eine Transaktion selbst im Fall eines Systemabsturzes entweder komplett oder gar nicht durchgeführt wird.

Konsistenzerhaltung:
Ein Programm kann bei negativem Resultat anwendungsspezifischer Konsistenzprüfung von sich aus den Abbruch der Transaktion erzwingen.

Isoliertheit:
Die Transaktion ist von Effekten anderer Transaktionen isoliert; sie kann z. B. ohne Nebeneffekte zurückgesetzt werden. Beim Multiuser-Zugriff werden die DB-Zugriffe so durch Sperren oder andere Techniken gesteuert, daß Serialisierbarkeit der Zugriffe gewährleistet ist.

Damit bleibt die Konsistenz des Datenbestandes hinsichtlich des Multiuser-Zugriffs erhalten.

Dauerhaftigkeit:
Nach Beendigung einer Transaktion wird die Dauerhaftigkeit aller Änderungen garantiert (Persistenz).

Mit dem Transaktionskonzept wird damit das *ACID-Prinzip* realisiert: Atomicy, Consistency, Isolation, Duration.
Im logischen Entwurf muß herausgearbeitet werden, welche Datenbankoperationen jeweils eine Transaktion bilden. Transaktionen sollten dabei nicht zu lang werden, weil dann die Parallelität im Multiuser-Verhalten zu stark eingeschränkt wird.
Ein wichtiger Aspekt, der bereits beim Entwurf berücksichtigt werden muß, ist die Wahrung der Integrität der Daten.

4.1.1 Asserts und Trigger

SQL stellt zwei Funktionen zur Sicherung der Integrität zur Verfügung. Mit *Asserts* können beim CREATE TABLE Integritätsbedingungen formuliert werden, die bei Änderungsoperationen überprüft werden. Im Mißachtungsfall wird die Transaktion zurückgesetzt. Damit wird auch die *Referentielle Integrität* gewährleistet. Beim Ändern eines Wertes in der Tabelle X wird der referenzierte Wert in der Tabelle Y, z. B. ein Fremdschlüssel, automatisch aktualisiert.
Der zweite Mechanismus ist die Möglichkeit, *Trigger* zu verwenden. Dabei handelt es sich um meist kleine Datenbank-Anwendungen (Prozeduren), die bei bestimmten Aktionen automatisch ausgeführt werden und beispielsweise Integritätsüberprüfungen realisieren.

4.1.2 Sperren

Beim Arbeiten mit einer Datenbank passiert es häufig, daß zwei Benutzer denselben Datensatz bearbeiten wollen. Es kann dabei z. B. folgende Situation entstehen, bei der parallele Transaktionen sich ins Gehege kommen können:

Buchungs-Transaktion des Benutzers 1:
    Abbuchen eines Betrags von Konto K1
    Zubuchen dieses Betrags auf Konto K2

Transaktion des Benutzers 2:
    Ermittelt Summe aller Konten

Falls nun durch das Time-Sharing der Benutzer 2 genau dann drankommt, wenn für den Benutzer 1 die Abbuchung bereits erfolgt, die Zubuchung aber noch realisiert werden muß, „sieht“ Benutzer 2 eine falsche Summe: die Datenbank ist kurzzeitig inkonsistent. Ähnliche Probleme entstehen, wenn mehrere Benutzer dieselben Daten ändern

wollen. Hier „gewinnt“ der Benutzer, der zuletzt den neuen Wert in die Datenbank wegschreibt (lost update).
Um diese Art der Inkonsistenz zu vermeiden, müssen die Teile der Datenbank, die von solchen Änderungen betroffen sind, vorübergehend gegen den Zugriff anderer Benutzer gesperrt werden. Dazu bieten die DBVSe die Möglichkeit, *Sperren* auf Tabellen oder Zeilen zu setzen und wieder freizugeben.
Bei den Sperren wird unterschieden, ob es sich um Lesesperren (Sperren des Objekts zum Lesen) oder Schreibsperren handelt. Eine Kompatibilitätsmatrix gibt an, welche Sperren miteinander verträglich sind, d. h. konkurrierenden Zugriff erlauben:

	Lesesperre	Schreibsperre
Lesesperre	**x**	-
Schreibsperre	-	-

Tab. 7: Sperrkompatibilität

Das bedeutet: wenn ein Benutzer ein Objekt zum Lesen angefordert hat, können andere, die das Objekt auch nur lesen wollen, parallel darauf zugreifen (**x**). Sobald ein Benutzer jedoch ein Objekt verändern möchte (INSERT, UPDATE, DELETE), kann kein anderer für die Dauer, die das Objekt gesperrt ist, darauf zugreifen (-). Versucht er es trotzdem, wird er in eine Warteschlange eingereiht bzw. erhält eine Rückmeldung. Damit nicht beliebig lange Wartezeiten auftreten, verfügen manche Datenbankverwaltungssysteme über einen timeout-Mechanismus.
Sperren werden innerhalb von Transaktionen implizit beim Änderungskommando oder explizit durch den Benutzer bzw. das Anwendungsprogramm angefordert. Freigegeben werden sie erst wieder bei Transaktionsende, um bei einem Zurücksetzen der Transaktion kaskadierte Rücksetzoperationen in anderen Transaktionen, die inzwischen auf die freigegebenen Objekte zugegriffen haben, zu vermeiden (Isolation).
In den DBVSen sind i. a. weitere Sperranforderungsmöglichkeiten realisiert, die dann per Anwendungsprogramm genutzt werden können.

4.2 Logging und Recovery

Die Atomaritäts-Eigenschaft einer Transaktion wird vom DBVS mittels eines *Logging* realisiert. Logging bedeutet, daß bestimmte Informationen in einem LOG-Protokoll (oder einfach Log) aufgezeichnet werden, um sie bei Bedarf zu benutzen. Für eine Transaktion gibt es zwei Arten des Logs:

- UNDO-Log
- REDO-LOG

Beim UNDO-Log wird der Zustand der Daten **vor** ihrer Änderung i. d. R. zeilenweise aufgezeichnet. Muß nun eine Transaktion zurückgesetzt werden, kann der alte Zustand der Daten vor der Transaktion mit Hilfe des UNDO-Log wiederhergestellt werden. Da das DBVS mit Pufferpooltechnik arbeitet, muß es das UNDO-Log vor der Datenänderung auf die permanenten Datenträger wegschreiben (write ahead log).
Das REDO-Log enthält den Zustand der Daten **nach** der Datenänderung. Mit ihm kann eine bereits abgeschlossene Transaktion, deren Änderungen noch nicht in der DB realisiert waren, nachgefahren werden.
Beim Hochfahren des DBS wird automatisch ermittelt, ob das letzte Abschalten der Datenbank ordnungsgemäß oder aufgrund eines Fehlers gewaltsam erfolgte. Im letzteren Fall wird beim Hochfahren ein *Recovery* durchgeführt, d. h. es werden die zum Zeitpunkt der Unterbrechung abgeschlossenen Transaktionen erneut zum Abschluß gebracht (REDO) und die offenen Transaktionen zurückgesetzt.
Ein besonderer Fehlerfall ist ein Plattendefekt. Üblicherweise schützt man sich gegen einen solchen Fehler mit regelmäßiger Datensicherung der Platte bzw. der wichtigen Daten. Zwischen zwei Komplettsicherungen, d. h. dem Abzug der ganzen Platte auf ein Sicherungsmedium) können mehrere incrementelle Sicherungen, das sind Sicherungen der Datenänderungen seit der letzten Komplettsicherung, durchgeführt werden. Um Datensicherungen zu starten, müssen i. d. R. alle Transaktionen abgeschlossen werden.
Falls ein Datenträger mit der gesamten oder Teilen der Datenbank einen Defekt hat, z. B. ein Headcrash der Platte, kann auf die letzte Sicherung zurückgegangen werden, indem diese von dem Sicherungsmedium eingespielt wird. Falls die letzte Sicherung ein oder mehrere Tage zurückliegt, sind jedoch viele Datenbankänderungen verloren. Um hier eine Situation zu erreichen, daß die zum Zeitpunkt des Datenträgerdefekts abgeschlossenen Transaktionen auf dem neuen Datenträger wiederhergestellt werden können, muß das o. g. REDO-Log für Transaktionen bis zur letzten Datensicherung zurückreichen und auf einem anderen Datenträger als dem zu sichernden aufgezeichnet sein.
Dann kann nach einem Plattenfehler der Zustand der Datenbank vor dem Fehler mit Hilfe der Datensicherung und dem REDO-Log bis auf die zum Zeitpunkt offenen Transaktionen wiederhergestellt werden (*Restauration*). Es sind also nur die nicht abgeschlossenen Transaktionen zu wiederholen.

4.3 Performance

Unter Performance versteht man bei Datenbank-Anwendungen die Geschwindigkeit der Abläufe. Hohe Performance bedeutet einerseits gute Antwortzeiten und andererseits einen hohen Durchsatz an Datenbank-Transaktionen. DBVSe werden nach Anzahl der Transaktionen pro Sekunde gemessen. Dabei handelt es sich um *OLTP-Anwendungen*

(Online Transaction Processing), wenn viele Lese- und Änderungstransaktionen parallel auf der Datenbank ablaufen. Leistungsfähige Computersysteme können heute mehrere hundert bis tausend Transaktionen pro Sekunde durchsetzen. Bei den *Decision Support Systemen* **(***DSS***)** dagegen werden fast nur Lesetransaktionen auf die Datenbank abgesetzt.
Es gibt inzwischen einige verbreitete Testprogramme, die die Transaktionsleistung eines DBVSs messen. Sie verhalten sich wie echte Anwendungen und werden *Benchmarks* genannt. Zu ihnen gehören

TP1	Nachahmung einer Kontenbuchung in einer Bankfiliale (Vorgänger: Debit-Credit-Benchmark).
TPC A	Client-Server-DB-Benchmark; verbesserte Version des TP1.
TPC B	wie TPC A, jedoch lokal.
TPC C	Plattformunabhängiger Benchmark, der das typische Lastscenario einer OLTP-Anwendung repräsentiert. Simuliert wird das Auftragsabwicklungssystem eines Großhandelsunternehmens mit regional verteilten Warenlagern sowie zugeordneten Vertriebsgebieten und Kunden. Fünf komplexe Transaktionstypen bilden die typischen Vorgänge Auftragseingang, Bezahlung, Auftragsstatus, Auslieferung und Lagerbestand nach. Zusätzlich werden max. Antwortzeiten und minimale Tabellengrößen vorgegeben.
TPC D	Simuliert die Eigenschaften von Entscheidungsunterstützenden Systemen (Data Warehouse bis 1 Gigabyte Speichergröße)
TPC W	Leistungstest für Web-Anwendungen.

TPC (Transaction Processing Performance Council) ist ein Gremium, das diese Standard-Benchmarks auf den Markt bringt.
Falls Anwenderprogramme nicht performant laufen, sind die Gründe häufig in der Struktur der Anwendung selbst zu suchen. Hier helfen Tuningmaßnahmen, die jedoch nur von Experten realisiert werden können.
Daneben haben auch andere, systemtechnische Eigenschaften einen Einfluß auf die Gesamt-Performance:

- Prozessorleistung (Taktrate, Multiprozessor)
- Systemarchitektur (symmetrisches Multiprocessing, massivparallele Systeme)
- Größe des Hauptspeichers
- Größe des Pufferpools (Plattencash)
- Platten (Anzahl, Zugriffszeit)

- Plattencontroller
- Netz (Topologie, Übertragungsraten)
- Einsatz von Stored Procedures
- Einsatz der Indexe

Ein Multiprozessorsystem wird im Fall eines Datenbank-Servers nur dann eine Performancesteigerung gegenüber einem Ein-Prozessorsystem bringen, wenn die Multiprozessor-Eigenschaft von dem Gespann Betriebssystem - DBVS ausgenutzt wird.
Ein großer Pufferpool bringt eine erhebliche Reduktion der Plattenzugriffe und damit eine große Durchsatzsteigerung.
Im Fall mehrerer Platten hat sich ein Disk-Striping als eine gute Alternative herausgestellt. Dabei werden sämtliche für das DBS genutzten Platten „streifenweise“ reihum beschrieben, um eine optimale Verteilung der Daten auf alle Platten zu erhalten.
Bei verteilten Anwendungen, z. B. bei *Client/Server*-Anwendungen, muß das Netzwerk schnell genug sein, um nicht der Performance-Engpaß zu sein. Entscheidend ist auch eine sinnvolle Schnittstelle zwischen Client und Server.
SQL-Abfragen müssen, falls sie nicht schon vorübersetzt sind, vom DBVS interpretiert werden. Um komplexe Abfragen effizient abzuarbeiten, z. B. um bei Joins geeignete Strategien für die Zusammenstellung der vorläufigen und endgültigen Ergebnismenge zu entwickeln, verfügen moderne DBVSe über *Query-Optimizer*.
In manchen Fällen ist eine *Denormalisierung* für eine ausreichende Performance erforderlich, um die Anzahl der Tabellen wieder zu reduzieren und damit die Joins zu eliminieren.

Nutzung eines Index:

Um den Nutzen eines Index zu verdeutlichen, soll folgender SELECT auf einer Datenbank analysiert werden:

```
SELECT      ART_NR, ART_BEZ
FROM        ARTIKEL
WHERE       ART_NR = 4711;
```

Falls die Daten in einfachen („flachen“) Dateien auf der Platte gespeichert sind, muß das Datenbanksystem sämtliche Zeilen aus der Tabelle ARTIKEL lesen und prüfen, ob ART_NR = 4711 ist. Dies kann bei großen Tabellen länger dauern.

Dazu ein Berechnungsbeispiel mit folgendem Mengenprofil:

- Artikel-Tabelle mit 80.000 Artikeln (Sätzen)
- Satzlänge: 100 Bytes
- ergibt ca: 8 MB Plattenplatz
- Plattenzugriffs-Blocklänge: 4 KB
- mittlere Zugriffszeit der Platte: 10 msec
- ergibt Anzahl Plattenzugriffe: 8 MB / 4 KB = 2.000

- Dauer dieser Zugriffe: 2.000 * 10 msec = 20 sec
- Mittlere Zugriffsdauer 20 sec/2 = 10 sec

10 Sekunden will kein Benutzer warten, bis der gesuchte Datensatz am Bildschirm angezeigt wird. Um die Plattenzugriffe deutlich zu reduzieren, verfügen DBVSe über Zugriffsbeschleunigungs-Verfahren. Das gebräuchlichste ist das *Index*-Verfahren auf der Basis eines B*-Baums. Dabei sind alle Datensätze sortiert und haben einen Primärschlüssel.

Verfahren zum Aufbau des B*-Baums:
Siehe Anhang: „Zugriffsbeschleunigung über Indexe“

dLege alle Datensätze sortiert in Plattenblöcken fester Länge (z. B. 4 KB) ab.
- Bilde aus den jeweils größten (bzgl. Alphabet, z. B. ANSI) Schlüsseln der Datensätze, die in einem Plattenblock gespeichert sind, eine Liste, wobei jedes Listenelement aus dem (Datensatz-) Schlüssel und dem Zeiger (Adresse) auf den betreffenden Plattenblock besteht.
- Wiederhole das Verfahren hinsichtlich der so entstandenen neuen Liste, falls diese länger als 1 Plattenblock ist, bis nur noch 1 Block übrig bleibt. Dadurch entsteht eine Hierarchie von Plattenblöcken, der B*-Baum, der eine bestimmte Höhe hat.
- Es sind ein- (unique) und mehrdeutige Schlüssel möglich.
- Einfügen und Löschen von Datensätzen hat Konsequenzen für den Indexbaum.

In dem obigen Beispiel wird die Höhe des Indexbaumes wie folgt berechnet:

Ebene 1	Datenblock-Ebene
Ebene 2	1. Indexblock-Ebene; falls Schlüssellänge = 6 Bytes und Plattenadresse = 4 Bytes (32-Bit-Adresse), hat ein Eintrag die Länge 10 Bytes; damit können 4 KB / 10 B = ca. 400 Einträge in einen Plattenblock aufgenommen werden. Bei 80.000 Sätzen werden 80000/400 = 200 Blöcke in dieser Ebene benötigt.
Ebene 3	Für die Einträge auf die 200 Blöcke der Ebene 2 reicht ein Block aus, da ja in 1 Block bis zu 400 Einträge passen.

Damit hat der B*-Baum die Höhe 3; d. h. bei einem Zugriff auf einen Artikel über die Artikel-Nummer muß zur Ermittlung des betreffenden Plattenblocks auf 3 Plattenblöcke zugegriffen werden. Da die obersten Baum-Ebenen meist im Pufferpool liegen (das wären 1 + 200 Blöcke zu je 4 KB, also ca. 800 KB Daten), ist i.d.R. für einen Zugriff der obigen Form nur 1 Plattenzugriff erforderlich!

5 Die Datenbank im Netz

Außer bei (immer weniger) Heim-PCs gibt es kaum noch Computer, die nicht vernetzt sind. Sie sind in ein Firmen – LAN (Local Area Network) eingebunden oder haben einen Zugang zu einem analogen oder digitalen Netz (ISDN) im WAN (Wide Area Network), um beispielsweise auf das Internet zuzugreifen. Im folgenden wird die Rolle der Datenbank in solchen Netzen betrachtet.

5.1 Client/Server – Architekturen

Client/Server – Architekturen verändern die EDV-Landschaft

Client/Server – Architekturen (C/S) haben seit Anfang der 90er Jahre die EDV-Landschaft erheblich verändert. Durch immer leistungsfähigere PCs ist die Präsentationsschicht und in vielen Fällen auch die Anwendung auf den PC, den Client, verlagert worden. Zusätzlich existiert ein Datenbankserver, der das DBS enthält. Häufig können noch 1 bis n Anwendungsserver die Architektur abrunden. Ein Beispiel für eine solche C/S – Architektur ist SAP R/3. Hier liegt die Präsentation auf dem Client (SAP-GUI), während Anwendung und Datenbank auf einem Server residieren oder auf mehreren Servern verteilt sein können. Greift dagegen ein Excel-Programm direkt auf die Datenbank zu (z. B. über ODBC), so befinden sich Präsentation und Anwendung auf dem Client, die Datenbank auf dem Datenbankserver.

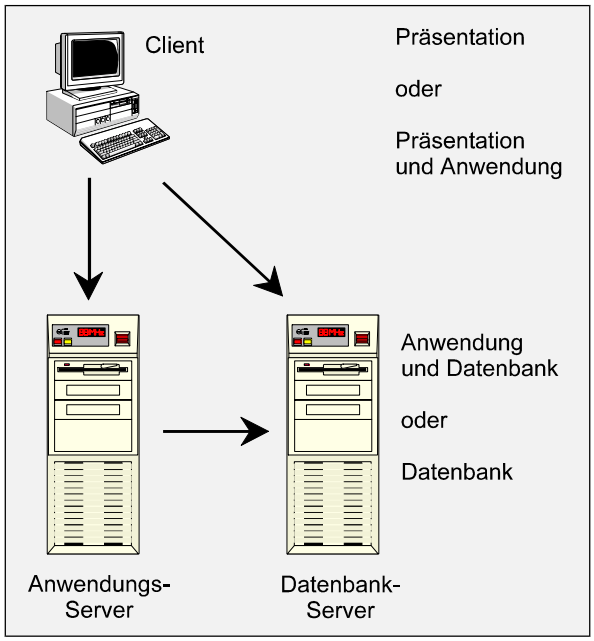


Abb. 5: Client/Server – Architektur

Gründe für eine solche C/S – Architektur sind u. a.

- Kostengünstige Hardware
- Entlastung des Hosts
- Unabhängigkeit der Fachabteilung
- Grafische Benutzeroberfläche
- Wunsch nach Einsatz von PC-Programmen

Inzwischen ist die C/S – Architektur stark verbreitet. Nun stellt man auch Nachteile dieser Architektur fest:

- Wildwuchs an Hardware und Software
- Schwierige Administration
- Benutzer haben oft Probleme und kommen in ihrer Anwendung nicht weiter, da sie die PCs nicht beherrschen

Um diese Probleme einigermaßen in den Griff zu bekommen, haben viele Unternehmen Benutzerservices eingerichtet, die die Benutzer mit Rat und Tat unterstützen. Mit Hilfe von Tools und Administrationssoftware (Helpdesk, Software-Verteilung, Lizenzmanagement, Datenbank- und Applikations-Management, ...) wird die C/S – Umgebung zentral unterstützt. Das verursacht jedoch erhebliche Kosten, die mit dem C/S – Ansatz ursprünglich aufgrund der billigeren Hardware eingespart werden sollten. Markforschungsinstitute (Gartner Group, IDC u. a.) haben errechnet, daß ein PC-Arbeitsplatz im Jahr bis zu 13.000 $ kostet, wobei die Hardware- und Software-Lizenzen daran nur einen geringen Anteil haben. Den größten Batzen verschlingt das Try and Error – Vorgehen zum Erlernen der Software, das viel Zeit kostet und die Produktivität des Anwenders reduziert.

5.2 Verteilte Datenbanken

Verteilte Datenbanken werden im wesentlichen aus zwei Gründen eingesetzt: aus Performancegründen und zur dezentralen Datenhaltung. Ein Beispiel für die dezentrale Datenhaltung sind Laptops von Außendienstmitarbeitern, deren Daten regelmäßig mit einem zentralen Datenbestand abgeglichen werden. Diesen Fall nennt man auch *Replikation* (der Daten).

5.2.1 Verteilte Datenbank aus Performancegründen

In einem LAN oder WAN kann eine logisch zusammengehörige Datenbank als verteilte Datenbank realisiert werden. Bei einem LAN kann so die Last eines Datenbankservers auf mehrere verteilt werden. Logisch zusammengehörig bedeutet, daß es sich aus der Sicht des Anwenders und des Anwendungsprogrammierers um **eine** Datenbank handelt, d. h. die Anwender sehen die Verteilung nicht und müssen sich auch nicht um die Probleme kümmern, die die Verteilung mit sich

bringt. Das DBVS sorgt im Betrieb dafür, daß die Anfragen und Änderungen der Anwender auf die richtigen Daten im Netz zugreifen.
Der DBA definiert die Regeln für die Verteilung der Daten. Dazu hat er wieder die Möglichkeiten von SQL zur Verfügung, wobei er die Verteilungsmengen im Prinzip wie Sichten definiert. Für diese wird dann festgelegt, in welchem Netzknoten sie gespeichert werden.

Beispiel:

> Das Lager einer Firma ist auf mehrere Standorte verteilt. Die zugehörige Lagerverwaltung basiert auf einer verteilten Datenbank. In der Firmenzentrale sind sämtliche Lagerdaten physisch gespeichert. In den einzelnen Lagerstandorten befindet sich eine Datenbank, die die Teilmenge der Lagerdaten zu den in diesem Standort befindlichen Lagerartikeln enthält.

In dem Beispiel haben wir redundante Teilkopien der Daten vorliegen. Bei einem Lesezugriff wird diejenige Datenbank genutzt, die dem Anwender am nächsten liegt. So arbeitet ein Lagerverwalter automatisch mit seiner Datenbank; das Netz und der zentrale Computer werden nicht belastet. Eine Änderung wird dagegen sofort in allen Kopien realisiert, falls das Netz funktionsfähig ist.
Wir haben es hier mit verteilten Transaktionen zu tun. Das bedeutet, daß eine Transaktion der Anwendung konkrete Auswirkungen auf mehreren Datenbankservern haben kann. Um die Forderungen des Transaktionskonzepts nach Konsistenzerhaltung und Atomarität bei einer verteilten Datenbank zu gewährleisten, wird ein spezielles Protokoll genutzt, das *Two Phase Commit* – Protokoll. Es funktioniert wie folgt:
Ein zentraler Transaktionsmanager als Softwarekomponente des DBVSs, der jeweils in demjenigen Datenbankserver agiert, von wo aus die Transaktion gestartet wird, kommuniziert mit den an der Transaktion beteiligten, dezentralen (aus der Sicht des zentralen) Transaktionsmanagern, um eine Transaktion durchzuführen und zu einem sauberen Abschluß zu bringen. Hier gehen wir davon aus, daß ein Anwender immer einem bestimmten Datenbankserver im Netz zugeordnet ist. Two Phase Commit bedeutet nun, daß bei Ende der Transaktion (Commit oder Rollback) der zentrale Transaktionsmanager diesen Endewunsch zunächst an alle dezentralen Transaktionsmanager übermittelt (erste Phase). Erst, wenn er von allen die Rückmeldung ok erhält, was bedeutet, daß das Netz funktioniert und alle dezentralen Transaktionsmanager ihre Vorbereitungen für das Transaktionsende (Schreiben des Log) getroffen haben, gibt er die Order, daß die Transaktion nun wirklich abgeschlossen werden kann (zweite Phase).
In dem Umfeld der verteilten Datenbanken existieren eine Reihe weiterer Konzepte, um das Problem, einerseits immer einen konsistenten Zustand zu gewährleisten, andererseits dem Anwender auch im Last- oder gar Störungsfall nicht zu große Anwortzeiten zuzumuten, zu lösen. Dazu gehören u. a.:

Primary Copy

Es existiert immer eine ausgezeichnete Kopie der Daten, die Primärkopie. Alle Änderungen werden zunächst in der Primärkopie realisiert. Sind sie dort erfolgreich, wird die Transaktion beendet. Die Änderungen der übrigen Kopien werden zu einem späteren Zeitpunkt nachgeholt. Der Datenbankserver mit der Primärkopie ist hier der kritische Kandidat; fällt er aus, sind keine Änderungen mehr möglich.

Optimistisches Sperren

Die Daten werden nicht schon vor der Änderung gesperrt, sondern erst bei Transaktionsende. Man geht also optimistisch davon aus, daß kein anderer die Daten gesperrt hat. Sind sie dennoch gesperrt, muß die Transaktion zurückgesetzt werden.

5.2.2 Replikation

Mit Lotus Notes ist die Datenverteilungstechnik der *Replikation* publik geworden. Auch hier existiert ein zentraler Datenbestand, der auf dezentralen Computern, meist PCs oder Laptops, repliziert (also kopiert) ist. Man geht nun davon aus, daß die dezentralen Computer nicht immer online sind, so daß eine sofortige Aktualisierung der Änderungen in allen beteiligten Datenbanken ausscheidet. Das Konzept sieht nun vor, daß dann, wenn ein dezentraler Computer eine Verbindung zur zentralen Datenbank herstellt, die Daten in beiden Richtungen, d. h. vom zentralen zum dezentralen und umgekehrt, aktualisiert werden. Falls dieselben Daten von beiden geändert worden sind, regelt eine Konfliktbehandlung die Situation, so daß am Ende ein konsistenter Zustand gegeben ist.

Beispiel:

Eine Firma hat ein Vertriebsinformationssystem realisiert. Ein zentraler Datenbankserver im Vertriebsinnendienst verwaltet alle Informationen über Kunden, Produkte, Preise, Angebote, Aufträge, Lagerbestand, Projekte, Mitbewerb usw. für den Außendienst. Jeder Außendienst-Mitarbeiter besitzt einen Laptop mit einer Anwendung für die Außendienstunterstützung. Diese basiert auf Lotus Notes, wobei die betreffende Datenbank einen Teil der Daten der Zentrale als Replikation enthält. Wenn der Außendienst-Mitarbeiter beim Kunden ist, hat er in seinem Laptop die (tages-) aktuellen Daten gespeichert und verfügbar.
Jeden Abend wählt er sich über sein Modem auf den Datenbankserver des Vertriebsinnendienstes ein. Dabei werden automatisch alle auf seinem Laptop gespeicherten Daten (Preise u. a.) aktualisiert. Ebenso werden die von ihm erfaßten Daten wie neue Auf-

träge, Projektinformationen, Besuchsberichte u. a. auf den zentralen Datenbankserver übertragen.

Inzwischen bieten viele Datenbankhersteller einen solchen Replikationsmechanismus an. Trotz der möglicherweise auftretenden Änderungskonflikte ist die Replikation für solche Vertriebsinformationssysteme gut geeignet.

5.3 ODBC

ODBC (Open Database Connectivity) hat sich als Marktstandard durchgesetzt. Im Client/Server-Umfeld wird ODBC häufig als Anwendungsschnittstelle für den herstellerunabhängigen Zugriff auf relationale Datenbanken und Dateisysteme genutzt. Der Einsatz von ODBC kann in bestimmten Fällen jedoch zu Performance-Problemen führen, da einerseits nicht alle Funktionen der Client-Anwendung von ODBC unterstützt werden und andererseits die ODBC-Zugriffe im Netzwerk stattfinden.

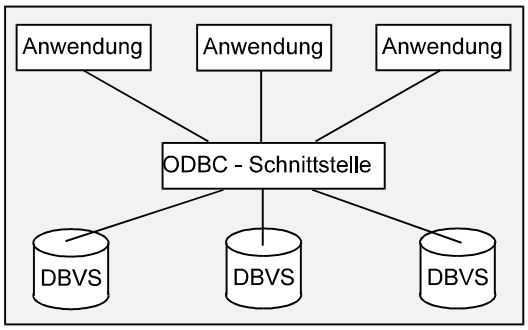


Abb. 6: Die ODBC-Schnittstelle

5.3.1 Was ist ODBC

ODBC ist ein ISO-Standard und weit verbreitet

ODBC wird dem Bereich „Middleware“ zugerechnet, einer Softwareschicht, die zwischen Anwendung und Datenhaltungssystem liegt. Mit Hilfe von ODBC greift ein Anwendungsprogramm über eine Standardschnittstelle auf relationale und nichtrelationale DBVSe zu. Die DBVSe können sich dabei auf PCs, Minicomputern oder Mainframes befinden. Der ODBC-Zugriff ist unabhängig von den jeweiligen Datenformaten und Programmierschnittstellen der Zielsysteme, so daß mit ODBC ein Standard-API (Application Programming Interface) zur Verfügung steht, das eine einheitliche Anwendungsprogrammierung für unterschiedliche DBVSe ermöglicht. ODBC basiert auf der CLI-Spezifikation der X/Open SQL Access Group und ist inzwischen ein ISO-Standard.
Eine Vielzahl von Windows-Anwendungen, darunter alle Microsoft-Programme und auch fast alle Datenhaltungssysteme, unterstützen den

ODBC-Standard auf der Client- oder der Server-Seite. Zu den Microsoft-Programmen als Clients gehören Excel, Word, Access, FoxPro, SQL Server, Visual Basic und Visual C++. Auf der Server-Seite sind bei den Datenhaltungssystemen vertreten: Oracle, INFORMIX, Sybase, OPENINGRES, DB2, SQL/DS, Gupta SQLBase, ADABAS-D, Teradata, Microsoft BackOffice, Lotus Notes, Btrieve, Clipper, dBase, Foxbase, FoxPro, Borland Interbase, Excel- und Textdateien u. a. In einigen Fällen wird die ODBC-Fähigkeit über Gateways erreicht. Als plattformübergreifende Lösung ist ODBC von seiner Konzeption her auf den Einsatz unter verschiedenen Betriebssystemen und für verschiedene Programmiersprachen ausgelegt, wobei die Portabilität der Anwendungen, die ODBC benutzen, ein vorrangiges Ziel ist. Als Betriebssysteme sind u. a. zu finden: Windows, Windows NT, OS/2, Macintosh und viele Unix-Varianten. Der ODBC-Ansatz hat sich also durchgesetzt und wird sich in der Zukunft weiterentwickeln.
Zusammengefaßt kann gesagt werden, daß ODBC Spezifikationen für die Programmierschnittstelle (API) und für die Syntax (SQL) festlegt.

5.3.2 Eigenschaften und Conformance Level

Zur Definition des ODBC-Funktionsumfangs wird sowohl die API-Schnittstelle als auch der SQL-Sprachumfang beschrieben. Zu beidem sind bestimmte Level definiert, die zusammengenommen den jeweiligen Conformance Level ausmachen.

ODBC mit verschiedenen Leveln

ODBC-API besteht zunächst aus einem Kernlevel, der Funktionen zum Verbindungsauf- und abbau zur Server-Datenbank, zur synchronen oder asynchronen Ausführung einer SQL-Anweisung, zur Beschreibung einer Ergebnismenge und zur Entgegennahme der Daten beinhaltet. Transaktionen werden ebenso unterstützt wie Long Binaries und ein Cursor-Konzept. Das ODBC-SQL-Sprachkonzept beinhaltet die deklarative referentielle Integrität und Mechanismen für den tabellenspezifischen Zugriffsschutz. Stored Procedures, Trigger und Autoinkrement-Felder werden nicht unterstützt.
Die Kernfunktionen werden durch die ODBC-Extensions ergänzt. Diese umfassen einerseits eine Reihe von Katalogfunktionen, die die dynamische Arbeitsweise mit einem Datenbanksystem unterstützen, indem Informationen über Tabellen, Zugriffsrechte, Indizes oder Stored Procedures abgefragt werden können. Andererseits enthalten die Extensions ODBC-Auskunftsfunktionen, mit denen die zielsystemspezifischen Eigenarten, insbesondere zum SQL-Sprachumfang, ermittelt werden.
Der SQL-Sprachumfang wird in drei Level eingeteilt, den Entry (Minimum), Intermediate (Core) und Full (Extended) Level.

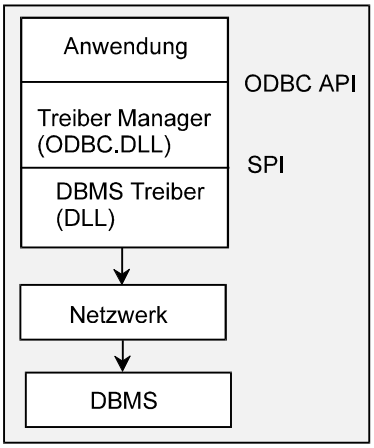


Abb. 7: Die Architektur von ODBC

5.3.3 Wann wird ODBC eingesetzt

Seit vielen Jahren hat sich der PC in den Fachabteilungen etabliert. Immer mehr Benutzer setzen auf ihrem PC nicht nur fertige Anwendungen ein. Sie wollen mit Programmen wie Tabellenkalkulationen, Textverarbeitungen oder PC-Datenbanken individuelle Applikationen erstellen oder von der DV-Abteilung erstellen lassen und einsetzen. Dabei handelt es sich um Ad hoc-Abfragen, Datenanalysen und -auswertungen, Standardberichte, Serienbriefe und ähnliches mehr. Diese meist einfachen Programme, die oft nur als Prototyp für eine oder wenige Personen erstellt werden, greifen häufig auf Daten zu, die nicht auf dem PC des jeweiligen Benutzers gespeichert sind. Vielmehr sind die Daten in Datenbanken oder Dateien eines PC-Netz-Servers, eines Minicomputers oder eines Mainframes zu finden – oft sogar auf mehreren dieser Datenspeicher oder in einem Data Warehouse. Die Daten liegen in verschiedenen Formaten vor und werden von unterschiedlichen Betriebssystemen und DBVSen verwaltet, was die Problematik erheblich verschärft.
Eine Möglichkeit, dem PC-Programm die gewünschten Daten zur Verfügung zu stellen, besteht darin, die erforderlichen Daten aus den verschiedenen Systemen zu exportieren und in eine PC-Datenbank zu importieren. Dies hat jedoch mehrere Nachteile. Zunächst sind die Daten nie ganz aktuell, da Änderungen ja erst wieder mit einem erneuten Export/Import in die PC-Datenbank übertragen werden müssen. Der Aufwand für den Export/Import dieser Änderungen ist erheblich, da im allgemeinen nicht mehr festgestellt werden kann, was sich tatsächlich geändert hat. Also müssen alle Daten – ob geändert oder nicht – immer wieder komplett in die PC-Datenbank transportiert werden. Hinzu kommt, daß ein Mischen der Daten aus verschiedenen Quellen ein erhebliches technisches Problem darstellt.
Der Anwendungsprogrammierer könnte das oben genannte Problem dadurch lösen, daß er eine Anwendung erstellt, die auf die verschiedenen Daten im Netz zugreift. Er müßte jedoch für dieselbe Abfrage jeweils eine Zugriffsvariante für die verschiedenen Zielsysteme erstellen, da die Programmierschnittstellen dieser Systeme in der Re-

gel nicht einheitlich sind. Von einer konsistenten Fehlerbehandlung oder einem Mischen der Daten in der Art eines Tabellenverbunds ganz zu schweigen.

ODBC: eine Schnittstelle für alle Datenspeicher im Netz

Mit ODBC erschließen sich nun ganz neue Möglichkeiten. Der Anwendungsentwickler kann nun *eine* Schnittstelle für *alle* Datenspeicher im Netz verwenden. Diese Interoperabilität bringt eine erhebliche Flexibilität, da er die ODBC-Schnittstelle für unterschiedliche Anwendungssysteme wie z. B. für Excel, Access oder andere benutzen kann. Damit reduziert sich die Vielfalt der heterogenen Systeme im Netz aus seiner Sicht erheblich, was sich in einer deutlich höheren Produktivität niederschlägt. Gerade Reportgeneratoren können von dieser Form des Zugriffs auf verschiedenartige Informationen profitieren.

Auch die Datenbankhersteller haben ein großes Interesse daran, die ODBC-Schnittstelle für ihre Produkte anzubieten. Neben der Tatsache, daß hier ein Standard vorliegt, der von den Herstellern erfüllt werden muß, um am Markt zu bestehen, wird von ihren Kunden zunehmend genau die Datenbank-Connectivity verlangt, die für die Lösung der oben genannten Problemstellung erforderlich ist.

ODBC für Anwendungsgeneratoren, Datenbankanalysen und Reports

Ein wichtiges Einsatzgebiet ist auch der Bereich der Anwendungsgeneratoren. Diese dürfen sich von vornherein nicht auf ein bestimmtes Datenbanksystem beschränken, um flexibel auf die konkreten Kundenwünsche reagieren zu können. Also müssen sie auf ODBC aufsetzen, um diese Flexibilität zu erlangen. Eine weitere Gruppe der Anwendungen, die sinnvollerweise mit ODBC arbeiten, sind die Datenbankanalyse- und reportingtools. Auch hier würde die Beschränkung auf ein bestimmtes Datenbanksystem den Käuferkreis stark einschränken. Als Beispiel sei DBExpert genannt, das neben dem direkten Zugriff auf DB2 auch eine ODBC-Schnittstelle zur Verfügung stellt.

Zusammenfassend noch einmal die Szenarien, bei denen der Einsatz von ODBC sinnvoll wird:

- einfacher Zugriff von verschiedenen PC-Anwendungen auf Daten, die auf Netz-Servern gespeichert sind
- Zugriff auf mehrere, verschiedene Datenbanken, die auf mehreren Servern installiert sind
- gleichzeitiger Zugriff auf Datenbanken und Dateisysteme
- Wunsch nach Verwendung einer Standard-Schnittstelle für den DB- und/oder den Dateizugriff, damit die Besonderheiten der Zielsysteme verborgen bleiben
- Zugriff auf aktuelle Daten ohne zeitraubenden und mühsamen Export/Import

Die ersten ODBC-Versionen besaßen noch erhebliche Sicherheits- und Performancemängel. Bei den neueren Versionen hat sich hier einiges getan, so daß ODBC heute durchaus eine echte Alternative in einer heterogenen Systemlandschaft ist. In Fachartikeln wird immer wieder auf Erfahrungen mit ODBC hingewiesen. Hier zusammenfassend ein kurzer Überblick:

Die CLI-Technologie (Common Language Interface) von ODBC hat sich verstärkt im PC-Umfeld durchgesetzt. ESQL (Embedded SQL) dagegen kommt eher aus dem Bereich der Mainframes. Dies liegt daran, daß CLI sich als DLL (Dynamic Link Library) besser in PC-Anwendungen integrieren läßt. ESQL dagegen verlangt Verfügbarkeit und Veränderbarkeit der Programmquellen und ihre Neuübersetzung. Bei einer Client/Server-Architektur mit ODBC arbeiten der Client und der Server nebenläufig, d. h. parallel zueinander. Dadurch bedingt ist die Performance im Netz (theoretisch) grundsätzlich besser als bei einer Netzwerkschnittstelle, die synchron auf Aufträge oder Antworten wartet.

5.3.4 Ein Beispiel für den ODBC-Einsatz

Im Fachbereich TBW der Märkischen Fachhochschule wird Oracle auf einer HP 9000 mit HP-UX eingesetzt. Über ODBC hat man von einigen der Windows NT – PCs einen Zugriff auf Oracle. Konkret werden Oracle-Tabellen in MS Access – Datenbanken eingebunden, damit Access-Anwendungen mit Oracle-Tabellen arbeiten können. ODBC setzt hier auf SQL*NET, einer Oracle-Netzkomponente auf Basis TCP/IP, auf.
Die ODBC-Schnittstelle wird auch genutzt, um von Data Mining – Anwendungen auf ein Data Warehouse zuzugreifen. Während Data Mining unter Windows NT läuft, ist das Data Warehouse mittels Oracle auf der HP realisiert.

5.4 Die Datenbank im Internet

ODBC und JDBC im Internet

Im Umfeld des WWW (World Wide Web) im Internet wird die objektorientierte Programmiersprache *Java* von JavaSoft, einer Tochter von Sun Microsystems, eingesetzt. Mit Java können Internet-Anwendungen, sogenannte Applets, programmiert werden. Diese kann der Internet-Benutzer wie HTML-Seiten (Hypertext Markup Language) aus dem Netz auf seinen lokalen PC runterladen und mit seinem Browser ausführen (genauer: interpretieren). Mit *JDBC* (Java Database Connectivity) wird ein API für den Zugriff der Applets auf lokale Datenbanken nach dem ODBC-Muster zur Verfügung gestellt. JDBC setzt meist wiederum auf ODBC auf.
Eine Alternative ist JSQL (Java SQL), eine ESQL – Variante für Java.

6 Markt- und Produktübersicht

6.1 Komponenten eines Datenbankverwaltungssystems

Normalerweise setzt ein DBVS auf einem Betriebssystem auf und nutzt dabei dessen Funktionen. Dabei kann das DBVS wie eine (systemnahe) Anwendung betrachtet werden. Falls es sich um einen Datenbankserver im Netz handelt, ist das DBVS i. d. R. die einzige Anwendung auf dem Computer. Im folgenden werden nur DBVSe betrachtet, die auf General Purpose Systemen mit Standard-Betriebssystemen laufen, also keine speziellen „Datenbank-Maschinen“. Ein DBVS besteht i. a. aus einem DBVS-Kern und einer Reihe von Tools. Der Kern enthält u. a. meist folgende Komponenten:

- Speicherungs- und Zugriffssystem mit physikalischer Platten-E/A, B*-Baumverwaltung und Pufferpoolverwaltung
- Netzwerkkomponente mit globalen und lokalen Funktionen
- Transaktions-Manager zur Verwaltung der Benutzertransaktionen
- Sperr-Manager zur Verwaltung der Sperrlisten
- Log-Manager zum Erzeugung des Logs (UNDO- und REDO-Log)

Als eigenständige Tools sind folgende Komponenten realisiert:

- Precompiler zum Vorübersetzen der Anwendungsprogramme mit Embedded SQL
- Query-Optimizer zur Optimierung der Abarbeitung der SQL-Kommandos
- Repository / Data Dictionary zur Verwaltung des Datenbank-Schemas
- Archivierungskomponente zur Datensicherung und -archivierung
- Endbenutzer-Tools zur Erstellung von Bildschirmmasken, und Druckreports, die in den Anwendungen genutzt werden können
- Interaktives SQL am Bildschirm (Kommando-Schnittstelle oder menügesteuertes, maskenorientiertes SQL)
- Sprache der 4. Generation (4GL) zur Entwicklung von Datenbank-Anwendungen
- Netzkomponente für Client/Server-Anwendungen oder für verteilte Datenbanken (z. B. SQL*NET bei ORACLE)

6.2 Produktübersicht

Viele DBVSe laufen nur auf einem bestimmten Betriebssystem eines bestimmten Herstellers. Andere sind auf fast allen bekannteren Be-

triebssystemen aller Rechnerklassen einsetzbar. Ein Beispiel hierfür ist ORACLE, das von MS-Windows bis OS/390, also vom PC bis zum Mainframe, verfügbar ist.
Relationale DBVSe unterstützen meist auch den Gedanken der offenen Systeme, indem sie sich an Standard-Schnittstellen halten (ANSI-SQL) und auf offenen (Betriebs-) Systemen zur Verfügung stehen.

Folgende Tabelle zeigt die wichtigsten relationalen Datenbanksysteme:

DBVS	Hersteller
DB2, DB2/6000	IBM (= IMS + relationales Datenmodell)
ORACLE	ORACLE
OPENINGRES	Computer Associates (CA)
INFORMIX	INFORMIX
ADABAS	Software AG
SQL Server	Microsoft
DBASE	Borland (früher Ashton Tate)
SYBASE	Sybase, Inc (Microsoft SQL-Server)
SQL-Windows	Gupta
Access	Microsoft
Foxpro	Microsoft
Paradox	Corel (ehemals Borland)

Tab. 8: Hersteller von relationalen DBVSen

Die Differenzierungen liegen unter anderem in den Bereichen

- welche Hardware-Plattform steht zur Verfügung
- welche Betriebssyteme werden unterstützt
- Qualität der Sicherheits- und Stabilitätsmechanismen
- welche Netzwerke können verwendet werden
- Verteilungsformen (Verteilte DB, Client-Server, Multi-Server)
- Integration in heterogene DBS-Netze
- Performance im OLTP (Anzahl Transaktionen pro Sekunde)
- Nähe zum Standard (ANSI-SQL)
- Spracheinbettung für welche Sprachen
- Funktionalität der Endbenutzer-Tools und der 4GL
- Kompilierbare Datenbankprozeduren (für mehr Performance von Anwendungsprogrammen)
- dynamisches SQL vorhanden
- welche Größenordnungen sind möglich (max. Anzahl Benutzer, max. Anzahl Tabellen usw.)
- Unterstützung von Multimedia-Datenbanken

- Integrationsmöglichkeiten zu Standard-Anwendungen, insbesondere im PC-Bereich, bzw. eigene Tools
- Verfügbare Anwendungen, die mit diesem DBVS arbeiten

Darüber hinaus spielen der Preis, die Wirtschaftskraft und die Supportleistung des Herstellers, sowie die Qualität von Broschüren, Dokumentationen, Schulungen und Installationstools ebenfalls eine wichtige Rolle.

6.3 Auswahlkriterien für ein DBVS

Betrachten wir folgendes Scenario: Ein Unternehmen setzt EDV bisher in kaufmännischen Bereichen ein, z. B. in der Auftragsverwaltung, der Fakturierung, der Kundenverwaltung und für das Produktions-Planungssystem (PPS). Zusätzlich existieren einzelne Lösungen als „Insellösungen“ in

- der Unternehmensplanung,
- der Kalkulation,
- der Dokumentation und Werbung,
- der Entwicklung,
- evtl. der Fertigung,
- der Lagerverwaltung.

Einige dieser Anwendungen haben zu anderen eine Datenschnittstelle auf der Basis von ASCII-Dateien, um Daten auf Anforderung auszutauschen. Der Austausch findet über ein Rechnernetz statt.
Während Datenbankverwaltungssysteme bei größeren Unternehmen bereits eine Selbstverständlichkeit sind, sind sie bei mittleren und kleineren noch nicht so verbreitet. Das liegt im wesentlichen daran, daß kleinere Unternehmen sich vorrangig für die Problemlösung, d. h. für die Anwendung interessieren, während die unterlagerte Technologie wenig beachtet wird. Ob die Anwendung mit oder ohne bzw. mit welchem DBVS arbeitet, ist nicht von Bedeutung. Ferner werden bei kleineren Unternehmen i. a. keine großen Programme erstellt, so daß das Argument „Produktivere Programmentwicklung bei Einsatz eines DBVS“ nicht zum Tragen kommt.
Inzwischen ist ein klarer Trend in Richtung Integration unterschiedlicher Anwendungen auch bei kleineren Unternehmen zu beobachten. Z. B. soll das PPS Schnittstellen zum CAD (Stücklisten) und zur Fertigungssteuerung (Feinplanung, BDE-Rückmeldung) haben.
Häufig werden die Interaktionen zwischen solchen Programmen jedoch immer stärker und dynamischer genutzt. Dann reicht eine ASCII-Schnittstelle nicht mehr aus. Hier hat sich der Einsatz einer (Unternehmens-) Datenbank als dem zentralen Integrationselement bewährt. Die unterschiedlichsten Anwendungen werden dann um die Datenbank herum implementiert und greifen auf diese zu. Die Daten werden nicht

mehr zwischen den Anwendungen transportiert, sondern die Anwendungen arbeiten direkt mit der zentralen Datenbasis.

Nimmt man noch die anderen offensichtlichen Vorteile der Datenbank-Technik wie:

- Datenunabhängigkeit (SQL, Views, Trennung in Schemata),
- Offenheit, Standards (SQL),
- Interaktiver Zugriff (Ad hoc – Queries) auf die Daten,
- Unterstützung der Erhaltung der Konsistenz (Transaktionskonzept),
- Höchstmögliche Benutzerparallelität (Sperrverfahren),
- Integrität (referentielle, benutzerdefinierte),
- Sicherheit (ausgefeilte Zugriffsrechte und Views, Recovery und Restauration) und
- Anpaßbarkeit an die Unternehmensorganisation durch Client/Server-Architekturen oder verteilte Datenbank

hinzu, so wird eine Entscheidung für ein Datenbanksystem, trotz der damit verbundenen höheren Aufwände in der Hardware (Performance) und der Ausbildung, eine gute Investition für die Zukunft sein.
Wenn die oben genannten Anforderungen aus Sicht der Anwendungen gestellt werden, so sollte eine Entscheidung für ein DBS getroffen werden. Abhängig von den konkreten Anforderungen müssen Kriterien für die Auswahl eines bestimmten DBVSs erarbeitet werden. Dabei ist i. a. eine Entscheidung für ein relationales Datenmodell und damit eine relationale Datenbank meist richtig, da hier die o.g. Vorteile am ehesten garantiert sind. Die o. g. Differenzierungsmerkmale der verschiedenen DBVSe sollten anhand der Auswahlkriterien beleuchtet und getestet werden, um das richtige DBVS auszuwählen.

6.4 Einführung und Betrieb eines DBVS

Bei der Einführung eines DBSs sind viele Schritte erforderlich:

- Ausbildung von Mitarbeitern zum Thema Datenbank
- Informations-Bedarfs – Analyse
- Analyse der Anforderungen an ein DBS
- Auswahl eines DBVSs, evtl. zusätzliche Hardware und ein Computer-Netzwerk
- Entwurf der Datenbank mit Erstellung eines Datenmodells; dazu Nutzung von Methoden und Tools
- Installation der Datenbank, evtl. auch der Zusatz-HW und des Netzes
- Anlegen der Tabellen; die betreffenden CREATE-TABLE – Kommandos sollten in Scripts in Form von ASCII-Dateien abgelegt sein, damit sie jederzeit wiederholbar sind
- Erstellung und Test eines Prototypen

- Laden bzw. Erfassen der Daten
- Testen der Performance mit Benchmarks
- Definition von Benutzern
- Probebetrieb und schließlich Aufnahme des Normalbetriebs

Nachdem die Entscheidung für das DBVS gefallen ist, können neue Anwendungen beschafft bzw. entwickelt oder vorhandene Anwendungen umgestellt werden. Bei letzterem können erhebliche Aufwände anfallen, so daß hier eine schrittweise Vorgehensweise ratsam ist.
Es sollte zunächst geprüft werden, welche Anwendungen mittel- und langfristig nach dem alten Verfahren laufen können, d. h. z. B. keine dynamische Integration zu anderen Anwendungen benötigen. Diese müssen nicht oder erst in einer zweiten Phase umgestellt werden.
Sinnvoll ist auch - zumindest für eine Übergangszeit - der Parallelbetrieb von konventioneller und neuer Technik. Dabei muß die Konsistenz der Daten wegen eventueller Redundanzen besonders beachtet werden.
Der Datenbank-Betrieb umfaßt neben der Weiterentwicklung (Erweiterung des Schemas, neue Anwendungen bzw. Integration vorhandener sowie neue, leistungsfähigere Hardware u. a.) im wesentlichen

- eine regelmäßige Datensicherung
- die Unterstützung von interaktiven Benutzern (Benutzerservice)
- Tuningmaßnahmen bei Leistungsengpässen
- den Problem-Support, d. h. die Behebung von Problemen aller Art

6.5 Multimedia - Datenbank

Multimedia – Datenbanken, d. h. Datenbanken mit der Möglichkeit, multimediale Informationen wie große Texte, Grafiken, Bilder, Videos und Ton zu speichern, werden zunehmend vom Markt verlangt. Man geht davon aus, daß in einigen Jahren der Anteil der Multimedia-Datenbanken an allen installierten Datenbanken 80% und mehr ausmachen wird.
Seit einiger Zeit bieten praktisch alle bekannten DBVSe Unterstützungen für solche Datentypen. Intern werden die Daten meist in sogen. BLOBs (Binary Large Objects) gespeichert. Eine Alternative dazu ist die Einbindung anstelle der Integration von Multimedia-Objekten, d. h. es wird nur eine Referenz auf das Objekt (bzw. auf die betreffende Datei) in der Datenbank gespeichert, anstatt das Objekt selbst in der Datenbank zu speichern.

7 Das Data Warehouse

Das Data Warehouse (DWH) ist ein neuer Begriff. Er bezeichnet ein Anfang der 90er Jahre entstandenes Konzept zur Unterstützung der Entscheidungsträger im Unternehmen durch fundierte Informationen über alle Geschäftsvorgänge. Das Data Warehouse stellt damit einen neuen Ansatz für ein wirkungsvolles Management Information System (MIS) dar, indem es die in den betriebsunterstützenden DV-Verfahren enthaltenen Daten dem Management als Faktenwissen zugänglich macht und damit eine fundierte Entscheidungsgrundlage liefert.
Der Grundgedanke des Data Warehouse ist, eine von allen operationalen Datenbanken des Unternehmens getrennte Faktensammlung zu führen, die einerseits die aktuellen Daten – teilweise in verdichteter Form – enthält und darüber hinaus auch deren zeitliche Entwicklung („Historie“). Hinzu kommt ein für alle erdenklichen ad-hoc-Anfragen geeignetes Abfragesystem, das gewünschte Auskünfte in einer auf das Management zugeschnittenen Form und Terminologie erstellt und das geeignet ist, von dieser Personengruppe selbst bedient zu werden.

7.1 Die Vorgänger

Die Idee des MIS kam schon sehr früh auf, nämlich Ende der 60er Jahre. Das MIS ist älter als das Datenbanksystem. Der Wunsch nach einem MIS hat die Entstehung der Datenbankidee mit ausgelöst. Den frühen MIS war jedoch kein Erfolg beschieden. Unter anderem überforderten die zu speichernden Datenmengen die technischen Möglichkeiten, und die damaligen Realisierungen der Rechercheprogramme (Information Retrieval Systeme) waren noch zu ineffizient. Sie führten auch bei Verwendung der teuersten Großrechner zu unakzeptabel langen Programmlaufzeiten, denn die Prozessorleistungen waren - im Vergleich zu heute - mehr als bescheiden.
Etliche Zeit später gaben die gestiegenen Fähigkeiten der Rechner, zusammen mit den aufkommenden Datenbank-Konzepten und dem zum Allgemeingut gewordenen Online-Zugang zum DV-System dem Gedanken des MIS neuen Auftrieb. Über längere Zeit war das Enterprise Information System (EIS) eines der beherrschenden Themen. Aber auch den vielen ElS-Lösungen, die diskutiert und angepriesen wurden, war kein Durchbruch in der Praxis beschert. Nur vereinzelt blieb ein solches EIS im Einsatz und war erfolgreich, d. h. dem Unternehmen nützlich und hatte eine positive Kosten/Nutzen-Bilanz. Das Thema EIS verschwand Anfang der 90er Jahre wieder aus den Schlagzeilen. Gründe für das Verschwinden waren u. a.:

- Man sah, daß die in den betrieblichen Datenbanken gespeicherten „operativen Daten“ einen zeitlich zu engen Ausschnitt darstellten, keine Historie der Geschäftsentwicklung enthielten und zur Beantwortung vieler Fragen des Managements, z. B. Trendverfolgungen, nicht ausreichten.
- Sie waren für die einzelnen Geschäftsfelder in getrennten Datenbanken (oft auch unterschiedlichen Typs) abgelegt, nicht einheitlich dargestellt und somit nicht über einen einheitlichen Zugriffsmechanismus erreichbar.
- Sie hatten keine gemeinsame Terminologie und die enthaltenen Daten waren nicht oder nur zufälligerweise wirklich konsistent.
- Weiterhin wirkte es sich aus, daß eine ausreichende Kenntnis zur Benutzung der Datenbank-Abfragesprachen beim Management nicht vorhanden war und auch keine Akzeptanz zur Einarbeitung in diese (aus Sicht des Managers viel zu spezielle) DV-Materie zu erreichen war. Auch Hilfslösungen wie z. B. die Vorformulierung von viel gebrauchten Abfragen und die Routine-Erstellung von Reports brachten nur gelegentlich Erfolg, denn das komplexer werdende Geschäftsgeschehen war zunehmend durch starke Veränderungen in kürzer werdenden Zeitabständen gekennzeichnet. Genützt hätten vor allem schnelle Antworten auf ad hoc formulierte Fragen statt regelmäßiger Reports auf Standardfragen.

7.2 Das Konzept

Es wurde nach Konzepten gesucht, die solche Mängel nicht haben. Im Zuge dieser Suche wurde das Data Warehouse entwickelt. Man datiert es auf 1992 und schreibt es Bill Inmon zu. Er setzte mit seinem Buch „Building a Data Warehouse“ [Inm96] einen Meilenstein. Die Definition lautet:

> „A Data Warehouse is a subject-oriented, integrated, non-volatile, time-variant collection of data organized to support management needs.“

Das DWH ist also eine themenorientierte, nicht nach Datenquellen sortierte Datensammlung; die Daten haben eine unternehmensweit einheitliche Darstellung; sie werden dauerhaft gespeichert, und schon Gespeichertes wird weder verändert noch gelöscht; es wird der ganze zeitliche Verlauf der gespeicherten Geschäftsgrößen und -werte abgelegt; die Organisation der Datenspeicherung und der Abfrage wird gezielt auf die Bedürfnisse des Managements ausgerichtet.
Der Kern des DWH ist eine eigene Datenbank. Es wird eine relationale oder eine *multidimensionale* Datenbank verwendet. Die Datenbank enthält die aus unterschiedlichsten Quellen entnommenen, aufbereiteten und konsistent gemachten Daten in einheitlicher Form. Dazu gehört auch ein Repository, das u. a. Informationen über die Daten des DWH, sowie Regeln, nach denen das DWH aufgebaut ist und arbeitet,

enthält. Das DWH arbeitet also im Gegensatz zu den früheren MISs und EISs nicht mit den „operationalen Daten“, welche in den diversen Datenbanken der operativen IV-Systeme des Unternehmens liegen. Der Datenbestand des DWH beinhaltet sowohl Daten aus internen Quellen, d. h. Daten aus den Datenbanken der operationalen IV-Systeme des Unternehmens, als auch Daten aus externen Quellen, z. B. Marktdaten aus Erhebungen anderer Institute, individuelle Daten einzelner Mitarbeiter, Daten aus allgemein zugänglichen Online-Datenbanken im Internet u.v.a.m. In der Praxis stammen typischerweise mindestens 90% der DWH-Daten aus internen Quellen und nur ein kleiner Anteil aus externen. In gewissen Abständen, etwa jede Nacht oder am Wochenende, wird der Datenbestand des DWH mit aktuellen internen und externen Daten ergänzt.
Folgende Abbildung zeigt die Struktur eines Data Warehouse:

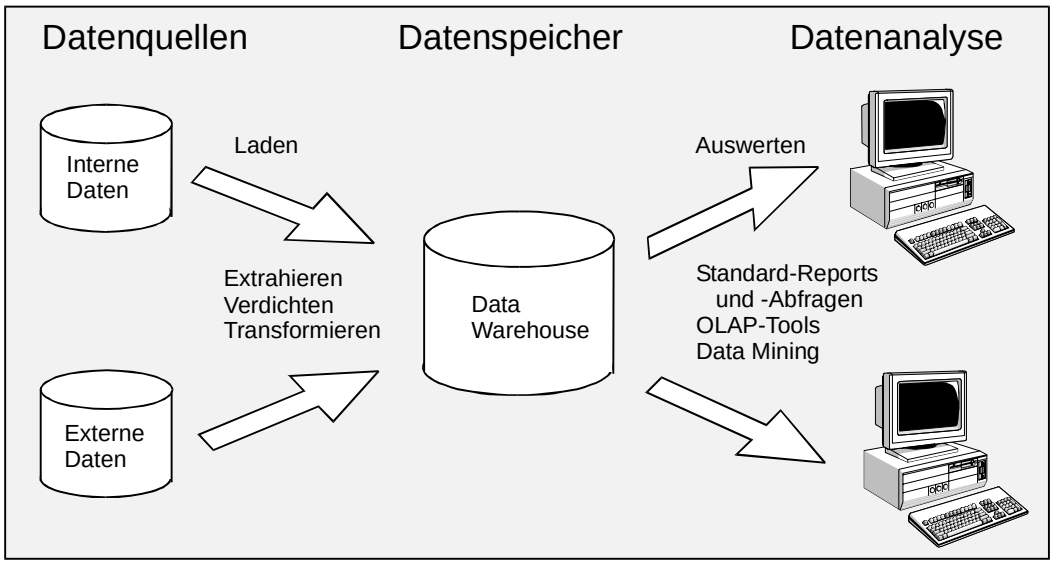


Abb. 8: Struktur eines Data Warehouse

Das im DWH verwendete DBVS kann eines der marktüblichen sein, es kann aber auch ein individuelles DBVS des DWH-Herstellers sein, was auf diese Aufgabenstellung spezialisiert ist, z. B. eine multidimensionale Datenbank. Sowohl für das Management der Datenbank wie auch zur Datenextraktion werden typischerweise rechnergestützte Werkzeuge eingesetzt.
Zur Befriedigung der typischen Fragestellungen an ein Data Warehouse wird eine Methode verwendet, die auf E.F. Codd (den Vater der relationalen Datenbanken) zurückgeht: „*Online Analytical Processing*“ (OLAP). OLAP wurde von Codd in Form von 12 Regeln beschrieben. Die Daten werden dabei in einer Art mehrdimensionalem Cubus dargestellt. Die „Achsen“ des Cubusses („Dimensionen“ genannt) sind unternehmensrelevante Größen wie z. B. Zeitverlauf, Produktspektrum, Absatzregionen. Um die Struktur der Daten zu beschreiben, hat sich ein bestimmtes Datenmodell herausgebildet, bei denen die Daten in einer Würfelform strukturiert sind: Das *multidimensionale Datenmodell*. Auf dem Markt sind DBVSe verfügbar, die

das multidimensionale Datenmodell direkt unterstützen, z. B. Oracle Express.
Für die Datenanalyse können diverse Tools eingesetzt werden. Darunter fallen Tools für

- Ad hoc-Auswertungen
- Reports
- Spreadsheets
- OLAP-Analyse-Tools
- Statistische Analysen
- Data Mining
- Datenvisualisierung

Dabei kann der Mitarbeiter Daten prüfen, durch die Informationen navigieren, Trends feststellen, Abweichungen erkennen, Ursachen erforschen und Prognosen erstellen.
Zu einem DWH gehören *Metadaten*. Damit bezeichnet man einen Informationskatalog, in dem der Benutzer DV-technische Informationen sowie betriebswirtschaftliche Erläuterungen zu den DWH-Daten findet. Metadaten-Tools werden i. a. mit einer grafischen Oberfläche angeboten.
Eine weitere Komponente des DWH ist die Archivierung. Historische Daten lassen sich nicht mehr aus den operationalen Systemen0 herleiten, so daß diese Daten für den Fall eines Datenverlustes archiviert werden müssen – bei den riesigen Datenmengen keine Kleinigkeit.

7.3 Technik und Kosten

7.3.1 Struktur

Wie bereits erwähnt, ist eine Würfelform, also ein multidimensionales Datenmodell, besonders für die Strukturierung eines Data Warehouse geeignet. Will man diese Struktur auf der Basis einer relationalen Datenbank realisieren, sind die Regeln, wie sie für OLTP-Datenbanken aufgestellt wurden (ERM, Normalisierung), praktisch nicht mehr gültig. Das liegt daran, daß wir es hier mit einer Datenbank zu tun haben, die nicht interaktiv geändert wird, die allerdings möglichst schnell umfangreiche und verdichtete Daten liefern soll. Gerade die Normalisierung hatte ja den Zweck, unerwünschte Nebeneffekte bei den Datenänderungen zu vermeiden.

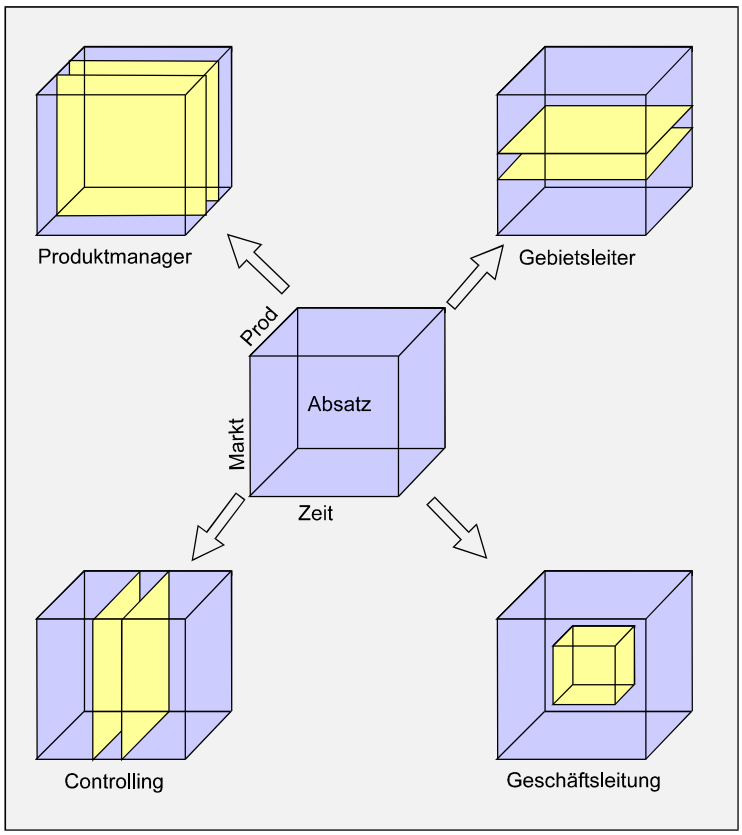


Abb. 9: Das multidimensionale Datenmodell

Das multidimensionale Datenmodell bietet den Nutzern unterschiedliche Sichten der Daten. Für die Abbildung des multidimensionalen Datenmodells auf eine relationale Datenbank haben sich im Prinzip zwei Techniken mit Untervarianten herausgebildet:

- das Starschema und
- das Snowflakeschema

Das *Starschema* hat seinen Namen von einer sternförmigen Struktur der Tabellen. In der Mitte des Sterns befindet sich die meist riesige *Faktentabelle*. Sie enthält einerseits Fakten, also z. B. die Daten zu Umsätzen und Stückzahlen einschließlich die verdichteten Daten wie etwa die Monatsumsätze, andererseits Fremdschlüssel aus den Dimensionstabellen. Die *Dimensionen* bilden die Kanten des Würfels, d. h. sie enthalten die Daten zu den Zeitachsen, den Produkten oder den Regionen. Damit kann ein solches Starschema wie folgt aussehen:

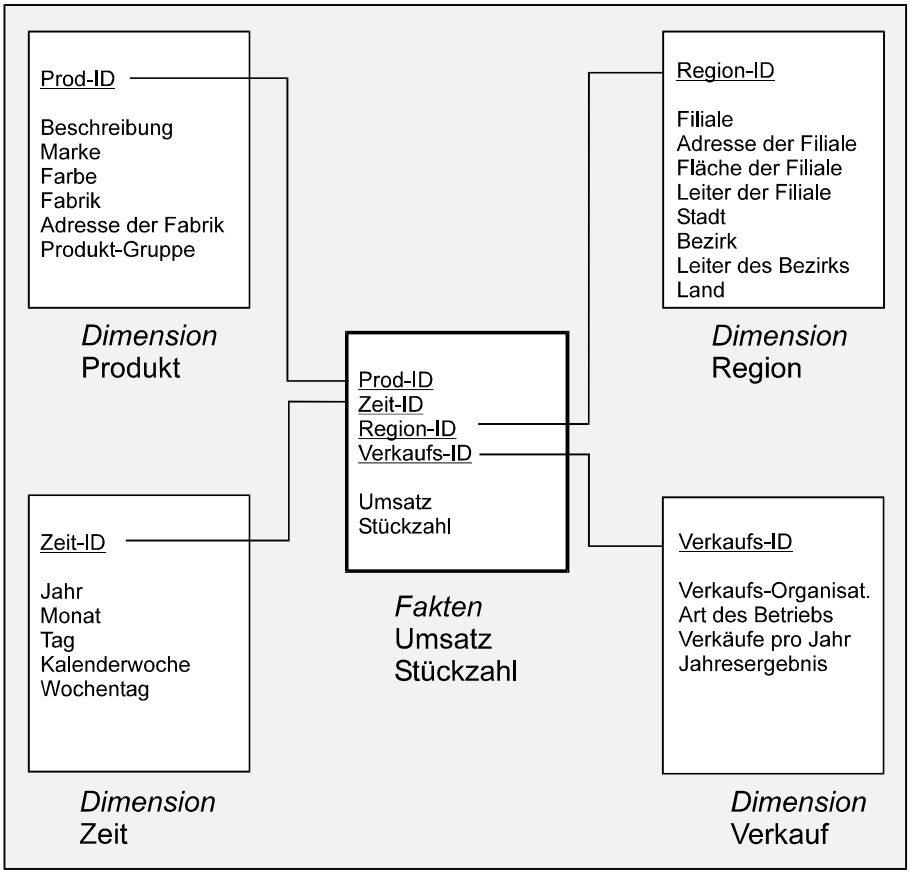


Abb. 10: Das Starschema

Einige Mängel des Starschemas werden durch Weiterentwicklungen eliminiert. Beispielsweise ist dem Benutzer nicht immer klar, zu welcher Hierarchieebene das betrachtete Element gehört. Dazu kann ein Ebenen-Indikator (Level) pro Dimensionstabelle eingeführt werden. Wenn sich die Dimensionen der verschiedenen Fakten zu stark unterscheiden, ist es sinnvoll, eine weitere Faktentabelle einzuführen.
Das Snowflakeschema ist eine Weiterentwicklung des Starschemas dahingehend, daß die Dimensionen normalisiert werden. Betrachten wir dazu die Dimension „Region“ in dem obigen Beispiel. Die Filialenattribute sind abhängig voneinander, also ein Verstoß gegen die 3. Normalform. Die Lösung ist eine Auslagerung der Filialendaten Filiale (= Filialenname), Adresse, Fläche und Leiter in eine eigene Tabelle Filiale, wobei in der Regionstabelle ein Fremdschlüssel Filialen-ID verbleibt. Das gleiche gilt für die Bezirksattribute.
Die Realisierung eines multidimensionalen Datenmodells mittels einer relationalen Datenbank wird auch als ROLAP (Relational OLAP) bezeichnet.

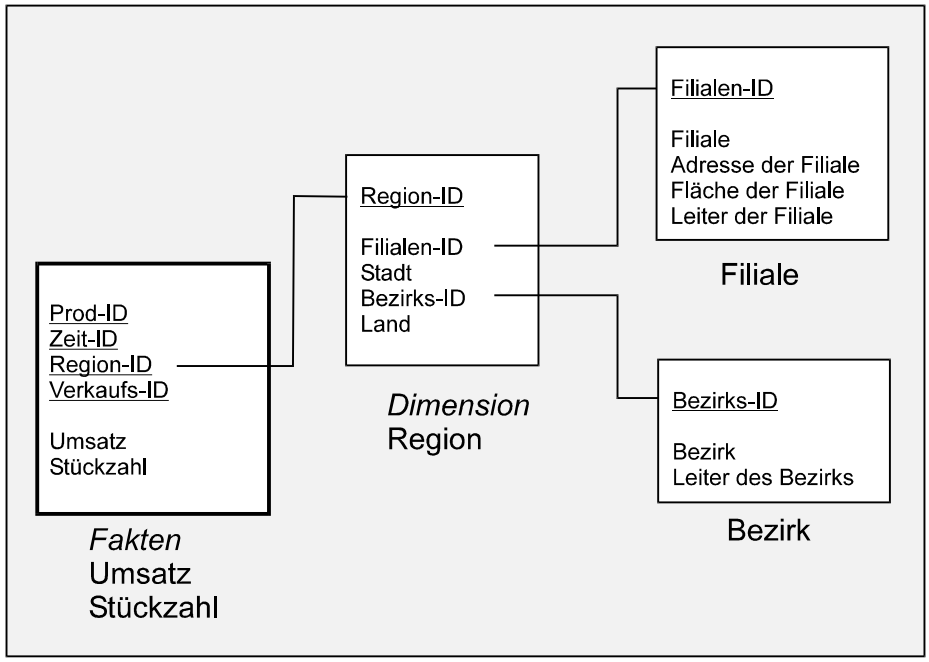


Abb. 11: Das Snowflakeschema

7.3.2 Laden der Daten

Sowohl das erstmalige Laden des DWH als auch das regelmäßige Aktualisieren wird „Warehouse Loading“ genannt. Warehouse Loading ist ein im Batch-Modus ablaufender Vorgang. Das Extrahieren und Einbringen der Daten aus verschiedenen Quellen in das DWH, sie konsistent zu machen, sie ggf. zu aggregieren/verdichten („Data reduction“), in das Zieldatenformat zu transformieren und über alle Quellen hinweg logisch zu verknüpfen heißt „Data extraction“ (Datenextraktion).

7.3.3 Realisierung

Als Kriterium für die Größenordnung eines DWH kann grob der benötigte (Platten-) Speicherplatz genommen werden. Ganz kleine DWH kommen schon mit 1 bis 3 GBytes aus; diese Größe hat nach eigenen Angaben das hausinterne DWH der Firma Informix. Zu den großen DWHs gehört das des USA-Telefon-Konzerns MCI, das 1,5 TeraBytes umfaßt. Daß ein DWH deutlich größeren Speicherbedarf hat als eine operational eingesetzte Datenbank, hat seinen Grund u. a. darin, daß das DWH die historische Entwicklung speichert, während die operationale Datenbank nur den aktuellen Datenbestand beinhaltet. Eine typische Größe ist schlecht auszumachen, aber Marktumfrageergebnisse großer Institute ergaben (Stand Anfang 1996) eine durchschnittliche Größenordnung von ca. 60 GBytes. Man beachte, daß dies ein Mittelwert über alle DWHs ist. Differenziert man danach, ob das DWH auf

einer Mainframe-Plattform realisiert ist, oder einer anderen (u. a. ist natürlich Unix eine starke Alternative), dann ergibt sich (ebenfalls Stand Anfang 1996) die erstaunliche Tatsache, daß die mainframe-basierten DWHs typischerweise bei 50 GBytes liegen, während die unix-basierten typischerweise eine Größe von 150 GBytes haben. Ganz offensichtlich realisiert man größere DWH mehrheitlich auf Unix-Plattformen und nicht auf Mainframe-Plattformen.
Es hat sich auch eine Kleinform des DWH herausgebildet, der *Data Mart* (engl. mart = Markt), als kleine Alternative zum Warehouse, dem großen Datenlager. Dem Data Mart liegt dieselbe Konzeption zugrunde wie dem DWH, aber sein Wirkungsbereich beschränkt sich auf einen begrenzten Teil des Unternehmens, z. B. auf eine Fachabteilung. Der Data Mart ist dementsprechend auch nicht auf die volle Sicht aller Unternehmensaspekte, sondern nur auf die für die Abteilung relevanten Teilaspekte ausgelegt. Dementsprechend benötigt ein Data Mart natürlich auch eine wesentlich kleinere Datenbasis und kommt mit einer kleineren Plattform aus.
Ein wichtiges Kriterium bei großen DWHs ist die Rechnerpower für gute Perfomance. Hier werden häufig neuere Technologien eingesetzt wie Symmetric Multi Processing (SMP) oder Massively Parallel Processing (MPP). Dadurch lassen sich viele Aktivitäten parallelisieren und damit beschleunigen, z. B. durch

- parallele Abfragen
- paralleles Laden und Indizieren der Daten
- parallele Join-Bildung
- parallele Sortierung
- parallele Aggregierung (Summe, Minimum, Maximum, ...)
- Partitionierung von Tabellen auf mehrere Platten bzw. Rechner
- gute Skalierbarkeit für größeren Ausbau

7.3.4 Einführungskosten

Das DWH ist ein ehrgeiziges Ziel und als solches nicht mit kleinen Kosten behaftet. Eine Untergrenze für die zu tätigende Investition liegt bei etwa 0,1 Mio DM (40 TDM für die DWH-Software, 60 TDM insgesamt für HW, System-SW, Datenbanksystem und Einführungsaufwand); hinzu kommen die jährlichen Betriebskosten. Ein solches DWH bedient dann eine Handvoll Abfrageterminals. Bezogen auf eine mittlere Unternehmensgröße handelt es sich bei einer solchen Installation aber noch nicht um ein DWH, sondern eher um einen Data Mart. Plant man für ein mittleres Unternehmen (2000 bis 3000 Mitarbeiter) ein DWH, dann ist die Gesamtinvestition höher: 2 bis 3 Mio DM sind in den ersten 2 Jahren aufzuwenden; einschließlich der Betriebskosten rechnet man mit ca. 5 Mio DM Gesamtaufwand. Ein solches DWH bedient dann 80 bis 100 Abfrageterminals. Der Datenbestand übersteigt nach mehreren Betriebsjahren ein TeraByte.

7.4 Anwendung des Data Warehouse

Die Benutzer des DWH sind die Entscheidungsträger aus der Geschäftsführung, dem Controlling, dem Marketing, dem Vertrieb usw. Sie benötigen die gewünschten Informationen in einer auf ihre Arbeit zugeschnittenen Terminologie. Sie haben typischerweise keine Kenntnisse in Datenbankabfragesprachen. Sie brauchen innerhalb kurzer Zeit Antworten auf geschäftsbezogene Fragen, z. B. Prognosen, Qualitätsanalysen, Ertragsanalysen, Liquiditätsberechnungen. Es handelt sich um ad-hoc-Abfragen, die i. a. nicht vorformuliert sind. Man benötigt dazu eine geeignete Methode und deren Realisierung in Form von Abfrage- und Auswerte-Werkzeugen.
Viele Analysten und Werkzeughersteller haben *Business Intelligence* als Oberbegriff für entscheidungsunterstützende Werkzeuge geprägt. Unter diesem Namen fassen sie Data Mining, OLAP, Data Warehouse und ähnliches zusammen.

7.4.1 DSS und OLAP

In einfachen Fällen genügen gängige Hilfen wie Tabellenkalkulation u. ä. Für komplexe Abfragen werden *OLAP-Methoden* eingesetzt; damit kann z. B. die Frage beantwortet werden: „Welche Produktreihe war im letzten Quartal in einer bestimmten Region am erfolgreichsten?“ Um solche Abfragen auf einfache Art formulieren zu können und damit die Benutzung des DWH jedem Manager selbst zu ermöglichen, gibt es Software-Werkzeuge. Sie sind auf dem PC des Managers implementiert bzw. von dort aufrufbar. Er kann nach kurzer Einarbeitung seine Fragen schnell stellen und bekommt sie in einer ihm verständlichen und geläufigen Darstellung beantwortet.
Diese Werkzeuge bieten die Möglichkeit, die Daten einer bestimmten Dimensionsebene herauszuschneiden oder zu drehen (*Slice and Dice*), siehe Abbildung „Das multidimensionale Datenmodell“. Man kann sich auch von einer Hierarchieebene zur nächst niedrigeren oder höheren bewegen *(Drill down, Roll up*).
Man sieht nun die Bedeutung des Wortes Data Warehouse (engl. Warehouse = Lagerhaus): Zusammengefaßte wie auch detaillierte Informationen werden bereitgehalten und sind mit einfachen Mitteln ohne spezielle Datenbank- oder DV-Kenntnisse zugreifbar, also Selbstbedienung wie an den Regalen eines wohlsortierten Waren/Lagerhauses. Das DWH kann man auch als „nachträgliche“ Datenbank auffassen, die Informationen des aktuellen Geschehens samt Zeitverlauf aus mehreren Quellen in aufbereiteter, konzentrierter und konsistenter Form bereithält und leicht abrufbar macht.
Solche DWH-Datenbanken und Anwendungen bezeichnet man im Gegensatz zu OLTP-Systemen mit *Decision Support System* (DSS).

7.4.2 Data Mining

Eine besonders zukunftsträchtige Anwendung der DSS ist die Suche nach noch unbekannten Geschäftszusammenhängen in Unternehmen. Ein typisches Beispiel ist die gezielte Analyse von Neukunden, mit denen das Unternehmen noch wenig Erfahrung hat. Ein anderes Beispiel ist die Suche nach „Bei-Käufen“, d. h. die Suche nach Artikeln, die der Kunde beim Kauf des von ihm eigentlich gesuchten Kaufgegenstandes spontan mitnimmt, obwohl er den Kauf dieses Artikels gar nicht vorhatte. Dazu verwendet man eine spezielle Analysetechnik, Assoziation und Affinität, die im Einzelhandel verbreitet ist.

Beispiel:

> In der US-Supermarktkette Wal-Mart hat man unter Nutzung von Data Mining herausgefunden, daß es nach Büroschluß besonders günstig ist, neben Windeln Bier zu plazieren. Durch Warenkorbanalyse war man treusorgenden Vätern auf die Spur gekommen, die am Feierabend häufig Windeln und auch den Sixpack für sich selbst mitnehmen.

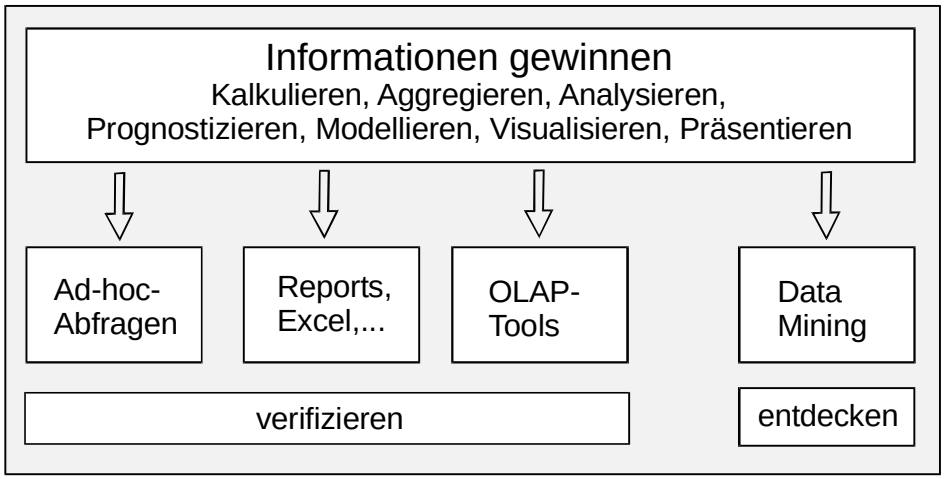


Abb. 12: Informationsgewinnung im DSS

Weitere Beispiele sind folgende Fragestellungen:

- Was kauft ein Kunde, obwohl er bei seiner Bank mit den Hypothekenraten schon beträchtlich in Verzug ist?
- Welche Kunden werden mit der größten Wahrscheinlichkeit auf eine Direct Mail Aktion antworten?
- Bei welchen Kunden ist die Wahrscheinlichkeit am größten, daß sie kündigen?
- Welche Merkmale haben Kunden, die gekündigt haben?
- Welche Haushalte werden mit hoher Wahrscheinlichkeit Mitglied in einem Buchclub?

Man nennt solche Suchvorgänge *Data Mining* (engl. mining = Bergbau, Schürfung). Auch das Data Mining erfolgt werkzeuggestützt. Die Tools ermöglichen Data Mining ohne besondere DV- oder Datenbank-Kenntnisse. Das Data Mining – Verfahren untersucht die Vergangenheit, entwirft davon ausgehend ein Modell bzw. ein Muster, nach dem aktuelle Daten verglichen werden. Das Ergebnis ist eine Prognose für die Zukunft, z. B. in Form einer Segmentierung, bei der die Gruppenmitglieder (Kunden) ähnliche Wahrscheinlichkeiten für ein bestimmtes Merkmal haben (z. B. das Merkmal, daß sie bald kündigen werden). Dabei werden auch sozio-demographische (Alter, Daten der Kunden für die Segmentierung herangezogen.
Insgesamt unterscheidet man fünf verschiedene Ergebnistypen bei Data Mining – Analysen:

- Beziehungen / Assoziationen
- Zeitreihenmuster
- Segmentierung / Clusterung / Klassifizierung
- Vorhersagen.

Zum Data Mining gehören eine Vielzahl von statistischen und anderen Verfahren wie

- Korrelationsanalyse
- Regressionsanalyse
- Clusteranalyse
- Neuronale Netze, evtl. mit Fuzzy Logic
- Entscheidungsbäume
- Regelinduktion
- Assoziation und Affinität

Hier sei auf die Literatur verwiesen. Typische Data Mining – Projekte werden unter anderem in folgenden Bereichen durchgeführt:

- Database Marketing wie Planung von Direktmarketing-Aktionen und Response-Analyse
- Kundenbindung bei der Analyse und Vorhersage der Abwanderungen von Kunden (z. B. bei Mobilfunk-Gesellschaften)
- Warenkorb-Analysen zur Plazierung gekoppelter Produkte in Regalen und Katalogen
- Risikomanagement wie Bonitätsbeurteilung oder Betrugserkennung im Kreditkartengeschäft, bei Versicherungsabschlüssen oder im Gesundheitswesen
- Prognosen
- Vertriebsunterstützung
- Analyse von technischen Prozessen zur Ermittlung der wichtigsten Einflußgrößen, z. B. der zu erwartenden Produktqualität
- Erstellung von Prozeßmodellen zu Simulation.

Abkürzungsverzeichnis

4GL	4th Generation Language
ABFAK	Auftragsbearbeitung und Fakturierung
API	Application Programming Interface
AV	Auftragsverarbeitung
BLOB	Binary Large Objects
BOT	Begin of Transaction
C/S	Client/Server
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CIM	Computer Integrated Manufacturing
CLI	Common Language Interface
CW	Computer Woche
DB(S)	Datenbank(system)
DBA	Datenbank-Administrator
DBMS	Data Base Management System (Synonym zu DBVS)
DBVS	Datenbankverwaltungssystem
DCL	Data Control Language
DD	Data Dictionary
DDL	Data Definition Language
DLL	Dynamic Link Library
DML	Data Manipulation Language
DSS	Decision Support System
DV	Datenverarbeitung
DWH	Data Warehouse
EDV	Elektronische Datenverarbeitung
EIS	Enterprise Information System
EOT	End of Transaction
ERM	Entity-Relationship-Modell
ESQL	Embedded SQL
FT	Fehlertoleranz
ggfs.	gegebenenfalls
GI	Gesellschaft für Informatik
HTML	Hypertext Transfer Markup Language
HW	Hardware
i. a.	im allgemeinen
i. d. R.	in der Regel
IFB	Informatik Fachberichte
IO	Input/Output
IS-A	steht für eine Teilmengenbeziehung
ISO	International Standardization Organisation
i. w.	im wesentlichen
ISQL	Interactive SQL
JDBC	Java Database Connectivity

JSQL	Java SQL
LAN	Local Area Network
LOG	Logging (Protokollieren der Datenbank-Zugriffe)
MIS	Management Information System
MOLAP	Multidimensional OLAP
MPP	Massively Parallel Processing
MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
NF^2	Non First Normal Form
ODBC	Open Database Connectivity
o. g.	oben genannte
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
OODB	Objektorientierte DB
PPS	Produktionsplanungs- und -steuerungssystem
QbF/QbE	Query by Forms / Query by Example
ROLAP	Relational OLAP
SMP	Symmetric Multi Processing
SQL	Structured Query Language
TCP/IP	Transport Control Protocol / Internet Protocol
TPC	Transaction Processing Performance Council
u. a.	und andere bzw. unter anderem
VDB	Verteilte Datenbank
WAL	Write Ahead Log
WAN	Wide Area Network
WWW	World Wide Web

Abbildungs- und Tabellenverzeichnis

Literaturempfehlungen

Behm96	W. Behme: Das Data Warehouse als zentrale Datenbank für Managementinformationssysteme in Hannig, Uwe (Hrsg.), Data Warehouse und Managementinformationssysteme, Schäffer-Poeschel, Stuttgart 1996
Berk92	T. Berkel: Die relationale Datenbanksprache SQL, FU Hagen, Addison Wesley, Bonn 1992
Celk95	Joe Celco's: SQL for Smarties, Advanced SQL-Programming, Morgan Kaufmann Publishers, San Francisco 1995
Bull95	H.-J. Bullinger: Data Warehouse und seine Anwendungen: Data Mining, OLAP und Führungsinformationen im betrieblichen Einsatz, IAO, Stuttgart 1995
Chen76	P.P.S Chen: The Entity-Relationship Model: Towards a Unified View of Data, ACM Transactions on Database Systems, 1: 9-36, 1976 (Chen ist der Erfinder des ERM)
Codd70	E.F. Codd: A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, Vol 13, No 6, June 1970 (Codd ist der Erfinder der relationalen DBVSe)
Date95	C.J. Date: An Introduction to Dabase Systems, sixth ed., Addison-Wesley, 1995
Dir97	W. Dirlewanger: Das Data Warehouse, K. G. Saur Verlag, München 1997, PIK 2/97
Eis96	R. Eisele: Der Stand der Standardisierung von SQL3, bei OO ist noch viel zu tun, DATENBANK FOKUS, 11/96
FKU91	H. Finkenzeller, U. Kracke, M. Unterstein: Systematischer Einsatz von SQL-ORACLE, Addison Wesley, 1991
Hald95	A. Hald, W. Nevermann: Datenbank-Engineering für Wirtschaftsinformatiker, vieweg, 1995

Heu92 A. Heuer: Objektorientierte Datenbanken, Addison-Wesley, Bonn 1992

Inm96: W.H. Inmon: Building the Data Warehouse, John Wiley & Sons, Inc., Second Edition, 1996

Melt93 J. Melton e.a.: Understanding the new SQL, Morgan Kaufmann Publishers, San Francisco 1993

Muck96 H. Mucksch: Das Data-Warehouse Konzept, Gabler, Wiesbaden 1996

Scheer95 A.-W. Scheer: Wirtschaftsinformatik, Referenzmodelle für industrielle Geschäftsprozesse, Springer 1995

Schw92 H. Schwinn: Relationale Datenbanksysteme, Hanser München Wien 1992

SQL89 International Standards Organization: Database Language SQL with integrity enhancement, Document ISO-9075-1989(E). Also available as American National Standards Institute Document X3.135-1989

SQL92 SQL2-Sprachstandard; Document ISO/IEC-9075: 1992

Stau91 J. L. Staud: Online Datenbanken, Addison-Wesley, Bonn 1991

Teor94 T.J. Teory: Database Modeling & Design, Second Edition, Morgan Kaufmann, San Francisco, 1994

Voss94 G. Vossen: Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme, Addison-Wesley, 1994

Index